Содержание

Аннотация к дисциплине

Предметом изучения являются модели данных, базы и банки данных. Объектами изучения выступают принципы построения баз и банков данных, основы проектирования баз данных и управления данными.

Место дисциплины в учебном процессе Академии.

Дисциплина включена в учебные планы Академии по всем программам подготовки специалистов по специальностям «Прикладная информатика (по областям) и Информационные системы и технологии. Дисциплина относится к циклу общепрофессиональных дисциплин и базируется на знании цикла естественно-научных дисциплин, в том числе математического анализа, информатики и основ программирования. Программа дисциплины ориентирована на формирование базовых профессиональных знаний, умений и навыков, развитие которых предполагается как в дисциплинах общепрофессионального, так и в дисциплинах специального цикла. Успешное усвоение материала данного курса поможет формированию целостного системного представления задач профессиональной деятельности.

Цель и задачи дисциплины.

Цель заключается в ознакомлении студентов с основными принципами организации баз и банков данных; с моделями данных; получении теоретических знаний и практических навыков по основам создания баз данных; в ознакомлении с современными СУБД и перспективами их развития.

Задачи:

- овладение понятийным аппаратом, описывающим различные аспекты теории баз данных и области применения баз и банков данных;
- ознакомление с историей, современными проблемами и перспективами развития баз и банков данных, СУБД, СУРБД;
- усвоение основных принципов построения различных моделей предметной области, методов и средств их создания, внедрения, анализа и сопровождения;
- приобретение опыта анализа предметной области и учета ее специфики при принятии проектных решений в процессе создания и использования баз и банков данных.

В результате изучения курса студент должен:

знать:

- основные модели данных и их организацию;
- средства и методику анализа и описания предметной области;
- основные понятия баз данных, банков данных и знаний, СУБД;
- типологию баз данных, банков данных и систем управления базами данных;
- основные принципы построения баз данных, банков данных и систем управления базами данных;
- об основных компонентах баз и банков данных, а также систем управления базами данных;
- методы построения баз данных;
- сущность информационного поиска, его задачи, объекты, виды, способы и технологии реализации;
- принципы построения языков запросов и манипулирования данными;
- о тенденциях развития баз данных, банков данных и систем управления базами данных;

уметь:

- пользоваться понятийным аппаратом, описывающим различные аспекты теории баз данных,
- анализировать и описывать информационные и функциональные процессы предметной области,
- осуществлять обоснованный выбор вида, метода и технологии создания и применения моделей предметной области
- разрабатывать концептуальные модели реальных проблемных областей, реализовывать их на ЭВМ,
- создавать различные информационные структуры средствами современных СУБД,
- строить SQL - запросы отбора данных и манипулирования ими;
- создавать базы данных средствами современных СУБД;

приобрести навыки:

- анализа информационных и функциональных процессов предметной области;
- выбор вида, метода и технологии создания и применения моделей предметной области;
- разработки концептуальных моделей реальных проблемных областей, реализации их на ЭВМ;
- создавать различные информационные структуры средствами современных СУБД,
- построения SQL - запросов отбора данных и манипулирования ими;
- создания баз данных средствами современных СУБД.

Тема 1. Введение в теорию баз данных

Вопрос 1. Основные понятия.

В настоящее время наибольшее распространение получили реляционные базы данных, в основе которых лежит представление данных в виде таблиц. Табличное представление данных понятно и привычно пользователю и не зависит от уровня его подготовки в IT-области и опыта работы с базами данных, что позволяет работать с базой данных практически любому. Неоспоримые преимущества дает относительная легкость добавления новых таблиц и связей между ними в отличие, например, от иерархической модели данных.

Для успешной работы с реляционными базами данных необходимо уметь оперировать следующими основными понятиями: первичный ключ, внешний ключ, домен, кортеж, кардинальность, атрибут, степень отношения, поле, запись, форма, запрос, отчет.

Первичный ключ [1] ***(идентификатор)*** – это столбец или некоторое подмножество столбцов, которые уникально, т.е. единственным образом определяют строки. первичный ключ не может быть полностью или частично пустым, т.е. иметь значение null.

Внешний ключ - это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. *Внешний ключ* таблицы является ссылкой на первичный ключ другой таблицы.

Домен – это совокупность допустимых значений, из которой берутся значения соответствующих атрибутов определенного отношения. С точки зрения программирования домен - это тип данных, определяемый системой (стандартный) или пользователем.

Кортеж – это строка или запись в таблице.

Кардинальность – это количество строк в таблице.

Атрибут – свойство, которое в реляционной модели становится столбцом таблицы, а в случае конкретного значения – полем.

Степень отношения – это количество столбцов.

Поле – элемент таблицы, содержащий данные определенного рода, например, фамилии. В режиме таблицы поле представляет собой ячейку.

Запись – полный набор данных об определенном объекте. В таблице запись изображается как строка.

Форма – объект Access, предназначенный в основном для ввода данных. В форме можно разместить элементы управления, применяемые для ввода, изображения и изменения данных в полях таблицы.

Запрос – объект, позволяющий получить нужные данные из одной или нескольких таблиц.

Отчет – объект БД Access, предназначенный для вывода сформированных данных на печать.

Типы отношений.

Существует три типа отношений между таблицами: *Один-ко-многим, Многие-ко-многим* и *Один-к-одному.*

Наиболее часто используется тип связи между таблицами «*Один-ко-многим»*. В этом случае каждой записи в таблице «А» может соответствовать несколько записей в таблице «В» (поля с этими записями называют *Внешними ключами),* а запись в таблице «В» не может иметь более одной соответствующей ей записи в таблице «А». Такая связь создается в случае, когда только одно из полей таблицы является ключевым или имеет уникальный индекс, т.е. значения в нем не повторяются.

При связи «*Многие-ко-многим»* одной записи в таблице «А» может соответствовать несколько записей в таблице «В», а одной записи в таблице «В» – несколько записей в таблице «А». Такая схема реализуется только с помощью третьей (связующей) таблицы, ключ которой состоит, по крайней мере, из двух полей; одно из них является общим с таблицей «А», другое – с таблицей «В». Она фактически представляет две связи типа «*один-ко-многим»* через третью таблицу.

При связи «*Один-к-одному»* запись в таблице «А» может иметь только одну связанную запись в таблице «В» и наоборот.

Банк данных (БНд) - это система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных.

Под *базой данных (БД)* обычно понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области.

Характерной чертой баз данных является *постоянство:* данные *постоянно* накапливаются и используются; состав и структура данных, необходимых для решения тех или иных прикладных задач обычно *постоянны* и стабильны во времени; отдельные или даже все элементы данных могут меняться - но и это есть проявление постоянства - *постоянная* актуальность.

Система управления базами данных (СУБД) - это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе банка данных выделяют *архивы*. Основанием для этого является особый режим использования данных - только часть данных находится под оперативным управлением СУБД. Все остальные данные (собственно архивы) обычно располагаются на носителях, оперативно не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в архивы. Банки данных могут не иметь архивов, но если они есть, то в состав банка данных может входить и система управления архивами.

Проблемы совместного использования данных и периферийных устройств компьютеров и рабочих станций породили модель вычислений, основанную на концепции файлового сервера - сеть создает основу для коллективной обработки, сохраняя простоту использования персонального компьютера, позволяет совместно использовать данные и периферию.

В этом смысле главной отличительной чертой баз данных является использование *централизованной системы управления данными*, причем как на уровне файлов, так и на уровне элементов данных.

Достоинства централизованного хранение совместно используемых данных:

- сокращение затрат на создание,
- поддержание данных в актуальном состоянии,
- сокращение избыточности информации,
- упрощение процедур поддержания непротиворечивости и целостности данных.

Эффективное управление внешней памятью является основной функцией СУБД.

Специализированные средства СУБД настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи уже потому, что их выполнение будет занимать слишком много времени.

Специализированные функции:

- построение индексов,
- буферизация данных,
- организация доступа
- оптимизация запросов
- являются невидимыми для пользователя.

Они *обеспечивают независимость между логическим и физическим уровнями системы:* прикладной программист не должен писать программы индексирования, распределять память на диске и т.д.

Вопрос 2. Основные функции СУБД.[2]

В прикладной программе, использующей при решении задачи один или несколько отдельных файлов, за сохранность и достоверность данных отвечал программист, работающий с этой задачей. Использование базы данных предполагает работу с ней нескольких прикладных программ, решающих задачи разных пользователей.

Естественно, что за сохранность и достоверность интегрированных данных программист, решающий одну из прикладных задач, отвечать уже не может. Кроме того, расширение круга решаемых с использованием базы данных задач может приводить к появлению новых типов записей и отношений между ними. Такое изменение структуры базы данных не должно вести к изменению множества ранее разработанных и успешно функционирующих прикладных программных систем, работающих с базой данных. С другой стороны, возможное изменение любой из прикладных программ, в свою очередь, не должно приводить к изменению структуры данных. Все вышесказанное обусловливает необходимость отделения данных от прикладных программ.

Роль интерфейса между прикладными программами и базой данных, обеспечивающего их независимость, играет программный комплекс – система управления базами данных (СУБД) (рис. 1).

СУБД – программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).

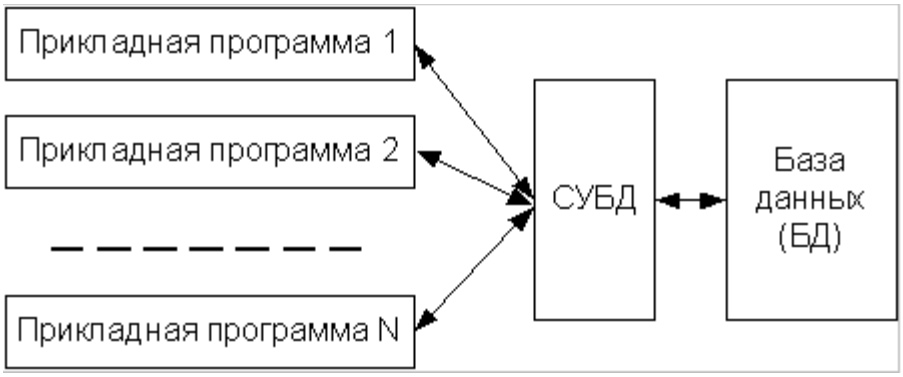


Рис. 1. Обеспечение независимости прикладных программ и базы данных

Определим еще одно понятие.

Банк данных – система языковых, алгоритмических, программных, технических и организационных средств поддержки интегрированной совокупности данных, а также сами эти данные, представленные в виде баз данных.

Перечислим основные *функции системы управления базами данных*.

1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки.

Как правило, создание структуры **базы данных** происходит в режиме диалога. СУБД последовательно запрашивает у пользователя необходимые данные. В большинстве современных СУБД **база данных** представляется в виде совокупности таблиц. Рассматриваемая функция позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы.

2. Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).

Такие возможности в СУБД представляются либо на основе использования специального языка программирования, входящего в состав СУБД, либо с помощью графического интерфейса. Для клиент-серверных СУБД существуют средства, позволяющие выполнять запросы, и программные средства, позволяющие создавать графический интерфейс пользователя.

3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).

Важнейшим свойством СУБД является возможность поддерживать два независимых взгляда на базу данных – «взгляд пользователя», воплощаемый в логическом представлении данных, и его отражения в прикладных программах; и «взгляд системы» – физическое представление данных в памяти ЭВМ. Обеспечение логической независимости данных предоставляет возможность изменения (в определенных пределах) логического представления базы данных без необходимости изменения физических структур хранения данных. Таким образом, изменение логического представления данных в прикладных программах не приводит к изменению структур хранения данных. Обеспечение физической независимости данных предоставляет возможность изменять (в определенных пределах) способы организации базы данных в памяти ЭВМ не вызывая необходимости изменения «логического» представления данных. Таким образом, изменение способов организации базы данных не приводит к изменению прикладных программ.

4. Защита логической целостности **базы данных**.

Основной целью реализации этой функции является повышение достоверности данных в базе данных. Достоверность данных может быть нарушена при их вводе в БД или при неправомерных действиях процедур обработки данных, получающих и заносящих в БД неправильные данные. Для повышения достоверности данных в системе объявляются так называемые ограничения целостности, которые в определенных случаях «отлавливают» неверные данные. Так, во всех современных СУБД проверяется соответствие вводимых данных их типу, описанному при создании структуры. Система не позволит ввести символ в поле числового

типа, не позволит ввести недопустимую дату и т.п. В развитых системах ограничения целостности описывает программист, исходя из содержательного смысла задачи, и их проверка осуществляется при каждом обновлении данных. Более подробно разные аспекты логической целостности базы данных будут рассматриваться в последующих разделах.

5. Защита физической целостности.

При работе ЭВМ возможны сбои в работе (например, из-за отключения электропитания), повреждение машинных носителей данных. При этом могут быть нарушены связи между данными, что приводит к невозможности дальнейшей работы. Развитые СУБД имеют средства восстановления базы данных. Важнейшим используемым понятием является понятие «транзакции». Транзакция – это единица действий, производимых с базой данных. В состав транзакции может входить несколько операторов изменения базы данных, но либо выполняются все эти операторы, либо не выполняется ни один. СУБД, кроме ведения собственно базы данных, ведет также журнал транзакций.

Необходимость использования транзакций в базах данных проиллюстрируем на упрощенном примере. Предположим, что база данных используется в некотором банке и один из клиентов желает перевести деньги на счет другого клиента банка. В базе данных хранится информация о количестве денег у каждого из клиентов. Нам нужно сделать два изменения в базе данных – уменьшить сумму денег на счете одного из клиентов и, соответственно, увеличить сумму денег на другом счете. Конечно, реальный перевод денег в банке представляет собой гораздо более сложный процесс, затрагивающий много таблиц, а возможно, и много баз данных. Однако суть остается та же – нужно либо совершить все действия (увеличить счет одного клиента и уменьшить счет другого), либо не выполнить ни одно из этих действий. Нельзя уменьшить сумму денег на одном счете, но не увеличить сумму денег на другом. Предположим также, что после выполнения первого из действий (уменьшения суммы денег на счете первого клиента) произошел сбой. Например, могла прерваться связь клиентского компьютера с базой данных или на клиентском компьютере мог произойти системный сбой, что привело к перезагрузке операционной системы. Что в этом случае стало с базой данных? Команда на уменьшение денег на счете первого клиента была выполнена, а вторая команда – на увеличение денег на другом счете – нет, что привело бы к противоречивому, неактуальному состоянию базы данных.

Использование механизма транзакций позволяет находить решение в этом и подобных случаях. Перед выполнением первого действия выдается команда начала транзакции. В транзакцию включается операция снятия денег на одном счете и увеличения суммы на другом счете. Оператор завершения транзакций обычно называется COMMIT. Поскольку после выполнения первого действия транзакция не была завершена, изменения не будут внесены в базу данных. Изменения вносятся (фиксируются) только после

завершения транзакции. До выдачи данного оператора сохранения данных в базе не произойдет.

В нашем примере, поскольку оператор фиксации транзакции не был выдан, база данных «откатится» в первоначальное состояние – иными словами, суммы на счетах клиентов останутся те же, что и были до начала транзакции. Администратор базы данных может отслеживать состояние транзакций и в необходимых случаях вручную «откатывать» транзакции. Кроме того, в очевидных случаях СУБД самостоятельно принимает решение об «откате» транзакции.

Транзакции не обязательно могут быть короткими. Бывают транзакции, которые длятся несколько часов или даже несколько дней. Увеличение количества действий в рамках одной транзакции требует увеличения занимаемых системных ресурсов. Поэтому желательно делать транзакции по возможности короткими. В журнал транзакций заносятся все транзакции – и зафиксированные, и завершившиеся «откатом». Ведение журнала транзакций совместно с созданием резервных копий базы данных позволяет достичь высокой надежности базы данных.

Предположим, что база данных была испорчена в результате аппаратного сбоя компьютера, на котором был установлен сервер СУБД. В этом случае нужно использовать последнюю сделанную резервную копию базы данных и журнал транзакций. Причем применить к базе данных нужно только те транзакции, которые были зафиксированы после создания резервной копии. Большинство современных СУБД позволяют администратору воссоздать базу данных исходя из резервной копии и журнала транзакций. В таких системах в определенный момент БД копируется на резервные носители. Все обращения к БД записываются программно в журнал изменений. Если база данных разрушена, запускается процедура восстановления, в процессе которой в резервную копию из журнала изменений вносятся все произведенные изменения.

6. Управление полномочиями пользователей на доступ к базе данных.

Разные пользователи могут иметь разные полномочия по работе с данными (некоторые данные должны быть недоступны; определенным пользователям не разрешается обновлять данные и т.п.). В СУБД предусматриваются механизмы разграничения полномочий доступа, основанные либо на принципах паролей, либо на описании полномочий.

7. Синхронизация работы нескольких пользователей.

Достаточно часто может иметь место ситуация, когда несколько пользователей одновременно выполняют операцию обновления одних и тех же данных. Такие коллизии могут привести к нарушению логической целостности данных, поэтому система должна предусматривать меры, не допускающие обновление данных другим пользователям, пока работающий с этими данными пользователь полностью не закончит с ними работать. Основным используемым здесь понятием является понятие

«блокировка». Блокировки необходимы для того, чтобы запретить различным пользователям возможность одновременно работать с базой данных, поскольку это может привести к ошибкам.

Для реализации этого запрета СУБД устанавливает блокировку на объекты, которые использует транзакция. Существуют разные типы блокировок – табличные, страничные, строчные и другие, которые отличаются друг от друга количеством заблокированных записей. Чаще других используется строчная блокировка – при обращении транзакции к одной строке блокируется только эта строка, остальные строки остаются доступными для изменения.

Таким образом, процесс внесения изменений в базу данных состоит из следующей последовательности действий: выдается оператор начала транзакции, выдается оператор изменения данных, СУБД анализирует оператор и пытается установить блокировки, необходимые для его выполнения, в случае успешной блокировки оператор выполняется, затем процесс повторяется для следующего оператора транзакции. После успешного выполнения всех операторов внутри транзакции выполняется оператор фиксации транзакции. СУБД фиксирует изменения, сделанные транзакцией, и снимает блокировки. В случае неуспеха выполнения какого-либо из операторов транзакция «откатывается», данные получают прежние значения, блокировки снимаются.

8. Управление ресурсами среды хранения.

БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память) и удаляются данные (освобождается память). СУБД выделяет ресурсы памяти для новых данных, перераспределяет освободившуюся память, организует ведение очереди запросов к внешней памяти и т.п.

9. Поддержка деятельности системного персонала.

При эксплуатации базы данных может возникать необходимость изменения параметров СУБД, выбора новых методов доступа, изменения (в определенных пределах) структуры хранимых данных, а также выполнения ряда других общесистемных действий. СУБД предоставляет возможность выполнения этих и других действий для поддержки деятельности БД обслуживающему БД системному персоналу, называемому администратором БД.

Тема 2. Принципы построения баз данных. Модели и структуры данных[3]

Вопрос 1. Принципы построения баз данных, банка данных, банка знаний.

Развитие теории и практики создания информационных систем, основанных на концепции баз данных, создание унифицированных методов и средств организации и поиска данных позволяют хранить и обрабатывать информацию о все более сложных объектах и их взаимосвязях, обеспечивая многоаспектные информационные потребности различных пользователей.

Основные требования, предъявляемые к банкам данных, можно сформулировать следующим образом.

- **Многократное использование данных:** пользователи должны иметь возможность использовать данные различным образом.
- **Простота:** пользователи должны иметь возможность легко узнать и понять, какие данные имеются в их распоряжении.
- **Легкость использования:** пользователи должны иметь возможность осуществлять (процедурно) простой доступ к данным, при этом все сложности доступа к данным должны быть скрыты в самой системе управления базами данных.
- **Гибкость использования:** обращение к данным или их поиск должен осуществляться с помощью различных методов доступа.
- **Быстрая обработка запросов на данные:** запросы на данные, в том числе незапланированные, должны обрабатываться с помощью высокоуровневого языка запросов, а не только прикладными программами, написанными с целью обработки конкретных запросов (разработка таких программ в каждом конкретном случае связана с большими затратами времени). Пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно «декларативным», т. е., упор должен быть сделан на «что», а не на «как». Кроме того, средства обработки запросов не должно зависеть от приложения, т. е., оно должно работать с любой возможной базой данных.
- **Язык взаимодействия конечных пользователей с системой** должен обеспечивать конечным пользователям возможность получения данных без использования прикладных программ.
- **База данных - это основа для будущего наращивания прикладных программ**: базы данных должны обеспечивать возможность быстрой и дешевой разработки новых приложений.
- **Сохранение затрат умственного труда:** существующие программы и логические структуры данных (на создание которых обычно затрачивается много человеко-лет) не должны переделываться при внесении изменений в базу данных.
- **Наличие интерфейса прикладного программирования:** Прикладные программы должны иметь возможность просто и эффективно выполнять запросы на данные; программы должны быть изолированы от расположения файлов и способов адресации данных.
- **Распределенная обработка данных:** система должна функционировать в условиях вычислительных сетей и обеспечивать

эффективный доступ пользователей к любым данным распределенной БД, размещенным в любой точке сети.

- **Адаптивность и расширяемость:** с целью увеличения производительности база данных должна быть настраиваемой, причем настройка не должна вызывать перезапись прикладных программ. Кроме того, поставляемый с СУБД набор предопределенных типов данных должен быть расширяемым - в системе должны быть средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов.
- **Контроль за целостностью данных: с**истема должна осуществлять контроль ошибок в данных и должна выполнять проверку взаимного логического соответствия данных.
- **Восстановление данных после сбоев:** Автоматическое восстановление без потери данных транзакции. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных.
- **Вспомогательные средства** должны позволять разработчику или администратору базы данных предсказать и оптимизировать производительность системы.
- **Автоматическая реорганизация и перемещение:** система должна обеспечивать возможность перемещения данных или автоматическую реорганизация физической структуры.

Вопрос 2. Компоненты банка данных.

Определение банка данных предполагает, что с функционально-организационной точки зрения банк данных является сложной человеко-машинной системой, включающей в себя все подсистемы, необходимые для надежного, эффективного и продолжительного во времени функционирования.

В структуре банка данных выделяют следующие компоненты (подсистемы):

- информационная база;
- лингвистические средства;
- программные средства;
- технические средства;
- организационно-административные подсистемы и нормативно-методическое обеспечение.

Информационная база.

Данные, отражающие состояние определенной предметной области и используемые информационной системой, принято называть *информационной базой.*

Информационная база состоит из двух компонентов:

1) коллекции записей собственно данных
2) описания этих данных -метаданных.

Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Уже из определения базы данных и приведенных ранее основных требований следует, что данные могут использоваться (т.е., представляться) по-разному.

С одной стороны, разные прикладные задачи требуют разных наборов данных, в совокупности обеспечивающих функциональную полноту информации, а с другой - они должны быть различны для различных категорий субъектов (разработчиков или пользователей).

Также должны быть различными и способы описания самих данных, их природы, формы хранения, условий взаимной непротиворечивости.

В литературе по базам данных упоминаются три уровня представления данных - концептуальный, внутренний и внешний (рис. 2.).

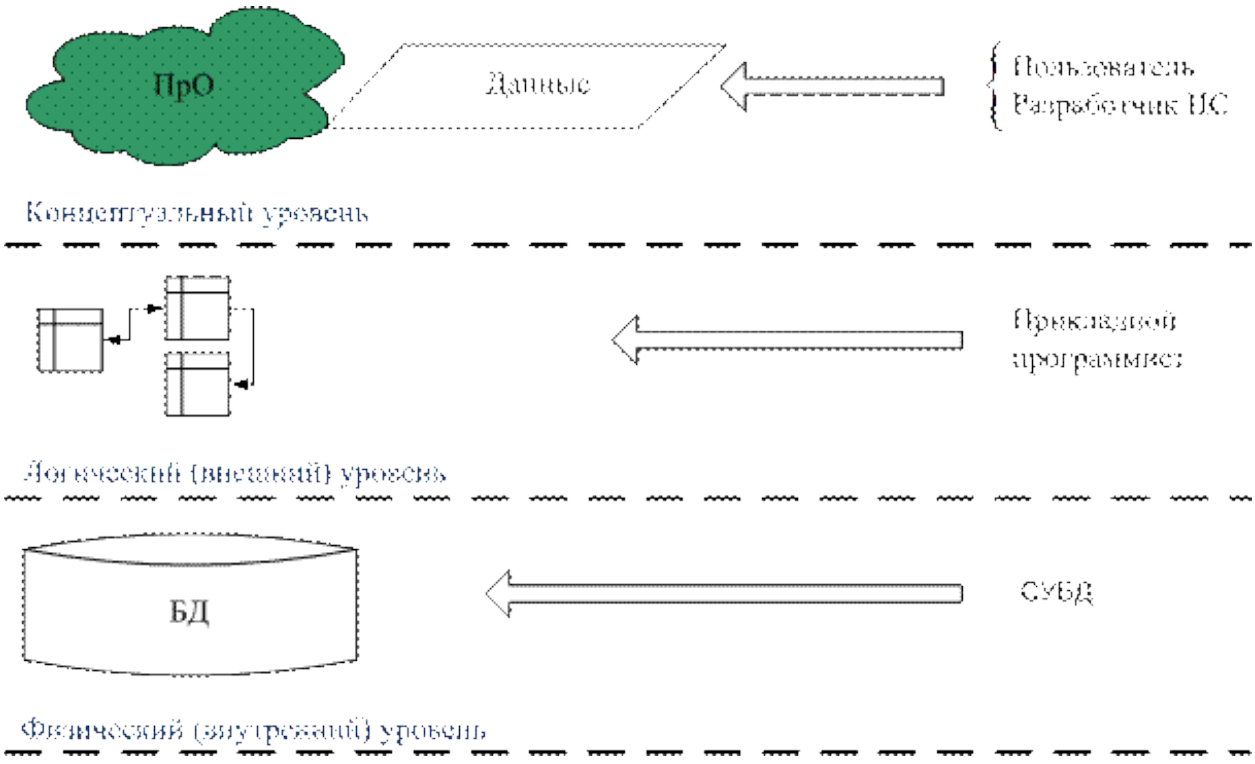


Рис. 2. Уровни представления данных

Эти уровни представлений введены исходя из различного рассмотрения БД. Например, прикладному программисту требуются не все данные БД, а только некоторая их часть, используемая в его программе. Внешний уровень представления обеспечивает именно эту форму обмена данными.

Внутренний уровень - глобальное представление БД, определяет необходимые условия для организации хранения данных на внешних запоминающих устройствах.

Описание БД на **концептуальном уровне п**редставляет собой обобщенный взгляд на данные с позиций предметной области (разработчика приложений, пользователя или внешней информационной системы).

Внешний уровень представления данных не затрагивает физической организации (размещения) данных во внешней памяти, поэтому его называют иногда логическим уровнем. Соответственно внутренний уровень называют физическим уровнем.

Лингвистические средства.

Многоуровневое представление БД предполагает соответствующие описания данных на каждом уровне и согласование одних и тех же данных на разных уровнях.

С этой целью в состав СУБД включаются специальные языки для описания представлений внутреннего и внешнего уровней. Кроме того, СУБД должна включать в себя язык манипулирования данными (ЯМД).

Желательно, также наличие тех или иных дополнительных сервисных средств, например, средств генерации отчетов.

Работа с базами данных предполагает несколько этапов:

- описание БД;
- описание частей БД, необходимых для конкретных приложений (задач, групп задач);
- программирование задач или описание запросов в соответствии с правилами конкретного языка и использованием языковых конструкций для обращения к БД;
- загрузка БД и т. д.

Для выражения обобщенного взгляда на данные применяют *язык описания данных (ЯОД)* внутреннего уровня, включаемый в состав СУБД. (Отсюда следует, что одна и та же БД может описываться по-разному на ЯОД различных СУБД.) Описание представляет собой модель данных и их отношений, т. е. структур, из которых образуется БД.

ЯОД позволяет определять схемы базы данных, характеристики хранимых и виртуальных данных и параметры организации их хранения в памяти, и может включать в себя средства поддержки целостности базы данных, ограничения доступа, секретности.

ЯМД обычно включает в себя средства запросов к базе данных и поддержания базы данных (добавление, удаление, обновление данных, создание и уничтожение БД, изменение определений БД, обеспечение запросов к справочнику БД).

Исторически первым типом структур данных, который был включен в языки программирования, была *иерархическая структура*. Некоторые

ранние СУБД также предполагали использование в качестве основной модели иерархические структуры типа дерева. Основанием для такого выбора было удобство представления (моделирования) естественных иерархических структур данных, существующих, например, в организациях.

В ряде предметных областей структура данных имеет более сложный вид, в котором поддерживаются связи типа «многие к одному», и которые могут быть представлены ориентированным графом. Такие структуры называют сетевыми.

Для управления БД сетевой структуры международной ассоциацией Кодасил была предложена обобщенная архитектура системы с ЯОД схемы (модели БД) и подсхемы (модели части БД для конкретного приложения), а также ЯМД для оперирования с данными БД в прикладных программах.

В настоящее время разработаны десятки языков, основанных на реляционном исчислении, различие которых обусловлено особенностями математических теорий, положенных в основу их построения. Среди этих языков, можно выделить языки, базирующиеся на *C*-исчислении, предложенном Коддом, и *P*-исчислении, предложенном Пиротти.

Функциональные характеристики языков отражают возможности:

- описания данных,
- средств представления запроса,
- обновления,
- поддержки целостности и секретности,
- включения в языки программирования,
- управления форматом ответов,
- средств запроса к словарю данных БД и т.д.

Качественные характеристики языков запросов могут определяться такими свойствами, как:

- полнота,
- селективная мощность,
- простота изучения и использования,
- степень процедурности
- степень модульности,
- унифицированность,
- производительность
- эффективность.

Рассмотрим некоторые из этих понятий.

Селективная мощность языков запросов характеризует возможность выбора данных по разным критериям. Данное понятие плохо поддается формализации: можно сказать, что язык с большей селективной мощностью позволяет сформулировать большинство запросов так, что ответ на них содержит меньше ненужных данных. Языки, обладающие малой селективной мощностью, в общем случае уже требуют привлечения дополнительных средств для анализа ответов на запросы (например, оценки пользователя).

Простота изучения является во многом субъективной оценкой и может быть в некоторой мере охарактеризована степенью его близости к естественному языку, требуемым для его освоения временем и необходимым уровнем подготовки пользователя.

Высокий ***уровень процедурности,*** свойственный реляционным языкам, определяется присущими реляционной модели свойствами, в частности, полным отделением логической структуры данных от структур хранения и стратегий доступа. Снижение уровня процедурности увеличивает свободу в выборе способов реализации языка, что позволяет осуществить его реализацию более оптимальным способом. Но необходимо отметить, что меньшая степень процедурности еще не означает автоматически меньшую сложность написания запросов. Некоторые сложные запросы можно более просто сформулировать в виде алгоритма поиска ответа, в то время как его формулировка в декларативном виде может оказаться достаточно трудной.

Модульность построения языка характеризует возможность существования нескольких уровней языка и зависит от специфических свойств математической теории, лежащей в его основе.

Минимальный уровень языка, обычно легко понимаемый пользователем, бывает достаточным для формулирования большинства запросов, и лишь формулировка сложных запросов может потребовать использования всех выразительных средств языка, о существовании которых пользователи начального уровня могут и не знать.

Языки, не обладающие модульностью, требуют от пользователя знания почти всего объема средств языка, что усложняет процесс их изучения.

Наиболее распространенным языком для работы с базами данных является SQL (Structured Query Language).

В последних реализациях предоставляет:

1) средства для спецификации и обработки запросов на выборку данных,
2) функции по созданию, обновлению, управлению доступом и т.д.

По существу SQL уже соединяет в себе и язык описания данных и язык манипулирования данными. *Он не является полноценным языком программирования и, в* случае его использования для организации доступа к БД из прикладных программ, SQL-выражения встраиваются в конструкции базового языка.

Являясь внутренним языком баз данных, SQL естественно отражает особенности конкретной СУБД.

Сегодня это единственный стандартизованный язык фактографических баз данных, достаточно мощный и в тоже время, простой для понимания и использования язык.

Сочетание этих факторов вместе с поддержкой ведущих производителей, таких как IBM и Microsoft, привели не только к широкому

его распространению, но и совершенствованию. Сегодня, благодаря независимости от конкретных СУБД и межплатформенной переносимости, SQL стал языком распределенных баз данных и языком шлюзов, позволяющим совместно использовать СУБД разного типа.

Программные средства.

Обработка данных и управление этой обработкой в вычислительной среде, а также взаимодействие с операционной системой и прикладными программами осуществляется комплексом программных средств, взаимосвязь которых иллюстрируется рис. 3.

В составе комплекса обычно выделяют следующие компоненты:

- **ядро**, обеспечивающее управление данными во внешней и оперативной памяти, а также протоколирование изменений;
- **процессор языка базы данных**, обеспечивающий обработку (трансляцию или компиляцию) и оптимизацию запросов на выборку и изменение данных;
- **подсистему (библиотеку) поддержки программных вызовов**, которая обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;
- **сервисные программы** (системные и внешние утилиты), обеспечивающие настройку СУБД, восстановление после сбоев и ряд дополнительных возможностей по обслуживанию.

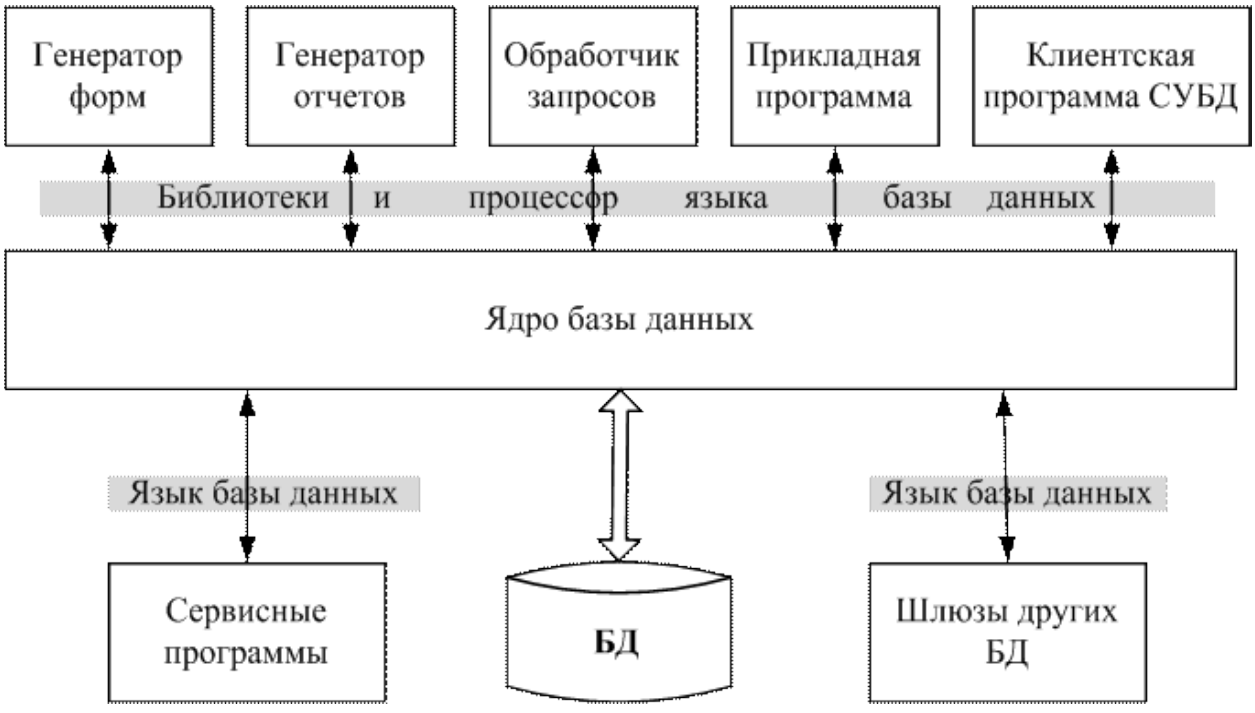


Рис. 3. Программные средства СУБД

Большинство СУБД работают в среде операционной системы и тесно с ней связаны. Многопользовательские приложения, обработка распределенных запросов, защита данных требуют эффективно использовать

ресурсы, управление которыми обычно является функцией ОС. Использование многопроцессорных систем и мультипоточных технологий обработки данных позволяет эффективно обслуживать параллельно выполняемые запросы, но требует координации использования ресурсов между ОС и СУБД. Соответственно, управление доступом и обеспечение защиты также обычно интегрируются с соответствующими средствами операционной системы.

Именно централизованное управление данными обеспечивает:

- сокращение избыточности в хранимых данных;
- совместное использование хранимых данных;
- стандартизацию представления данных, упрощающую эксплуатацию БД;
- разграничение доступа к данным;
- целостность данных, обеспечиваемую процедурами, предотвращающими включение в БД неверных данных и ее восстановление после отказов системы.

Технические средства.

Для больших баз данных, функционирующих в промышленном режиме, обеспечение эффективной и бесперебойной работы должно основываться на использовании адекватных аппаратных средств.

Устройства ввода-вывода и накопители внешней памяти - традиционно узкое место любой базы данных. Объем и быстродействие накопителей являются, очевидно, важными параметрами.

Однако, столь же значима и **отказоустойчивость**. Здесь следует отметить необходимость согласованных решений при распределении ролей между аппаратными и программными компонентами управления операциями ввода-вывода.

Например, наличие буферной памяти в накопителе ускоряющей ввод-вывод (аппаратное кэширование) при сбоях системы во время выполнения операции записи в БД может привести к потере данных: переданные для записи данные еще будут находиться в буфере, а т.к. СУБД уже отметит операцию записи как уже завершившуюся и откат для восстановления данных станет невозможен.

Для повышения надежности хранения часто используют специализированные дисковые подсистемы - RAID (Redundant Array of Inexpensive Disk).

Один логический RAID-диск - это несколько физических дисков, объединенных в одно устройство, управляемое специализированным контроллером, что позволяет распределять основные и системные данные между несколькими носителями (дисками), в том числе дублировать данные. Таким образом, в случае повреждения одного из дисков, можно оперативно восстановить потерянные данные.

Не менее значима роль **центрального процессора**. Многие промышленные СУБД поддерживают многопроцессорную обработку запросов.

Теоретически использование еще одного процессора позволит ускорить обработку. Однако на практике многопроцессорные системы требуют повышенного внимания при приобретении оборудования: надежно работают только сертифицированные системы, использующие соответствующие периферийные устройства.

Для распределенных и удаленно используемых баз данных также важно **сетевое окружение:** связное оборудование и сетевые протоколы. Здесь важны не только показатели быстродействия, но и поддерживаемые ими возможности обеспечения безопасности.

Организационно-административные подсистемы.

Организационно-методические средства не являются технической компонентой системы, однако трудно рассчитывать на устойчивое и долговременное функционирование банка данных, если будут отсутствовать **необходимые методические и инструктивные материалы, регламентирующие работу пользователей, различных по своему статусу и уровню подготовленности.**

Вопрос 3. Понятия и модели предметной области. Принципы построения и проектирования БД как составляющей информационных систем.

Информационная система (ИС) - программно-аппаратный комплекс, предназначенный для хранения и обработки информации какой-либо предметной области.

База данных - важнейший компонент любой информационной системы.

Хорошо структурированная информация в базе данных позволяет:

- без проблем эксплуатировать систему и выполнять ее текущее обслуживание,
- модифицировать и развивать ее при модернизации предприятия и изменении информационных потоков, законодательства и форм отчетности.

В настоящее время в эксплуатации на крупных предприятиях находятся комплексные ИС управления предприятиями (КИС, корпоративные системы, ERP-системы), такие как R/3 фирмы SAP, Oracle E-Business Suite, BaanERP. Среди российских разработок приближаются по функциональности к системам класса ERP «Галактика», «Флагман», «Парус».

Многие ERP-системы могут устанавливаться и функционировать на различных операционных системах и серверах баз данных (многоплатформенные системы). База данных подобных систем состоит из нескольких тысяч таблиц (BaanERP 5.0c - более 2500 таблиц информации по одному предприятию).

Любая сложная система для обеспечения ее надежного функционирования строится как иерархическая система, состоящая из отдельных подсистем и модулей, которые взаимодействуют между собой и используют общую базу данных.

При описании информационной системы предполагается, что она содержит два типа сущностей:

- операционные сущности, которые выполняют какую-либо обработку (некоторый аналог программы),
- пассивные сущности, которые хранят информацию, доступную для пополнения, изменения, поиска, чтения (база данных).

При проектировании сложных информационных систем используется метод декомпозиции - система разбивается на составные части, которые связаны, взаимодействуют друг с другом и образуют иерархическую структуру.

Иерархический характер сложных систем хорошо согласуется с принципом групповой разработки. В этом случае деятельность каждого участника проекта ограничивается соответствующим иерархическим уровнем.

Классический подход к разработке сложных систем представляет собой структурное проектирование, при котором осуществляется алгоритмическая декомпозиция системы по методу *«сверху вниз»*. Именно в этом случае можно построить хорошо функционирующую систему с общей базой данных, согласованными форматами использования и обработки информации на всех участках, с оптимальным взаимодействием всех подсистем.

Исторически сложилось так, что некоторые системы разрабатывались по методу *«снизу вверх»*: вначале создавались отдельные автоматизированные рабочие места (АРМы), затем предпринимались попытки объединения их в единую информационную систему. Подобные разработки для крупных систем не могут быть успешны.

При создании проекта информационной системы для проектирования ее базы данных следует определить:

- объекты информационной системы (сущности в концептуальной модели);
- их свойства (атрибуты);
- взаимодействие объектов (связи) и информационные потоки внутри и между ними.

При этом очень важен анализ существующей практики реализации информационных процессов и нормативной информации (законов, постановлений правительства, отраслевых стандартов), определяющих необходимый объем и формат хранения и передачи информации. Если радикальной перестройки сложившегося информационного процесса не

предвидится, следует учитывать имеющиеся формы хранения и обработки информации в виде журналов, ведомостей, таблиц и т.п. бумажных носителей. Однако предварительно необходимо выполнить анализ возможности перехода на новые системы учета, хранения и обработки информации, возможно, исходя из имеющихся на рынке программных продуктов-аналогов, разработанных крупными информационными компаниями и частично или полностью соответствующими поставленной задаче.

Схема формирования информационной модели представлена на рис. 4.

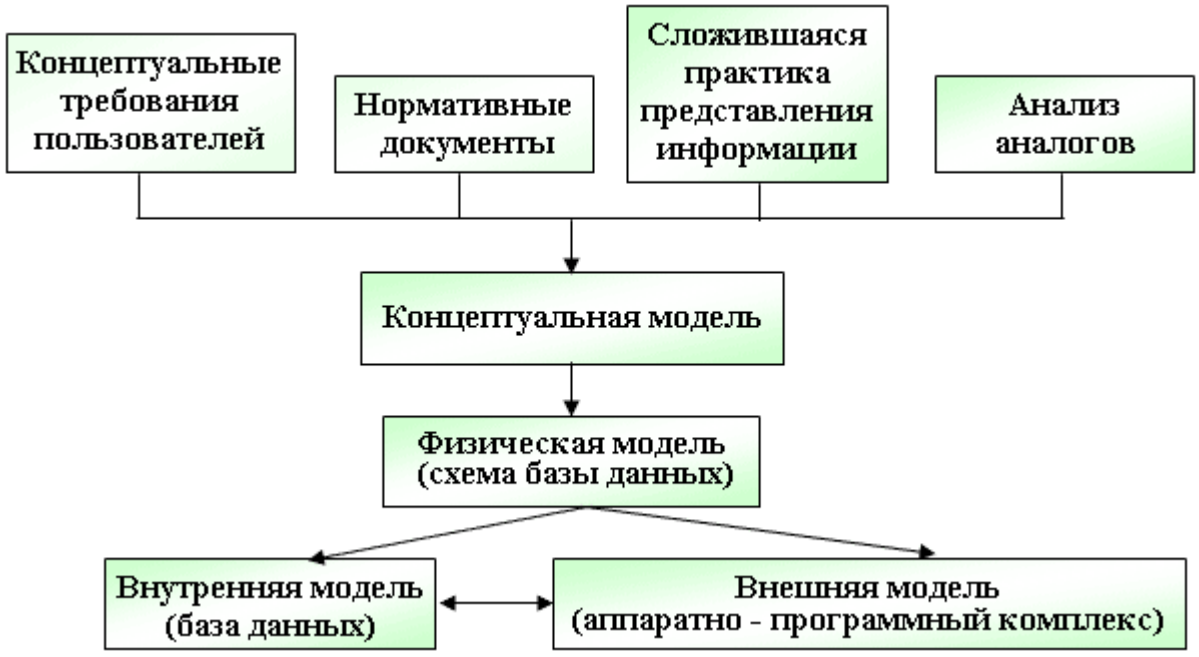


Рис. 4. Схема формирования информационной модели

Концептуальная модель - отображает информационные объекты, их свойства и связи между ними без указания способов физического хранения информации (модель предметной области, иногда ее также называют информационно-логической или инфологической моделью).

Информационными объектами обычно являются сущности - обособленные объекты или события, информацию о которых необходимо сохранять, имеющие определенные наборы свойств - атрибутов.

Физическая модель - отражает все свойства (атрибуты) информационных объектов базы и связи между ними с учетом способа их хранения - используемой СУБД.

Внутренняя модель - база данных, соответствующая определенной физической модели.

Внешняя модель - комплекс программных и аппаратных средств для работы с базой данных, обеспечивающий процессы создания, хранения, редактирования, удаления и поиска информации, а также решающий задачи выполнения необходимых расчетов и создания выходных печатных форм.

Создание информационной системы ведется в несколько этапов, на каждом из которых конкретизируются и уточняются элементы разрабатываемой системы.

Вопрос 4. Жизненный цикл базы данных. Модели жизненного цикла. [4]

Под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует.

Известны следующие базовые модели жизненного цикла: каскадная, поэтапная (каскадная с обратной связью), спиральная.

Каскадная модель.

Каскадная модель, в которой переход на следующий этап означает полное завершение работ на предыдущем этапе.

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ (или «водопад»).

Его основной характеристикой является разбиение всей разработки на этапы, при этом переход на следующий этап происходит только после полного завершения работ на текущем (рис. 5).

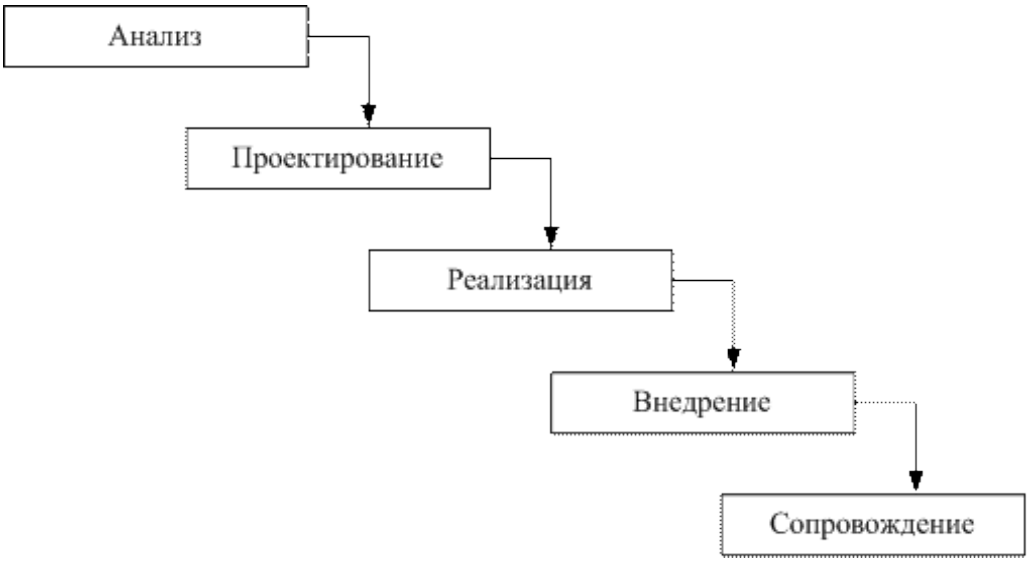


Рис. 5. Каскадная схема разработки ПО

Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков. При этом этапы работ выполняются в логичной последовательности, что позволяет планировать сроки завершения всех работ и соответствующие затраты. Этот подход хорошо зарекомендовал себя при построении ИС, для которых в начале разработки можно достаточно точно и

полно сформулировать все требования и предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения.

Его недостатки связаны с тем, что реальный процесс создания ПО ИС обычно не укладывается в такую жёсткую схему. Практически постоянно возникает потребность возвращаться к предыдущим этапам, уточнять или пересматривать принятые решения. В результате затягиваются сроки выполнения работы, пользователи могут вносить замечания лишь по завершению всех работ с системой. При этом модели автоматизируемого объекта могут устареть к моменту их утверждения.

Поэтапная модель.

Поэтапная модель с промежуточным контролем. (См. рис. 6) Разработка ПО ведётся итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют уменьшить трудоёмкость процесса разработки по сравнению с каскадной моделью. Время жизни каждого из этапов растягивается на весь период разработки.

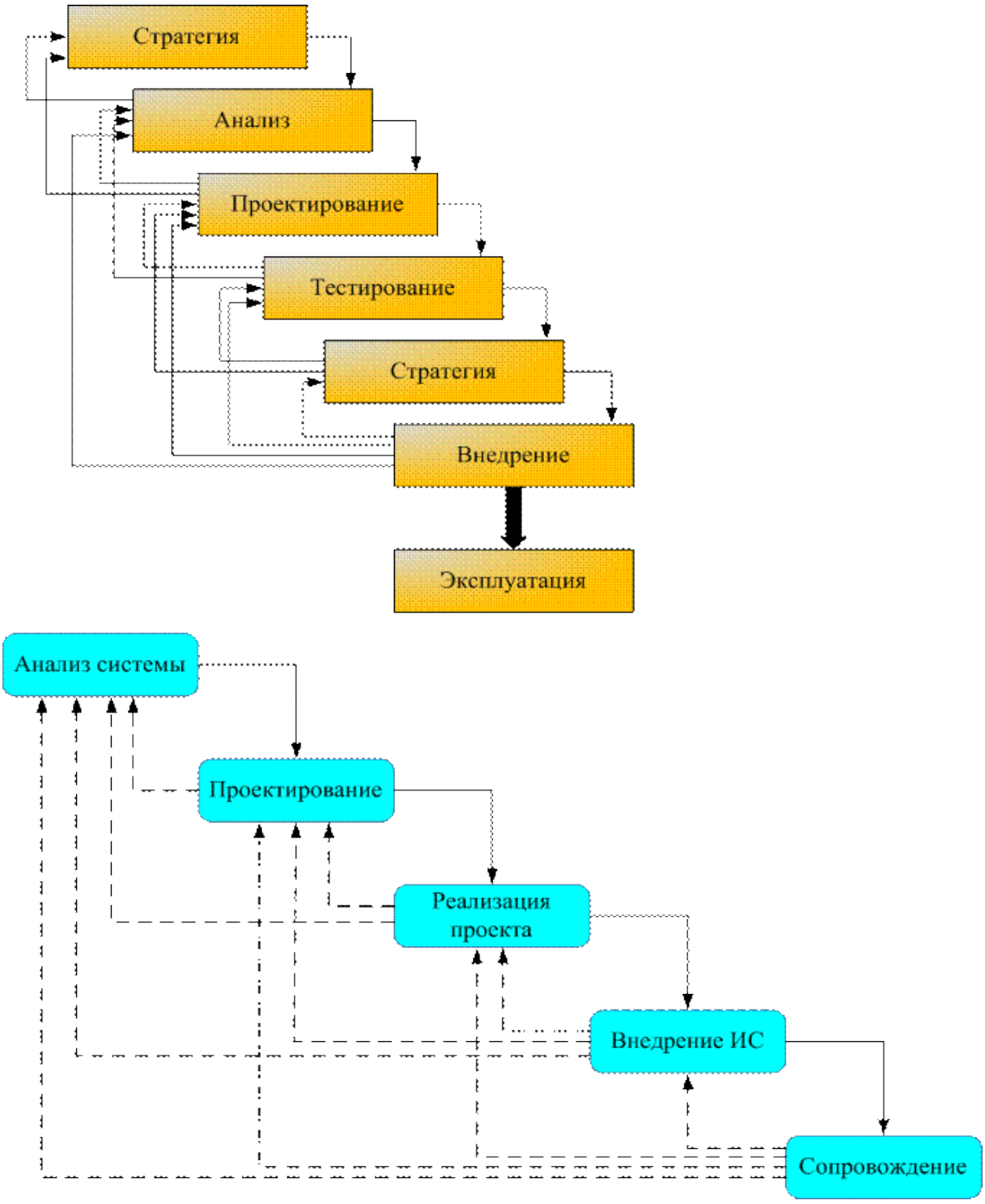


Рис. 6. Поэтапная модель с промежуточным контролем

Спиральная модель.

В этой модели (рис. 7) особое внимание уделяется начальным этапам разработки – выработке стратегии, анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

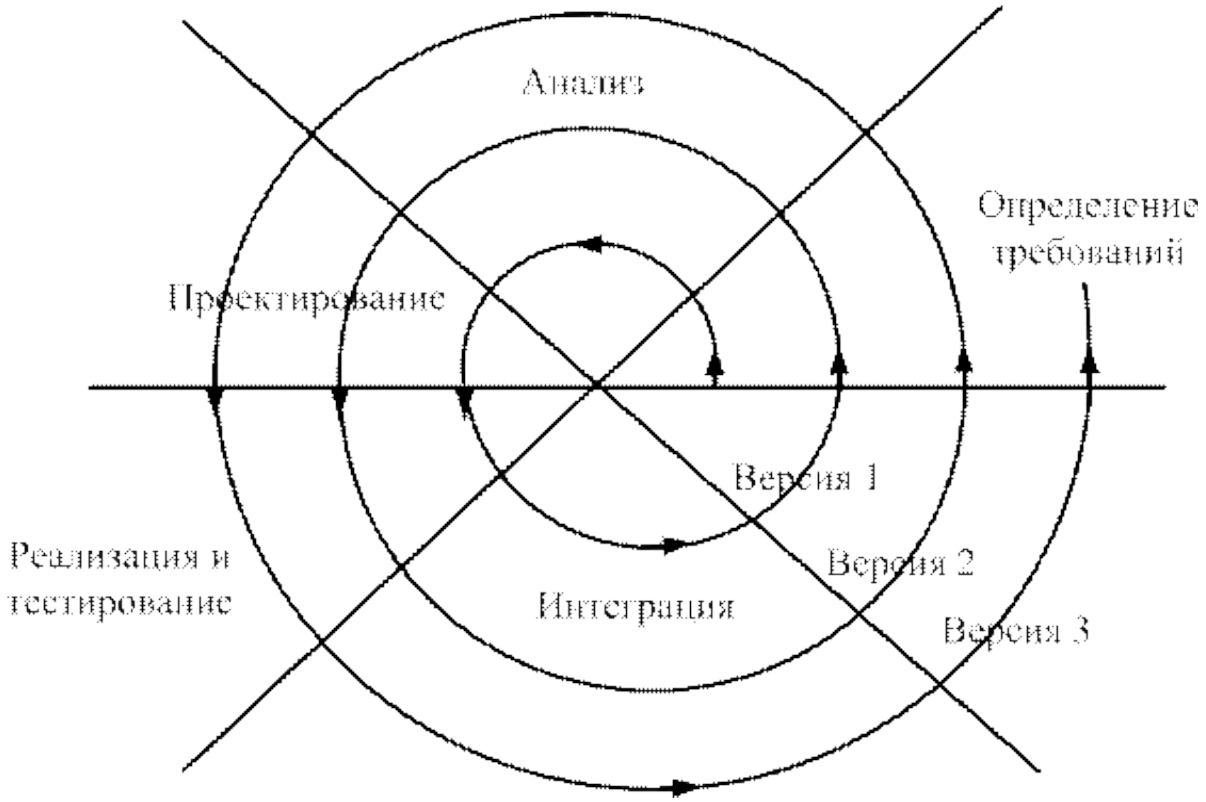


Рис. 7. Спиральная модель

Каждый виток спирали предполагает создание фрагмента (компонента) или версии программного продукта. На них уточняются цели и характеристики проекта, определяется его качество, планируются работы следующего витка спирали. Таким образом, углубляются, последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который доводится до реализации.

Полный жизненный цикл ИС должен поддерживаться комплексом инструментальных средств с учётом необходимости: адаптации типового проекта к различным системно-техническим платформам (техническим средствам, операционным системам и СУБД) и организационно-экономическим особенностям объектов внедрения; интеграции с существующими разработками (включая реинжиниринг приложений и конвертирование БД); обеспечения целостности проекта и контроля за его состоянием (наличие единой технологической среды создания, сопровождения и развития ИС, а также целостность репозитария). При этом желательно обеспечить независимость от программно-аппаратной платформы и СУБД, поддержку одновременной работы групп разработчиков, открытую архитектуру и возможности экспорта/импорта.

Вопрос 5. Методологии и стандарты.

Для решения задач проектирования сложных систем существуют специальные методологии и стандарты. К таким стандартам относятся методологии семейства

IDEF (Icam DEFinition, ICAM - Integrated Computer-Aided Manufacturing - первоначально разработанная в конце 70-х гг. программа ВВС США интегрированной компьютерной поддержки производства). С их помощью можно эффективно проектировать, отображать и анализировать модели деятельности широкого спектра сложных систем в различных разрезах.

Другие методологии, используемые при моделировании сложных систем:

- DFD-технология анализа «потока данных» (Data Flow Diagrams);
- Workflow-технология анализа «потока работ».

Важное место в моделировании информационных систем занимает **методология и системы, использующие UML** - унифицированный язык моделирования (Unified Modeling Language). UML - язык для спецификации, визуализации, конструирования и документирования сложных информационно-насыщенных объектных систем. В настоящее время зарегистрирован как международный стандарт ISO/IEC 19501:2005 «Information technology - Open Distributed Processing - Unified Modeling Language (UML)».

Одной из последних разработок в области моделирования предприятия являет**ся создание специального унифицированного языка моделирования** UEML (Unified Enterprise Modeling Language).

Разработка UEML - сетевой проект (IST-2001-34229), финансируемый Евросоюзом (см. http://athena.troux.com/akmii/Default.aspx?WebID=249).

Проект UEML включает создание:

- общего, визуального, базированного на шаблонах языка для коммерческих инструментальных средств моделирования предприятий и программных систем класса workflow;
- стандартизованных, независимых от инструментов механизмов передачи моделей между проектами;
- репозитория моделей предприятий.

Данный проект осуществляется в соответствии с международными стандартами:

Таблица 1.

ISO 14258	Rules and Guidelines for Enterprise Models (Правила и руководящие принципы для моделей предприятия);
ISO 15704	Requirements for enterprise-reference architectures and methodologies (Требования и методологии по описанию архитектуры предприятия).

Инструментальные средства моделирования предприятий, поддерживающие язык UEML, Metis (Computas), e-MAGIM (GraiSoft), MOzGO (IPK) и др.

В нашей стране в списке действующих ГОСТов по разработке автоматизированных систем (по данным Стандартинформ http://www.vniiki.ru/catalog_v.asp?page=1) следующие:

Таблица 2.

ГОСТ 34.003-90	«Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения»;
ГОСТ 34.201-89	«Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплектность и обозначение документов при создании автоматизированных систем»;
ГОСТ 34.601-90	«Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания»;
ГОСТ 34.602-89	«Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».
ГОСТ 19.101-77	«Единая система программной документации. Общие положения» и т.д.(На разработку программной документации действуют стандарты класса ЕСПД)

Стадии и этапы создания автоматизированных систем (АС) в соответствии с ГОСТ 34.601-90 приведены в табл 3:

Таблица 3.

Стадии	**Этапы работ**
1. Формирование требований к АС	1.1. Обследование объекта и обоснование необходимости создания АС. 1.2. Формирование требований пользователя к АС. 1.3. Оформление отчета о выполненной работе и заявки на разработку АС (тактико-технического задания)
2. Разработка концепции АС	2.1. Изучение объекта. 2.2. Проведение необходимых научно-исследовательских работ. 2.3. Разработка вариантов концепции АС, удовлетворяющего требованиям пользователя. 2.4. Оформление отчета о выполненной работе
3. Техническое задание	3.1. Разработка и утверждение технического задания на создание АС
4. Эскизный проект	4.1. Разработка предварительных проектных решений по системе и ее частям. 4.2. Разработка документации на АС и ее части
5. Технический проект	5.1. Разработка проектных решений по системе и ее частям. 5.2. Разработка документации на АС и ее части. 5.3. Разработка и оформление документации на поставку изделий для комплектования АС и (или) технических требований (технических заданий) на их разработку. 5.4. Разработка заданий на проектирование в смежных частях проекта объекта автоматизации
6. Рабочая документация	6.1. Разработка рабочей документации на систему и ее части. 6.2. Разработка или адаптация программ
7. Ввод в действие	7.1. Подготовка объекта автоматизации к вводу АС в действие. 7.2. Подготовка персонала. 7.3. Комплектация АС поставляемыми изделиями (программными и

	техническими средствами, программно-техническими комплексами, информационными изделиями). 7.4. Строительно-монтажные работы. 7.5. Пусконаладочные работы. 7.6. Проведение предварительных испытаний. 7.7. Проведение опытной эксплуатации. 7.8. Проведение приемочных испытаний
8. Сопровождение	8.1. Выполнение работ в соответствии с гарантийными обязательствами. 8.2. Послегарантийное обслуживание

Для проектирования концептуальной модели и формирования физической модели базы данных информационной системы можно использовать инструментальные CASE-средства (Computer-Aided Software System Engineering). Например, Case Studio, SyBase Power Disigner, ERWin Data Modeler и др.

Данные системы применяются при описании модели данных стандарт IDEF1X и позволяют генерировать программный код на языках SQL, VBScript, JScript, либо работать с другими технологиями для переноса физической модели в реальные СУБД, которыми могут быть Oracle, Microsoft SQL Server, IBM DB2, Informix, Microsoft Access и др.

На рис. 8. приведена схема, показывающая взаимосвязь основных терминов в области проектирования баз данных и работы с ними.

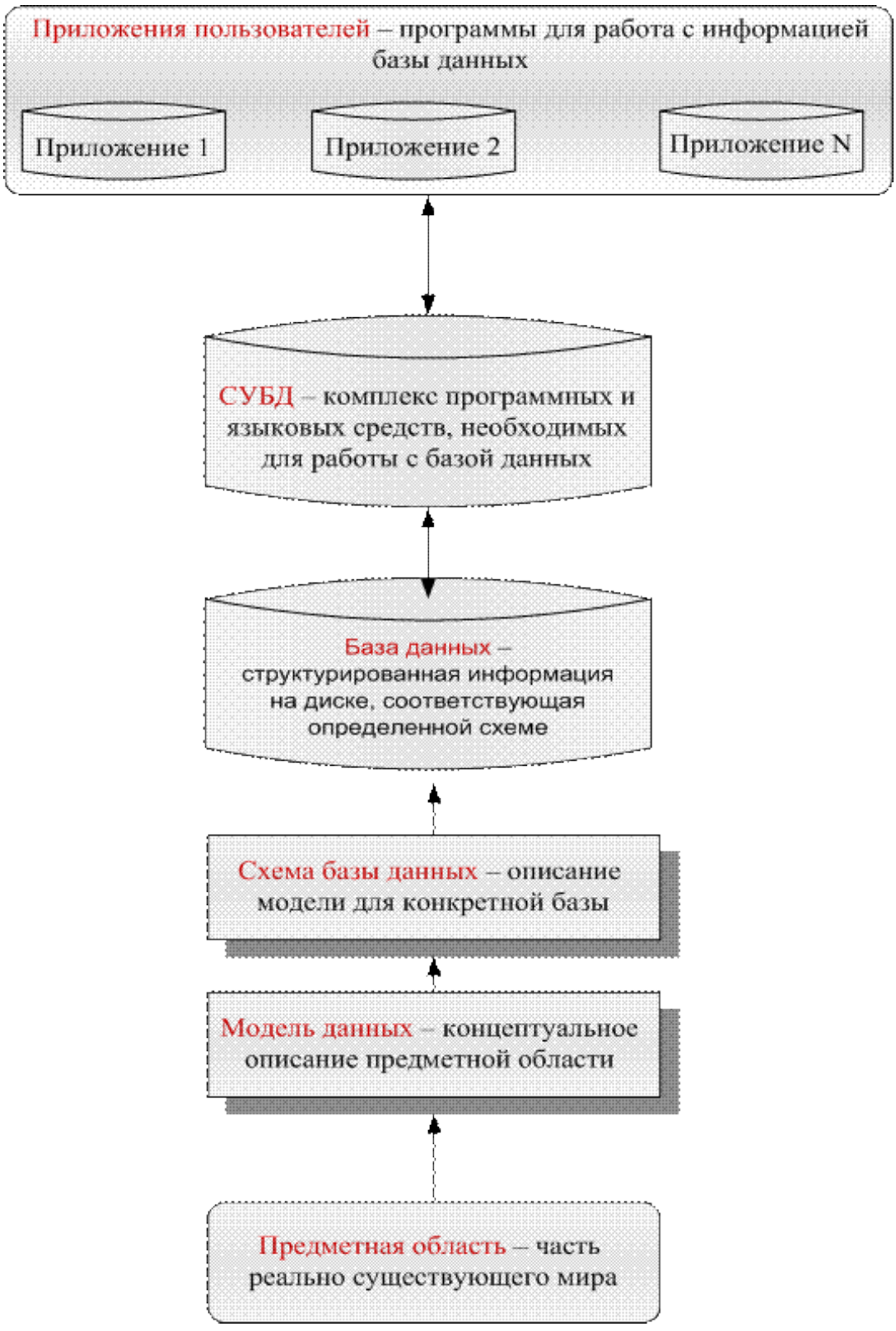


Рис. 8. Взаимосвязь основных терминов в области проектирования баз данных и работы с ними

Выбор системы для разработки приложений, работающих с базой данных, сложное и ответственное решение. Такую возможность имеют универсальные системы программирования: Visual C++ и C#, Delphi, Visual

Basic и др. Однако специализированные языки программирования в составе СУБД располагают огромным количеством процедур и функций для работы с базами данных, специальными библиотеками, механизмами для ускорения работы с большими базами данных.

В связи с повсеместным распространением Интранета, Экстранета и Интернета, многие системы имеют возможность создания трехуровневой сервис-ориентированной архитектуры Web-приложений для работы с базами данных.

По принятым сегодня нормам, над любым проектом ИС работают:

- бизнес-аналитики, изучающие и моделирующие бизнес-процессы предметной области;
- системные аналитики и архитекторы, проектирующие архитектуру решения, приложений и данных;
- авторы кода приложений;
- специалисты по тестированию и оценке качества;
- авторы документации;
- авторы дистрибутивов;
- специалисты по внедрению.

Причем обычно эти функции распределяются между различными специалистами, хотя совмещение ролей все еще практикуется. На этапах проектирования и программирования могут использоваться методы объектно-ориентированного подхода к разработке объектов информационной системы (наследование, инкапсуляция, полиморфизм).

Вопрос 6. Пользователи баз данных.

В информационных системах, создаваемых на основе СУБД, способы организации данных и методы доступа к ним перестали играть решающую роль, поскольку оказались скрытыми внутри СУБД. Массовый, так называемый *конечный пользователь,* как правило, имеет дело только с внешним интерфейсом, поддерживаемым СУБД. Эти преимущества, как уже понятно, не могут быть реализованы путем механического объединения данных в БД.

Предполагается, что в системе существует (как неотъемлемая составная часть) специальное должностное лицо (группа лиц) - *администратор базы данных (АБД),* который несет ответственность за проектирование и общее управление базой данных.

АБД определяет информационное содержание БД. С этой целью он идентифицирует объекты БД и моделирует базу, используя язык описания данных. Получаемая модель служит в дальнейшем справочным документом для администраторов приложений и пользователей. Администратор решает также все вопросы, связанные с размещением БД в памяти, выбором стратегии и ограничений доступа к данным. В функции АБД входят также организация загрузки, ведения и восстановления БД и многие другие

действия, которые не могут быть полностью формализованы и автоматизированы.

Администратор приложений (или, если таковой специально не выделяется - администратор БД) определяет для приложений подмодели данных. Тем самым разные приложения обеспечиваются собственным «взглядом» но не на всю БД, а только на требуемую для конкретного приложения («видимую») ее часть. Вся остальная часть БД для данного приложения будет «прозрачна».

Прикладные программисты имеют, как правило, в своем распоряжении один или несколько языков программирования, с помощью которых генерируются прикладные программы.

Тема 3. Проектирование баз данных

Вопрос 1. Многоуровневые модели предметной области.[5]

Рассматриваемые в контексте понятия «информационная система» элементы реального мира, информацию о которых мы сохраняем и обрабатываем, будем называть *объектами.*

Объект может быть

- материальным (например, служащий, изделие или населенный пункт),
- нематериальным (например, имя, понятие, абстрактная идея).

Набором объектов будем называть совокупность объектов, однородных с некоторой точки зрения (например, объектов *нашего* внимания, пусть даже и разнородных по своей внутренней природе).

Объект имеет различные *свойства* (например, цвет, вес, имя), которые важны для нас в то время, когда мы обращаемся к объекту (например, выбираем среди множества других) с какой-либо целью его использования.

Свойства могут быть заданы как отдельными однозначно интерпретируемыми количественными показателями, так и словесными нечеткими описаниями, допускающими разную трактовку, иногда зависящую от точки зрения и наличных знаний воспринимающего субъекта.

Во всех случаях человек, работая с информацией, имеет дело с *абстракцией,* представляющей интересующий его фрагмент реального мира - той совокупностью *характеристических свойств (атрибутов),* которые важны для решения его прикладной задачи.

Абстрагирование - это способ *упрощения* совокупности фактов, относящихся к реальному объекту (по своей сути бесконечно сложному и разнообразному при изучении его человеком). При этом некоторые свойства объекта игнорируются, поскольку считается, что для решения данной

прикладной задачи (или совокупности задач) они не являются определяющими и не влияют на конечный результат действий при решении.

Цель такого абстрагирования - построение *конструктивного* операбельного *описания* (рабочей модели), *удобного в обработке*, как для человека, так и для машины, *позволяющего* организовать *эффективную обработку больших объемов информации*, причем высоко-производительной должна быть работа не только вычислительной системы, но и взаимодействующего с ней человека.

Обычно отдельная база данных содержит (отражает) информацию о некоторой *предметной области* - наборе объектов, представляющих интерес для актуальных или предполагаемых пользователей. То есть, реальный мир отображается совокупностью конкретных и абстрактных понятий, между которыми существуют (и соответственно, фиксируются) определенные связи.

Выбор для описания предметной области (ПрО) существенных понятий и связей является предпосылкой того, что пользователь будет иметь *практически все необходимые ему в рамках задачи* знания об объектах предметной области. Но, следует отметить, что пользователь, который хочет работать с базой данных, должен владеть основными понятиями, представляющими предметную область. И в этом смысле *абстрагирование позволяет построить такое описание* (модель предметной области), которое другой *человек сможет* не только *воспринять*, но и безошибочно использовать для работы с описаниями экземпляров объектов, хранимых в базе данных.

Пример. Модель предметной области соотносится с реальными объектами и связями так же, как схема маршрутов городского пассажирского транспорта с фактической траекторией движения автобуса. Схема адекватно отражает действительность на уровне основных понятий - маршрутов и остановок: выбрав по схеме маршрут, пассажир достигнет цели (прибудет на нужную остановку) независимо от того, в каком транспортном ряду будет двигаться автобус.

Наиболее простой способ представления предметных областей в БД реализуется поэтапно:

1) фиксацией *логической точки зрения* на данные (т.е. данные рассматриваются независимо от особенностей их хранения и поиска в конкретной вычислительной среде);
2) определением физического представления данных с учетом выбранных структур хранения данных и архитектуры ЭВМ.

Абстрагированное описание предметной области с фиксированной (логической) точки зрения будем называть *концептуальной схемой.*

Соответственно, систематизация понятий и связей предметной области называется *логическим* или *концептуальным проектированием.*

Моделью данных будем называть совокупность функциональных характеристик объектов и особенностей представления информации (например, в числовой или текстовой форме).

Внутренней схемой будем называть отображение концептуальной схемы на физический уровень.

Соотношение этих понятий приведено на рис. 9:

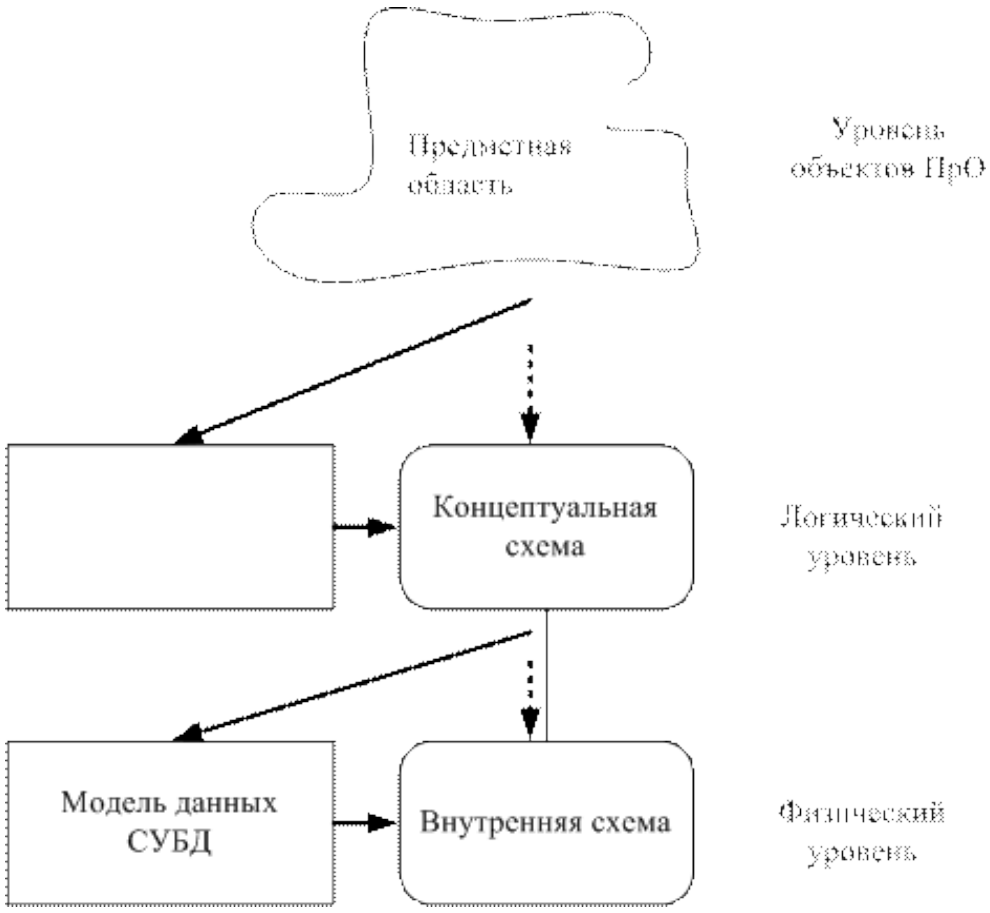


Рис. 9. Соотношение понятий концептуальной и внутренней схем

Внешней схемой называется отражение взгляда (точки зрения) отдельного пользователя на концептуальную схему (как *вариант восприятия предметной области*). Внешняя схема использует те же абстрактные категории, что и концептуальная, а на практике соответствует логической организации данных в прикладной программе.

Теоретически вопрос о многообразии уровней абстракции был решен еще в 60 - 70-х годах. Основой для его решения является концепция многоуровневой архитектуры системы базы данных. В частном случае, на внешнем уровне может поддерживаться совсем иная модель данных (или даже несколько моделей), чем на концептуальном уровне. Поддержка разнообразных возможностей абстрагирования в такой системе достигается благодаря средствам определения и поддержки межуровневого отображения моделей данных. Помимо этого, для решения вопроса о многообразии

уровней абстракции может использоваться внутримодельная структура, например, механизмы *представлений* (view).

В объектных системах для этих целей может использоваться отношение *наследования.*

В общем случае концепция трехуровнего представления не требует более трех уровней, однако с практической точки зрения иногда удобно включать схемы дополнительных уровней. На рис. 10. приведены некоторые варианты решений.

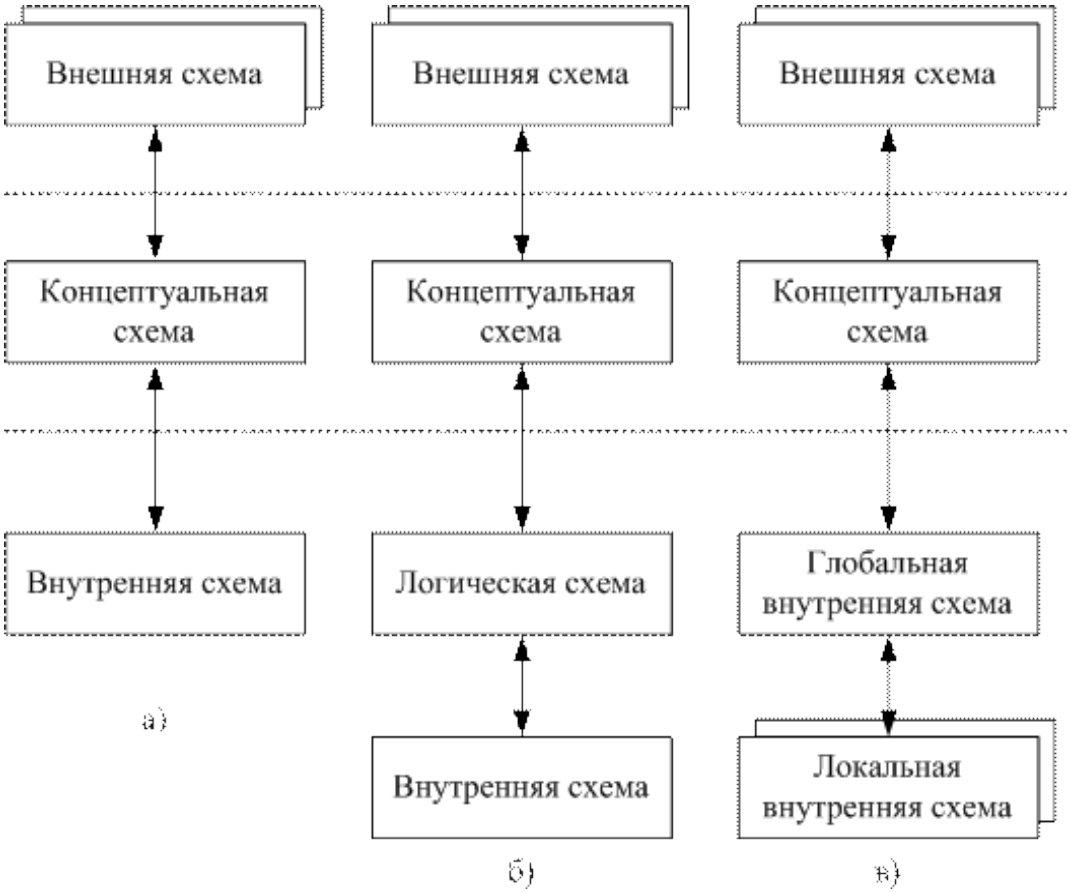


Рис. 10. Примеры трехуровнего представления

б) выделена логическая схема, учитывающая особенности СУБД.

в) приведенный пример характерен для варианта распределенной базы данных, объединяющей информацию, представленную разными внутренними схемами.

Рассмотренная трехуровневая архитектура обеспечивает выполнение основных требований, предъявляемых к системам баз данных:

- адекватность отображения предметной области;
- возможность взаимодействия с БД разных пользователей при решении разных прикладных задач;
- обеспечение независимости программ и данных;
- надежность функционирования БД и защиту от несанкционированного доступа.

С точки зрения пользователей различных категорий трехуровневая архитектура имеет следующие достоинства:

- системный аналитик, создающий модель предметной области, не обязательно должен быть специалистом в области программирования и вычислительной техники;
- администратор баз данных, обеспечивающий отражение концептуальной схемы во внутреннюю, не должен беспокоиться о корректности представления предметной области;
- конечные пользователи, используя внешнюю схему, могут не вдаваться полностью в предметную область, обращаясь только к необходимым составляющим. При этом исключается возможность несанкционированного обращения к данным вне объявленных внешней схемой, так как формирование ее находится в сфере деятельности администратора базы данных;
- системный аналитик, как и конечный пользователь не вмешивается во внутреннее представление данных.

Все это отражает распространенную практику специализации и разделения ответственности.

Главное же заключается в том, что работу по проектированию и эксплуатации баз данных можно разделить на три достаточно самостоятельных этапа.

Отметим, что на практике создание концептуальной схемы не всегда предшествует построению внешней. Иногда трудно с самого начала полностью определить предметную область, но, с другой стороны, уже известны требования пользователей (именно поэтому создание базы уже имеет смысл). И, кроме того, адекватность модели предметной области, в конце концов, должна подтверждаться практикой пользовательских представлений.

Вопрос 2. Идентификация объектов и записей.[6]

В задачах обработки информации, и в первую очередь в алгоритмизации и программировании, атрибуты ***именуют*** (обозначают) и приписывают им ***значения.***

При обработке информации мы имеем дело с совокупностью объектов, ***информацию о свойствах*** каждого из которых надо сохранять (записывать) как ***данные,*** чтобы при решении задач их можно было найти и выполнить необходимые преобразования.

Таким образом, любое состояние объекта характеризуется совокупностью актуализированных атрибутов, которые фиксируются на некотором материальном носителе в виде ***записи*** - совокупности ***(группы)*** формализованных ***элементов данных*** (значений атрибутов, представленных в том или ином формате).

В общем случае объект может описываться совокупностью записей, относящихся к его составным частям или отражающих динамику изменения состояния. Кроме того, в контексте задач хранения и поиска можно говорить, что значение атрибута *идентифицирует* объект: использование значения в качестве поискового признака позволяет реализовать простой критерий отбора по условию сравнения.

Отметим, что некоторые семантические проблемы идентификации через значение атрибута. Значение атрибута идентифицирует запись о **состоянии** объекта, и в случае изменения значения, например – табельного номера служащего, будет невозможно ответить на вопрос: идет ли речь о том же служащем, или о новом.

Также как и в реальном мире, отдельный **объект всегда уникален** (уже хотя бы потому, что мы *именно его* выделяем среди других). Соответственно, запись, содержащая данные о нем, также должна быть узнаваема однозначно (по крайней мере, в рамках предметной области), т.е. - иметь уникальный идентификатор, причем никакой другой объект не должен иметь такой же идентификатор. Поскольку идентификатор - суть значение элемента данных, в некоторых случаях для обеспечения уникальности требуется использовать более одного элемента.

Например, для однозначной идентификации записей о дисциплинах учебного плана необходимо использовать элементы СЕМЕСТР и НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ, так как одна дисциплина может быть прочитана в разных семестрах.

Предложенная выше схема представляет атрибутивный способ идентификации содержания объекта (см. рис. 11).

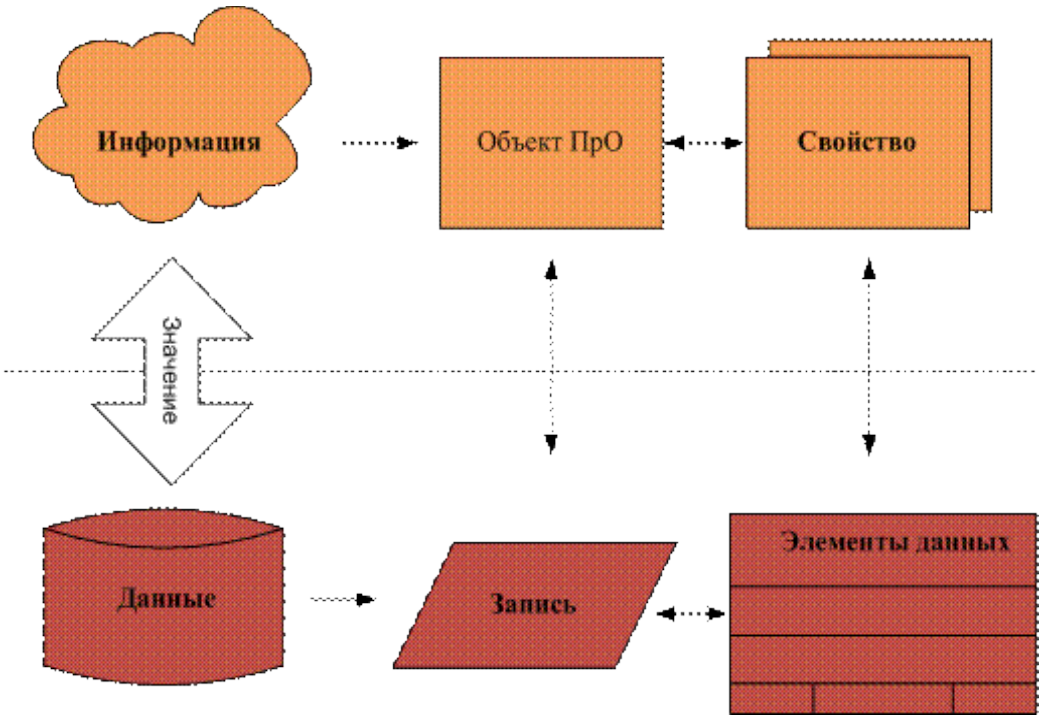


Рис. 11. Атрибутивный способ идентификации

Она является достаточно **естественной** для данных, имеющих **фактографическую природу**, и описывающих обычно материальные объекты. Информацию, представляемую такого рода данными, называют *хорошо структурированной.*

Здесь важно отметить, что структурированность относится не только к форме представления данных (формат, способ хранения), но и к способу интерпретации значения пользователем: значение параметра не только представлено в предопределенной форме, но и обычно сопровождается указанием размерности величины, что позволяет пользователю понимать ее смысл без дополнительных комментариев.

Таким образом, фактографические данные предполагают возможность их *непосредственной* интерпретации.

Однако атрибутивный способ практически не подходит для идентификации ***слабо структурированной информации***, связанной с объектами, имеющими обычно *идеальную* (умозрительную) природу - категориями, понятиями, знаковыми системами. Такие объекты зачастую определяются логически и опосредованно - через другие объекты.

Для описания таких объектов используются естественные или искусственные языки (например, язык алгебры).

Соответственно, для понимания смысла пользователю необходимо использовать соответствующие правила языка, и, более того, часто необходимо уже располагать некоторой информацией, позволяющей идентифицировать и связать получаемую информацию с наличным знанием. Т.е., процесс интерпретации такого рода данных имеет ***опосредованный*** характер и требует использования дополнительной информации, причем такой, которая не обязательно присутствует в формализованном виде в базе данных.

Такое разделение нашло отражение в традиционном разделении баз данных на *фактографические* и *документальные.*

Вопрос 3. Поиск записей.[7]

Программисту или пользователю необходимо иметь возможность обращаться к отдельным, нужным ему записям (описаниям объектов) или отдельным элементам данных.

В зависимости от уровня программного обеспечения прикладной программист может использовать следующие способы:

- задать машинный адрес данных и в соответствии с физическим форматом записи прочитать значение (это случай, когда программист должен быть «навигатором»);
- сообщить системе имя записи или элемента данных, которые он хочет получить, и, возможно, организацию набора данных.

В этом случае система сама произведет выборку (по предыдущей схеме), но для этого она должна будет использовать вспомогательную информацию о структуре данных и организации набора. Такая информация по существу будет избыточной по отношению к объекту, однако общение с базой данных не будет требовать от пользователя знаний программиста и позволит переложить заботы о размещении данных на систему.

В качестве ключа, обеспечивающего доступ к записи, можно использовать **идентификатор** - отдельный элемент данных.

Ключ, который идентифицирует запись единственным образом, называется *первичным (главным).* В том случае, когда ключ идентифицирует некоторую группу записей, имеющих определенное общее свойство, ключ называется *вторичным (альтернативным)*. Набор данных может иметь несколько вторичных ключей, необходимость введения которых определяется практической необходимостью - оптимизацией процессов нахождения записей по соответствующему ключу.

Иногда в качестве идентификатора используют составной **сцепленный ключ -** несколько элементов данных, которые в совокупности, например, обеспечат уникальность идентификации каждой записи набора данных. При этом ключ может храниться в составе записи или отдельно.

Например, ключ для записей, имеющих неуникальные значения атрибутов, для устранения избыточности может храниться отдельно.

На рис. 12 приведены два таких способа хранения ключей и атрибутов для набора простейшей структуры.

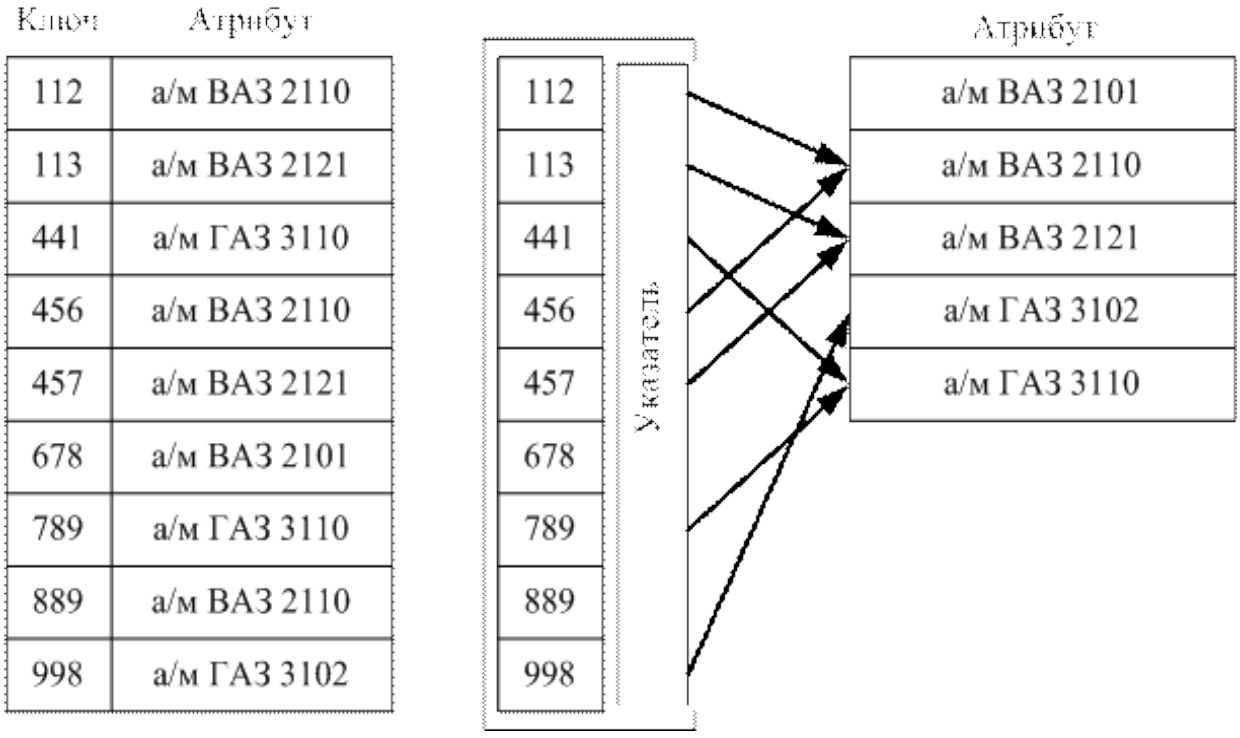


Рис. 12. Способы хранения ключа и атрибута

Введенное понятие ключа является логическим и его не следует путать с физической реализацией ключа - *индексом,* обеспечивающим доступ к записям, соответствующим отдельным значениям ключа.

Один из способов использования вторичного ключа в качестве входа - организация инвертированного списка, каждый вход которого содержит значение ключа вместе со списком идентификаторов соответствующих записей. Данные в индексе располагаются в возрастающем или убывающем порядке, поэтому алгоритм нахождения нужного значения довольно прост и эффективен, а после нахождения значения запись локализуется по указателю физического расположения. **Недостатком индекса я**вляется то, что он занимает дополнительное пространство и его надо обновлять каждый раз, когда удаляется, обновляется или добавляется запись. На рис. 13 приведен инвертированный список для предыдущего примера.

а/м ВАЗ 2101	678
а/м ВАЗ 2110	112, 456, 889
а/м ВАЗ 2121	113, 457
а/м ГАЗ 3102	998
а/м ГАЗ 3110	441, 789

Рис. 13. Инвертированный список для ключа «Марка автомобиля»

В общем случае инвертированный список может быть построен для любого ключа, в том числе, составного.

В контексте задач поиска можно сказать, что существуют два основных способа организации данных. Первый способ соответствует первому примеру и представляет прямую организацию массива. Второй способ является инверсией первого.

Прямая организация массива удобна для поиска по условию «Каковы свойства указанного объекта?», а инвертированная – для поиска по условию «Какие объекты обладают указанным свойством?».

Существует следующая типология простых (атомарных) запросов:

1. **A(E) = ?** Каково значение атрибута A для объекта E?
2. **A(?) = V** Какие объекты имеют значение атрибута равное V?
3. **?(E) = V** Какие атрибуты объекта E имеют значение равное V?
4. **?(E) = ?** Какие значения атрибутов имеет объект E?
5. **A(?) = ?** Какие значения имеет атрибут A в наборе?
6. **?(?) = V** Какие атрибуты объектов набора имеют значение равное V?

Здесь в запросах типов 2, 3, 6 вместо оператора равенства может быть использован другой оператор сравнения *(больше, меньше, не равно* или другие).

Запросы типа 1 выполняются поиском по «прямому» массиву: доступ к записи производится по первичному ключу.

Запросы типа 2 выполняются поиском по инвертированному списку: доступ к записи (ям) производится по указателю, выбираемому из списка по значению вторичного ключа. Ответом в этих случаях будет *значение* атрибута или идентификатора.

Запросы типа 3 имеют ответом *имя* атрибута.

Запросы типа 2, 5, 6 относятся к нескольким атрибутам, и в этом случае могут быть построены несколько индексов, облегчающих поиск по этим ключам.

Составные условия поиска могут использовать несколько простых условий, обычно связанных логическими (булевыми) операторами.

Следует отметить, что в контексте обработки запросов 2-го типа «Какие объекты имеют заданное значение атрибута?» можно выделить три следующих типа архитектур доступа:

1. Системы с вторичными индексами.

В этих системах последовательность расположения записей соответствует последовательности значений первичного ключа. Как правило, используется один первичный индекс и несколько вторичных.

2. Системы частично инвертированных файлов.

В этих системах записи могут располагаться в произвольной последовательности. В отличие от систем первого типа первичный индекс отсутствует. Вторичные индексы применяются для прямой адресации записей, что существенно облегчает включение в файл новых записей, так как допускается их размещение в любом свободном участке файла.

3. Системы полностью инвертированных файлов. В этих системах предусмотрено наличие файлов, содержащих значения отдельных элементов данных, входящих в состав записей - допускается раздельное хранение элементов данных записи.

Значения элементов данных, составляющих конкретную запись или кортеж, в общем случае могут размещаться в памяти произвольно.

Для ускорения процесса поиска в системе используют два набора индексов: *индекс экземпляров* (значений ключей) и *индекс данных* (инвертированный список). С помощью индекса экземпляров можно найти в файле элементы данных, имеющих заданное значение. С помощью индекса данных можно найти записи, связанные с заданными значениями элементов. Такая организация характерна для организации данных *документальных информационных систем.*

Вопрос 4. Представление предметной области и модели данных.[8]

Если бы назначением базы данных было только хранение и поиск данных в массивах записей, то структура системы и самой базы была бы простой.

Причина сложности в том, что практически любой объект характеризуется не только параметрами-величинами, но и взаимосвязями частей или состояний.

Есть различия и в характере взаимосвязей между объектами предметной области:

одни объекты могут использоваться только как характеристики остальных объектов,

другие - независимы и имеют самостоятельное значение (рис. 14).

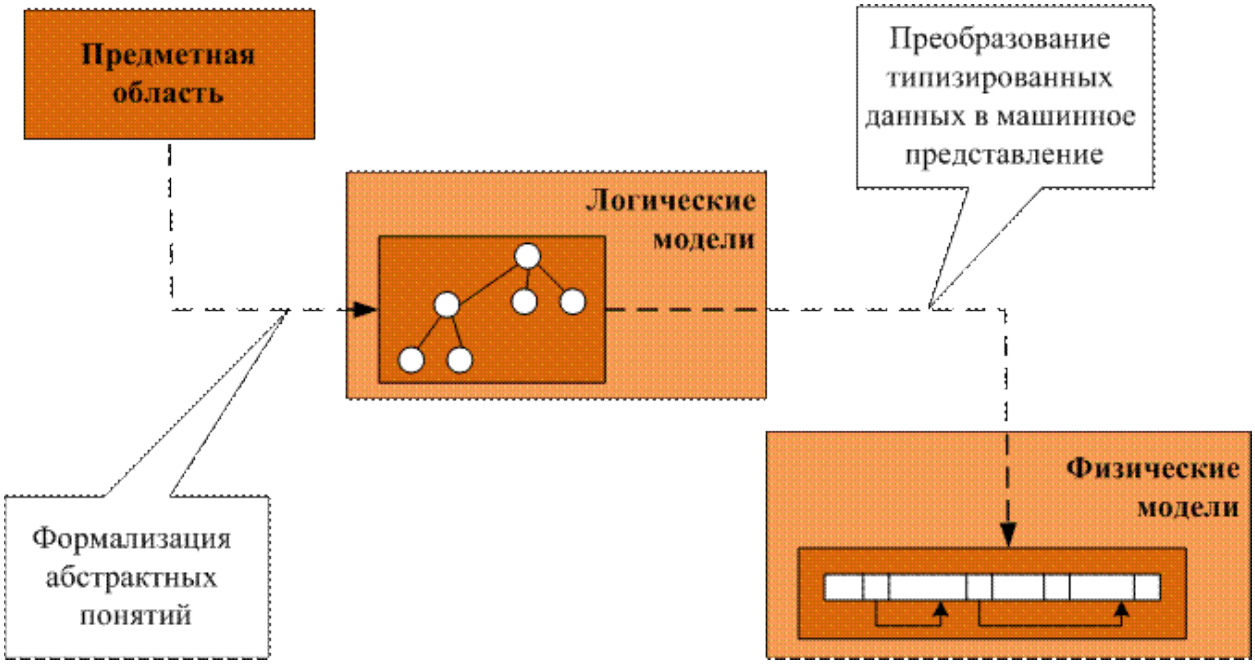


Рис. 14. Этапы преобразования представлений ПрО

Отдельный элемент данных (его значение) ничего не представляет. Он приобретает смысл только тогда, когда связан с атрибутом (природой значения, что позволит интерпретировать значение) и другими элементами данных. Поэтому **физическому размещению данных** (и, соответственно, определению структуры физической записи) **должно предшествовать описание логической структуры предметной области** - построение *модели* соответствующего фрагмента реального мира, выделяющей только те объекты, которые будут интересны будущим пользователям, и представленные только теми параметрами, которые будут значимы при решении прикладных задач. Такая модель будет иметь очень мало физического сходства с реальностью, но будет полезна как ***представление*** пользователя о реальном мире. Причем это представление будет задаваться (описываться) ***удобными для пользователя*** средствами в не адекватной человеку жесткой

вычислительной среде с двоичной логикой и числовым представлением информации.

Таким образом, прежде чем описывать физическую реализацию объектов и связей между ними, необходимо определить:

1) способ, с помощью которого внешние пользователи представляют (описывают) объекты и связи;
2) форму и методы внутримашинного представления элементов данных и взаимосвязей;
3) средства, обеспечивающие взаимно однозначные преобразования внешнего и внутримашинного представлений.

Такой подход является компромиссом, свойственным языкам программирования: за счет *предварительно определяемого множества абстракций,* общих для большинства задач обработки данных, обеспечивается возможность построения *надежных* программ обработки.

Пользователь, используя *ограниченное множество формальных, но достаточно знакомых понятий,* выделяя сущности и связи, описывает объекты и связи предметной области;

Программист (или система автоматизации проектирования БД), используя такие *типовые абстрактные понятия* (как например числа, множества, последовательности, агрегаты), определяет соответствующие информационные структуры.

Система управления данными, используя *двоичные формы представления типизированных* данных, обеспечивает эффективные процедуры хранения и обработки данных.

Именно введение **промежуточного** уровня абстракции позволяет иметь раздельное описание логического и физического представления, освобождает конечного пользователя от необходимости беспокоиться о деталях внутримашинного представления и обработки, поскольку он может быть уверен, что программистом выбрана наиболее эффективная форма для данной ситуации. Однако **эффективность** здесь **имеет определенные пределы**. Чем ближе система абстракций к особенностям вычислительной среды, тем выше эффективность выполнения программы, но вынужденная «специализация» абстракций увеличивает вероятность того, что они станут неподходящими для некоторых других применений.

Модель данных должна, так или иначе, дать основу для описания данных и манипулирования данными, а также дать средства анализа и синтеза структур данных.

Необходимо отметить, что предметные среды с точки зрения описания целесообразно условно разделить на два полярных случая:

1. Предметная среда характеризуется сравнительно небольшим количеством типов отношений, но каждое отношение само есть большое множество.

Эти отношения сравнительно устойчивы, а изменений в пределах каждого множества существенно меньше мощности самого отношения.

Например, отношение «вхождения» элементов изделий, содержащееся в конструкторских спецификациях, для среднего предприятия содержит сотни тысяч записей. В этом случае, задав схемы отношений и ориентировочные значения их мощностей, можно достаточно полно представить структуру и масштаб предметной среды.

2. Для предметной среды характерно большое число типов отношений между объектами, но каждое отношение есть множество сравнительно малой мощности.

При этом мощность потока изменений для отношений сравнима с мощностью самих отношений.

Первый случай характерен для отображения процессов на уровне автоматизированных систем управления предприятиями. Современные системы управления базами данных наиболее эффективны именно в подобном случае, при отображении статических в указанном смысле предметных сред. Обычно при этом речь идет о целых классах объектов, например, о деталях данного типа и обычно не отображается состояние каждой конкретной детали.

Второй случай характерен для описания производственного *технологического процесса,* с учетом временных и пространственных факторов нахождения конкретных объектов.

Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором - о семантических сетях и фреймах.

Основное отличие этих методов заключается в том, что первые задают четкую схему (так называемую схему базы данных), в рамках которой и отображается предметная область.

Подобное построение по сути своей является довольно статичным, требует априорного знания типов отношений, в которых может находиться объект, однако зафиксированная схема базы данных позволяет довольно эффективно организовать поиск необходимой информации.

Во втором случае предметная среда отображается (по крайней мере, на уровне модели) **в виде однородной сети**, любые изменения которой, как по вводу новых классов объектов, так и новых типов отношений, не связаны с какими-либо структурными преобразованиями сети.

В силу большого количества типов отношений манипулирование подобной «элементарной» информацией достаточно затруднено, поэтому для данного случая характерно введение большого количества общих понятий (и соответствующих им отношений), что упрощает работу с таким представлением.

В контексте машинного представления модель данных может быть использована следующим образом:

- как средство спецификации типов данных и их организации, разрешенных в конкретной БД;
- как основа разработки общей методологии построения баз данных;

- как основа минимизации влияния эволюции баз данных на уже существующие прикладные программы и работу конечных пользователей;
- как основа разработки семейства языков запросов и языков манипулирования данных;
- как основа архитектуры СУБД;
- как основа изучения динамических свойств различных организаций данных.

Таким образом, **модель данных** - это базовый инструментарий, обеспечивающий на формальном абстрактном уровне конкретные способы представления объектов и связей.

Модель базы данных охватывает более широкий спектр понятий.

Основное назначение модели базы данных состоит в том, чтобы:

- определить ясную границу между логическим и физическим аспектами управления базой данных *(независимость данных);*
- обеспечить конечным пользователям и программистам, создающим БД, возможность и средства общего понимания смысла данных *(коммуникабельность);*
- определить языковые понятия высокого уровня, обеспечивающие возможность выполнения однотипных операций над большими совокупностями записей (в общем случае разнотипных данных) как единую операцию *(обработка множеств).*

Вопрос 5. Структуры данных (линейные, нелинейные, сетевые).[9]

Абстрактное понятие **структуры** ближе всего находится к так называемой концептуальной модели предметной среды и часто лежит в основе последней.

Понятие структуры используется на всех уровнях представления предметной области и реализуется как:

- ***структура информации*** - схематичная форма представления сложных композиционных объектов и связей реальной ПрО, выделяемых как актуально необходимые для решения прикладных задач;
- ***структура данных*** - атрибутивная форма представления свойств и связей ПрО, ориентированная на выражение описания данных средствами формальных языков (т.е. учитывающая возможности и ограничения конкретных средств с целью сведения описаний к стандартным типам и регулярным связям);
- ***структура записей*** - целесообразная (учитывающая особенности физической среды) реализация способов хранения данных и организации доступа к ним как на уровне отдельных записей, так и их элементов (с целью определения основных и вспомогательных функциональных массивов, а также совокупности унифицированных процедур манипулирования данными).

Структура является общепринятым и удобным инструментом, одинаково эффективно используемым как на уровне сознания человека при работе с абстрактными понятиями, так и на уровне логики машинных алгоритмов.

Структура позволяет простыми способами свести многомерность содержательного описания к линейной последовательности записей. Именно это позволяет формализовать на общей понятийной основе взаимосвязь представлений информации в разных средах: обеспечить контролируемое сведение бесконечного разнообразия объектов и видов взаимосвязей реального мира к жестко детерминированному описанию - совокупности двоичных данных и машинно-ориентированных алгоритмов их обработки.

Выделение трех видов структур, относящихся к представлению объектов ПрО, имеет, в некотором смысле, принципиальный характер.

Структура информации - это неотъемлемое свойство информации (сведений, сигналов, воспринимаемых субъектом) о некоторой совокупности объектов предметной области, в контексте практической задачи (решаемой субъектом), в общем случае без учета того, будут ли для ее решения использованы средства программирования и вычислительные машины.

Структурирование информации осуществляется системным аналитиком и сводится к выделению операционных объектов и определению их характеристических свойств и взаимосвязей.

Структура данных - это определение информационных массивов (состава и взаимосвязей данных на логическом уровне, соответствующих характеру информации и видам соответствующих преобразований).

При определении структур данных необходимо:

- установить состав массива,
- определить оптимальную их взаимосвязь (и, соответственно, определить критерии и методы оценки эффективности),

Пример, выделение групп или агрегатов, имеющих иерархическую идентификацию. Эффективность в этом случае связывается с процессом построения программы («решателя» прикладной задачи) и, в каком-то смысле - с эффективностью работы программиста.

Пример, при функциональной обработке массива необходимо обращаться к отдельным элементам, в то время как в операциях присваивания или при записи массива в файл поэлементное обращении приведет к увеличению размера текста программы, а в ряде случаев - к увеличению времени выполнения.

Структура записей - это определение структуры физической памяти:

- выделение,
- освобождение,
- защита областей физического носителя,
- способы адресации и пересылки.

Эффективность в этом случае связывается с процессами обмена между устройствами оперативной и внешней памяти, искусственно вводимой для обеспечения функциональной эффективности отдельных операций (например, поиска по ключам) избыточностью данных,

Рассмотрим разновидности и типологию «компьютерных» логических структур данных с точки зрения особенности их организации.

Заметим, что мы не будем рассматривать простейшие типы, к которым относятся **стандартные типы –** целые и вещественные числа, логические переменные, символы. Их состав и структура определяется в основном набором **встроенных** базовых типов данных и операций, свойственных конкретному типу ЭВМ.

Итак, структура в первую очередь определяет алгоритм выборки отдельных элементов данных, но в то же время необходимо отметить, что она отражает и особенности «технологии» организации и обработки информации, свойственные человеку в его повседневной деятельности.

Физически понятию *структура* соответствует *запись данных.*

Запись - это упорядоченная в соответствии с характером взаимосвязей совокупность полей (элементов) данных, размещаемых в памяти в соответствии с их типом.

Память, отводимая для хранения значения элемента данных (поле данных), должна выбираться в соответствии с диапазоном значений, которые может иметь этот элемент. Поскольку для выполнения операции присвоения значения элементу данных (установление соответствующих битов в «0» или «1») необходимо сначала выделить память, для чего используются две схемы– статическая и динамическая. Для первой характерно выделение памяти до того, как реально появляются значения (обычно на этапе трансляции программы); для второй – в тот момент, когда программа во время исполнения получает конкретное значение. Кроме того, характер данных (тип данных) определяет способ представления и, соответственно, некоторое множество стандартных операций (примитивов). Поле представляет собой минимальную адресуемую (идентифицируемую) часть памяти - единицу данных, на которую можно ссылаться при обращении к данным.

Итак, **структура данных** - это способ отображения значений в памяти:

- размер области
- порядок ее выделения (который и определит характер процедуры адресации/выборки).

Зачастую именно успешность структурирования данных определяет сложность процедур их обработки.

Классификация структур данных должна проводиться с двух точек зрения.

1) с точки зрения порядка их размещения/выборки:

по характеру взаимосвязи элементов структуры виды структур можно разделить на

- линейные
- нелинейные.

2) с точки зрения однородности и «элементарности» типов данных, отражающих понятийную структуру ПрО.

по характеру информации, представляемой структурой:

- однородные структуры, где все элементы находятся на одном понятийном уровне и имеют один тип данных,
- неоднородные (композиционные), где элементы относятся к нескольким понятийным уровням или имеют разную природу.

Линейные структуры.

К линейным структурам относятся массивы и последовательности, таблицы.

Порядок следования (и, соответственно, выборки) элементов таких структур имеет **линейный характер** и соответствует порядку расположения элементов в памяти: один за другим без каких-либо промежутков.

Адрес элемента соответствует его положению и определяется индексом - порядковым номером элемента в последовательности размещения. К элементу имеется прямой доступ, если известен его индекс.

Особенностью линейной структуры является то, что при последовательной организации (размещении) она допускает **возможность прямого доступа к произвольному элементу. Это возможно** потому, что условие однородности (однотипности) предполагает, что все элементы занимают расположенные строго последовательно области одинакового размера, что и позволяет достаточно просто вычислять значение физического адреса элемента по значению его индекса.

Массив представляет собой совокупность однотипных элементов, причем число элементов массива известно до его размещения, что позволяет строить гибкие многомерные системы адресации.

Последовательность, так же, как и массив, представляет собой совокупность однотипных элементов. Однако число элементов до размещения неизвестно.

Хотя, каждая конкретная последовательность имеет конечную длину, до начала обработки (и, соответственно, размещения) необходимо считать длину последовательности бесконечной.

Принципиальность такого предположения выражается в том, что необходимо предусматривать

специальную процедуру использования памяти (выделения/освобождения),

алгоритм обработки последовательности по частям (в ряде случаем, когда это требуется).

Этот тип данных превалирует в операциях ввода/вывода с устройствами внешней памяти.

Последовательный доступ позволяет организовать «потоковые» операции: однородность позволяет рассматривать пересылаемые данные как непрерывный поток.

Поток не может быть прерван по контекстно определяемому условию.

Например, при пересылке текста - по значению кода «перевод строки», и это не заставляет программу анализировать значение каждого очередного элемента. И, кроме того, последовательный доступ - это простота управления памятью и устройством ввода-вывода.

Но мы, не рассматриваем *очереди* и *стеки.* Они отличаются тем, что для них устанавливаются особые схемы добавления (размещения в памяти) новых элементов и их выборки.

Очередь: порядок размещения/выборки определяется правилом «первым размещен – первым выбран».

Стек - «первым размещен – последним выбран». Выборка элемента влечет его удаление из последовательности.)

Таблица - это последовательности, обычно представляемые строками - совокупностями разнотипных элементов. Таблица -это множество записей, каждая из которых представляет набор поименованных полей.

Однако с точки зрения размещения элементов таблица может быть представлена как одномерный массив (или, в случае БД - последовательность) с однородными композиционными элементами, каждый из которых представляет собой совокупность разнотипных элементов.

Это позволяет свести ввод/вывод таких типов структур к последовательным элементарным операциям.

Разнотипность элементов позволяет ввести отличную от перечислительной схему идентификации записей, определив одно из полей как ключ записи. Обычно ключ содержит значение, используемое в процедурах упорядочения и поиска записей.

Нелинейные структуры.

В качестве примеров нелинейных структур рассмотрим **списки, деревья и сети**.

Порядок следования (и, соответственно, выборки) элементов таких структур может не соответствовать порядку расположения элементов в памяти.

Списки представляют собой пример линейного упорядочения,

Деревья - двумерного,

Сети - произвольного.

Соответственно различаются методы и средства, обеспечивающие последовательность выборки элементов данных. Обычно для обеспечения возможности прямого доступа к произвольному элементу необходимо *использовать вспомогательные структуры типа инвертированных списков.*

Список представляет собой совокупность однотипных элементов. Порядок выборки элементов может отличаться от порядка следования в памяти, определенного при размещении. Наиболее очевидный способ установления однонаправленного порядка выборки элементов - это сопоставить каждому элементу списка ссылку, указывающую на следующий элемент. Соответственно, для организации двунаправленного списка, допускающего также выборку в обратном порядке, каждый элемент должен иметь ссылку на предыдущий.

Такая организация уже не допускает возможности прямого доступа, например, по номеру элемента.

Число элементов списка, как и в случае последовательностей, может быть неизвестно до размещения, и до начала обработки (и, соответственно, размещения) необходимо считать длину списка бесконечной, что ведет к необходимости предусматривать специальную процедуру выделения/освобождения памяти.

Таким образом, с точки зрения физической реализации элемент списка должен быть *составным,* включающим собственно информативные данных, несущий смысловое значение, и дополнительные данные *(ссылки),* определяющие порядок доступа к информативным элементам.

Понятие списка достаточно универсально. В общем случае ссылки могут указывать ответвления к другим спискам - **подспискам.**

В зависимости от способа построения списка и предполагаемых путей доступа к элементам различают следующие **виды ссылок:**

- перекрестные,
- боковые,
- иерархические,
- множественные,

Они позволяют изменять «естественный» последовательный порядок прохода по элементам списка.

Деревья.

Дерево (рис. 15) представляет собой иерархию элементов, называемых *узлами.* На самом верхнем уровне иерархии имеется только один узел - корень. Каждый узел, кроме корня, связан с одним узлом на более высоком уровне, называемым *исходным узлом* для данного узла. Каждый элемент имеет только один исходный. Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне, которые называются *порожденными.* Элементы, расположенные в конце ветви, т. е. не имеющие порожденных, называются *листьями.*

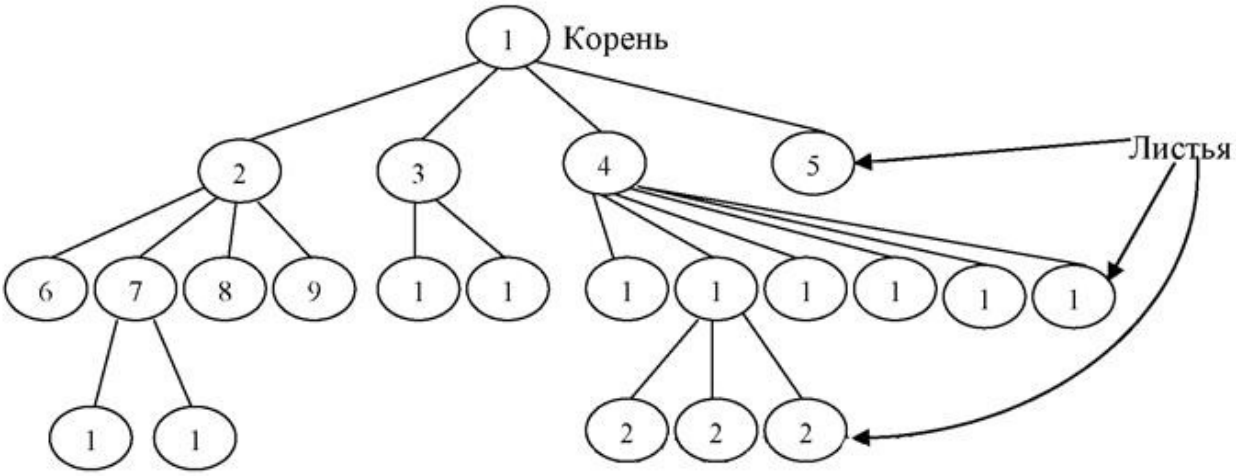


Рис. 15. Пример структуры дерева

Существует несколько способов представления структуры дерева.

Например, дерево может быть определено как иерархия узлов с попарными связями, в которой:

1. Самый верхний уровень иерархии имеет один узел, называемый *корнем.*

2. Все узлы, кроме корня, связываются с одним и только одним узлом на более высоком уровне по отношению к ним самим.

Такое определение в части организации связей совпадает со списком, и, в частности, список представляет вырожденный случай дерева, в котором каждая вершина имеет не более одного поддерева. Заметим, что деревья мы рассматриваем как средство и для логического, и для физического представления данных.

В **логическом описании** данных они используются для определения связей между элементами структуры. При **определении физической организации данных** - для определения набора указателей, реализующих связи между ними.

Использование ссылок для организации доступа к отдельным элементам структуры не позволяет сократить процедуру поиска, в основу которой положен последовательный перебор. Процедура поиска будет эффективнее, если будет предварительно установлен некоторый порядок перехода к следующему элементу дерева. Такой порядок в ряде случаев определяется в отношении *метода обхода* и *регулярности* итераций, определяемой длиной пути и кратностью деления вершины.

Выделяют три метода обхода: сверху вниз, слева направо, снизу вверх. Регулярность обхода дерева может быть связана с *упорядоченными деревьями,* к которым относятся *сбалансированные* (рис. 16) и *двоичные деревья* (рис. 17).

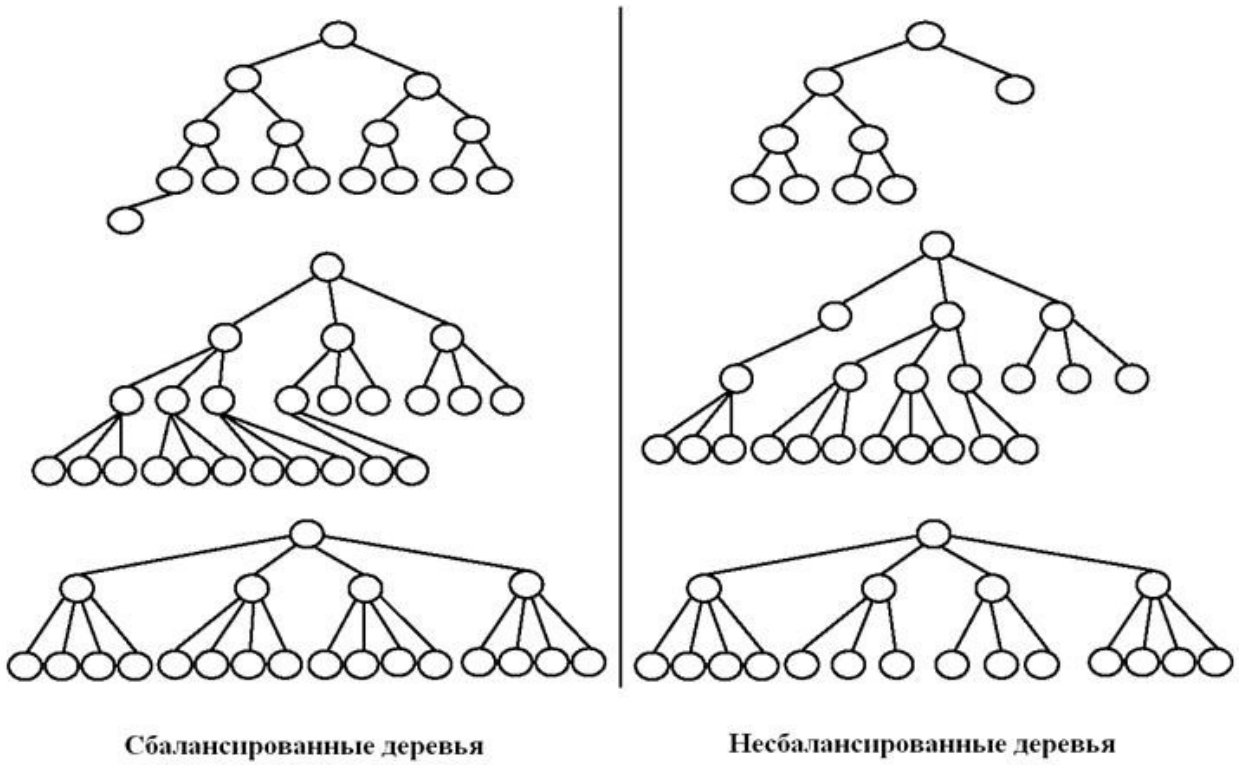


Рис. 16. Примеры сбалансированных и несбалансированных деревьев

Сбалансированное дерево в каждом узле имеет одинаковое число ветвей, причем процесс включения новых ветвей в узлы дерева идет сверху вниз, а на каждом уровне дерева — слева направо. Для дерева с фиксированным числом ветвей физическая организация данных будет более простой. Однако большая часть логических организаций данных не может быть задана в виде сбалансированной древовидной структуры, и для их представления требуется переменное число ветвей в каждом узле. В то же время индексы могут быть построены в виде сбалансированных древовидных структур.

Двоичные деревья - это особая категория сбалансированных древовидных структур, в которой допускается не более двух ветвей для одного узла. На рис. 17 показано несбалансированное двоичное дерево.

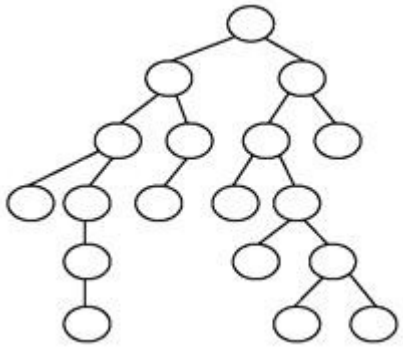


Рис. 17. Пример несбалансированного двоичного дерева

Любые связи в дереве можно представить в виде двоичных древовидных структур. Рис. 18 иллюстрирует представление дерева в виде двоичного дерева.

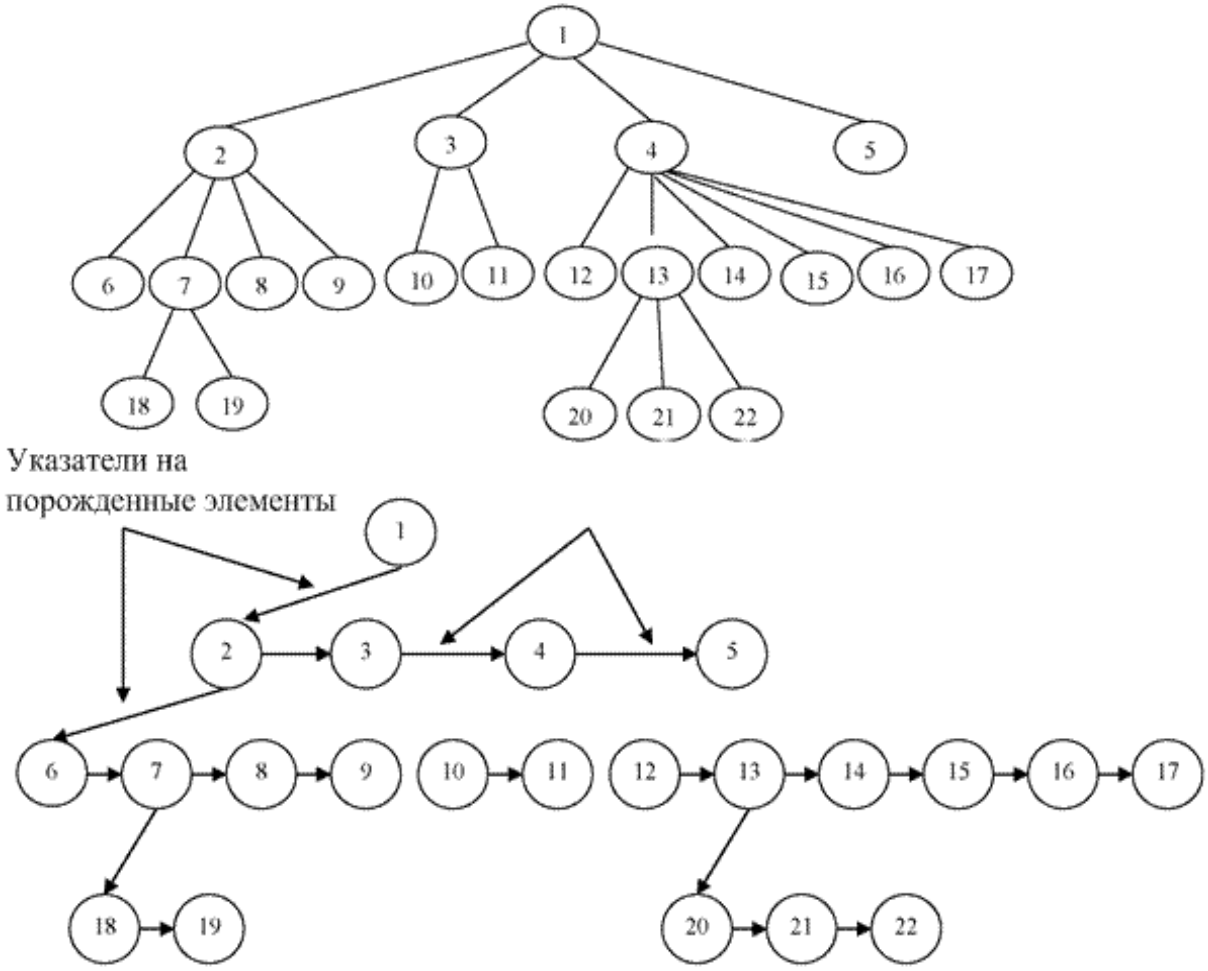


Рис. 18. Пример представления дерева в виде двоичного

При таком представлении каждый элемент может иметь указатели как на порожденные, так и на подобные элементы.

Различные виды двоичных деревьев, для которых характерно наличие жесткой схемы управления их ростом, достаточно эффективно используются для построения больших поисковых индексов, размещаемых обычно на устройствах внешней памяти. Кроме того, для таких деревьев можно организовать специальное «страничное» хранение поддеревьев, что сократит число физических обращений к устройству. Заметим, что деревья поисковых индексов являются *однородными* структурами: каждый узел представлен элементами одного типа. Однако большинство баз должно поддерживать организацию данных, имеющих различную природу. В этом случае при работе с неоднородными структурами разной глубины, гарантировать

регулярность, обеспечивающую эффективность процедур доступа, становится затруднительно.

Сетевые структуры.

Иерархические структуры характерны для многих областей, однако во многих случаях отдельная запись требует более одного представления или связана с несколькими другими. В результате получаются обычно более сложные структуры по сравнению с древовидными. *Например*, генеалогическое дерево может быть представлено в виде древовидной структуры, только если для каждого элемента (личности) будет показан только один исходный элемент (родитель). Если бы были показаны оба родителя, то это была бы более сложная структура.

В сетевой структуре любой элемент может быть связан с любым другим элементом.

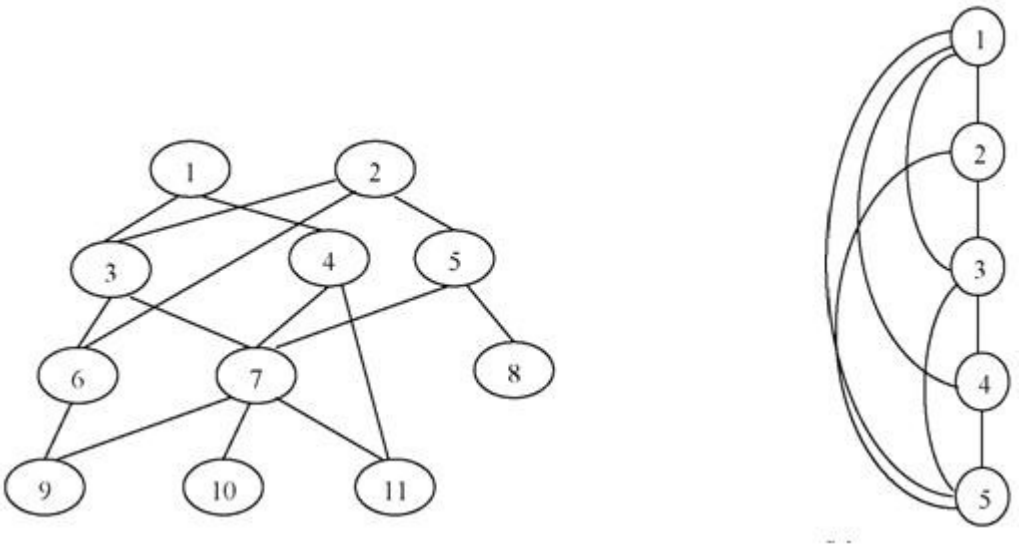


Рис. 19. Пример сетевых структур

Так же как и в случае древовидных структур, сетевую структуру можно описать с помощью исходных и порожденных элементов. Удобно представлять ее так, чтобы порожденные элементы располагались ниже исходных. При рассмотрении некоторых сетевых структур естественно говорить об уровнях, так же как и в случае древовидных структур.

Во многих сетевых структурах, задающих связи между элементами, представление отношений между исходными и порожденными элементами аналогично представлению отношений в случае дерева:

- отношение исходный-порожденный является сложным (указывается сдвоенными стрелками),
- отношение порожденный-исходный - простым (указывается одинарными стрелками).

На рис. 20 показана неоднородная сетевая структура с пятью типами элементов. Ни одна из их соединяющих линий не имеет сдвоенных стрелок в

обоих направлениях. Каждое отношение может рассматриваться как отношение «исходный-порожденный».Запись ЗАКАЗ-НА-ЗАКУПКУ является порожденной по отношению к записи ИЗДЕЛИЕ и исходной по отношению к записи ПАРТИЯ-ТОВАРА.

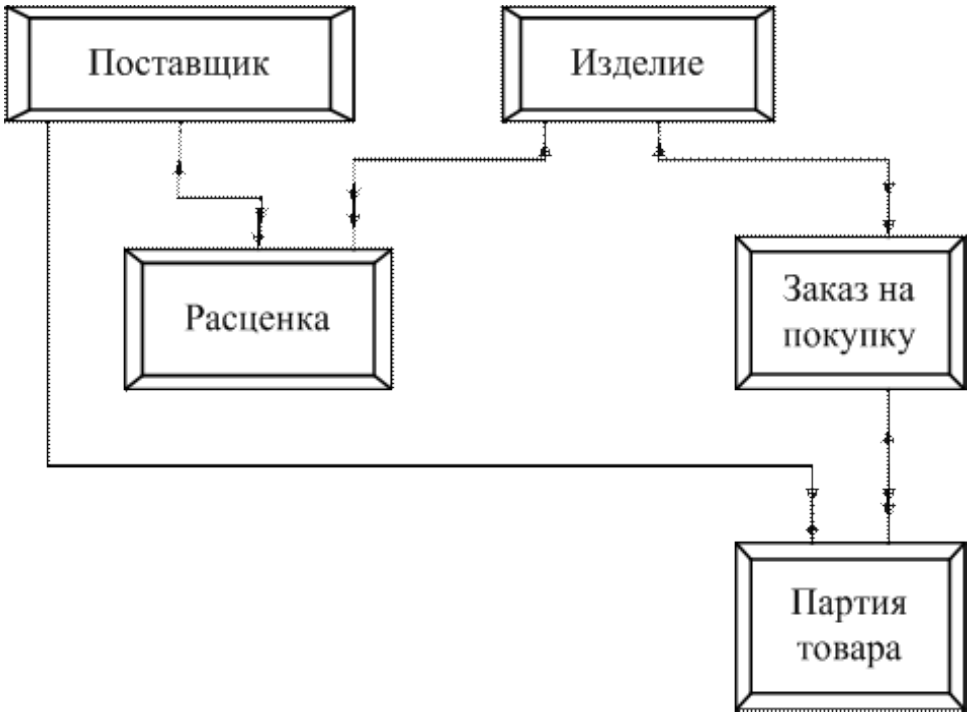


Рис. 20. Пример простой сетевой структуры

Желательно отличать структуры, в которых представление отношений «порожденный-исходный» является *простым* или не используется, от структур, в которых представление отношений между какими-то двумя типами данных является *сложным* в обоих направлениях.

Для структур второго типа на одной из линий схемы будут сдвоенные стрелки, указывающие в разные стороны. Этот тип схемы назовем *сложной* сетевой структурой,

Схему, в которой ни на одной из линий нет сдвоенных стрелок в обоих направлениях,- *простой* сетевой структурой. На рис. 20 показана простая сетевая структура. Она станет *сложной,* если ввести отношение ЗАКАЗ-НА-ЗАКУПКУ - ИЗДЕЛИЕ, когда один заказ может быть сделан сразу на несколько изделий. *Для образования сложной сетевой структуры достаточно двух типов элементов. Например,* ПОСТАВЩИК может иметь несколько порожденных записей, потому что может поставляться более одного вида изделий. С другой стороны, элемент ИЗДЕЛИЕ может иметь более одного исходного элемента, поскольку это изделие может поставляться различными поставщиками.

В некоторых случаях один элемент данных может быть связан с целой *совокупностью* других элементов данных. *Например*, одно изделие может поставляться несколькими поставщиками, каждый из которых

установил свою цену на это изделие. Элемент данных ЦЕНА не может быть ассоциирован только с элементом ИЗДЕЛИЕ или только с элементом ПОСТАВЩИК, а должен быть связан одновременно с двумя. Информация такого рода, т. е. данные, ассоциированные с совокупностью элементов, называют иногда **данными пересечения**.

Некоторые структуры содержат циклы. **Циклом** считается ситуация, в которой предшественник узла является в то же время его последователем. Отношения «исходный-порожденный» образуют при этом замкнутый контур.

Например, завод выпускает различную продукцию. Некоторые изделия производятся на других заводах-субподрядчиках. С одним контрактом может быть связано производство нескольких изделий. Представление этих отношений и образует цикл.

Иногда элементы связаны с другими элементами того же типа. Такая ситуация называется **петлей**. На рис. 21 приведены две достаточно распространенные ситуации, где могут использоваться петли. В массиве служащих специфицированы связи, существующие между некоторыми служащими. В базу данных списка материалов введено дополнительное усложнение: некоторые узлы сами состоят из узлов.

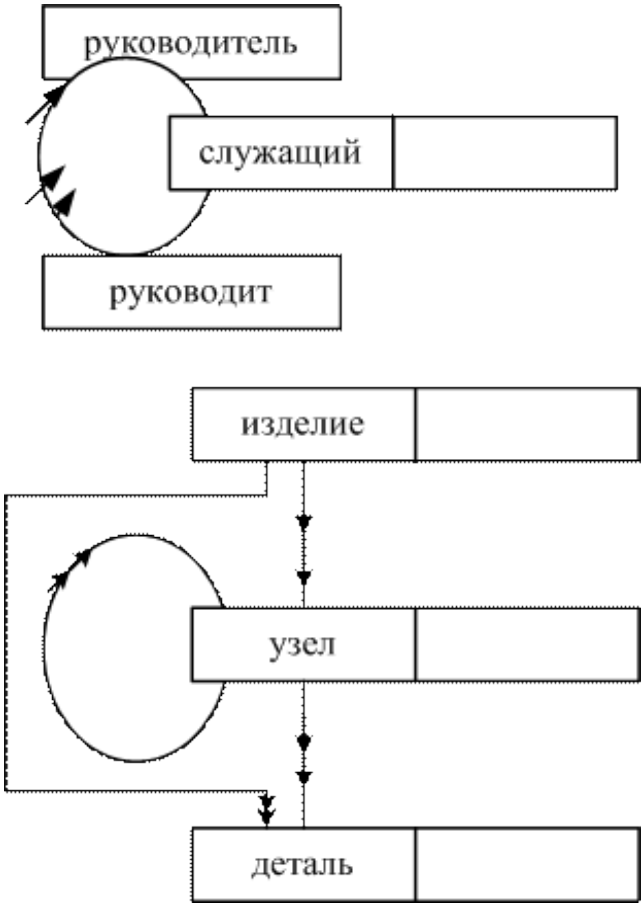


Рис. 21. Пример сетевой структуры с петлей

Разделение сетевых структур на простые и сложные необходимо потому, что сложные структуры требуют более сложных

методов *физического* представления. Это не всегда является недостатком, поскольку сложную сетевую структуру можно (а в большинстве случаев и следует) преобразовать к простому виду.

Вопрос 6. Реляционная модель данных.[10]

Реляционная[11] модель является удобной и наиболее привычной формой представления данных в виде таблицы. В отличие от иерархической и сетевой модели, такой способ представления 1) понятен пользователю-непрограммисту; 2) позволяет легко изменять схему - присоединять новые элементы данных и записи без изменения соответствующих подсхем; 3) обеспечивает необходимую гибкость при обработке непредвиденных запросов. К тому же любая сетевая или иерархическая схема может быть представлена двумерными отношениями.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат. Каждая строка в таблице представляет некоторый объект реального мира или соотношение между объектами. Пользователь модели сам должен для себя решить вопрос, обладают ли соответствующие сущности реального мира однородностью. Этим самым решается проблема пригодности модели для предполагаемого применения.

Основными понятиями, с помощью которых определяется реляционная модель, являются следующие: *домен, отношение, кортеж, кардинальность, атрибуты, степень, первичный ключ.* Соотношение этих понятий иллюстрируется рис. 22. Эти понятия представляют специальную терминологию, введенную авторами теоретических основ, однако они имеют и более привычные аналоги (но не во всем эквиваленты!), соответствие которых приведено в следующей таблице 4.

Таблица 4.

Домен	Совокупность допустимых значений
Кортеж	Строка таблицы
Кардинальность	Количество строк в таблице
Атрибут	Поле, столбец таблицы
Степень отношения	Количество полей (столбцов)
Первичный ключ	Уникальный идентификатор

Домен – это совокупность значений, из которой берутся значения соответствующих атрибутов определенного отношения. С точки зрения программирования домен - это тип данных, определяемый системой (стандартный) или пользователем.

Первичный ключ - это столбец или некоторое подмножество столбцов, которые уникально, т.е. единственным образом определяют строки.

Первичный ключ, который включает более одного столбца, называется множественным, или комбинированным, или составным. Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т.е. иметь значение null.

Остальные ключи, которые можно также использовать в качестве первичных, называются потенциальными или *альтернативными* ключами.

Внешний ключ - это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. *Внешний ключ* таблицы является ссылкой на первичный ключ другой таблицы. Правило ссылочной целостности гласит, что внешний ключ может быть либо пустым, либо соответствовать значению первичного ключа, на который он ссылается. Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют связи между таблицами базы данных.

Внешний ключ, как и первичный ключ, тоже может представлять собой комбинацию столбцов. На практике внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей.

Модель предъявляет к таблицам следующие требования:

1) данные в ячейках таблицы должны быть структурно неделимыми;
2) данные в одном столбце должны быть одного типа;
3) каждый столбец должен быть уникальным (недопустимо дублирование столбцов);
4) столбцы размещаются в произвольном порядке;
5) строки размещаются в таблице также в произвольном порядке;
6) столбцы имеют уникальные наименования.

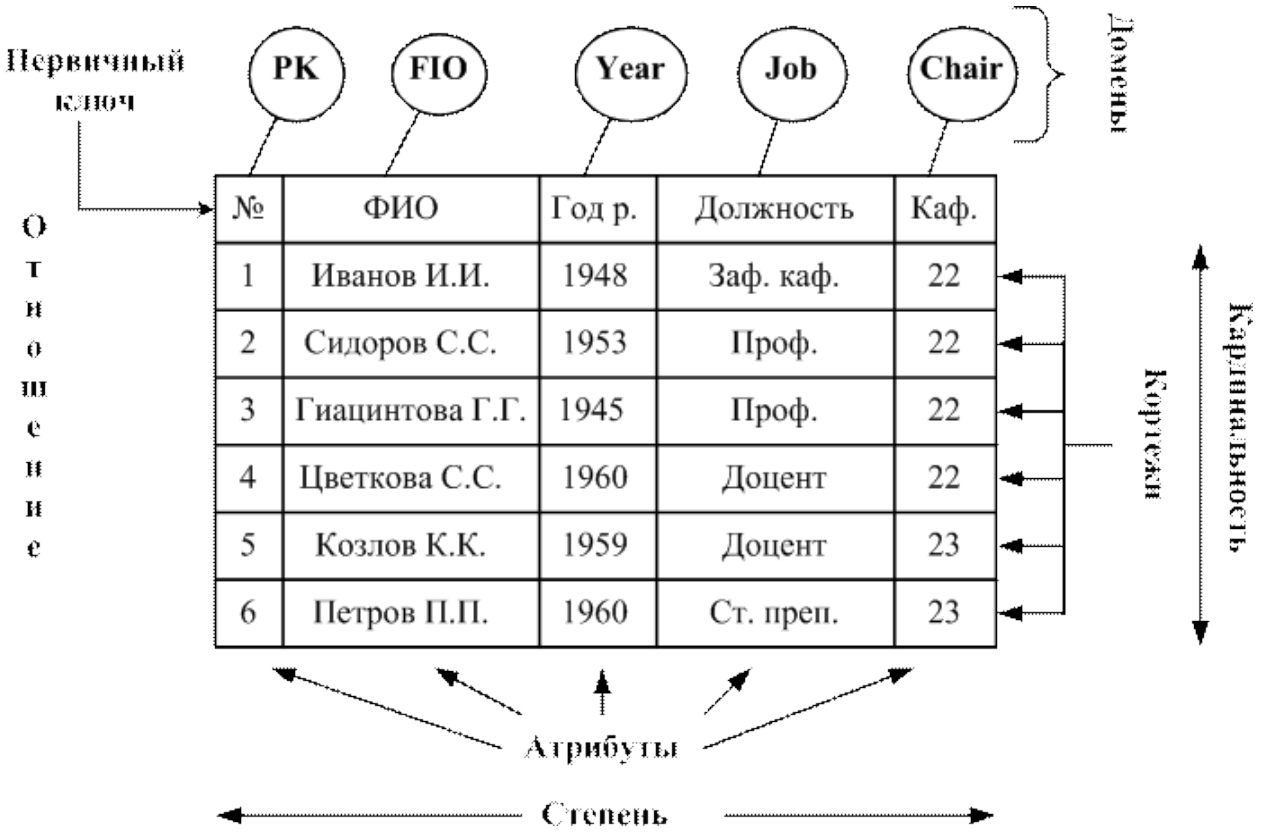


Рис. 22. Основные понятия реляционной модели

В целом концепция реляционной модели определяется следующими двенадцатью правилами Кодда:

1. *Правило информации.* Вся информация в базе данных должна быть предоставлена исключительно на логическом уровне и только одним способом - в виде значений, содержащихся в таблицах.
2. *Правило гарантированного доступа.* Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путём использования комбинации имени таблицы, первичного ключа и имени столбца.
3. *Правило поддержки недействительных значений.* В реляционной базе данных должна быть реализована поддержка недействительных значений, которые отличаются от строки символов нулевой длинны, строки пробельных символов, от нуля или любого другого числа и используются для представления отсутствующих данных независимо от типа этих данных.
4. *Правило динамического каталога, основанного на реляционной модели.* Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользователи, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.
5. *Правило исчерпывающего подъязыка данных.* Реляционная система может поддерживать различные языки и режимы взаимодействия с пользователем (например, режим вопросов и ответов). Однако должен

существовать по крайней мере один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает определение данных; определение представлений; обработку данных (интерактивную и программную); условия целостности; идентификация прав доступа; границы транзакций (начало, завершение и отмена).

6. *Правило обновления представлений.* Все представления, которые теоретически можно обновить, должны быть доступны для обновления.

7. *Правило добавления, обновления и удаления.* Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. *Правило независимости физических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.

9. *Правило независимости логических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.

10. *Правило независимости условий целостности.* Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подъязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. *Правило независимости распространения.* Реляционная СУБД не должна зависеть от потребностей конкретного клиента.

12. *Правило единственности.* Если в реляционной системе есть низкоуровневой язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз).

Правило 2 указывает на роль первичных ключей при поиске информации в базе данных. Имя таблицы позволяет найти требуемую таблицу, имя столбца позволяет найти требуемый столбец, а первичный ключ позволяет найти строку, содержащую искомый элемент данных.

Правило 3 требует, чтобы отсутствующие данные можно было представить с помощью недействительных значений (NULL).

Правило 4 гласит, что реляционная база данных должна сама себя описывать. Другими словами, база данных должна содержать набор *системных таблиц,* описывающих структуру самой базы данных.

Правило 5 требует, чтобы СУБД использовала язык реляционной базы данных, например SQL. Такой язык должен поддерживать все основные функции СУБД - создание базы данных, чтение и ввод данных, реализацию защиты базы данных и т.д.

Правило 6 касается *представлений,* которые являются виртуальными таблицами, позволяющими показывать различным пользователям различные фрагменты структуры базы данных. Это одно из правил, которые сложнее всего реализовать на практике.

Правило 7 акцентирует внимание на том, что базы данных по своей природе ориентированы на множества. Оно требует, чтобы операции добавления, удаления и обновления можно было выполнять над множествами строк. Это правило предназначено для того, чтобы запретить реализации, в которых поддерживаются только операции над одной строкой.

Правила 8 и 9 означают отделение пользователя и прикладной программы от низкоуровневой реализации базы данных. Они утверждают, что конкретные способы реализации хранения или доступа, используемые в СУБД, и даже изменения структуры таблиц базы данных не должны влиять на возможность пользователя работать с данными.

Правило 10 гласит, что язык базы данных должен поддерживать ограничительные условия, налагаемые на вводимые данные и действия, которые могут быть выполнены над данными.

Правило 11 гласит, что язык базы данных должен обеспечивать возможность работы с распределенными данными, расположенными на других компьютерных системах.

Правило 12 предотвращает использование других возможностей для работы с базой данных, помимо языка базы данных, поскольку это может нарушить ее целостность.

Вопрос 7. Основы реляционной алгебры.[12]

С точки зрения внешнего представления (абстрагирования на логическом уровне) объектов реального мира *модель данных* - это основные понятия и способы, используемые при анализе и описании предметной области.

Среди многих попыток представить обработку данных на формальном абстрактном уровне реляционная модель, предложенная Э.Ф. Коддом, стала по существу первой работоспособной *моделью данных,* поскольку помимо средств описания объектов имела эффективный инструментарий преобразований этих описаний - операции реляционной алгебры.

Реляционная алгебра в том виде, в котором она была определена Э.Ф. Коддом, состоит из двух групп по четыре оператора.

1. Традиционные операции над множествами (но модифицированные с учетом того, что их операндами являются отношения, а не произвольные множества): объединение, пересечение, разность и декартово произведение.
2. Специальные реляционные операции: выборка, проекция, соединение, деление.

Рассмотрим подробнее операции реляционной алгебры.

Объединение возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим.

Объединение

FIO	*Year*	*Job*	*Chair*
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

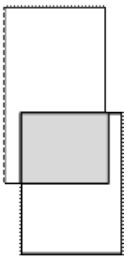
FIO	*Year*	*Job*	*Chair*
Иванов. И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	*Year*	*Job*	*Chair*
Иванов И.И.	1943	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

Рис. 23

Пересечение возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям.

Пересечение



FIO	*Year*	*Job*	*Chair*
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	*Year*	*Job*	*Chair*
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

FIO	*Year*	*Job*	*Chair*
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

Рис. 24

Разность возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух заданных отношений и не принадлежат второму.

Разность

FIO	Year	Job	Chair
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

FIO	Year	Job	Chair
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютникова Л.Л.	1977	Ассистент	24

FIO	Year	Job	Chair
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22

Рис. 25

Произведение - возвращает отношение, содержащее все возможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум заданным отношениям.

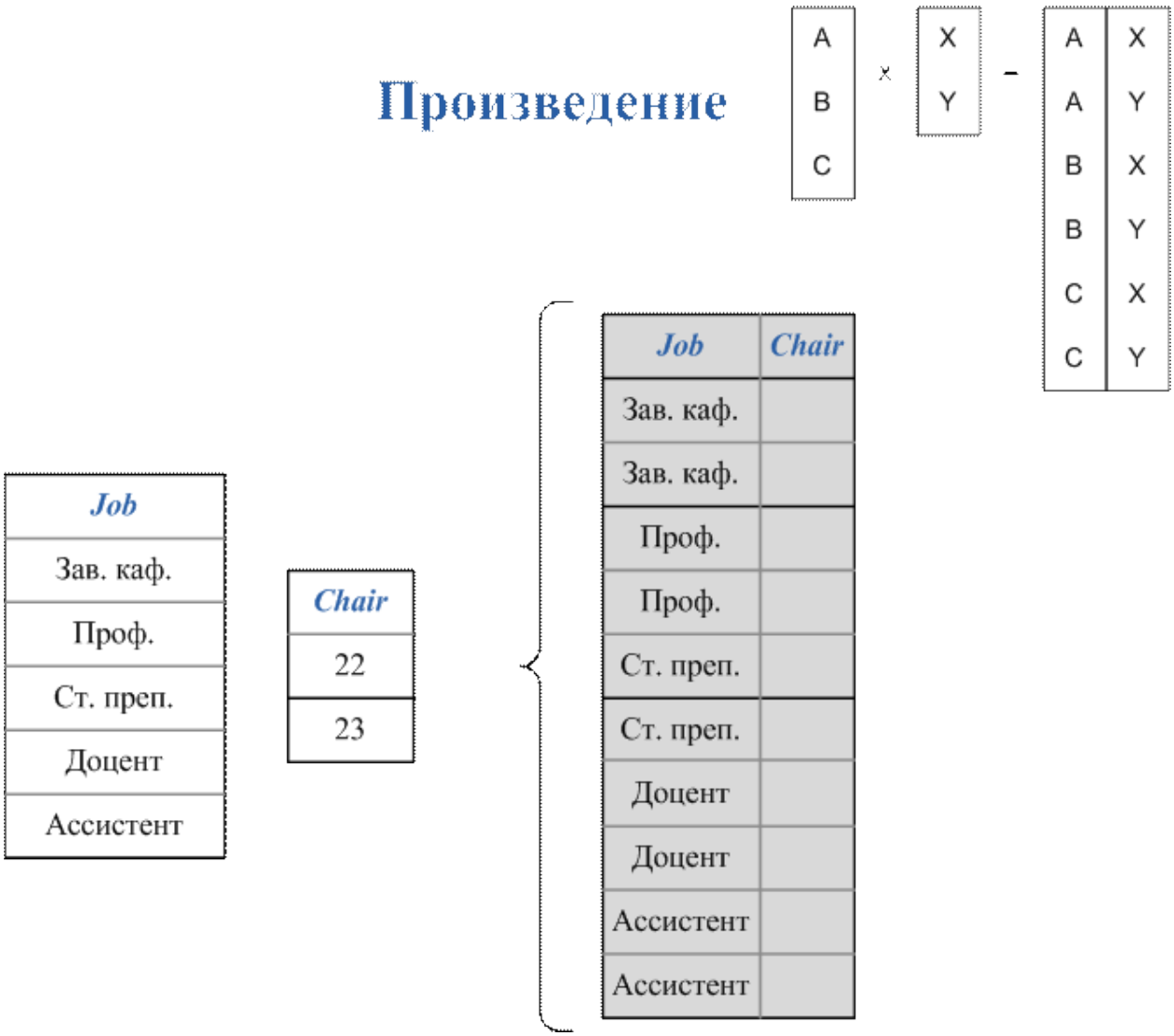


Рис. 26

Выборка - возвращает отношение, содержащие все кортежи из заданного отношения, которые удовлетворяют указанным условиям.

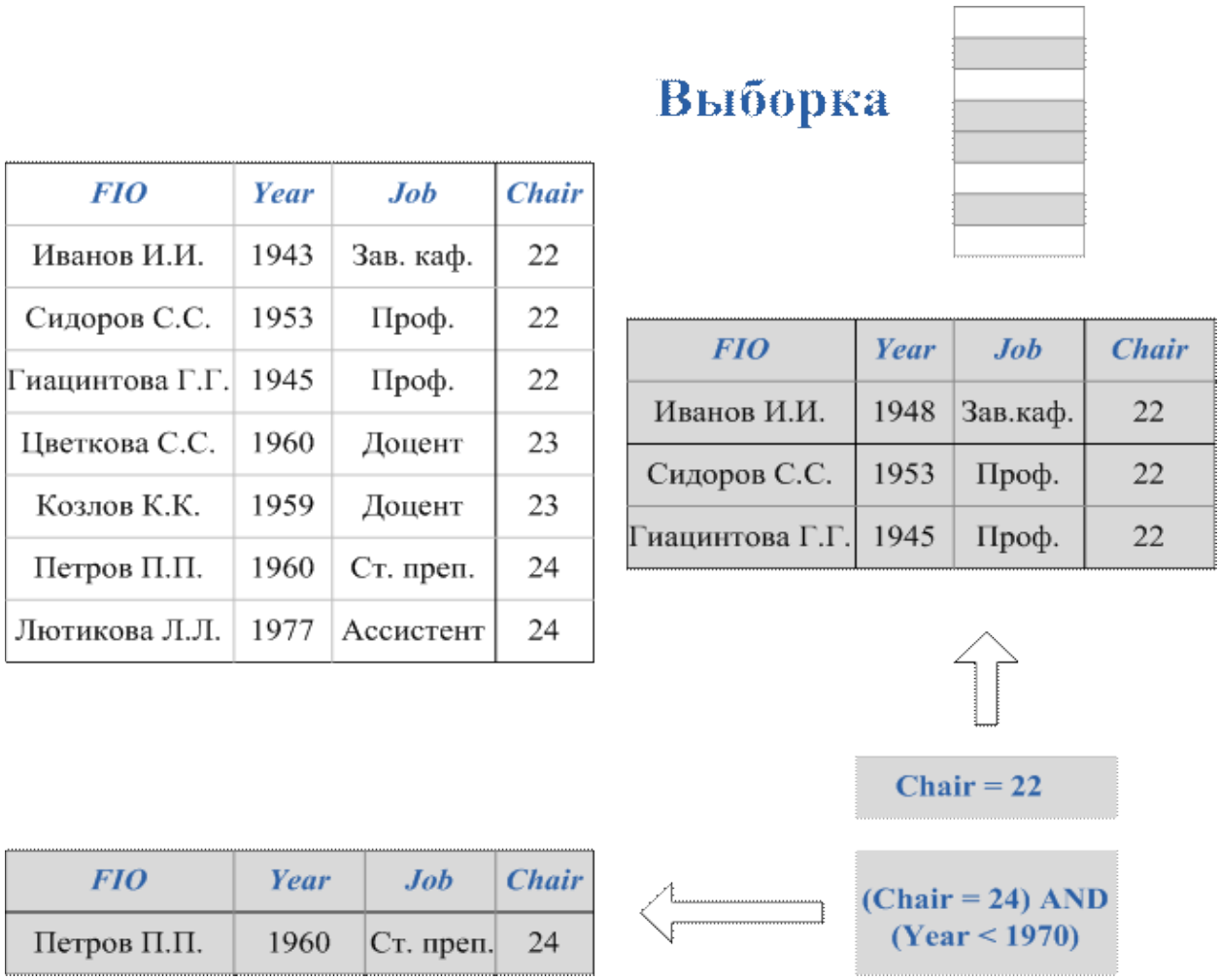
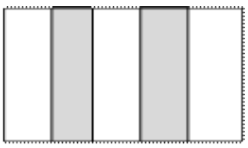


Рис. 27

Проекция возвращает отношение, содержащее все кортежи (подкортежи) заданного отношения, которые остались в этом отношении после исключения из него некоторых атрибутов.

Проекция 

FIO	*Year*	*Job*	*Chair*
Иванов И.И.	1943	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

FIO	*Job*
Иванов И.И.	Зав. каф.
Сидоров С.С.	Проф.
Гиацинтова Г.Г.	Проф.
Цветкова С.С.	Доцент
Козлов К.К.	Доцент
Петров П.П.	Ст. преп.
Лютикова Л.Л.	Ассистент

Рис. 28

Соединение возвращает отношение, содержащее все возможные кортежи, которые представляют собой комбинацию атрибутов двух кортежей, принадлежащих двум заданным, при условии, что в этих двух комбинированных кортежах присутствуют одинаковые значения в одном или нескольких общих для исходных отношений атрибутах (причем эти общие значения в результирующем кортеже появляются один раз, а не дважды).

Соединение

A1	B1		B1	C1		A1	B1	C1
A2	B1		B2	C1		A2	B1	C1
A3	B3		B3	C3		A3	B2	C2

FIO	*Job*	*Chair*
Иванов И.И.	Зав. каф.	22
Сидоров С.С.	Проф.	22
Гиацинтова Г.Г.	Проф.	22
Цветкова С.С.	Доцент	23
Козлов К.К.	Доцент	23
Петров П.П.	Ст. преп.	24
Лютикова Л.Л.	Ассистент	24

Job	*Pay*
Зав.каф.	3000
Проф.	2500
Доцент	2000
Ст.преп.	1500
Ассистент	1200

FIO	*Job*	*Chair*	*Pay*
Иванов И.И.	Зав.каф.	22	3000
Сидоров С.С.	Проф.	22	2500
Гиацинтова Г.Г.	Проф.	22	2500
Цветкова С.С.	Доцент	23	2000
Козлов К.К.	Доцент	23	2000
Петров П.П.	Ст.преп.	24	1500
Лютикова Л.Л.	Ассистент	24	1200

Рис. 29

Деление для заданных двух унарных отношений и одного бинарного возвращает отношение, содержащее все кортежи из первого унарного отношения, которые содержатся также в бинарном отношении и соответствуют всем кортежам во втором унарном отношении.

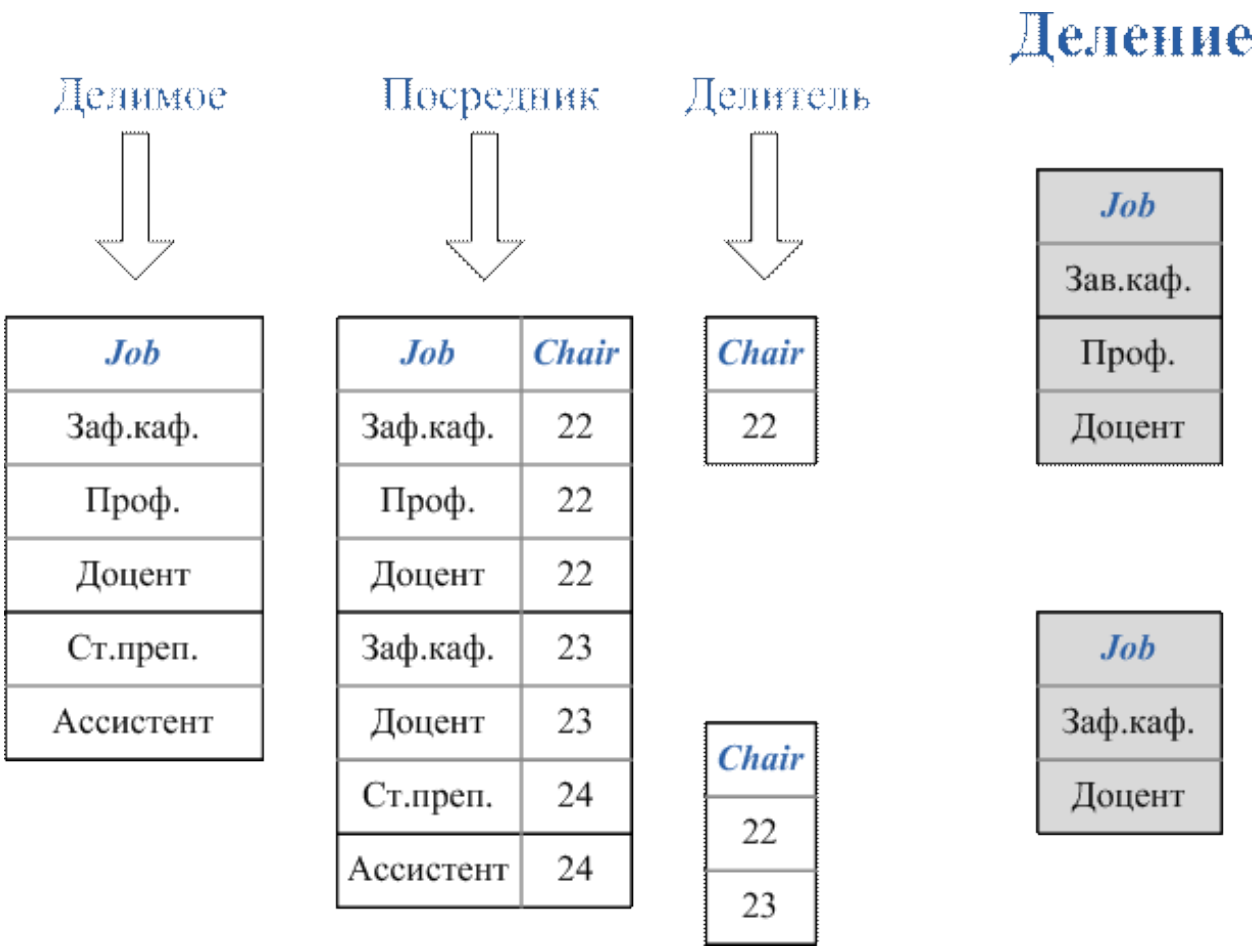


Рис. 30

Результат выполнения любой операции над отношением также является отношением, поэтому результат одной операции может использоваться в качестве исходных данных для другой. Другими словами, можно записывать вложенные реляционные выражения, т.е. выражения, в которых операторы сами представлены реляционными выражениями, причем произвольной сложности. Эта особенность называется *свойством реляционной замкнутости.*

Важно, что отношение имеет две части - заголовок и тело. Нестрого говоря, заголовок - это атрибуты, а тело - это картежи. Заголовок для базового отношения, т.е. значение базовой переменной-отношения, очевидно, вполне конкретен и известен системе, поскольку он задается как часть определения соответствующей базовой переменной-отношения. Т.к. результат обязательно должен иметь вполне определенный тип отношения, поэтому, если рассматривать свойство реляционной замкнутости более строго, каждая реляционная операция должна быть определена таким образом, чтобы выдавать результат с надлежащим типом отношения (в частности, с соответствующим набором имен атрибутов или заголовком).

Реляционная алгебра имеет набор правил вывода типов (отношений), позволяющих вывести тип (отношение) на выходе произвольной реляционной операции, зная типы (отношения) на входе этой операции.

Задав такие правила для всех операций, можно гарантировать, что для реляционного выражения любой сложности будет вычисляться результат, имеющий вполне определенный тип (отношение) и, в частности, известный набор имен атрибутов.

Рассмотренные восемь операторов Кодда не являются минимальным набором, так как не все из них примитивны, т.е. часть из них можно определить через другие операторы. Действительно, операции соединения, пересечения и деления можно определить через остальные пять. Эти пять операций (выборка, проекция, произведение, объединение и разность) можно рассматривать как примитивные в том смысле, что ни одна из них не выражается через другие. Они образуют минимальный набор, но, тем не менее, необязательно единственно возможный. Кроме того, остальные три операции (в особенности операция соединения) на практике используются настолько часто, что, несмотря на то, что они не являются примитивными, имеет смысл обеспечить их непосредственную поддержку.

Предшествующее рассмотрение алгебры представлено в контексте только операций выборки данных. Однако, как отмечается в классических введениях к реляционной алгебре, ее основная цель - обеспечить запись реляционных выражений позволяющих определять:

- области выборки, т.е. тех данных, которые должны быть доставлены в результате выполнения операции выборки;
- области обновления, т.е. данных, которые должны быть вставлены, изменены или удалены в результате выполнения операции обновления;
- правила поддержки целостности данных, т.е. некоторых особых требований, которым должна удовлетворять база данных;
- производные переменные-отношения, т.е. те данные, которые должны быть включены в представления базы данных;
- требования устойчивости, т.е. данные, которые должны быть включены в контролируемую область для некоторых операций управления параллельным доступом к информации;
- ограничения защиты, т.е. данные, для которых осуществляется тот или иной тип контроля доступа.

В целом, выражения реляционной алгебры служат для символического высокоуровневого представления намерений пользователя (например, в отношении некоторого определенного запроса). И именно потому, что подобные выражения являются символическими и высокоуровневыми, ими можно манипулировать в соответствии с различными высокоуровневыми правилами преобразования, в том числе и для оптимизации процедур выполнения запросов на данные.

Вопрос 8. Модели и технологии инфологического проектирования реляционных БД. [13]

Как отмечалось ранее, представляется вполне очевидным начинать создание базы данных с определения самих данных, выполняя проектирование базы данных в терминах отношений на основе механизма нормализации. Однако, забегая вперед, отметим, что такой подход часто представляет собой очень сложный и неудобный процесс для самого проектировщика.

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- реляционная модель не предоставляет достаточных средств для фиксации смысла данных, т.е. семантика предметной области не фиксируется непосредственно в отношениях;
- для многих приложений трудно моделировать предметную область на основе плоских таблиц;
- хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не имеет средств представления (отражения семантики) этих зависимостей;
- несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области («сущностей») и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для различения сущностей и связей в базе данных.

На практике семантическое моделирование обычно производится на первой стадии проектирования. **Полученный результат** - концептуальная схема базы данных (в терминах семантической модели) затем вручную или автоматически преобразуется к реляционной схеме.

Инфологическое проектирование и семантическая модель.

Начальной стадией проектирования системы баз данных является построение **семантической модели** предметной области, которая базируется на анализе свойств и природы объектов предметной области и информационных потребностей будущих пользователей разрабатываемой системы. Эту стадию принято называть **концептуальным проектированием системы,** а ее результат - **концептуальной моделью** предметной области (объектом моделирования здесь является предметная область будущей системы!). Этой стадии соответствуют также ранее упомянутые термины **«инфологическое проектирование»** и **«инфологическая модель»**.

Инфологические модели обобщенно представляют информационные потребности пользователей создаваемой системы в части использования хранимых данных и, по существу, являются средством коммуникации как разработчиков, так и пользователей на разных стадиях жизненного цикла базы данных.

Отметим, что они, в отличие от моделей данных, используемых в качестве инструмента моделирования конкретных баз данных, не

обязательно поддерживаются механизмами используемой СУБД, хотя это и может иметь место на практике

Назначение инфологических моделей определяет и некоторые специфические требования к средствам их представления.

Помимо упомянутой независимости от среды (оборудования) и требования адекватности отражения предметной области отметим следующие:

- **формализованность**, обеспечивающую возможность автоматизированной обработки и, в том числе, например, автоматический контроль непротиворечивости;
- **дружественность**, обеспечивающую возможность использования наглядных графических средств отображения и обработки их пользователем.

К инфологическим моделям относятся различные компоненты, по-разному и разными средствами отражающие предметную область. К инфологическому уровню описания предметной области можно отнести следующие компоненты:

- систему описания объектов и связей между ними (модель «сущность-связь»)
- систему атрибутов и средств описания предметной области.

Например, логические (алгоритмические) связи между показателями или лингвистические свойства языка (синонимию, синтаксис и т.д.), используемого для вербального представления объектов;

- ограничения целостности, определяющие допустимость значения отдельных полей и взаимосвязей как на уровне семантики содержимого БД, так и ее физической структуры (отдельных файлов данных и взаимосвязей между ними);
- описание информационных потребностей пользователей, например, в виде типовых запросов, отражающих процедурные особенности обращения к данным.

Модель «Сущность - связь».

Одной из наиболее популярных средств формализованного представления предметной области систем, ориентированных на обработку фактографической информации, является модель «сущность-связь», которая положена в основу значительного количества коммерческих CASE-продуктов, поддерживающих полный цикл разработки систем баз данных или отдельные его стадии.

При этом многие из них поддерживают стадию концептуального проектирования предметной области разрабатываемой системы (*все поддерживают*), позволяют осуществить на основе построенной их средствами модели стадию логического проектирования путем автоматической генерации концептуальной схемы базы данных для

выбранной СУБД (*например, схемы базы данных для какого-либо SQL-сервера или объектной СУБД*).

Моделирование предметной области в этом случае базируется на использовании графических диаграмм, включающих сравнительно небольшое число компонентов и, самое важное - **технологию построения** таких диаграмм.

Существует много версий ER-диаграмм, которые по-разному представляют связи, сущности и атрибуты, и которые имеют различные ограничения и условия применимости. Приведенный здесь пример не является полным и не претендует на точное соответствие какой либо версии построения ER-диаграмм или CASE-продукту.

Семантическую основу ER-модели составляют следующие предположения:

- та часть реального мира (совокупность взаимосвязанных объектов), сведения о которых должны быть помещены в базу данных, может быть *представлена* как совокупность **сущностей;**
- каждая сущность обладает характеристическими свойствами (атрибутами), отличающими ее от других сущностей и позволяющими ее *идентифицировать;*
- сущности можно классифицировать по типам сущностей: каждый экземпляр сущности (представляющий некоторый объект) может быть отнесен классу - **типу сущностей**, каждый экземпляр которого обладает общими для них свойствами и отличающим их от сущностей других классов;
- систематизация представления, основанная на классах, в общем случае предполагает иерархическую зависимость типов: сущность типа *A* является **подтипом** сущности *B,* если каждый экземпляр типа *A* является экземпляром сущности типа *B;*
- взаимосвязи объектов могут быть представлены как **связи** - сущности[39], которые служат для фиксирования (представления) взаимозависимости двух или нескольких сущностей.

Такое определение связи, как сущности особого рода, отражает существо реляционного подхода, для которого характерно единообразное представление сущностей всех типов, включая связи, посредством переменных-отношений.

Здесь следует еще раз подчеркнуть информационную природу понятия *сущность* и его соотношение с материальными или воображаемыми объектами предметной области. Любой объект предметной области обладает свойствами, часть из которых выделяется как характеристические - значимые с точки зрения прикладной задачи. При этом, например, в процессе анализа и систематизации предметной области, обычно выделяются *классы* - совокупности объектов, обладающих одинаковым набором свойств, задаваемых в виде *наборов атрибутов* (значения атрибутов для объектов одного класса естественно могут различаться).

Соответственно, на уровне представления предметной области (т.е. - ее инфологической модели)

- объекту, рассматриваемому как понятие (объект в сознании человека), соответствует понятие *сущность;*
- объекту, как части материального мира (и существующему независимо от сознания человека), соответствует понятие *экземпляр сущности;*
- классу объектов соответствует понятие *тип сущности.*

В дальнейшем, поскольку в инфологической модели рассматриваются не отдельные экземпляры объектов, а классы, мы не будем различать соответствующие понятия этих двух уровней, т.е. будем предполагать тождественность понятий *объект* и *сущность, свойство объекта* и *свойство сущности.*

ER-модель, как описание предметной области, должна определить объекты и взаимосвязи между ними, т.е. установить связи следующих двух типов:

- связи между объектами и наборами характеристических свойств и таким образом определить сами объекты;
- связи между объектами, задающие характер и функциональную природу их взаимозависимости.

ER-моделирование предметной области базируется на использовании графических диаграмм, как простого (привычного), наглядного и, в то же время, информативного и многоаспектного способа отображения компонентов проекта.

Поэтому изложение основных положений ER-модели будет иллюстрироваться материалом примера ER-диаграммы, приведенной на рис. 31.

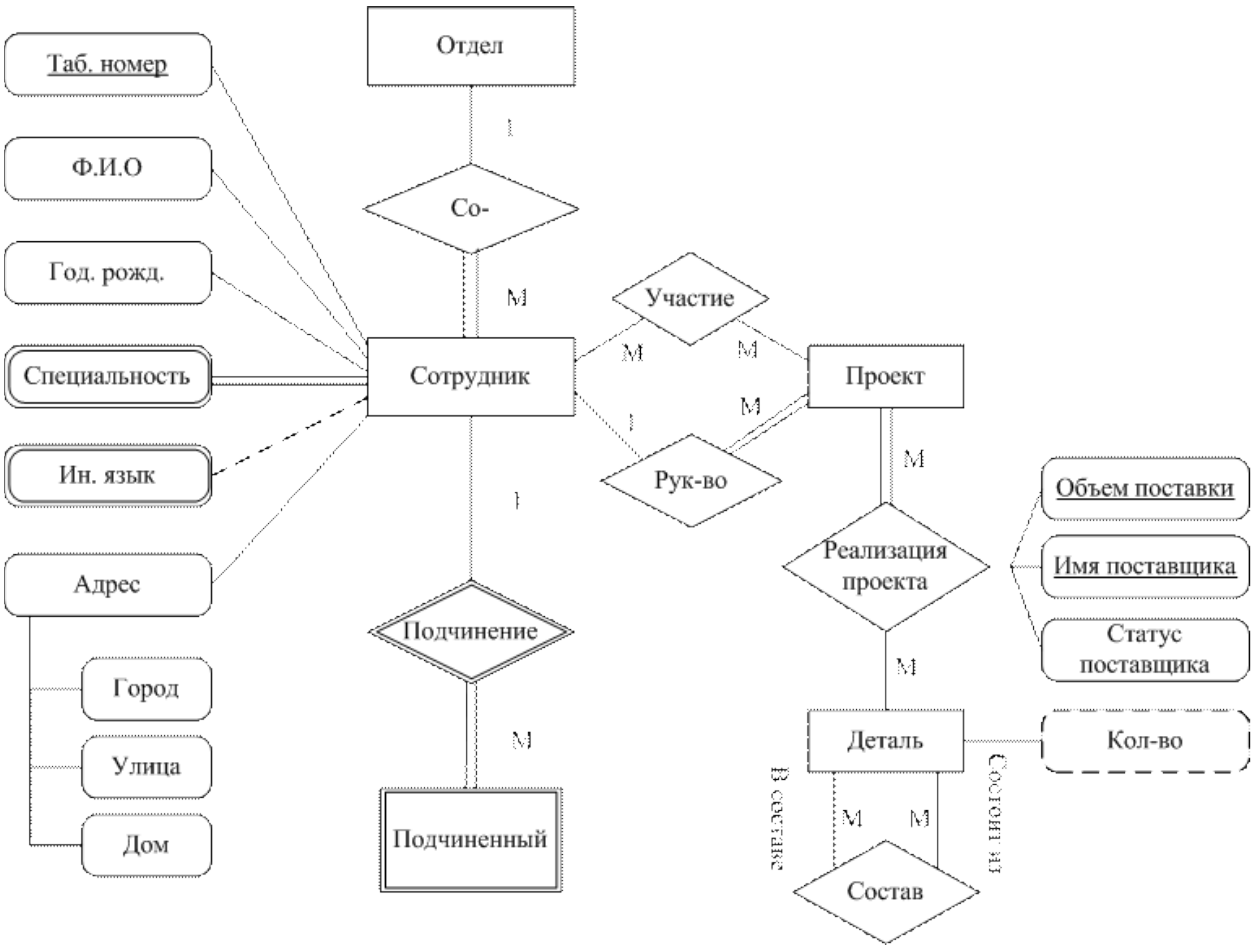


Рис. 31. Пример ER-диаграммы

Сущность.

Сущность, с помощью которой моделируется класс однотипных объектов, определяется как «предмет, который может быть четко идентифицирован».

Так же, как каждый объект уникально характеризуется набором значений свойств, сущность должна *определяться* таким *набором атрибутов,* который позволял бы различать отдельные экземпляры сущности.

Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах).

Например, для однозначной идентификации каждого экземпляра сущности «Сотрудник» вводится атрибут «Таб.номер», который вследствие своей природы будет всегда иметь уникальное значение в рамках предприятия.

Т.е., уникальным идентификатором сущности может являться атрибут, комбинация атрибутов, комбинация связей или комбинация связей и атрибутов, однозначно отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

Сущность имеет *имя,* уникальное в пределах модели.

При этом *имя сущности* - это *имя типа,* а не некоторого конкретного экземпляра.

Сущности подразделяются на *сильные* и *слабые.* Сущность является слабой, если ее существование зависит от другой сущности - сильной, по отношению к ней. Например, сущность «Подчиненный» является слабой по отношению к сущности «Сотрудник»: если будет удалена запись, соответствующая некоторому сотруднику, имеющему подчиненных, то сведения о подчинении также должны быть удалены.

Свойства.

Природа свойства, как *характер связи* свойства с сущностью (объектом), может быть различной. Рассмотрим основные виды свойств. Свойство может быть *множественным* или *единичным* - т.е. атрибут, задающий свойство, может одновременно иметь несколько значений или, соответственно, только одно. Например, сотрудник может иметь несколько специальностей, но единственное значение «Таб. номер». Свойство может быть *простым* (не подлежащих дальнейшему делению с точки зрения прикладных задач) или *составным* - если его значение составляется из значений простых свойств. Например, свойство «Год рождения» является простым, а свойство «Адрес» - составным, т.к. включает значения простых свойств «Город», «Улица», «Дом».

В некоторых случаях полезно различать *базовые* и *производные* свойства. Например, «Поставщик» может иметь свойство «Общее количество поставляемых деталей», которое вычисляется суммированием количества деталей, поставляемых им по проекту.

Если наличие некоторого свойства для всех экземпляров сущности не является обязательным, то такое свойство называется *условным.* Например, не все сотрудники обладают свойством «ученая степень».

Значения свойств могут быть постоянными - *статическими,* или *динамическими,* т.е. меняться со временем. Например, свойство «Таб. номер» является статическим, а «Адрес» - динамическим. Свойство может быть *неопределенным,* если оно является динамическим, но его текущее значение еще не задано.

Свойство может рассматриваться как *ключевое,* если его значение уникально и, возможно, в определенном контексте, однозначно идентифицирует сущность. Например, подчиненный некоторого определенного сотрудника.

Связи.

Кроме связей между объектом и его свойствами, инфологическая модель отражает связи между объектами разных классов.

Связь - ассоциация, объединяющая несколько сущностей. Эта ассоциация всегда может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Как и сущность, связь

является *типовым* понятием, т.е. все экземпляры связываемых сущностей подчиняются правилам связывания типов.

Принципиальность различия типов связей между типами и экземплярами иллюстрируется ER-диаграммами для типов и экземпляров, представленными на рис. 32.

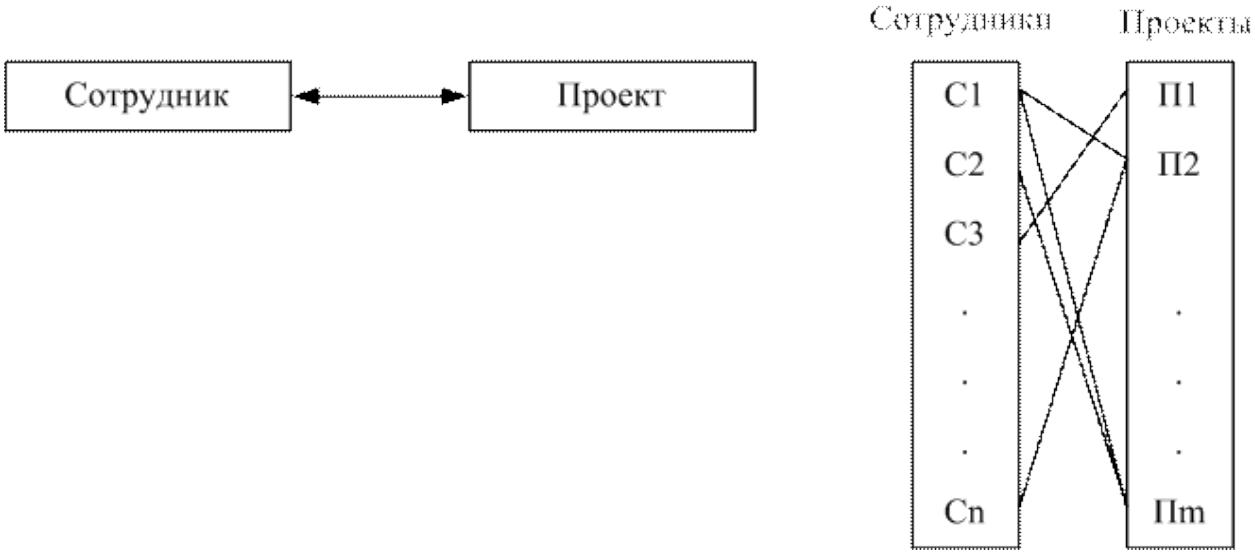


Рис. 32. Примеры ER-диаграмм для типов и экземпляров сущностей

Сущности, объединяемые связью, называются *участниками.*

Степень связи определяется количеством участников связи. В большинстве CASE-систем принята упрощенная форма: эта ассоциация всегда **должна быть бинарной** и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). Если каждый экземпляр сущности участвует, по крайней мере, в одном экземпляре связи, то такое участие этой сущности называется полным (или обязательным); в противном случае - неполным (или необязательным).

Количественный характер участия экземпляров сущностей (один или многие) задается *типом связи* (или *мощностью связи).* Возможны следующие типы: «один к одному» (1:1), «один ко многим» (1:M), «многие к одному» (M:1), «многие ко многим» (M:M).

Следует отметить, что инструмент связей - это средство представления *сложных объектов,* каждый из которых может рассматриваться как множество некоторым образом взаимосвязанных *простых объектов.* Деление на простые и сложные объекты, также как и характер взаимосвязи, является условным и определяется особенностям анализа предметной области, т.е. в конце концов - характером использования данных о предметах в решаемых прикладных задачах. При этом с точки зрения, например, конструктора, ДЕТАЛЬ является сложным объектом, а с точки зрения поставщика - простым.

Среди многих разновидностей взаимосвязей наиболее частыми являются такие отношения иерархического типа, как «часть-целое», «род-вид». Отношение «часть-целое» используются для представления *составных объектов.* Например, МАШИНЫ состоят из УЗЛОВ, УЗЛЫ состоят из ДЕТАЛЕЙ. Здесь возможны как отношения *«один ко многим», т*ак *и «многие ко многим».*

Отношение «род-вид» - для представления *обобщенных объектов.*

Например, СОТРУДНИКИ подразделяются по профессии на КОНСТРУКТОРОВ, ПРОГРАММИСТОВ, РАБОЧИХ; ПРОГРАММИСТЫ - на ПРИКЛАДНЫХ ПРОГРАММИСТОВ и СИСТЕМНЫХ ПРОГРАММИСТОВ.

Иерархические отношения, и, в частности - «родовидовые», обычно используются как основа классификации объектов по наборам характеристических признаков. Причем «видовые» объекты *наследуют* свойства «родовых».

Другой широко используемой разновидностью взаимосвязи является агрегирование - объединение простых объектов в сложный по принципу их принадлежности *агрегату* или их совместного участия в некотором процессе. Агрегирование, рассматриваемое здесь как более общий случай иерархических отношений, объединяет объекты разной природы с единственным общим свойством «совместное участие». Агрегированные объекты именуются обычно отглагольными существительными, например, *«Состав»:* ПОДРАЗДЕЛЕНИЕ *состоит из* СОТРУДНИКОВ; *«Поставка»:* ПОСТАВЩИК *поставляет* ДЕТАЛИ.

Супертипы и подтипы.

Сущность может быть расщеплена на два или более взаимно исключающих *подтипов,* каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе выделение подтипов может продолжаться на более низких уровнях, но в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называется *супертипом.* Подтипы должны образовывать полное множество, т.е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для полноты множества надо определять дополнительный подтип, например ПРОЧИЕ.

Подтип наследует свойства и связи супертипа. Например, тип сущности ПРОГРАММИСТ является подтипом сущности СОТРУДНИК. Программисты обладают всеми свойствами сотрудников и участвуют во всех связях, однако обратные утверждения неверны.

Тип сущности, его подтипы, подтипы этих подтипов и т.д. образуют *иерархию типов сущности,* пример которой приведен на рис. 33.

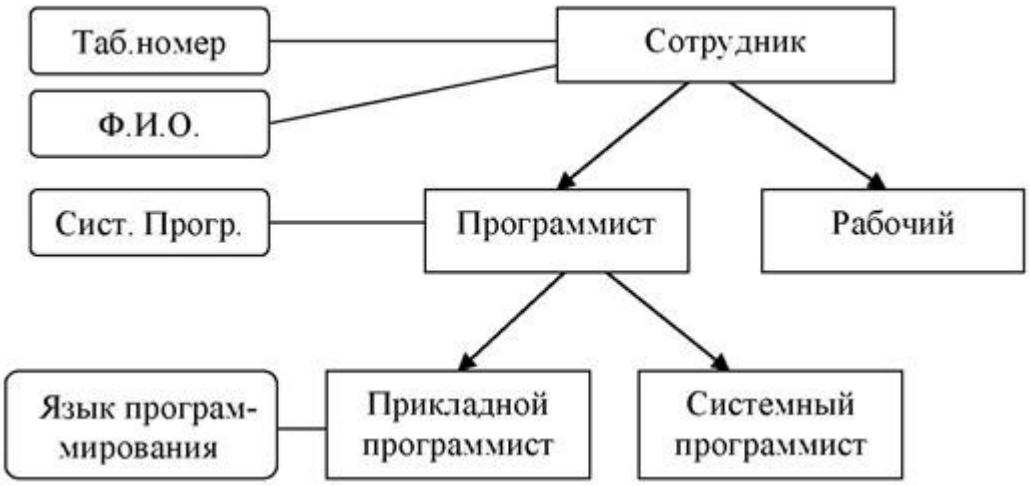


Рис. 33. Пример иерархии типов сущности

ER - диаграмма.

Как отмечалось ранее, одна из основных целей семантического моделирования состоит в том, чтобы результаты анализа предметной области были отражены в достаточно простом, наглядном но, в то же время, формализованном и достаточно информативном виде. В этом смысле ER-диаграмма является очень удачным решением. В ней сочетаются функциональный и информационный подходы, что позволяет представлять как совокупность выполняемых функций, так и отношения между элементами системы, задаваемые структурами данных.

При этом графическая форма позволяет отобразить в компактном виде (за счет наглядных условных обозначений) типологию и свойства сущностей и связей, а формализмы, положенные в основу ER-диаграмм, позволяют использовать на следующем шаге проектирования логической структуры базы данных строгий аппарат нормализации.

Сущности.

Каждый тип сущности в ER-диаграммах представляется в виде прямоугольника, содержащего имя сущности. В качестве имени обычно используются существительные (или обороты существительного) в единственном числе. Для отражения сущностей слабых типов используются прямоугольники, стороны которых рисуются двойными линиями.

Например, в рассматриваемой далее ER-диаграмме, приведенной на рис. 31, ПОДЧИНЕННЫЙ - сущность слабого типа.

Свойства.

Свойства служат для уточнения, идентификации, характеристики или выражения состояния сущности или связи. Свойства отображаются в виде эллипсов, содержащих имя свойства. В большинстве CASE-систем имена свойств заносятся в прямоугольник, изображающий сущность, под именем

сущности и изображаются малыми буквами, возможно, с примерами. Эллипс соединяется с соответствующей сущностью или связью линией.

Имена ключевых свойств подчеркиваются. Например, свойство «Таб.номер» сущности СОТРУДНИК.

Контур эллипса рисуется двойной линией, если свойство многозначное. Например, свойство «специальность» сущности СОТРУДНИК.

Контур эллипса рисуется штриховой линией, если свойство производное. Например, свойство «кол-во» сущности ПОСТАВЩИК.

Эллипс соединяется пунктирной линией, если свойство условное. Например, свойство «Ин. язык» сущности СОТРУДНИК.

Если свойство составное, то составляющие его свойства отображаются другими эллипсами, соединенными с эллипсом составного. Например, свойство «Адрес» сущности СОТРУДНИК состоит из простых свойств «Город», «Улица», «Дом».

Связи. Связь - это графически изображаемая ассоциация, устанавливаемая между сущностями. Каждый тип связи на ER-диаграмме отображается в виде ромба с именем связи внутри. В качестве имени обычно используются отглагольные существительные.

В большинстве CASE-систем связь всегда должна быть бинарной или рекурсивной. В этом случае в ER-диаграмме связь представляется в виде **линии**, связывающей две сущности или ведущей от сущности к ней же самой. В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи). При этом в месте «стыковки» связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

Стороны ромба рисуют двойными линиями, если это связь сущности слабого типа с сущностью от которой она зависит. Например, связь «Подчинение», связывающая сущность слабого типа ПОДЧИНЕННЫЙ с сущностью СОТРУДНИК, от которой она зависит.

Участники связи соединены со связью линиями. Двойная линия обозначает полное участие сущности в связи с данной стороны. Например, связь «Подчинение», со стороны сущности ПОДЧИНЕННЫЙ.

Связь может быть модифицирована указанием роли. Например, для рекурсивной связи «Состав», указаны роли: «Деталь *состоит из …»* и «Деталь входит *в сосав* …».

Тип связи указывается индексами «1» или «М» над соответствующей линией. Например, связь «Руководство» имеет тип «один ко многим»: один сотрудник может руководить многими проектами; связь «Участие» имеет тип «многие ко многим»: один сотрудник может участвовать во многих проектах, и в проекте могут участвовать многие сотрудники.

Как и в реляционных схемах баз данных, в ER-диаграммах вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу реляционных нормальных форм.

Приведем краткие и неформальные определения трех первых нормальных форм.

В первой нормальной форме ER-диаграммы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, «замаскированных» под атрибуты.

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

На рис. 34 представлена ER-диаграмма рис. 31 в третьей нормальной форме.

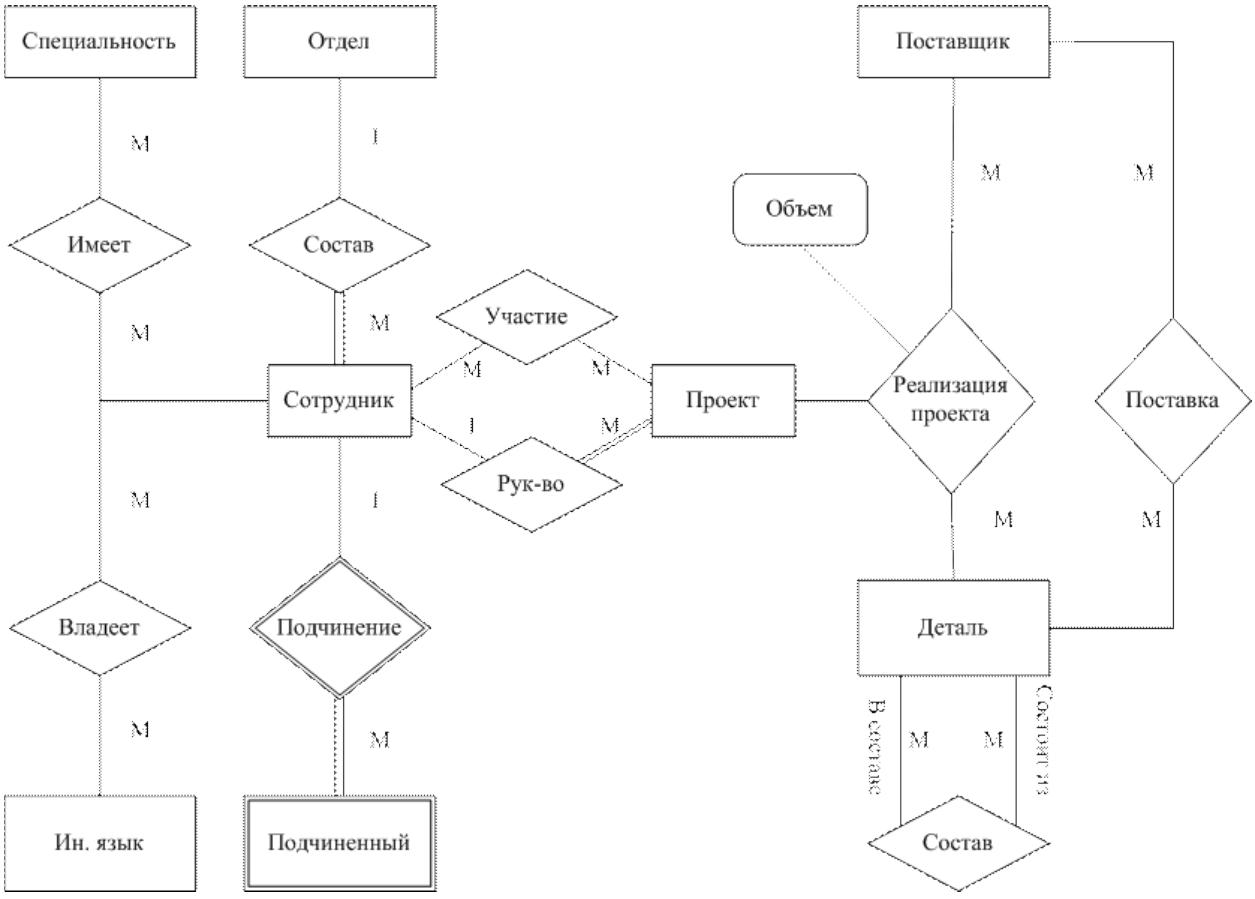


Рис. 34. Пример ER-диаграммы в третьей нормальной форме

Даталогические модели.

Задачей следующей стадии проектирования системы базы данных является выбор подходящей СУБД и отображение в ее среду (структур данных) спецификаций инфологической модели предметной области. Другими словами, модель предметной области разрабатываемой системы должна быть представлена в терминах модели данных концептуального уровня выбранной конкретной СУБД. Эту стадию называют логическим (или даталогическим) проектированием базы данных, а ее результатом является концептуальная схема базы данных, включающая определение всех информационных элементов (единиц) и связей, в том числе задание типов, характеристик и имен. Хотя даталогическое проектирование оперирует не физическими записями, а логическими понятиями, связанными со структурой базы данных, тем не менее, особенности представления данных, правила и языки агрегирования и манипулирования данными имеют определяющее влияние. Не все виды связей, например, «многие ко многим», могут быть непосредственно отображены в логической модели.

Кроме того, может быть много вариантов отображения инфологической модели предметной области в даталогическую модель базы.

Здесь следует учитывать влияние двух следующих значимых факторов, связанных с практикой разработки базы данных.

Во-первых, связи предметной области могут отображаться двумя путями, как декларативным - в логической схеме, так и процедурным - отработкой связей через программные модули, обрабатывающие (связывающие) соответствующие хранимые данные.

Во-вторых, существенным фактором может оказаться характер обработки информации. Например, частые обращения к совместно обрабатываемым данным очевидно предполагают их совместное хранение, а данные (особенно большой размерности), к которым обращаются редко, целесообразно хранить отдельно от часто используемых.

Рассмотрим по шагам общий подход к построению реляционной базы данных на основе инфологической модели, представленной ER-диаграммой.

Получение реляционной схемы из ER-диаграммы.

1. Каждая простая сущность превращается в таблицу (отношение). Имя сущности становится именем таблицы.

2. Каждый атрибут становится возможным столбцом с тем же именем. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут. Если атрибут является множественным, то для него строится отдельное отношение.

3. Компоненты уникального идентификатора сущности превращаются в первичный ключ. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав

уникального идентификатора входят связи, то к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

4. Связи «многие к одному» и «один к одному» становятся внешними ключами. Т.е. создается копия уникального идентификатора с конца связи «один», и соответствующие столбцы составляют внешний ключ.

5. Индексы создаются для первичного ключа (уникальный индекс), а также внешних ключей и тех атрибутов, которые будут часто использоваться в запросах.

6. Если в концептуальной схеме присутствуют подтипы, то возможны два варианта.

Все подтипы хранятся в одной таблице, которая создается для самого внешнего супертипа, а для подтипов создаются представления. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА, и он становится частью первичного ключа.

Во втором случае для каждого подтипа создается отдельная таблица и для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

7. Если остающиеся внешние ключи все принадлежат одному домену, т.е. имеют общий формат, то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей.

Физические модели.

Стадия физического проектирования базы данных в общем случае включает:

- выбор способа организации базы данных, основные из которых были рассмотрены ранее в главе 4;
- разработку спецификации внутренней схемы средствами модели данных ее внутреннего уровня;
- описание отображения концептуальной схемы во внутреннюю.

Важно заметить, что в отличие от ранних СУБД, многие современные системы не предоставляют разработчику какого-либо выбора на этой стадии. Реально к вопросам проектирования физической модели можно отнести выбор схемы размещения данных (разделение по файлам или тип RAID-

массива) и определение числа и типа индексов (например, кластеризованный или некластеризованный в случае MS SQL Server).

Способ хранения базы данных определяется механизмами СУБД автоматически «по умолчанию» на основе спецификаций концептуальной схемы базы данных, и внутренняя схема в явном виде в таких системах не используется.

Следует также отметить, что внешние схемы базы данных обычно конструируются на стадии разработки приложений.

Вопрос 9. Проектирование реляционной БД с использованием нормализации.

Первая нормальная форма.

Сначала будет рассмотрен классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны.

Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости. Для дальнейшего изложения нам потребуются несколько определений.

Определение 1. Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y.

Определение 2. Полная функциональная зависимость

Функциональная зависимость R.X (r) R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение 3. Транзитивная функциональная зависимость

Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z -> R.X. (При отсутствии последнего требования мы имели бы «неинтересные» транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Определение 4. Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, первичного).

Определение 5. Взаимно независимые атрибуты

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Вторая нормальная форма.

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ
(СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ: СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР -> СОТР_ЗАРП

СОТР_НОМЕР -> ОТД_НОМЕР
ОТД_НОМЕР -> СОТР_ЗАРП
СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Как видно, хотя первичным ключом является составной атрибут СОТР_НОМЕР, ПРО_НОМЕР, атрибуты СОТР_ЗАРП и ОТД_НОМЕР функционально зависят от части первичного ключа, атрибута СОТР_НОМЕР.

В результате мы не сможем вставить в отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ кортеж, описывающий сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенное значение).

При удалении кортежа мы не только разрушаем связь данного сотрудника с данным проектом, но утрачиваем информацию о том, что он работает в некотором отделе.

При переводе сотрудника в другой отдел мы будем вынуждены модифицировать все кортежи, описывающие этого сотрудника, или получим несогласованный результат. Такие неприятные явления называются аномалиями схемы отношения. Они устраняются путем нормализации.

Определение 6. Вторая нормальная форма (в этом определении предполагается, что единственным ключом отношения является первичный ключ)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа.

Можно произвести следующую декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ в два отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР)

Первичный ключ: СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР -> СОТР_ЗАРП
СОТР_НОМЕР -> ОТД_НОМЕР
ОТД_НОМЕР -> СОТР_ЗАРП
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ: СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости: СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Каждое из этих двух отношений находится в 2NF, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Определение 6. Вторая нормальная форма (допускается наличие нескольких ключей)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Здесь и далее мы не будем приводить примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

Третья нормальная форма.

Рассмотрим еще раз отношение СОТРУДНИКИ-ОТДЕЛЫ, находящееся в 2NF. Заметим, что функциональная зависимость СОТР_НОМЕР -> СОТР_ЗАРП является транзитивной; она является следствием функциональных зависимостей СОТР_НОМЕР -> ОТД_НОМЕР и ОТД_НОМЕР -> СОТР_ЗАРП. Другими словами, заработная плата сотрудника на самом деле является характеристикой не сотрудника, а отдела, в котором он работает (это не очень естественное предположение, но достаточное для примера).

В результате мы не сможем занести в базу данных информацию, характеризующую заработную плату отдела, до тех пор, пока в этом отделе не появится хотя бы один сотрудник (первичный ключ не может содержать неопределенное значение). При удалении кортежа, описывающего последнего сотрудника данного отдела, мы лишимся информации о заработной плате отдела. Чтобы согласованным образом изменить заработную плату отдела, мы будем вынуждены предварительно найти все кортежи, описывающие сотрудников этого отдела. Т.е. в отношении СОТРУДИКИ-ОТДЕЛЫ по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации.

Определение 7. Третья нормальная форма. (Снова определение дается в предположении существования единственного ключа.)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Можно произвести декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ в два отношения СОТРУДНИКИ и ОТДЕЛЫ:

СОТРУДНИКИ (СОТР_НОМЕР, ОТД_НОМЕР)

Первичный ключ: СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР -> ОТД_НОМЕ
ОТДЕЛЫ (ОТД_НОМЕР, СОТР_ЗАРП)

Первичный ключ: ОТД_НОМЕР
Функциональные зависимости: ТД_НОМЕР -> СОТР_ЗАРП

Каждое из этих двух отношений находится в 3NF и свободно от отмеченных аномалий. Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Определение 7 Третья нормальная форма (допускается наличие нескольких ключей)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 1NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

Нормальная форма Бойса-Кодда.

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, СОТР_ИМЯ, ПРО_НОМЕР, СОТР_ЗАДАН)

Возможные ключи:
СОТР_НОМЕР, ПРО_НОМЕР
СОТР_ИМЯ, ПРО_НОМЕР

Функциональные зависимости:
СОТР_НОМЕР -> СОТР_ИМЯ
СОТР_НОМЕР -> ПРО_НОМЕР
СОТР_ИМЯ -> СОТР_НОМЕР
СОТР_ИМЯ -> ПРО_НОМЕР
СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН
СОТР_ИМЯ, ПРО_НОМЕР -> СОТР_ЗАДАН

В этом примере мы предполагаем, что личность сотрудника полностью определяется как его номером, так и именем (это снова не очень жизненное предположение, но достаточное для примера).

В соответствии с определением 7~ отношение СОТРУДНИКИ-ПРОЕКТЫ находится в 3NF. Однако тот факт, что имеются функциональные зависимости атрибутов отношения от атрибута, являющегося частью первичного ключа, приводит к аномалиям. Например, для того, чтобы изменить имя сотрудника с данным номером согласованным образом, нам потребуется модифицировать все кортежи, включающие его номер.

Определение 8. Детерминант
Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Определение 9. Нормальная форма Бойса-Кодда
Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.

Очевидно, что это требование не выполнено для отношения СОТРУДНИКИ-ПРОЕКТЫ. Можно произвести его декомпозицию к отношениям СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ)

Возможные ключи:
СОТР_НОМЕР
СОТР_ИМЯ

Функциональные зависимости:
СОТР_НОМЕР -> СОТР_ИМЯ
СОТР_ИМЯ -> СОТР_НОМЕР
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Возможный ключ: СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Возможна альтернативная декомпозиция, если выбрать за основу СОТР_ИМЯ. В обоих случаях получаемые отношения СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ находятся в BCNF, и им не свойственны отмеченные аномалии.

Четвертая нормальная форма.

Рассмотрим пример следующей схемы отношения:

ПРОЕКТЫ (ПРО_НОМЕР,ПРО_СОТР, ПРО_ЗАДАН)

Отношение ПРОЕКТЫ содержит номера проектов, для каждого проекта список сотрудников, которые могут выполнять проект, и список заданий, предусматриваемых проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут включать одинаковые задания.

Каждый кортеж отношения связывает некоторый проект с сотрудником, участвующим в этом проекте, и заданием, который сотрудник выполняет в рамках данного проекта (мы предполагаем, что любой сотрудник, участвующий в проекте, выполняет все задания, предусмотренные этим проектом). По причине сформулированных выше условий единственным возможным ключем отношения является составной атрибут ПРО_НОМЕР, ПРО_СОТР, ПРО_ЗАДАН, и нет никаких других детерминантов. Следовательно, отношение ПРОЕКТЫ находится в BCNF. Но при этом оно обладает недостатками: если, например, некоторый сотрудник присоединяется к данному проекту, необходимо вставить в отношение ПРОЕКТЫ столько кортежей, сколько заданий в нем предусмотрено.

Определение 10. Многозначные зависимости

В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

В отношении ПРОЕКТЫ существуют следующие две многозначные зависимости:

ПРО_НОМЕР -> -> ПРО_СОТР

ПРО_НОМЕР -> -> ПРО_ЗАДАН

Легко показать, что в общем случае в отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, когда существует многозначная зависимость R.A -> -> R.C.

Дальнейшая нормализация отношений, подобных отношению ПРОЕКТЫ, основывается на следующей теореме:

Теорема Фейджина

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A -> -> B | C.

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений.

Определение 11. Четвертая нормальная форма

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A -> -> B все остальные атрибуты R функционально зависят от A.

В нашем примере можно произвести декомпозицию отношения ПРОЕКТЫ в два отношения ПРОЕКТЫ-СОТРУДНИКИ и ПРОЕКТЫ-ЗАДАНИЯ:

ПРОЕКТЫ-СОТРУДНИКИ (ПРО_НОМЕР, ПРО_СОТР)
ПРОЕКТЫ-ЗАДАНИЯ (ПРО_НОМЕР, ПРО_ЗАДАН)

Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий.

Пятая нормальная форма.

Во всех рассмотренных до этого момента нормализациях производилась декомпозиция одного отношения в два. Иногда это сделать не удается, но возможна декомпозиция в большее число отношений, каждое из которых обладает лучшими свойствами.

Рассмотрим, например, отношение
СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (СОТР_НОМЕР, ОТД_НОМЕР, ПРО_НОМЕР)

Предположим, что один и тот же сотрудник может работать в нескольких отделах и работать в каждом отделе над несколькими проектами. Первичным ключем этого отношения является полная совокупность его атрибутов, отсутствуют функциональные и многозначные зависимости.

Поэтому отношение находится в 4NF. Однако в нем могут существовать аномалии, которые можно устранить путем декомпозиции в три отношения.

Определение 12. Зависимость соединения

Отношение R (X, Y, ..., Z) удовлетворяет зависимости соединения * (X, Y, ..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z.

Определение 13. Пятая нормальная форма

Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Введем следующие имена составных атрибутов:
СО = {СОТР_НОМЕР, ОТД_НОМЕР}
СП = {СОТР_НОМЕР, ПРО_НОМЕР}
ОП = {ОТД_НОМЕР, ПРО_НОМЕР}

Предположим, что в отношении СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ существует зависимость соединения:

* (СО, СП, ОП)

На примерах легко показать, что при вставках и удалениях кортежей могут возникнуть проблемы. Их можно устранить путем декомпозиции исходного отношения в три новых отношения:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, ОТД_НОМЕР)
СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР)
ОТДЕЛЫ-ПРОЕКТЫ (ОТД_НОМЕР, ПРО_НОМЕР)

Пятая нормальная форма - это последняя нормальная форма, которую можно получить путем декомпозиции. Ее условия достаточно нетривиальны, и на практике 5NF не используется. Заметим, что зависимость соединения является обобщением как многозначной зависимости, так и функциональной зависимости.

Тема 4. Основы SQL[14]

Вопрос 1. Основные понятия и функции структурированного языка запросов SQL.

Рост количества данных, необходимость их хранения и обработки привели к тому, что возникла потребность в создании стандартного языка баз данных, который мог бы функционировать в многочисленных компьютерных системах различных видов. Действительно, с его помощью пользователи могут манипулировать данными независимо от того, работают ли они на персональном компьютере, сетевой рабочей станции или универсальной ЭВМ.

Одним из языков, появившихся в результате разработки реляционной модели данных, является язык SQL (Structured Query Language), который в настоящее время получил очень широкое распространение и фактически превратился в стандартный язык реляционных баз данных. Стандарт на язык

SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного. Нынешний стандарт SQL известен под названием SQL/92.

С использованием любых стандартов связаны не только многочисленные и вполне очевидные преимущества, но и определенные недостатки. Прежде всего, стандарты направляют в определенное русло развитие соответствующей индустрии; в случае языка SQL наличие твердых основополагающих принципов приводит, в конечном счете, к совместимости его различных реализаций и способствует как повышению переносимости программного обеспечения и баз данных в целом, так и универсальности работы администраторов баз данных. С другой стороны, стандарты ограничивают гибкость и функциональные возможности конкретной реализации. Под реализацией языка SQL понимается программный продукт SQL соответствующего производителя. Для расширения функциональных возможностей многие разработчики, придерживающиеся принятых стандартов, добавляют к стандартному языку SQL различные расширения. Следует отметить, что стандарты требуют от любой законченной реализации языка SQL наличия определенных характеристик и в общих чертах отражают основные тенденции, которые не только приводят к совместимости между всеми конкурирующими реализациями, но и способствуют повышению значимости программистов SQL и пользователей реляционных баз данных на современном рынке программного обеспечения.

Все конкретные реализации языка несколько отличаются друг от друга. В интересах самих же производителей гарантировать, чтобы их реализация соответствовала современным стандартам ANSI в части переносимости и удобства работы пользователей. Тем не менее каждая реализация SQL содержит усовершенствования, отвечающие требованиям того или иного сервера баз данных. Эти усовершенствования или расширения языка SQL представляют собой дополнительные команды и опции, являющиеся добавлениями к стандартному пакету и доступные в данной конкретной реализации.

В настоящее время язык SQL поддерживается многими десятками СУБД различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мейнфреймами.

Все языки манипулирования данными, созданные для многих СУБД до появления реляционных баз данных, были ориентированы на операции с данными, представленными в виде логических записей файлов. Разумеется, это требовало от пользователя детального знания организации хранения данных и серьезных усилий для указания того, какие данные необходимы, где они размещаются и как их получить.

Рассматриваемый язык SQL ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц-

отношений. Важнейшая особенность его структур – ориентация на конечный результат обработки данных, а не на процедуру этой обработки. Язык SQL сам определяет, где находятся данные, индексы и даже какие наиболее эффективные последовательности операций следует использовать для получения результата, а потому указывать эти детали в запросе к базе данных не требуется.

Вопрос 2. Типы команд SQL.

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.

Основные категории команд языка SQL:

DDL – язык определения данных;

DML – язык манипулирования данными;

DQL – язык запросов;

DCL – язык управления данными;

команды администрирования данных;

команды управления транзакциями

Определение структур базы данных (DDL).

Язык определения данных (Data Definition Language, DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными командами языка DDL являются следующие: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.

Манипулирование данными (DML).

Язык манипулирования данными (Data Manipulation Language, DML) используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE.

Выборка данных (DQL).

Язык запросов DQL наиболее известен пользователям реляционной базы данных, несмотря на то, что он включает одну команду SELECT. Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования запросов к реляционной базе данных.

Язык управления данными (DCL - Data Control Language).

Команды управления данными позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они

используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие: GRANT, REVOKE.

Команды администрирования данных.

С помощью команд администрирования данных пользователь осуществляет контроль за выполняемыми действиями и анализирует операции базы данных; они также могут оказаться полезными при анализе производительности системы. Не следует путать администрирование данных с администрированием базы данных, которое представляет собой общее управление базой данных и подразумевает использование команд всех уровней.

Команды управления транзакциями.

Существуют следующие команды, позволяющие управлять транзакциями базы данных: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Преимущества языка SQL.

Язык SQL является основой многих СУБД, т.к. отвечает за физическое структурирование и запись данных на диск, а также за чтение данных с диска, позволяет принимать SQL-запросы от других компонентов СУБД и пользовательских приложений. Таким образом, SQL – мощный инструмент, который обеспечивает пользователям, программам и вычислительным системам доступ к информации, содержащейся в реляционных базах данных.

Основные достоинства языка SQL заключаются в следующем:

- стандартность – как уже было сказано, использование языка SQL в программах стандартизировано международными организациями;
- независимость от конкретных СУБД – все распространенные СУБД используют SQL, т.к. реляционную базу данных можно перенести с одной СУБД на другую с минимальными доработками;
- возможность переноса с одной вычислительной системы на другую – СУБД может быть ориентирована на различные вычислительные системы, однако приложения, созданные с помощью SQL, допускают использование как для локальных БД, так и для крупных многопользовательских систем;
- реляционная основа языка – SQL является языком реляционных БД, поэтому он стал популярным тогда, когда получила широкое распространение реляционная модель представления данных. Табличная структура реляционной БД хорошо понятна, а потому язык SQL прост для изучения;
- возможность создания интерактивных запросов – SQL обеспечивает пользователям немедленный доступ к данным, при этом в интерактивном режиме можно получить результат запроса за очень короткое время без написания сложной программы;
- возможность программного доступа к БД – язык SQL легко использовать в приложениях, которым необходимо обращаться к базам

данных. Одни и те же операторы SQL употребляются как для интерактивного, так и программного доступа, поэтому части программ, содержащие обращение к БД, можно вначале проверить в интерактивном режиме, а затем встраивать в программу;

- обеспечение различного представления данных – с помощью SQL можно представить такую структуру данных, что тот или иной пользователь будет видеть различные их представления. Кроме того, данные из разных частей БД могут быть скомбинированы и представлены в виде одной простой таблицы, а значит, представления пригодны для усиления защиты БД и ее настройки под конкретные требования отдельных пользователей;
- возможность динамического изменения и расширения структуры БД – язык SQL позволяет манипулировать структурой БД, тем самым обеспечивая гибкость с точки зрения приспособленности БД к изменяющимся требованиям предметной области;
- поддержка архитектуры клиент-сервер – SQL – одно из лучших средств для реализации приложений на платформе клиент-сервер. SQL служит связующим звеном между взаимодействующей с пользователем клиентской системой и серверной системой, управляющей БД, позволяя каждой из них сосредоточиться на выполнении своих функций.

Любой язык работы с базами данных должен предоставлять пользователю следующие возможности:

- создавать базы данных и таблицы с полным описанием их структуры;
- выполнять основные операции манипулирования данными, в частности, вставку, модификацию и удаление данных из таблиц;
- выполнять простые и сложные запросы, осуществляющие преобразование данных.

Кроме того, язык работы с базами данных должен решать все указанные выше задачи при минимальных усилиях со стороны пользователя, а структура и синтаксис его команд – достаточно просты и доступны для изучения. И наконец, он должен быть универсальным, т.е. отвечать некоторому признанному стандарту, что позволит использовать один и тот же синтаксис и структуру команд при переходе от одной СУБД к другой. Язык SQL удовлетворяет практически всем этим требованиям.

Язык SQL является примером языка с трансформирующейся ориентацией, или же языка, предназначенного для работы с таблицами с целью преобразования входных данных к требуемому выходному виду. Он включает только команды определения и манипулирования данными и не содержит каких-либо команд управления ходом вычислений. Подобные задачи должны решаться либо с помощью языков программирования или управления заданиями, либо интерактивно, в результате действий, выполняемых самим пользователем. По причине подобной незавершенности в плане организации вычислительного процесса язык SQL может

использоваться двумя способами. Первый предусматривает интерактивную работу, заключающуюся во вводе пользователем с терминала отдельных SQL-операторов. Второй состоит во внедрении SQL-операторов в программы на процедурных языках. Язык SQL относительно прост в изучении. Поскольку это не процедурный язык, в нем необходимо указывать, какая информация должна быть получена, а не как ее можно получить. Иначе говоря, SQL не требует указания методов доступа к данным. Как и большинство современных языков, он поддерживает свободный формат записи операторов. Это означает, что при вводе отдельные элементы операторов не связаны с фиксированными позициями экрана. Язык SQL может использоваться широким кругом специалистов, включая администраторов баз данных, прикладных программистов и множество других конечных пользователей.

Язык SQL – первый и пока единственный стандартный язык для работы с базами данных, который получил достаточно широкое распространение. Практически все крупнейшие разработчики СУБД в настоящее время создают свои продукты с использованием языка SQL либо с SQL-интерфейсом. В него сделаны огромные инвестиции как со стороны разработчиков, так и со стороны пользователей. Он стал частью архитектуры приложений, является стратегическим выбором многих крупных и влиятельных организаций.

Язык SQL используется в других стандартах и даже оказывает влияние на разработку иных стандартов как инструмент определения (например, стандарт Remote Data Access, RDA). Создание языка способствовало не только выработке необходимых теоретических основ, но и подготовке успешно реализованных технических решений. Это особенно справедливо в отношении оптимизации запросов, методов распределения данных и реализации средств защиты. Начали появляться специализированные реализации языка, предназначенные для новых рынков: системы управления обработкой транзакций (OnLine Transaction Processing, OLTP) и системы оперативной аналитической обработки или системы поддержки принятия решений (OnLine Analytical Processing, OLAP). Уже известны планы дальнейших расширений стандарта, включающих поддержку распределенной обработки, объектно-ориентированного программирования, расширений пользователей и мультимедиа.

Запись SQL-операторов.

Для успешного изучения языка SQL необходимо привести краткое описание структуры SQL-операторов и нотации, которые используются для определения формата различных конструкций языка. Оператор SQL состоит из зарезервированных слов, а также из слов, определяемых пользователем. Зарезервированные слова являются постоянной частью языка SQL и имеют фиксированное значение. Их следует записывать в точности так, как это установлено, нельзя разбивать на части для переноса с одной строки на другую. Слова, определяемые пользователем, задаются им самим (в соответствии с синтаксическими правилами) и представляют

собой идентификаторы или имена различных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных и являются именами таблиц, представлений, столбцов и других объектов базы данных. Символы, которые могут использоваться в создаваемых пользователем идентификаторах языка SQL, должны быть определены как набор символов. Стандарт SQL задает набор символов, который используется по умолчанию, – он включает строчные и прописные буквы латинского алфавита (A-Z, a-z), цифры (0-9) и символ подчеркивания (_). На формат идентификатора накладываются следующие ограничения:

- идентификатор может иметь длину до 128 символов;
- идентификатор должен начинаться с буквы;
- идентификатор не может содержать пробелы.

<идентификатор>::=<буква>
{<буква>|<цифра>}[...n]

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка SQL свободный формат, отдельные SQL-операторы и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Язык, в терминах которого дается описание языка SQL, называется метаязыком. Синтаксические определения обычно задают с помощью специальной металингвистической символики, называемой Бэкуса-Науэра формулами (БНФ). Прописные буквы используются для записи зарезервированных слов и должны указываться в операторах точно так, как это будет показано. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации БНФ символы и их обозначения показаны в таблице 5.

Таблица 5.

Символ	Обозначение
::=	Равно по определению
|	Необходимость выбора одного из нескольких приведенных значений
<…>	Описанная с помощью метаязыка структура языка
{…}	Обязательный выбор некоторой конструкции из списка
[…]	Необязательный выбор некоторой конструкции из списка
[,…n]	Необязательная возможность повторения конструкции от нуля до нескольких раз

Описание учебной базы данных.

В дальнейшем изложении в качестве примера будет использоваться небольшая база данных, отражающая процесс поставки или продажи некоторого товара постоянным клиентам.

Исходя из анализа предметной области, можно выделить два типа сущностей – ТОВАР и КЛИЕНТ, которые связаны между собой отношением «многие–ко–многим», т.к. каждый покупатель может купить много наименований товара, а каждый товар может быть куплен многими покупателями. Однако реляционная модель данных требует заменить отношение «многие–ко-многим» на несколько отношений «один–ко-многим». Добавим еще один тип сущностей, отображающий процесс продажи товаров, – СДЕЛКА.

Установим связи между объектами. Один покупатель может неоднократно покупать товары, поэтому между объектами КЛИЕНТ и СДЕЛКА имеется связь «один–ко–многим». Каждое наименование товара может неоднократно участвовать в сделках, в результате между объектами ТОВАР и СДЕЛКА имеется связь «один-ко-многим».

Определим атрибуты и свяжем их с сущностями и связями. К объекту ТОВАР относятся такие характеристики, как название, тип, цена, сорт. К объекту КЛИЕНТ – имя, отчество, фамилия, фирма, город, телефон. Тип сущности СДЕЛКА может быть охарактеризован такими признаками, как дата и количество проданного товара.

Важным этапом в создании базы данных является определение атрибутов, которые однозначно определяют каждый экземпляр сущности, т.е. выявление первичных ключей.

Для таблицы ТОВАР название не может служить первичным ключом, т.к. товары разных типов могут иметь одинаковые названия, поэтому введем первичный ключ КодТовара, под которым можно понимать, например, артикул товара. Точно так же ни Имя, ни Фирма, ни Город не могут служить первичным ключом в таблице КЛИЕНТ. Введем первичный ключ КодКлиента, под которым можно понимать номер паспорта, идентификационный номер налогоплательщика или любой другой атрибут, однозначно определяющий каждого клиента.
Для таблицы СДЕЛКА первичным ключом является поле КодСделки, т.к. оно однозначно определяет дату, покупателя и другие элементы данных. В качестве первичного ключа можно было бы выбрать не одно поле, а некоторую совокупность полей, но для иллюстрации конструкций языка ограничимся простыми первичными ключами.

Установим связи между таблицами. Один покупатель может неоднократно покупать товары. Поэтому между таблицами КЛИЕНТ и СДЕЛКА имеется связь «один–ко–многим» по полю КодКлиента.

Каждый покупатель может приобрести несколько различных товаров. Поэтому между таблицами ТОВАР и СДЕЛКА имеется связь «один–ко–многим» по полю КодТовара.

Теперь нужно создать связи между таблицами базы данных. Для этого поместим копии первичных ключей из родительской таблицы (таблицы со стороны «один») в дочернюю таблицу (таблицу со стороны «много»). Для

организации связи между таблицами ТОВАР и СДЕЛКА поместим копию поля КодТовара из таблицы ТОВАР в таблицу СДЕЛКА. Для организации связи между таблицами КЛИЕНТ и СДЕЛКА поместим копию поля КодКлиента из таблицы КЛИЕНТ в таблицу СДЕЛКА.
Для таблицы СДЕЛКА поля КодКлиента и КодТовара являются внешними (чужими) ключами. В результате получим следующую структуру базы данных.

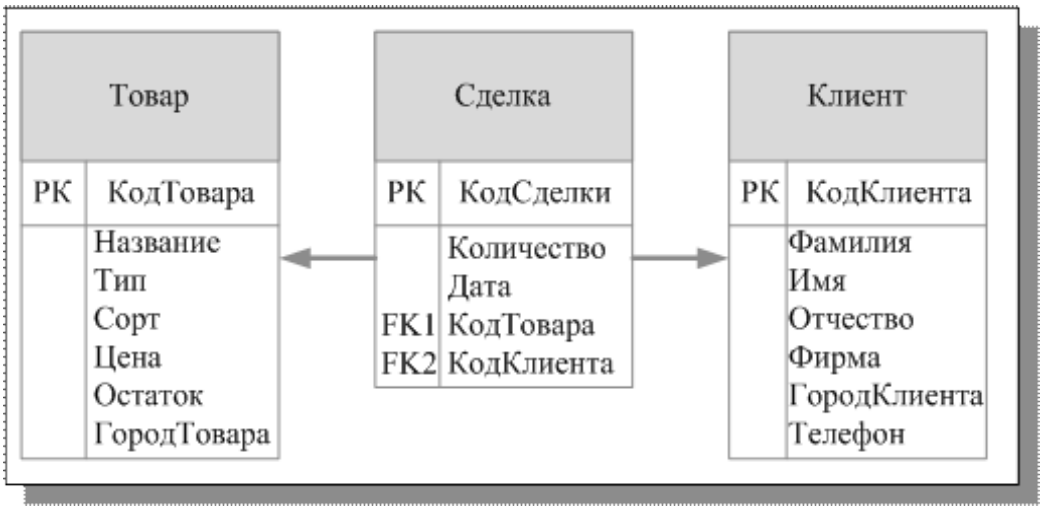


Рис. 35. Пример структуры базы данных

Вопрос 3. Типы данных SQL.

Данные – это совокупная информация, хранимая в базе данных в виде одного из нескольких различных типов. С помощью типов данных устанавливаются основные правила для данных, содержащихся в конкретном столбце таблицы, в том числе размер выделяемой для них памяти.

В языке SQL имеется шесть скалярных типов данных, определенных стандартом. Их краткое описание представлено в таблице 6.

Таблица 6.

Тип данных	Объявления
Символьный	CHAR | VARCHAR
Битовый	BIT | BIT VARYING
Точные числа	NUMERIC | DECIMAL | INTEGER | SMALLINT
Округленные числа	FLOAT | REAL | DOUBLE PRECISION
Дата/время	DATE | TIME | TIMESTAMP
Интервал	INTERVAL

Символьные данные.

Символьные данные состоят из последовательности символов, входящих в определенный создателями СУБД набор символов. Поскольку

наборы символов являются специфическими для различных диалектов языка SQL, перечень символов, которые могут входить в состав значений данных символьного типа, также зависит от конкретной реализации. Чаще всего используются наборы символов ASCII и EBCDIC. Для определения данных символьного типа используется следующий формат:

```
<символьный_тип>::=
{ CHARACTER [ VARYING][длина] | [CHAR |
VARCHAR][длина]}
```

При определении столбца с символьным типом данных параметр длина применяется для указания максимального количества символов, которые могут быть помещены в данный столбец (по умолчанию принимается значение 1). Символьная строка может быть определена как имеющая фиксированную или переменную (VARYING) длину. Если строка определена с фиксированной длиной значений, то при вводе в нее меньшего количества символов значение дополняется до указанной длины пробелами, добавляемыми справа. Если строка определена с переменной длиной значений, то при вводе в нее меньшего количества символов в базе данных будут сохранены только введенные символы, что позволит достичь определенной экономии внешней памяти.

Битовые данные.

Битовый тип данных используется для определения битовых строк, т.е. последовательности двоичных цифр (битов), каждая из которых может иметь значение либо 0, либо 1. Данные битового типа определяются при помощи следующего формата:

```
<битовый_тип>::=
BIT [VARYING][длина]
```

Точные числа.

Тип точных числовых данных применяется для определения чисел, которые имеют точное представление, т.е. числа состоят из цифр, необязательной десятичной точки и необязательного символа знака. Данные точного числового типа определяются точностью и длиной дробной части. Точность задает общее количество значащих десятичных цифр числа, в которое входит длина как целой части, так и дробной, но без учета самой десятичной точки. Масштаб указывает количество дробных десятичных разрядов числа.

```
<фиксированный_тип>::=
{NUMERIC[точность[,масштаб]|{DECIMAL|DEC}
  [точность[, масштаб]
| {INTEGER |INT}| SMALLINT}
```

Типы NUMERIC и DECIMAL предназначены для хранения чисел в десятичном формате. По умолчанию длина дробной части равна нулю, а принимаемая по умолчанию точность зависит от реализации. Тип INTEGER (INT) используется для хранения больших положительных или отрицательных целых чисел. Тип SMALLINT – для хранения небольших положительных или отрицательных целых чисел; в этом случае расход внешней памяти существенно сокращается.

Округленные числа.

Тип округленных чисел применяется для описания данных, которые нельзя точно представить в компьютере, в частности действительных чисел. Округленные числа или числа с плавающей точкой представляются в научной нотации, при которой число записывается с помощью мантиссы, умноженной на определенную степень десяти (порядок), например: 10E3, +5.2E6, -0.2E-4. Для определения данных вещественного типа используется формат:

```
<вещественный_тип>::=
{ FLOAT [точность]| REAL |
  DOUBLE PRECISION}
```

Параметр точность задает количество значащих цифр мантиссы. Точность типов REAL и DOUBLE PRECISION зависит от конкретной реализации.

Дата и время.

Тип данных «дата/время» используется для определения моментов времени с некоторой установленной точностью. Стандарт SQL поддерживает следующий формат:

```
<тип_даты/времени>::=
{DATE | TIME[точность][WITH TIME ZONE]|
  TIMESTAMP[точность][WITH TIME ZONE]}
```

Тип данных DATE используется для хранения календарных дат, включающих поля YEAR (год), MONTH (месяц) и DAY (день). Тип данных TIME – для хранения отметок времени, включающих поля HOUR (часы), MINUTE (минуты) и SECOND (секунды). Тип данных TIMESTAMP – для совместного хранения даты и времени. Параметр точность задает количество дробных десятичных знаков, определяющих точность сохранения значения в поле SECOND. Если этот параметр опускается, по умолчанию его значение для столбцов типа TIME принимается равным нулю (т.е. сохраняются целые секунды), тогда как для полей типа TIMESTAMP он принимается равным 6 (т.е. отметки времени сохраняются с точностью до миллисекунд). Наличие ключевого слова WITH TIME ZONE определяет использование полей TIMEZONE HOUR и TIMEZONE MINUTE, тем самым задаются час и

минуты сдвига зонального времени по отношению к универсальному координатному времени (Гринвичскому времени).

Данные типа INTERVAL используются для представления периодов времени.

Понятие домена.

Домен – это набор допустимых значений для одного или нескольких атрибутов. Если в таблице базы данных или в нескольких таблицах присутствуют столбцы, обладающие одними и теми же характеристиками, можно описать тип такого столбца и его поведение через домен, а затем поставить в соответствие каждому из одинаковых столбцов имя домена. Домен определяет все потенциальные значения, которые могут быть присвоены атрибуту.

Стандарт SQL позволяет определить домен с помощью следующего оператора:

```
<определение_домена>::=
  CREATE DOMAIN имя_домена [AS]
    тип_данных
  [ DEFAULT значение]
  [ CHECK (допустимые_значения)]
```

Каждому создаваемому домену присваивается имя, тип данных, значение по умолчанию и набор допустимых значений. Следует отметить, что приведенный формат оператора является неполным. Теперь при создании таблицы можно указать вместо типа данных имя домена.

Удаление доменов из базы данных выполняется с помощью оператора:

```
DROP DOMAIN имя_домена [ RESTRICT |
  CASCADE]
```

В случае указания ключевого слова CASCADE любые столбцы таблиц, созданные с использованием удаляемого домена, будут автоматически изменены и описаны как содержащие данные того типа, который был указан в определении удаляемого домена.

Альтернативой доменам в среде SQL Server являются пользовательские типы данных.

Типы данных, используемые в SQL-сервере.

Системные типы данных.

Один из основных моментов процесса создания таблицы – определение типов данных для ее полей. Тип данных поля таблицы определяет тип информации, которая будет размещаться в этом поле. Понятие типа данных в SQL Server полностью адекватно понятию типа данных в современных языках программирования. SQL-сервер поддерживает большое число различных типов данных: текстовые, числовые, двоичные:

Таблица 7.

image	smalldatetime	bit	binary
text	real	decimal	char
uniqueidentifier	money	numeric	timestamp
tinyint	datetime	smallmoney	nvarchar
smallint	float	varbinary	nchar
int	ntext	varchar	sysname

Краткий обзор типов данных SQL Server.

Для хранения символьной информации используются символьные типы данных, к которым относятся CHAR (длина), VARCHAR (длина), NCHAR (длина), NVARCHAR (длина). Последние два предназначены для хранения символов Unicode. Максимальное значение длины ограничено 8000 знаками (4000 – для символов Unicode). Хранение символьных данных большого объема (до 2 Гб) осуществляется при помощи текстовых типов данных TEXT и NTEXT. К целочисленным типам данных относятся INT (INTEGER), SMALLINT, TINYINT, BIGINT. Для хранения данных целочисленного типа используется, соответственно, 4 байта (диапазон от -2^{31} до $2^{31}-1$), 2 байта (диапазон от -2^{15} до $2^{15}-1$), 1 байт (диапазон от 0 до 255) или 8 байт (диапазон от -2^{63} до $2^{63}-1$). Объекты и выражения целочисленного типа могут применяться в любых математических операциях. Числа, в составе которых есть десятичная точка, называются нецелочисленными. Нецелочисленные данные разделяются на два типа – десятичные и приблизительные. К десятичным типам данных относятся типы DECIMAL [(точность[,масштаб])] или DEC и NUMERIC [(точность[,масштаб])]. Типы данных DECIMAL и NUMERIC позволяют самостоятельно определить формат точности числа с плавающей запятой. Параметр точность указывает максимальное количество цифр вводимых данных этого типа (до и после десятичной точки в сумме), а параметр масштаб – максимальное количество цифр, расположенных после десятичной точки. В обычном режиме сервер позволяет вводить не более 28 цифр, используемых в типах DECIMAL и NUMERIC (от 2 до 17 байт).

К приблизительным типам данных относятся FLOAT (точность до 15 цифр, 8 байт) и REAL (точность до 7 цифр, 4 байта). Эти типы представляют данные в формате с плавающей запятой, т.е. для представления чисел используется мантисса и порядок, что обеспечивает одинаковую точность вычислений независимо от того, насколько мало или велико значение.

Для хранения информации о дате и времени предназначены такие типы данных, как DATETIME и SMALLDATETIME, использующие для представления даты и времени 8 и 4 байта соответственно.

Типы данных MONEY и SMALLMONEY делают возможным хранение информации денежного типа; они обеспечивают точность значений до 4 знаков после запятой и используют 8 и 4 байта соответственно.

Тип данных BIT позволяет хранить один бит, который принимает значения 0 или 1.

В среде SQL Server реализован ряд специальных типов данных.

Тип данных TIMESTAMP применяется в качестве индикатора изменения версии строки в пределах базы данных.

Тип данных UNIQUEIDENTIFIER используется для хранения глобальных уникальных идентификационных номеров.

Тип данных SYSNAME предназначен для идентификаторов объектов.

Тип данных SQL_VARIANT позволяет хранить значения любого из поддерживаемых SQL Server типов данных за исключением TEXT, NTEXT, IMAGE и TIMESTAMP.

Тип данных TABLE, подобно временным таблицам, обеспечивает хранение набора строк, предназначенных для последующей обработки. Тип данных TABLE может применяться только для определения локальных переменных и возвращаемых пользовательскими функциями значений. Пример использования типа данных TABLE приведен в лекции, посвященной функциям пользователя.

Тип данных CURSOR нужен для работы с такими объектами, как курсоры, и может быть востребован только для переменных и параметров хранимых процедур. Курсоры SQL Server представляют собой механизм обмена данными между сервером и клиентом. Курсор позволяет клиентским приложениям работать не с полным набором данных, а лишь с одной или несколькими строками. Примеры использования данных типа CURSOR мы рассмотрим в лекциях, посвященных курсорам и хранимым процедурам.

Создание пользовательского типа данных.

В системе SQL-сервера имеется поддержка пользовательских типов данных. Они могут использоваться при определении какого-либо специфического или часто употребляемого формата.

Создание пользовательского типа данных осуществляется выполнением системной процедуры:

```
sp_addtype [@typename=]type,[@phystype=]
  system_data_type
    [,[@nulltype=]'null_type']
```

Тип данных system_data_type выбирается из следующей таблицы:

Таблица 8.

image	smalldatetime	decimal	bit
text	real	'decimal[(p[,s])]'	'binary(n)'
uniqueidentifier	datetime	numeric	'char(n)'
smallint	float	'numeric[(p[,s])]'	'nvarchar(n)'

int	‘float(n)’	‘varbinary(n)’	
	ntext	‘varchar(n)’	‘nchar(n)’

Пример. Создание пользовательского типа данных bir:

```
EXEC sp_addtype bir, DATETIME, 'NULL'
или
EXEC sp_addtype bir, DATETIME, ‘NOT NULL’
```

Пример. Использование пользовательского типа данных bir при создании таблицы:

```
CREATE TABLE tab
(id_n        INT IDENTITY(1,1) PRIMARY KEY,
names        VARCHAR(40),
birthday     BIR)
```

Удаление пользовательского типа данных происходит в результате выполнения процедуры sp_droptype type:

```
EXEC sp_droptype 'bir'
```

Получение информации о типах данных.

Получить список всех типов данных, включая пользовательские, можно из системной таблицы systypes:

```
SELECT * FROM  systypes
```

Преобразование типов.

Нередко требуется конвертировать значения одного типа в значения другого. Наиболее часто выполняется конвертирование чисел в символьные данные и наоборот, для этого используется специализированная функция STR. Для выполнения других преобразований SQL Server предлагает универсальные функции CONVERT и CAST, с помощью которых значения одного типа преобразовываются в значения другого типа, если такие изменения вообще возможны. CONVERT и CAST примерно одинаковы и могут быть взаимозаменяемыми.

```
CAST(выражение AS тип_данных)
CONVERT(тип_данных[(длина)],
  выражение [,стиль])
```

С помощью аргумента стиль можно управлять стилем представления значений следующих типов данных: дата/время, денежный или нецелочисленный.

Пример. Преобразование данных символьного типа к данным типа дата/время:

```
DECLARE @d DATETIME
DECLARE  @s CHAR(8)
SET @s=’29.10.01’
SET @d=CAST(@s AS DATETIME)
```

Наряду с типами данных основополагающими понятиями при работе с языком SQL в среде MS SQL Server являются выражения, операторы, переменные, управляющие конструкции.

Выражения.

Выражения представляют собой комбинацию идентификаторов, функций, знаков логических и арифметических операций, констант и других объектов. Выражение может быть использовано в качестве аргумента в командах, хранимых процедурах или запросах.

Выражение состоит из операндов (собственно данных) и операторов (знаков операций, производимых над операндами). В качестве операндов могут выступать константы, переменные, имена столбцов, функции, подзапросы.

Операторы – это знаки операций над одним или несколькими выражениями для создания нового выражения. Среди операторов можно выделить унарные операторы, операторы присваивания, арифметические операторы, строковые операторы, операторы сравнения, логические операторы, битовые операторы.

Переменные.

В среде SQL Server существует несколько способов передачи данных между командами. Один из них – передача данных через локальные переменные. Прежде чем использовать какую-либо переменную, ее следует объявить. Объявление переменной выполняется командой DECLARE, имеющей следующий формат:

```
DECLARE {@имя_переменной тип_данных }
  [,...n]
```

Значения переменной можно присвоить посредством команд SET и SELECT. С помощью команды SELECT переменной можно присвоить не только конкретное значение, но и результат вычисления выражения.

Пример. Использование SET для присваивания значения локальной переменной:

```
DECLARE @a INT
SET @a=10
```

Пример. Использование SELECT для присваивания локальной переменной результата вычислений:

```
DECLARE @k INT
SELECT @k=SUM(количество) FROM Товар
```

Управляющие конструкции SQL.

Язык SQL является непроцедурным, но тем не менее в среде SQL Server предусмотрен ряд различных управляющих конструкций, без которых невозможно написание эффективных алгоритмов.

Группировка двух и более команд в единый блок осуществляется с использованием ключевых слов BEGIN и END:

```
<блок_операторов>::=
BEGIN
{ sql_оператор | блок_операторов }
END
```

Сгруппированные команды воспринимаются интерпретатором SQL как одна команда. Подобная группировка требуется для конструкций поливариантных
ветвлений, условных и циклических конструкций. Блоки BEGIN...END могут быть вложенными.

Некоторые команды SQL не должны выполняться вместе с другими командами (речь идет о командах резервного копирования, изменения структуры таблиц, хранимых процедур и им подобных), поэтому их совместное включение в конструкцию BEGIN...END не допускается.

Нередко определенная часть программы должна выполняться только при реализации некоторого логического условия. Синтаксис условного оператора показан ниже:

```
<условный_оператор>::=
IF лог_выражение
  { sql_оператор | блок_операторов }
[ ELSE
  {sql_оператор | блок_операторов } ]
```

Циклы организуются с помощью следующей конструкции:

```
<оператор_цикла>::=
WHILE лог_выражение
   { sql_оператор | блок_операторов }
  [ BREAK ]
   { sql_оператор | блок_операторов }
  [ CONTINUE ]
```

Цикл можно принудительно остановить, если в его теле выполнить команду BREAK. Если же нужно начать цикл заново, не дожидаясь

выполнения всех команд в теле, необходимо выполнить команду CONTINUE.

Для замены множества одиночных или вложенных условных операторов используется следующая конструкция:

```
<оператор_поливариантных_ветвлений>::=
CASE входное_значение
WHEN {значение_для_сравнения |
  лог_выражение } THEN
  вых_выражение [,...n]
[ ELSE иначе_вых_значение ]
END
```

Если входное значение и значение для сравнения совпадают, то конструкция возвращает выходное значение. Если же значение входного параметра не найдено ни в одной из строк WHEN...THEN, то тогда будет возвращено значение, указанное после ключевого слова ELSE.

Основные объекты структуры базы данных SQL-сервера.

Рассмотрим логическую структуру базы данных. Логическая структура определяет структуру таблиц, взаимоотношения между ними, список пользователей, хранимые процедуры, правила, умолчания и другие объекты базы данных. Логически данные в SQL Server организованы в виде объектов. К основным объектам базы данных SQL Server относятся объекты, представленные ниже (табл. 9).

Таблица 9.

Tables	Таблицы базы данных, в которых хранятся собственно данные
Views	Просмотры (виртуальные таблицы) для отображения данных из таблиц
Stored Procedures	Хранимые процедуры
Triggers	Триггеры – специальные хранимые процедуры, вызываемые при изменении данных в таблице
User Defined function	Создаваемые пользователем функции
Indexes	Индексы – дополнительные структуры, призванные повысить производительность работы с данными
User Defined Data Types	Определяемые пользователем типы данных
Keys	Ключи – один из видов ограничений целостности данных
Constraints	Ограничение целостности – объекты для обеспечения логической целостности данных
Users	Пользователи, обладающие доступом к базе данных
Roles	Роли, позволяющие объединять пользователей в группы
Rules	Правила базы данных, позволяющие контролировать логическую целостность данных
Defaults	Умолчания или стандартные установки базы данных

Краткий обзор основных объектов баз данных.

Таблицы.

Все данные в SQL содержатся в объектах, называемых таблицами. Таблицы представляют собой совокупность каких-либо сведений об объектах, явлениях, процессах реального мира. Никакие другие объекты не хранят данные, но они могут обращаться к данным в таблице. Таблицы в SQL имеют такую же структуру, что и таблицы всех других СУБД и содержат:

- строки; каждая строка (или запись) представляет собой совокупность атрибутов (свойств) конкретного экземпляра объекта;
- столбцы; каждый столбец (поле) представляет собой атрибут или совокупность атрибутов. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, тип данных и размер.

Представления.

Представлениями (просмотрами) называют виртуальные таблицы, содержимое которых определяется запросом. Подобно реальным таблицам, представления содержат именованные столбцы и строки с данными. Для конечных пользователей представление выглядит как таблица, но в действительности оно не содержит данных, а лишь представляет данные, расположенные в одной или нескольких таблицах. Информация, которую видит пользователь через представление, не сохраняется в базе данных как самостоятельный объект.

Хранимые процедуры.

Хранимые процедуры представляют собой группу команд SQL, объединенных в один модуль. Такая группа команд компилируется и выполняется как единое целое.

Триггеры.

Триггерами называется специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении или удалении данных из таблицы.

Функции.

Функции в языках программирования – это конструкции, содержащие часто исполняемый код. Функция выполняет какие-либо действия над данными и возвращает некоторое значение.

Индексы.

Индекс – структура, связанная с таблицей или представлением и предназначенная для ускорения поиска информации в них. Индекс определяется для одного или нескольких столбцов, называемых индексированными столбцами. Он содержит отсортированные значения индексированного столбца или столбцов со ссылками на соответствующую строку исходной таблицы или представления. Повышение производительности достигается за счет сортировки данных. Использование индексов может существенно повысить производительность поиска, однако для хранения индексов необходимо дополнительное пространство в базе данных.

Пользовательские типы данных.

Пользовательские типы данных – это типы данных, которые создает пользователь на основе системных типов данных, когда в нескольких таблицах необходимо хранить однотипные значения; причем нужно гарантировать, что столбцы в таблице будут иметь одинаковый размер, тип данных и чувствительность к значениям NULL.

Ограничения целостности.

Ограничения целостности – механизм, обеспечивающий автоматический контроль соответствия данных установленным условиям (или ограничениям). Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся: ограничение на значение NULL, проверочные ограничения, ограничение уникальности (уникальный ключ), ограничение первичного ключа и ограничение внешнего ключа. Последние три ограничения тесно связаны с понятием ключей.

Правила.

Правила используются для ограничения значений, хранимых в столбце таблицы или в пользовательском типе данных. Они существуют как самостоятельные объекты базы данных, которые связываются со столбцами таблиц и пользовательскими типами данных. Контроль значений данных может быть реализован и с помощью ограничений целостности.

Умолчания.

Умолчания – самостоятельный объект базы данных, представляющий значение, которое будет присвоено элементу таблицы при вставке строки, если в команде вставки явно не указано значение для этого столбца.

Вопрос 4. Построение запросов на выборку данных.

Предложение SELECT.

Оператор SELECT – один из наиболее важных и самых распространенных операторов SQL. Он позволяет производить выборки данных из таблиц и преобразовывать к нужному виду полученные результаты. Будучи очень мощным, он способен выполнять действия, эквивалентные операторам реляционной алгебры, причем в пределах единственной выполняемой команды. При его помощи можно реализовать сложные и громоздкие условия отбора данных из различных таблиц.

Оператор SELECT – средство, которое полностью абстрагировано от вопросов представления данных, что помогает сконцентрировать внимание на проблемах доступа к данным. Примеры его использования наглядно демонстрируют один из основополагающих принципов больших (промышленных) СУБД: средства хранения данных и доступа к ним отделены от средств представления данных. Операции над данными производятся в масштабе наборов данных, а не отдельных записей.

Оператор SELECT имеет следующий формат:

```
SELECT [ALL | DISTINCT ] {*|[имя_столбца
  [AS         новое_имя]]} [,...n]
  FROM  имя_таблицы [[AS] псевдоним] [,...n]
  [WHERE  <условие_поиска>]
  [GROUP BY  имя_столбца [,...n]]
  [HAVING <критерии выбора групп>]
  [ORDER BY имя_столбца [,...n]]
```

Оператор SELECT определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в результате запроса. Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки. Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен.

Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. Имя_таблицы.Имя_поля.

Предложение FROM.

Предложение FROM задает имена таблиц и просмотров, которые содержат поля, перечисленные в операторе SELECT. Необязательный параметр псевдонима – это сокращение, устанавливаемое для имени таблицы.

Обработка элементов оператора SELECT выполняется в следующей последовательности:

FROM – определяются имена используемых таблиц;

WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;

GROUP BY – образуются группы строк , имеющих одно и то же значение в указанном столбце;

HAVING – фильтруются группы строк объекта в соответствии с указанным условием;

SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;

ORDER BY – определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения SELECT и FROM являются обязательными, все остальные могут быть опущены. SELECT – закрытая операция: результат запроса к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Пример. Составить список сведений о всех клиентах.

```
SELECT * FROM Клиент
```

Параметр WHERE определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат ALL задает включение в выходной набор всех дубликатов, отобранных по критерию WHERE. Нет необходимости указывать ALL явно, поскольку это значение действует по умолчанию.

Пример. Составить список всех фирм.

```
SELECT ALL Клиент.Фирма FROM Клиент
```

Или (что эквивалентно)

```
SELECT Клиент.Фирма FROM Клиент
```

Результат выполнения запроса может содержать дублирующиеся значения, поскольку в отличие от операций реляционной алгебры оператор SELECT не исключает повторяющихся значений при выполнении выборки данных.

Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие дублирующие записи в выбранных полях. Значения для каждого из приведенных в инструкции SELECT полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.

Причиной ограничения в применении DISTINCT является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректированный **пример** Составить список всех фирм.

пример 4.2 выглядит следующим образом:

```
SELECT DISTINCT Клиент.Фирма
  FROM Клиент
```

Предложение WHERE.

С помощью WHERE-параметра пользователь определяет, какие блоки данных из приведенных в списке FROM таблиц появятся в результате запроса. За ключевым словом WHERE следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска (или предикатов):

Сравнение: сравниваются результаты вычисления одного выражения с результатами вычисления другого.

Диапазон: проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.

Принадлежность множеству: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.

Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.

Значение NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Сравнение.

В языке SQL можно использовать следующие операторы сравнения: = – равенство; < – меньше; > – больше; <= – меньше или равно; >= – больше или равно; <> – не равно.

Пример. Показать все операции отпуска товаров объемом больше 20.

```
SELECT * FROM Сделка
   WHERE Количество>20
```

Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также скобок, используемых для определения порядка вычисления выражения. Вычисление выражения в условиях выполняется по следующим правилам:

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операторы NOT выполняются до выполнения операторов AND и OR.
- Операторы AND выполняются до выполнения операторов OR.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Пример. Вывести список товаров, цена которых больше или равна 100 и меньше или равна 150.

```
SELECT Название, Цена
   FROM Товар
   WHERE Цена>=100 And Цена<=150
```

Пример. Вывести список клиентов из Москвы или из Самары.

```
SELECT Фамилия, ГородКлиента
   FROM Клиент
   WHERE ГородКлиента=«Москва» Or
       ГородКлиента=«Самара»
```

Диапазон.

Оператор BETWEEN используется для поиска значения внутри некоторого интервала, определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в условие поиска.

Пример. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150 (запрос эквивалентен рассмотреному ранее).

```
SELECT Название, Цена
   FROM Товар
   WHERE Цена Between 100 And 150
```

При использовании отрицания NOT BETWEEN требуется, чтобы проверяемое значение лежало вне границ заданного диапазона.

Пример. Вывести список товаров, цена которых не лежит в диапазоне от 100 до 150.

```
SELECT Товар.Название, Товар.Цена
   FROM Товар
   WHERE Товар.Цена Not Between 100 And 150
```

Или (что эквивалентно)

```
SELECT Товар.Название, Товар.Цена
  FROM Товар
  WHERE (Товар.Цена<100) OR (Товар.Цена>150)
```

Принадлежность множеству.

Оператор IN используется для сравнения некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора IN может быть достигнут тот же результат, что и в случае применения оператора OR, однако оператор IN выполняется быстрее.

Пример. Вывести список клиентов из Москвы или из Самары.

```
SELECT Фамилия, ГородКлиента
  FROM Клиент
  WHERE ГородКлиента in («Москва», «Самара»)
```

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример. Вывести список клиентов, проживающих не в Москве и не в Самаре.

```
SELECT Фамилия, ГородКлиента
   FROM Клиент
   WHERE ГородКлиента
    Not in («Москва»,»Самара»)
```

Соответствие шаблону.

С помощью оператора LIKE можно выполнять сравнение выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

Символ % – вместо этого символа может быть подставлено любое количество произвольных символов.

Символ _ заменяет один символ строки.

[] – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.

[^] – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример. Найти клиентов, у которых в номере телефона вторая цифра – 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
   FROM Клиент
   WHERE Клиент.Телефон Like «_4%»
```

Пример. Найти клиентов, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
   FROM Клиент
   WHERE Клиент.Телефон Like «_[24]%»
```

Пример. Найти клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
   FROM Клиент
   WHERE Клиент.Телефон Like «_[2-4]%»
```

Пример. Найти клиентов, у которых в фамилии встречается слог «ро».

```
SELECT Клиент.Фамилия
   FROM Клиент
   WHERE Клиент.Фамилия Like «%ро%»
```

Значение NULL.

Оператор IS NULL используется для сравнения текущего значения со значением NULL – специальным значением, указывающим на отсутствие любого значения. NULL – это не то же самое, что знак пробела (пробел – допустимый символ) или ноль (0 – допустимое число). NULL отличается и от строки нулевой длины (пустой строки).

Пример. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
   FROM Клиент
   WHERE Телефон Is Null
```

IS NOT NULL используется для проверки присутствия значения в поле.

Пример. Выборка сотрудников, у которых есть телефон (поле Телефон содержит какое-либо значение).

```
SELECT Клиент.Фамилия, Клиент.Телефон
   FROM Клиент
   WHERE Клиент.Телефон Is Not Null
```

Предложение ORDER BY.

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Однако их можно требуемым образом отсортировать, для чего в оператор SELECT помещается фраза ORDER BY, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ORDER BY через запятую. Способ сортировки задается ключевым словом, указываемым в рамках параметра ORDER BY следом за названием поля, по которому выполняется сортировка. По умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC. Для выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать ключевое слово DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример.Вывести список клиентов в алфавитном порядке.

```
SELECT Клиент.Фамилия, Клиент.Фирма
   FROM Клиент
   ORDER BY Клиент.Фамилия
```

Во фразе ORDER BY может быть указано и больше одного элемента. Главный (первый) ключ сортировки определяет общую упорядоченность строк результирующей таблицы. Если во всех строках результирующей таблицы значения главного ключа сортировки являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако, если значения главного ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае, возможно, придется упорядочить строки с одним и тем же значением главного ключа по какому-либо дополнительному ключу сортировки.

Пример. Вывести список фирм и клиентов. Названия фирм упорядочить в алфавитном порядке, имена клиентов в каждой фирме отсортировать в обратном порядке.

```
SELECT Клиент.Фирма, Клиент.Фамилия
```

```
FROM Клиент
ORDER BY Клиент.Фирма, Клиент.Фамилия DESC
```

Вопрос 5. Вычисления и подведение итогов в запросах.

Построение вычисляемых полей.

В общем случае для создания вычисляемого (производного) поля в списке SELECT следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого столбца (поля) таблицы или запроса, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM соответствующей инструкции. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS.

Пример. Рассчитать общую стоимость для каждой сделки. Этот запрос использует расчет результирующих столбцов на основе арифметических выражений.

```
SELECT Товар.Название, Товар.Цена,
   Сделка.Количество,
   Товар.Цена*Сделка.Количество AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
```

Пример. Получить список фирм с указанием фамилии и инициалов клиентов.

```
SELECT Фирма, Фамилия+»«+
   Left(Имя,1)+».»+Left(Отчество,1)+».»AS ФИО
FROM Клиент
```

В запросе использована встроенная функция Left, позволяющая вырезать в текстовой переменной один символ слева в данном случае.

Пример. Получить список товаров с указанием года и месяца продажи.

```
SELECT Товар.Название, Year(Сделка.Дата)
   AS Год, Month(Сделка.Дата) AS Месяц
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
```

В запросе использованы встроенные функции Year и Month для выделения года и месяца из даты.

Использование итоговых функций.

С помощью итоговых (агрегатных) функций в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные итоговые функции:

Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;

Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;

Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.

Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Чаще всего в качестве выражения выступают имена столбцов. Выражение может вычисляться и по значениям нескольких таблиц.

Все эти функции оперируют со значениями в единственном столбце таблицы или с арифметическим выражением и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей, за исключением COUNT(*). При вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся конкретным значениям столбца. Вариант COUNT(*) - особый случай использования функции COUNT, его назначение состоит в подсчете всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово DISTINCT. Оно не имеет смысла для функций MIN и MAX, однако его использование может повлиять на результаты выполнения функций SUM и AVG, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT может быть указано в любом запросе не более одного раза.

Очень важно отметить, что итоговые функции могут использоваться только в списке предложения SELECT и в составе предложения HAVING. Во всех других случаях это недопустимо. Если список в предложении SELECT содержит итоговые функции, а в тексте запроса отсутствует фраза GROUP BY, обеспечивающая объединение данных в группы, то ни один из элементов списка предложения SELECT не может включать каких-либо ссылок на поля, за исключением ситуации, когда поля выступают в качестве аргументов итоговых функций.

Пример. Определить первое по алфавиту название товара.

```
SELECT Min(Товар.Название) AS Min_Название
FROM Товар
```

Пример. Определить количество сделок.

```
SELECT Count(*) AS Количество_сделок
FROM Сделка
```

Пример. Определить суммарное количество проданного товара.

```
SELECT Sum(Сделка.Количество)
   AS Количество_товара
FROM Сделка
```

Пример. Определить среднюю цену проданного товара.

```
SELECT Avg(Товар.Цена) AS Avg_Цена
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара;
```

Пример. Подсчитать общую стоимость проданных товаров.

```
SELECT Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
ON Товар.КодТовара=Сделка.КодТовара
```

Предложение GROUP BY.

Часто в запросах требуется формировать промежуточные итоги, что обычно отображается появлением в запросе фразы «для каждого...». Для этой цели в операторе SELECT используется предложение GROUP BY. Запрос, в котором присутствует GROUP BY, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная суммарная строка. Стандарт SQL требует, чтобы предложение SELECT и фраза GROUP BY были тесно связаны между собой. При наличии в операторе SELECT фразы GROUP BY каждый элемент списка в предложении SELECT должен иметь единственное значение для всей группы. Более того, предложение SELECT может включать только следующие типы элементов: имена полей, итоговые функции, константы и выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения SELECT, должны присутствовать и во фразе GROUP BY - за исключением случаев, когда имя столбца используется в итоговой функции. Обратное правило не является справедливым - во фразе GROUP BY могут быть имена столбцов, отсутствующие в списке предложения SELECT.

Если совместно с GROUP BY используется предложение WHERE, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом SQL определено, что при проведении группирования все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значение NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Пример. Вычислить средний объем покупок, совершенных каждым покупателем.

```
SELECT Клиент.Фамилия, Avg(Сделка.Количество)
   AS Среднее_количество
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фамилия
```

Фраза «каждым покупателем» нашла свое отражение в SQL-запросе в виде предложения GROUP BY Клиент.Фамилия.

Пример. Определить, на какую сумму был продан товар каждого наименования.

```
SELECT Товар.Название,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название
```

Пример. Подсчитать количество сделок, осуществленных каждой фирмой.

```
SELECT Клиент.Фирма, Count(Сделка.КодСделки)
   AS Количество_сделок
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фирма
```

Пример. Подсчитать общее количество купленного для каждой фирмы товара и его стоимость.

```
SELECT Клиент.Фирма, Sum(Сделка.Количество)
   AS Общее_Количество,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
```

```
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Клиент.Фирма
```

Пример. Определить суммарную стоимость каждого товара за каждый месяц.

```
SELECT Товар.Название, Month(Сделка.Дата)
   AS Месяц,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Пример. Определить суммарную стоимость каждого товара первого сорта за каждый месяц.

```
SELECT Товар.Название, Month(Сделка.Дата)
   AS Месяц,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
WHERE Товар.Сорт=«Первый»
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Предложение HAVING.

При помощи HAVING отражаются все предварительно сгруппированные посредством GROUP BY блоки данных, удовлетворяющие заданным в HAVING условиям. Это дополнительная возможность «профильтровать» выходной набор.

Условия в HAVING отличаются от условий в WHERE:

HAVING исключает из результирующего набора данных группы с результатами агрегированных значений;

WHERE исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;

в условии поиска WHERE нельзя задавать агрегатные функции.

Пример. Определить фирмы, у которых общее количество сделок превысило три.

```
SELECT Клиент.Фирма, Count(Сделка.Количество)
   AS Количество_сделок
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
GROUP BY Клиент.Фирма
HAVING Count(Сделка.Количество)>3
```

Пример. Вывести список товаров, проданных на сумму более 10000 руб.

```
SELECT Товар.Название,
   Sum(Товар.Цена*Сделка.Количество)
   AS Стоимость
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Сделка.Количество)>10000
```

Пример. Вывести список товаров, проданных на сумму более 10000 без указания суммы.

```
SELECT Товар.Название
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Товар.Название
HAVING Sum(Товар.Цена*Сделка.Количество)>10000
```

Вопрос 6. Построение вложенных подзапросов.

Понятие подзапроса.

Часто невозможно решить поставленную задачу путем одного запроса. Это особенно актуально, когда при использовании условия поиска в предложении WHERE значение, с которым надо сравнивать, заранее не определено и должно быть вычислено в момент выполнения оператора SELECT. В таком случае приходят на помощь законченные операторы SELECT, внедренные в тело другого оператора SELECT. Внутренний подзапрос представляет собой также оператор SELECT, а кодирование его предложений подчиняется тем же правилам, что и основного оператора SELECT. Внешний оператор SELECT использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут быть помещены непосредственно после оператора сравнения (=, <, >, <=, >=, <>) в предложения WHERE и HAVING внешнего оператора SELECT – они получают название подзапросов или вложенных запросов. Кроме того, внутренние операторы SELECT могут применяться в операторах INSERT, UPDATE и DELETE.

Подзапрос – это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки.
К подзапросам применяются следующие правила и ограничения:

- фраза ORDER BY не используется, хотя и может присутствовать во внешнем подзапросе;

- список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в подзапросе присутствует ключевое слово EXISTS;
- по умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются квалифицированные имена столбцов (т.е. с указанием таблицы);
- если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.

Существует два типа подзапросов:

Скалярный подзапрос возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.

Табличный подзапрос возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.

Использование подзапросов, возвращающих единичное значение.

Пример. Определить дату продажи максимальной партии товара.

```
SELECT Дата, Количество
FROM Сделка
WHERE Количество=(SELECT Max(Количество) FROM Сделка)
```

Во вложенном подзапросе определяется максимальное количество товара. Во внешнем подзапросе – дата, для которой количество товара оказалось равным максимальному. Необходимо отметить, что нельзя прямо использовать предложение WHERE Количество=Max(Количество), поскольку применять обобщающие функции в предложениях WHERE запрещено. Для достижения желаемого результата следует создать подзапрос, вычисляющий максимальное значение количества, а затем использовать его во внешнем операторе SELECT, предназначенном для выборки дат сделок, где количество товара совпало с максимальным значением.

Пример. Определить даты сделок, превысивших по количеству товара среднее значение и указать для этих сделок превышение над средним уровнем.

```
SELECT Дата, Количество,
Количество-(SELECT Avg(Количество)
      FROM Сделка) AS Превышение
FROM Сделка
WHERE Количество>
   (SELECT Avg(Количество)
```

```
FROM Сделка)
```

В приведенном примере результат подзапроса, представляющий собой среднее значение количества товара по всем сделкам вообще, используется во внешнем операторе SELECT как для вычисления отклонения количества от среднего уровня, так и для отбора сведений о датах.

Пример. Определить клиентов, совершивших сделки с максимальным количеством товара.

```
SELECT Клиент.Фамилия
FROM Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE  Сделка.Количество=
   (SELECT Max(Сделка.Количество)
   FROM Сделка)
```

Здесь показан пример использования подзапроса при выборке данных из разных таблиц.

Пример. Определить клиентов, в сделках которых количество товара отличается от максимального не более чем на 10%.

```
SELECT Клиент.Фамилия,
   Сделка.Количество
FROM Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=
   Сделка.КодКлиента
WHERE  Сделка.Количество>=0.9*
   (SELECT Max(Сделка.Количество)
   FROM Сделка)
```

Покажем, как применяются подзапросы в предложении HAVING.

Пример. Определить даты, когда среднее количество проданного за день товара оказалось больше 20 единиц.

```
SELECT Сделка.Дата, Avg(Сделка.Количество) AS
   Среднее_за_день
FROM Сделка
GROUP BY Сделка.Дата
HAVING Avg(Сделка.Количество)>20
```

За каждый день определяется среднее количество товара, которое сравнивается с числом 20. Добавим в запрос подзапрос.

Пример. Определить даты, когда среднее количество проданного за день товара оказалось больше среднего показателя по всем сделкам вообще.

```
SELECT Сделка.Дата,
   Avg(Сделка.Количество)
   AS Среднее_за_день
```

```
FROM Сделка
GROUP BY Сделка.Дата
HAVING Avg(Сделка.Количество)>
   (SELECT Avg(Сделка.Количество)
      FROM Сделка)
```

Внутренний подзапрос определяет средний по всем сделкам показатель, с которым во внешнем запросе сравнивается среднее за каждый день количество товара.

Использование подзапросов, возвращающих множество значений.

Во многих случаях значение, подлежащее сравнению в предложениях WHERE или HAVING, представляет собой не одно, а несколько значений. Вложенные подзапросы генерируют непоименованное промежуточное отношение, временную таблицу. Оно может использоваться только в том месте, где появляется в подзапросе. К такому отношению невозможно обратиться по имени из какого-либо другого места запроса. Применяемые к подзапросу операции основаны на тех операциях, которые, в свою очередь, применяются к множеству, а именно:

{ WHERE | HAVING } выражение [NOT] IN (подзапрос);

{ WHERE | HAVING } выражение оператор_сравнения { ALL | SOME | ANY }(подзапрос);

{WHERE | HAVING } [NOT] EXISTS (подзапрос);

Использование операций IN и NOT IN.

Оператор IN используется для сравнения некоторого значения со списком значений, при этом проверяется, входит ли значение в предоставленный список или сравниваемое значение не является элементом представленного списка.

Пример. Определить список товаров, которые имеются на складе.

```
SELECT  Название
FROM  Товар
WHERE КодТовара In
   (SELECT КодТовара FROM Склад)
```

Пример. Определить список отсутствующих на складе товаров.

```
SELECT  Название
FROM  Товар
WHERE КодТовара Not In (SELECT КодТовара
   FROM Склад)
```

Пример. Определить товары, которые покупают клиенты из Москвы.

```
SELECT DISTINCT Товар.Название,
   Клиент.ГородКлиента
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
```

```
ON Клиент.КодКлиента=Сделка.КодКлиента)
ON Товар.КодТовара=Сделка.КодТовара
WHERE Клиент.ГородКлиента='Москва'
```

В результат включаются товары, приобретенные клиентами из Москвы, однако не исключено, что покупателями таких товаров были и клиенты из других городов.

Введение в запрос фразы «только» требует использования операции NOT IN.

Пример. Определить товары, покупку которых осуществляют только клиенты из Москвы, и никто другой.

```
SELECT DISTINCT Товар.Название,
   Клиент.ГородКлиента
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента)
ON Товар.КодТовара=Сделка.КодТовара
WHERE Товар.Название NOT IN
   (SELECT Товар.Название
   FROM Товар INNER JOIN
     (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
   WHERE Клиент.ГородКлиента<>'Москва')
```

Пример. Какие товары ни разу не купили московские клиенты?

```
 SELECT DISTINCT Товар.Название,
   Клиент.ГородКлиента
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
WHERE Товар.Название NOT IN
   (SELECT Товар.Название
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
WHERE Клиент.ГородКлиента='Москва')
```

Во вложенном запросе определяется список товаров, приобретаемых клиентами из Москвы. Во внешнем запросе выбираются только те товары, которые не входят в этот список.

Пример. Определить фирмы, покупающие товары местного производства.

```
SELECT DISTINCT Клиент.Фирма, Клиент.ГородКлиента,
```

```
    Товар.ГородТовара
FROM Товар INNER JOIN
    (Клиент INNER JOIN Сделка
    ON Клиент.КодКлиента=Сделка.КодКлиента)
    ON Товар.КодТовара=Сделка.КодТовара
WHERE Клиент.ГородКлиента=Товар.ГородТовара
```

В результате выполнения запроса перечисляются сделки, когда клиенту был продан товар, изготовленный в его городе, что совсем не исключает наличие сделок этих же клиентов, связанных с приобретением товара из другого города.

Введем в запрос фразу «только» – сразу потребуется привлечение операции NOT IN.

Пример. Определить фирмы, которые покупают только товары, произведенные в своем городе, и никакие другие.

```
SELECT DISTINCT Клиент.Фирма,
    Клиент.ГородКлиента,
    Товар.ГородТовара
FROM Товар INNER JOIN
    (Клиент INNER JOIN Сделка
    ON Клиент.КодКлиента=Сделка.КодКлиента)
    ON Товар.КодТовара=Сделка.КодТовара
WHERE Клиент.ГородКлиента NOT IN
    (SELECT DISTINCT  Клиент.ГородКлиента
    FROM Товар INNER JOIN
    (Клиент INNER JOIN Сделка
    ON Клиент.КодКлиента=Сделка.КодКлиента)
    ON Товар.КодТовара=Сделка.КодТовара
    WHERE Клиент.ГородКлиента<>
      Товар.ГородТовара)
```

Во вложенном запросе определяется множество фирм, совершивших хотя бы одну покупку товара из чужого города. Затем определяются фирмы, не входящие в это множество.

Использование ключевых слов ANY и ALL.

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел.

Если подзапросу будет предшествовать ключевое слово ALL, условие сравнения считается выполненным, только когда оно выполняется для всех значений в результирующем столбце подзапроса.

Если записи подзапроса предшествует ключевое слово ANY, то условие сравнения считается выполненным, когда оно выполняется хотя бы для одного из значений в результирующем столбце подзапроса.

Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY – невыполненным. Ключевое слово SOME является синонимом слова ANY.

Пример. Определить клиентов, совершивших сделки с максимальным количеством товара.

```
SELECT Клиент.Фамилия, Сделка.Количество
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE Сделка.Количество>=ALL(SELECT Количество
   FROM Сделка)
```

В примере определены клиенты, в сделках которых количество товара больше или равно количеству товара в каждой из всех сделок.

Пример. Найти фирму, купившую товаров на сумму, превышающую 10000 руб.

```
SELECT Клиент.Фирма,
   Sum(Товар.Цена*Сделка.Количество)
   AS Общ_стоимость
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Клиент.Фирма
HAVING Sum(Товар.Цена*Сделка.Количество)>10000
```

Добавим в запрос подзапрос.

Пример. Найти фирму, которая приобрела товаров на самую большую сумму.

```
SELECT Клиент.Фирма,
   Sum(Товар.Цена*Сделка.Количество)
   AS Общ_стоимость
FROM Товар INNER JOIN
   (Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента)
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Клиент.Фирма
HAVING Sum(Товар.Цена*Сделка.Количество)>=
ALL(SELECT Sum(Товар.Цена*Сделка.Количество)
   FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=Сделка.КодТовара
GROUP BY Сделка.КодКлиента)
```

Вложенный подзапрос подсчитывает общую стоимость покупок каждого клиента. Внешний подзапрос также подсчитывает общую стоимость покупок каждого клиента и определяет тех, для кого эта сумма, по сравнению с другими покупателями, оказалась больше или точно такой же.

Пример. Найти фирмы, в сделках которых количество товара превышает такой же показатель хотя бы в одной сделке клиентов из Самары.

```
SELECT Клиент.Фирма, Сделка.Количество
FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE Сделка.Количество>
ANY(SELECT Сделка.Количество
   FROM Клиент INNER JOIN Сделка
   ON Клиент.КодКлиента=Сделка.КодКлиента
WHERE Клиент.ГородКлиента='Самара')
```

Использование операций EXISTS и NOT EXISTS.

Ключевые слова EXISTS и NOT EXISTS предназначены для использования только совместно с подзапросами. Результат их обработки представляет собой логическое значение TRUE или FALSE. Для ключевого слова EXISTS результат равен TRUE в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки операции EXISTS будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову EXISTS. Поскольку по ключевым словам EXISTS и NOT EXISTS проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Пример. Определить список имеющихся на складе товаров.

```
SELECT Название
FROM Товар
WHERE EXISTS (SELECT КодТовара
   FROM Склад
WHERE Товар.КодТовара=Склад.КодТовара)
```

Пример. Определить список отсутствующих на складе товаров.

```
SELECT Название
FROM Товар
WHERE NOT EXISTS (SELECT КодТовара
   FROM Склад
WHERE Товар.КодТовара=Склад.КодТовара)
```

Вопрос 7. Запросы модификации данных.

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью запросов действия пользователь может добавить,

удалить или обновить блоки данных. Существует три вида запросов действия:

INSERT INTO – запрос добавления;

DELETE – запрос удаления;

UPDATE – запрос обновления.

Запрос добавления.

Оператор INSERT применяется для добавления записей в таблицу. Формат оператора:

```
<оператор_вставки>::=INSERT INTO <имя_таблицы>
   [(имя_столбца [,...n])]
   {VALUES (значение[,...n])|
   <SELECT_оператор>}
```

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Первая форма оператора INSERT с параметром VALUES предназначена для вставки единственной строки в указанную таблицу. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERT указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL, за исключением тех случаев, когда при описании столбца использовался параметр DEFAULT. Список значений должен следующим образом соответствовать списку столбцов:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
- типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример. Добавить в таблицу ТОВАР новую запись.

```
INSERT INTO Товар (Название, Тип, Цена)
   VALUES(‘Славянский’, ‘шоколад’, 12)
```

Если столбцы таблицы ТОВАР указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы ТОВАР, оператор можно упростить.

```
INSERT INTO Товар VALUES (‘Славянский’, ‘шоколад’, 12)
```

Вторая форма оператора INSERT с параметром SELECT позволяет скопировать множество строк из одной таблицы в другую. Предложение SELECT может представлять собой любой допустимый оператор SELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора SELECT, применимы и в этом случае.

Поскольку оператор SELECT в общем случае возвращает множество записей, то оператор INSERT в такой форме приводит к добавлению в таблицу аналогичного числа новых записей.

Пример. Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог
  (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата)
  AS Месяц, Sum(Товар.Цена*Сделка.Количество)
  AS Стоимость
FROM Товар INNER JOIN Сделка
  ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Запрос удаления.

Оператор DELETE предназначен для удаления группы записей из таблицы. Формат оператора:

```
<оператор_удаления> ::=DELETE
   FROM <имя_таблицы>[WHERE <условие_отбора>]
```

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Если предложение WHERE присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложение WHERE, из таблицы будут удалены все записи, однако сама таблица сохранится.

Пример. Удалить все прошлогодние сделки.

```
DELETE
FROM Сделка
WHERE Year(Сделка.Дата)=Year(GETDATE())-1
```

В приведенном примере условие отбора формируется с учетом года (функция Year) от текущей даты (функция GETDATE()).

Запрос обновления.

Оператор UPDATE применяется для изменения значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

```
<оператор_изменения> ::=
   UPDATE имя_таблицы SET имя_столбца=
      <выражение>[,...n]
   [WHERE <условие_отбора>]
```

Параметр имя_таблицы – это либо имя таблицы базы данных, либо имя обновляемого представления. В предложении SET указываются имена одного и более столбцов, данные в которых необходимо изменить. Предложение WHERE является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложение WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

Пример. Для товаров первого сорта установить цену в значение 140 и остаток – в значение 20 единиц.

```
UPDATE Товар SET Товар.Цена=140, Товар.Остаток=20
WHERE Товар.Сорт=« Первый «
```

Пример. Увеличить цену товаров первого сорта на 25%.

```
UPDATE Товар SET Товар.Цена=Товар.Цена*1.25
WHERE Товар.Сорт=« Первый «
```

Пример. В сделке с максимальным количеством товара увеличить число товаров на 10%.

```
UPDATE Сделка SET Сделка.Количество=
   Сделка.Количество*1.1
WHERE Сделка.Количество=
   (SELECT Max(Сделка.Количество) FROM Сделка)
```

Введение в понятие «целостность данных».

Выполнение операторов модификации данных в таблицах базы данных INSERT, DELETE и UPDATE может привести к нарушению целостности данных и их корректности, т.е. к потере их достоверности и непротиворечивости.

Чтобы информация, хранящаяся в базе данных, была однозначной и непротиворечивой, в реляционной модели устанавливаются некоторые ограничительные условия – правила, определяющие возможные значения данных и обеспечивающие логическую основу для поддержания корректных значений. Ограничения целостности позволяют свести к минимуму ошибки, возникающие при обновлении и обработке данных.

В базе данных, построенной на реляционной модели, задается ряд правил целостности, которые, по сути, являются ограничениями для всех допустимых состояний базы данных и гарантируют корректность данных. Рассмотрим следующие типы ограничений целостности данных:

- обязательные данные;
- ограничения для доменов полей;
- корпоративные ограничения;
- целостность сущностей;
- ссылочная целостность.

Обязательные данные.

Некоторые поля всегда должны содержать одно из допустимых значений, другими словами, эти поля не могут иметь пустого значения.

Ограничения для доменов полей.

Каждое поле имеет свой домен, представляющий собой набор его допустимых значений.

Корпоративные ограничения целостности.

Существует понятие «корпоративные ограничения целостности» как дополнительные правила поддержки целостности данных, определяемые пользователями, принятые на предприятии или администраторами баз данных. Ограничения предприятия называются бизнес-правилами.

Целостность сущностей.

Это ограничение целостности касается первичных ключей базовых таблиц. По определению, первичный ключ – минимальный идентификатор (одно или несколько полей), который используется для уникальной идентификации записей в таблице. Таким образом, никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации записей.

Целостность сущностей определяет, что в базовой таблице ни одно поле первичного ключа не может содержать отсутствующих значений, обозначенных NULL.

Если допустить присутствие определителя NULL в любой части первичного ключа, это равносильно утверждению, что не все его поля необходимы для уникальной идентификации записей, и противоречит определению первичного ключа.

Ссылочная целостность.

Указанное ограничение целостности касается внешних ключей. Внешний ключ – это поле (или множество полей) одной таблицы, являющееся ключом другой (или той же самой) таблицы. Внешние ключи используются для установления логических связей между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям ключа другой.

Между двумя или более таблицами базы данных могут существовать отношения подчиненности, которые определяют, что для каждой записи главной таблицы (называемой еще родительской) может

существовать одна или несколько записей в подчиненной таблице (называемой также дочерней).

Существует три разновидности связи между таблицами базы данных:

- «один-ко-многим»;
- «один-к-одному»;
- «многие-ко-многим».

Отношение «один-ко-многим» имеет место, когда одной записи родительской таблицы может соответствовать несколько записей дочерней. Связь «один-ко-многим» иногда называют связью «многие-к-одному». И в том, и в другом случае сущность связи между таблицами остается неизменной.

Связь «один-ко-многим» наиболее распространена для реляционных баз данных. Она позволяет моделировать также иерархические структуры данных.

Отношение «один-к-одному» имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней. Это отношение встречается намного реже, чем отношение «один-ко-многим». Его используют, если не хотят, чтобы таблица БД «распухала» от второстепенной информации. Использование связи «один-к-одному» приводит к тому, что для чтения связанной информации в нескольких таблицах приходится производить несколько операций чтения вместо одной, когда данные хранятся в одной таблице.

Отношение «многие-ко-многим» имеет место в следующих случаях:

- одной записи в родительской таблице соответствует более одной записи в дочерней таблице;
- одной записи в дочерней таблице соответствует более одной записи в родительской таблице.

Считается, что всякая связь «многие-ко-многим» может быть заменена на связь «один-ко-многим» (одну или несколько).

Часто связь между таблицами устанавливается по первичному ключу, т.е. значениям внешнего ключа одной таблицы присваиваются значения первичного ключа другой. Однако это не является обязательным – в общем случае связь может устанавливаться и с помощью вторичных ключей. Кроме того, при установлении связей между таблицами не требуется непременная уникальность ключа, обеспечивающего установление связи. Поля внешнего ключа не обязаны иметь те же имена, что и имена ключей, которым они соответствуют. Внешний ключ может ссылаться на свою собственную таблицу – в таком случае внешний ключ называется рекурсивным.

Ссылочная целостность определяет: если в таблице существует внешний ключ, то его значение должно либо соответствовать значению первичного ключа некоторой записи в базовой таблице, либо задаваться определителем NULL.

Существует несколько важных моментов, связанных с внешними ключами. Во-первых, следует проанализировать, допустимо ли использование во внешних ключах пустых значений. В общем случае, если участие дочерней таблицы в связи является обязательным, то рекомендуется запрещать применение пустых значений в соответствующем внешнем ключе. В то же время, если имеет место частичное участие дочерней таблицы в связи, то помещение пустых значений в поле внешнего ключа должно быть разрешено. Например, если в операции фиксации сделок некоторой торговой фирмы необходимо указать покупателя, то поле КодКлиента должно иметь атрибут NOT NULL. Если допускается продажа или покупка товара без указания клиента, то для поля КодКлиента можно указать атрибут NULL.

Следующая проблема связана с организацией поддержки ссылочной целостности при выполнении операций модификации данных в базе. Здесь возможны следующие ситуации:

Вставка новой строки в дочернюю таблицу. Для обеспечения ссылочной целостности необходимо убедиться, что значение внешнего ключа новой строки дочерней таблицы равно пустому значению либо некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.

Удаление строки из дочерней таблицы. Никаких нарушений ссылочной целостности не происходит.

Обновление внешнего ключа в строке дочерней таблицы. Этот случай подобен описанной выше первой ситуации. Для сохранения ссылочной целостности необходимо убедиться, что значение внешнего ключа в обновленной строке дочерней таблицы равно пустому значению либо некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.

Вставка строки в родительскую таблицу. Такая вставка не может вызвать нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов.

Удаление строки из родительской таблицы. Ссылочная целостность окажется нарушенной, если в дочерней таблице будут существовать строки, ссылающиеся на удаленную строку родительской таблицы. В этом случае может использоваться одна из следующих стратегий:

- NO ACTION. Удаление строки из родительской таблицы запрещается, если в дочерней таблице существует хотя бы одна ссылающаяся на нее строка.
- CASCADE. При удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы. Если любая из удаляемых строк дочерней таблицы выступает в качестве родительской стороны в какой-либо другой связи, то операция удаления применяется ко всем строкам дочерней таблицы этой связи и т.д. Другими словами, удаление строки родительской таблицы автоматически распространяется на любые дочерние таблицы.

- SET NULL. При удалении строки из родительской таблицы во всех ссылающихся на нее строках дочернего отношения в поле внешнего ключа, соответствующего первичному ключу удаленной строки, записывается пустое значение. Следовательно, удаление строк из родительской таблицы вызовет занесение пустого значения в соответствующее поле дочерней таблицы. Эта стратегия может использоваться, только когда в поле внешнего ключа дочерней таблицы разрешается помещать пустые значения.
- SET DEFAULT. При удалении строки из родительской таблицы в поле внешнего ключа всех ссылающихся на нее строк дочерней таблицы автоматически помещается значение, указанное для этого поля как значение по умолчанию. Таким образом, удаление строки из родительской таблицы вызывает помещение принимаемого по умолчанию значения в поле внешнего ключа всех строк дочерней таблицы, ссылающихся на удаленную строку. Эта стратегия применима лишь в тех случаях, когда полю внешнего ключа дочерней таблицы назначено некоторое значение, принимаемое по умолчанию.
- NO CHECK. При удалении строки из родительской таблицы никаких действий по сохранению ссылочной целостности данных не предпринимается.

Обновление первичного ключа в строке родительской таблицы. Если значение первичного ключа некоторой строки родительской таблицы будет обновлено, нарушение ссылочной целостности случится при том условии, что в дочернем отношении существуют строки, ссылающиеся на исходное значение первичного ключа. Для сохранения ссылочной целостности может применяться любая из описанных выше стратегий. При использовании стратегии CASCADE обновление значения первичного ключа в строке родительской таблицы будет отображено в любой строке дочерней таблицы, ссылающейся на данную строку.

Существует и другой вид целостности – смысловая (семантическая) целостность базы данных. Требование смысловой целостности определяет, что данные в базе данных должны изменяться таким образом, чтобы не нарушалась сложившаяся между ними смысловая связь.

Уровень поддержания целостности данных в разных системах существенно варьируется.

Идеология архитектуры клиент-сервер требует переноса максимально возможного числа правил целостности данных на сервер. К преимуществам такого подхода относятся:

- гарантия целостности базы данных, поскольку все правила сосредоточены в одном месте (в базе данных);
- автоматическое применение определенных на сервере ограничений целостности для любых приложений;
- отсутствие различных реализаций ограничений в разных клиентских приложениях, работающих с базой данных;

- быстрое срабатывание ограничений, поскольку они реализованы на сервере и, следовательно, нет необходимости посылать данные клиенту, увеличивая при этом сетевой трафик;
- доступность внесенных в ограничения на сервере изменений для всех клиентских приложений, работающих с базой данных, и отсутствие необходимости повторного распространения измененных приложений клиентов среди пользователей.

К недостаткам хранения ограничений целостности на сервере можно отнести:

- отсутствие у клиентского приложения возможности реагировать на некоторые ошибочные ситуации, возникающие на сервере при реализации тех или иных правил (например, ошибок при выполнении хранимых процедур на сервере);
- ограниченность возможностей языка SQL и языка хранимых процедур и триггеров для реализации всех возникающих потребностей определения целостности данных.

На практике в клиентских приложениях реализуют лишь такие правила, которые тяжело или невозможно реализовать с применением средств сервера. Все остальные ограничения целостности данных переносятся на сервер.

Вопрос 8. Создание и удаление таблиц.

Создание базы данных.

В различных СУБД процедура создания баз данных обычно закрепляется только за администратором баз данных. В однопользовательских системах принимаемая по умолчанию база данных может быть сформирована непосредственно в процессе установки и настройки самой СУБД. Стандарт SQL не определяет, как должны создаваться базы данных, поэтому в каждом из диалектов языка SQL обычно используется свой подход. В соответствии со стандартом SQL, таблицы и другие объекты базы данных существуют в некоторой среде. Помимо всего прочего, каждая среда состоит из одного или более каталогов, а каждый каталог – из набора схем. Схема представляет собой поименованную коллекцию объектов базы данных, некоторым образом связанных друг с другом (все объекты в базе данных должны быть описаны в той или иной схеме). Объектами схемы могут быть таблицы, представления, домены, утверждения, сопоставления, толкования и наборы символов. Все они имеют одного и того же владельца и множество общих значений, принимаемых по умолчанию.

Стандарт SQL оставляет за разработчиками СУБД право выбора конкретного механизма создания и уничтожения каталогов, однако механизм создания и удаления схем регламентируется посредством операторов CREATE SCHEMA и DROP SCHEMA. В стандарте также

указано, что в рамках оператора создания схемы должна существовать возможность определения диапазона привилегий, доступных пользователям создаваемой схемы. Однако конкретные способы определения подобных привилегий в разных СУБД различаются.

В настоящее время операторы CREATE SCHEMA и DROP SCHEMA реализованы в очень немногих СУБД. В других реализациях, например, в СУБД MS SQL Server, используется оператор CREATE DATABASE.

Создание базы данных в среде MS SQL Server.

Процесс создания базы данных в системе SQL-сервера состоит из двух этапов: сначала организуется сама база данных, а затем принадлежащий ей журнал транзакций. Информация размещается в соответствующих файлах, имеющих расширения *.mdf (для базы данных) и *.ldf. (для журнала транзакций). В файле базы данных записываются сведения об основных объектах (таблицах, индексах, просмотрах и т.д.), а в файле журнала транзакций – о процессе работы с транзакциями (контроль целостности данных, состояния базы данных до и после выполнения транзакций).

Создание базы данных в системе SQL-сервер осуществляется командой CREATE DATABASE. Следует отметить, что процедура создания базы данных в SQL-сервере требует наличия прав администратора сервера.

```
<определение_базы_данных> ::=
  CREATE DATABASE имя_базы_данных
  [ON [PRIMARY]
  [ <определение_файла>  [,...n] ]
  [,<определение_группы> [,...n] ] ]
  [ LOG ON {<определение_файла>[,...n] } ]
  [ FOR LOAD | FOR ATTACH ]
```

Рассмотрим основные параметры представленного оператора.

При выборе имени базы данных следует руководствоваться общими правилами именования объектов. Если имя базы данных содержит пробелы или любые другие недопустимые символы, оно заключается в ограничители (двойные кавычки или квадратные скобки). Имя базы данных должно быть уникальным в пределах сервера и не может превышать 128 символов.

При создании и изменении базы данных можно указать имя файла, который будет для нее создан, изменить имя, путь и исходный размер этого файла. Если в процессе использования базы данных планируется ее размещение на нескольких дисках, то можно создать так называемые вторичные файлы базы данных с расширением *.ndf. В этом случае основная информация о базе данных располагается в первичном (PRIMARY) файле, а при нехватке для него свободного места добавляемая информация будет размещаться во вторичном файле. Подход, используемый в SQL-сервере, позволяет распределять содержимое базы данных по нескольким дисковым томам.

Параметр ON определяет список файлов на диске для размещения информации, хранящейся в базе данных.

Параметр PRIMARY определяет первичный файл. Если он опущен, то первичным является первый файл в списке.

Параметр LOG ON определяет список файлов на диске для размещения журнала транзакций. Имя файла для журнала транзакций генерируется на основе имени базы данных, и в конце к нему добавляются символы _log.

При создании базы данных можно определить набор файлов, из которых она будет состоять. Файл определяется с помощью следующей конструкции:

```
<определение_файла>::=
  ([ NAME=логическое_имя_файла,]
  FILENAME='физическое_имя_файла'
  [,SIZE=размер_файла ]
  [,MAXSIZE={max_размер_файла |UNLIMITED } ]
  [, FILEGROWTH=величина_прироста ] )[,...n]
```

Здесь логическое имя файла – это имя файла, под которым он будет опознаваться при выполнении различных SQL-команд.

Физическое имя файла предназначено для указания полного пути и названия соответствующего физического файла, который будет создан на жестком диске. Это имя останется за файлом на уровне операционной системы.

Параметр SIZE определяет первоначальный размер файла; минимальный размер параметра – 512 Кб, если он не указан, по умолчанию принимается 1 Мб.

Параметр MAXSIZE определяет максимальный размер файла базы данных. При значении параметра UNLIMITED максимальный размер базы данных ограничивается свободным местом на диске.

При создании базы данных можно разрешить или запретить автоматический рост ее размера (это определяется параметром FILEGROWTH) и указать приращение с помощью абсолютной величины в Мб или процентным соотношением.

Дополнительные файлы могут быть включены в группу:

```
<определение_группы>::=FILEGROUP имя_группы_файлов
   <определение_файла>[,...n]
```

Пример. Создать базу данных, причем для данных определить три файла на диске С, для журнала транзакций – два файла на диске С.

```
CREATE DATABASE Archive
ON PRIMARY ( NAME=Arch1,
  FILENAME=’c:\user\data\archdat1.mdf’,
```

```
SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch2,
  FILENAME='c:\user\data\archdat2.mdf',
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Arch3,
  FILENAME='c:\user\data\archdat3.mdf',
  SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
LOG ON
(NAME=Archlog1,
  FILENAME='c:\user\data\archlog1.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
(NAME=Archlog2,
  FILENAME='c:\user\data\archlog2.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

Изменение базы данных.

Большинство действий по изменению конфигурации базы данных выполняется с помощью следующей конструкции:

```
<изменение_базы_данных> ::=
  ALTER DATABASE имя_базы_данных
  { ADD FILE <определение_файла>[,...n]
    [TO FILEGROUP имя_группы_файлов ]
  | ADD LOG FILE <определение_файла>[,...n]
  | REMOVE FILE логическое_имя_файла
  | ADD FILEGROUP имя_группы_файлов
  | REMOVE FILEGROUP имя_группы_файлов
  | MODIFY FILE <определение_файла>
  | MODIFY FILEGROUP имя_группы_файлов
  <свойства_группы_файлов>}
```

Как видно из синтаксиса, за один вызов команды может быть изменено не более одного параметра конфигурации базы данных. Если необходимо выполнить несколько изменений, придется разбить процесс на ряд отдельных шагов.

В базу данных можно добавить (ADD) новые файлы данных (в указанную группу файлов или в группу, принятую по умолчанию) или файлы журнала транзакций.

Параметры файлов и групп файлов можно изменять (MODIFY).

Для удаления из базы данных файлов или групп файлов используется параметр REMOVE. Однако удаление файла возможно лишь при условии его освобождения от данных. В противном случае сервер не разрешит удаление.

В качестве свойств группы файлов используются следующие:

READONLY – группа файлов используется только для чтения; READWRITE – в группе файлов разрешаются изменения; DEFAULT – указанная группа файлов принимается по умолчанию.

Удаление базы данных.

Удаление базы данных осуществляется командой:

```
DROP DATABASE имя_базы_данных [,...n]
```

Удаляются все содержащиеся в базе данных объекты, а также файлы, в которых она размещается. Для исполнения операции удаления базы данных пользователь должен обладать соответствующими правами.

Таблица.

Создание таблицы.

После создания общей структуры базы данных можно приступить к созданию таблиц, которые представляют собой отношения, входящие в состав проекта базы данных.

Таблица – основной объект для хранения информации в реляционной базе данных. Она состоит из содержащих данные строк и столбцов, занимает в базе данных физическое пространство и может быть постоянной или временной.

Поле, также называемое в реляционной базе данных столбцом, является частью таблицы, за которой закреплен определенный тип данных. Каждая таблица базы данных должна содержать хотя бы один столбец. Строка данных – это запись в таблице базы данных, она включает поля, содержащие данные из одной записи таблицы.

Приступая к созданию таблицы, необходимо иметь ответы на ряд вопросов:

- Как будет называться таблица?
- Как будут называться столбцы (поля) таблицы?
- Какие типы данных будут закреплены за каждым столбцом?
- Какой размер памяти должен быть выделен для хранения каждого столбца?
- Какие столбцы таблицы требуют обязательного ввода?
- Из каких столбцов будет состоять первичный ключ?

Базовый синтаксис оператора создания таблицы имеет следующий вид:

```
<определение_таблицы> ::=
  CREATE TABLE имя_таблицы
  (имя_столбца тип_данных
    [NULL | NOT NULL ] [,...n])
```

Приведенный стандарт совпадает с реализацией оператора создания таблицы в среде MS SQL Server.

Главное в команде создания таблицы – определение имени таблицы и описание набора имен полей, которые указываются в соответствующем порядке. Кроме того, этой командой оговариваются типы данных и размеры полей таблицы.

Ключевое слово NULL используется для указания того, что в данном столбце могут содержаться значения NULL. Значение NULL отличается от пробела или нуля – к нему прибегают, когда необходимо указать, что данные недоступны, опущены или недопустимы. Если указано ключевое слово NOT NULL, то будут отклонены любые попытки поместить значение NULL в данный столбец. Если указан параметр NULL, помещение значений NULL в столбец разрешено. По умолчанию стандарт SQL предполагает наличие ключевого слова NULL.

Мы использовали упрощенную версию оператора CREATE TABLE стандарта SQL. Его полная версия приводится при обсуждении вопросов обеспечения целостности данных.

Пример. Создать таблицу для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. Необходимо учесть такие сведения, как название и тип товара, его цена, сорт и город, где товар производится.

```
CREATE TABLE Товар
(Название      VARCHAR(50) NOT NULL,
   Цена        MONEY NOT    NULL,
   Тип         VARCHAR(50) NOT NULL,
   Сорт        VARCHAR(50),
   ГородТовара VARCHAR(50))
```

Пример.Создать таблицу для сохранения сведений о постоянных клиентах с указанием названий города и фирмы, фамилии, имени и отчества клиента, номера его телефона.

```
CREATE TABLE  Клиент
   (Фирма       VARCHAR(50) NOT NULL,
   Фамилия      VARCHAR(50) NOT NULL,
   Имя          VARCHAR(50) NOT NULL,
   Отчество     VARCHAR(50),
   ГородКлиента VARCHAR(50),
   Телефон      CHAR(10) NOT NULL)
```

Изменение таблицы.

Структура существующей таблицы может быть модифицирована с помощью команды ALTER TABLE, упрощенный синтаксис которой представлен ниже:

```
ALTER TABLE имя_таблицы
  {[ADD [COLUMN] имя_столбца тип_данных [
    NULL | NOT NULL ]]
  | [DROP [COLUMN] имя_столбца]}
```

В среде MS SQL Server упрощенный синтаксис команды модификации таблицы имеет вид:

```
ALTER TABLE имя_таблицы
 {[ALTER COLUMN имя_столбца
 {новый_тип_данных [(точность[,масштаб])]
  [ NULL | NOT NULL ]}]
 | ADD { [имя_столбца тип_данных]
 | имя_столбца AS выражение } [,...n]
 | DROP {COLUMN имя_столбца}[,...n]
 }
```

Команда позволяет добавлять и удалять столбцы, изменять их определения.

Одно из основных правил при добавлении столбцов в существующую таблицу гласит: когда в таблице уже содержатся данные, добавляемый столбец не может быть определен с атрибутом NOT NULL. Этот атрибут означает, что для каждой строки данных соответствующий столбец должен содержать некоторое значение, поэтому добавление столбца с атрибутом NOT NULL приводит к появлению противоречия – уже существующие строки данных таблицы не будут иметь в новом столбце ненулевых значений.

Тем не менее существует способ добавления обязательных полей в существующую таблицу. Для этого необходимо:

- добавить в таблицу новый столбец, определив его с атрибутом NULL (т.е. столбец не обязан содержать каких-либо значений);
- ввести в новый столбец какие-либо значения для каждой строки данных таблицы;
- убедившись, что новый столбец содержит ненулевые значения для каждой строки данных, изменить структуру таблицы, заменив атрибут этого столбца на NOT NULL.

При изменении определений столбцов следует принимать во внимание некоторые общепринятые правила:

- размер столбца может быть увеличен до максимального значения, допускаемого соответствующим типом данных;
- размер столбца может быть уменьшен только в том случае, если содержащееся в нем наибольшее значение не будет превосходить его нового размера;
- количество разрядов числового типа данных всегда может быть увеличено;
- количество разрядов числового типа данных может быть уменьшено только в том случае, если количество разрядов наибольшего значения в соответствующем столбце не будет превосходить нового числа разрядов, определенного для этого столбца;
- количество десятичных знаков числового типа данных может быть уменьшено или увеличено;
- тип данных столбца, как правило, может быть изменен.

Некоторые реализации фактически могут ограничить разработчика в использовании некоторых опций команды ALTER TABLE. Например, может оказаться недопустимым удаление столбцов из существующей таблицы. Чтобы добиться этого, сначала потребуется удалить саму таблицу и только потом заново ее построить с нужными столбцами. Причем уже внесенные в таблицу данные будут потеряны.

Возможны трудности, связанные с удалением из таблицы столбца, который зависит от некоторого столбца другой таблицы. В таком случае сначала придется удалить ограничение столбца, а затем сам столбец.

Пример. Добавить в таблицу Клиент поле для номера расчетного счета.

```
ALTER TABLE Клиент ADD Рас_счет CHAR(20)
```

Удаление таблицы.

С течением времени структура базы данных меняется: создаются новые таблицы, а прежние становятся ненужными и удаляются из базы данных с помощью оператора:

```
DROP TABLE имя_таблицы [RESTRICT | CASCADE]
```

Следует отметить, что эта команда удалит не только указанную таблицу, но и все входящие в нее строки данных. Если требуется удалить из таблицы лишь данные, сохранив структуру таблицы, следует воспользоваться командой DELETE.

Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно. Если в операторе указано ключевое слово RESTRICT, то при наличии в базе данных хотя бы одного объекта, существование которого зависит от удаляемой таблицы, выполнение оператора DROP TABLE будет отменено. Если указано ключевое слово CASCADE, автоматически удаляются и все прочие объекты базы данных, чье существование зависит от удаляемой таблицы, а также другие объекты, зависящие от удаляемых объектов. Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может оказаться весьма ощутимым, поэтому подобные операторы следует использовать с максимальной осторожностью.

Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с некорректной структурой, можно воспользоваться оператором DROP TABLE для ее удаления, после чего создать таблицу заново.

Индексы.

Индексы в стандарте языка.

Индексы представляют собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы на основе значений одного или более

ее столбцов. Наличие индекса может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. Индекс – это набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Хотя индекс и связан с конкретным столбцом (или столбцами) таблицы, все же он является самостоятельным объектом базы данных.

Физически индекс – всего лишь упорядоченный набор значений из индексированного столбца с указателями на места физического размещения исходных строк в структуре базы данных. Когда пользователь выполняет обращающийся к индексированному столбцу запрос, СУБД автоматически анализирует индекс для поиска требуемых значений.

Однако, поскольку индексы должны обновляться системой при каждом внесении изменений в их базовую таблицу, они создают дополнительную нагрузку на систему.

Индексы обычно создаются с целью удовлетворения определенных критериев поиска после того, как таблица уже находилась некоторое время в работе и увеличилась в размерах. Создание индексов не предусмотрено стандартом SQL, однако большинство диалектов поддерживают как минимум следующий оператор:

```
CREATE [ UNIQUE ] INDEX имя_индекса
  ON имя_таблицы(имя_столбца[ASC|DESC][,...n])
```

Указанные в операторе столбцы составляют ключ индекса. Индексы могут создаваться только для базовых таблиц, но не для представлений. Если в операторе указано ключевое слово UNIQUE, уникальность значений ключа индекса будет автоматически поддерживаться системой. Требование уникальности значений обязательно для первичных ключей, а также возможно и для других столбцов таблицы (например, для альтернативных ключей). Хотя создание индекса допускается в любой момент, при его построении для уже заполненной данными таблицы могут возникнуть проблемы, связанные с дублированием данных в различных строках. Следовательно, уникальные индексы (по крайней мере, для первичного ключа) имеет смысл создавать непосредственно при формировании таблицы. В результате система сразу возьмет на себя контроль за уникальностью значений данных в соответствующих столбцах.

Если созданный индекс впоследствии окажется ненужным, его можно удалить с помощью оператора

```
DROP INDEX имя_индекса
```

Индексы в среде MS SQL Server.

Индекс представляет собой средство, помогающее ускорить поиск необходимых данных за счет физического или логического их

упорядочивания. Индекс представляет собой набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Индексы - это наборы уникальных значений для некоторой таблицы с соответствующими ссылками на данные. Они расположены в самой таблице и являются удобным внутренним механизмом системы SQL-сервера, с помощью которого осуществляется доступ к данным наиболее оптимальным способом. В среде SQL Server реализованы эффективные алгоритмы поиска нужного значения в строго определенной последовательности данных. Ускорение поиска достигается именно за счет того, что данные представляются упорядоченными (хотя физически, в зависимости от типа индекса, они могут храниться в соответствии с очередностью их добавления в таблицу). К настоящему времени разработаны эффективные математические алгоритмы поиска данных в упорядоченной последовательности. Наиболее эффективной структурой для поиска данных в машинном представлении являются B-деревья – многоуровневая иерархическая структура с переменным количеством элементов в каждом узле.

Создание индекса.

Если выборка данных из таблицы требует значительного времени, это означает, что для нее необходимо создать индекс. Индексы могут существенно повысить производительность выполнения операций поиска и выборки данных. При выборе столбца для индекса следует проанализировать, какие типы запросов чаще всего выполняются пользователями и какие столбцы являются ключевыми, т.е. задающими критерии выборки данных, например, порядок сортировки.

В среде SQL Server реализовано несколько типов индексов:

- кластерные индексы;
- некластерные индексы;
- уникальные индексы.

Некластерный индекс.

Некластерные индексы – наиболее типичные представители семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру таблицы, а лишь организуют ссылки на соответствующие строки.

Для идентификации нужной строки в таблице некластерный индекс организует специальные указатели, включающие в себя:

- информацию об идентификационном номере файла, в котором хранится строка;
- идентификационный номер страницы соответствующих данных;
- номер искомой строки на соответствующей странице;
- содержимое столбца.

В большинстве случаев следует ограничиваться 4-5 индексами.

Кластерный индекс.

Принципиальным отличием кластерного индекса от индексов других типов является то, что при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Логическая структура таблицы в этом случае представляет собой скорее словарь, чем индекс. Данные в словаре физически упорядочены, например по алфавиту.

Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными индексами. Увеличение производительности особенно заметно при работе с последовательными данными. Если в таблице определен некластерный индекс, то сервер должен сначала обратиться к индексу, а затем найти нужную строку в таблице. При использовании кластерных индексов следующая порция данных располагается сразу после найденных ранее данных. Благодаря этому отпадают лишние операции, связанные с обращением к индексу и новым поиском нужной строки в таблице.

Естественно, в таблице может быть определен только один кластерный индекс. В качестве такового следует выбирать наиболее часто используемые столбцы. При этом стоит следовать общим рекомендациям создания индексов и не индексировать слишком длинные столбцы.

Кластерный индекс может включать несколько столбцов. Однако количество таких столбцов рекомендуется по возможности свести к минимуму.

Необходимо избегать создания кластерного индекса для часто изменяемых столбцов, поскольку сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии, как того требует кластерный индекс. Для интенсивно изменяемых столбцов лучше подходит некластерный индекс.

При создании в таблице первичного ключа (PRIMARY KEY) сервер автоматически создает для него кластерный индекс, если его не существовало ранее или если при определении ключа не был явно указан другой тип индекса.

Когда же в таблице определен еще и некластерный индекс, то его указатель ссылается не на физическое положение строки в базе данных, а на соответствующий элемент кластерного индекса, описывающего эту строку, что позволяет не перестраивать структуру некластерных индексов всякий раз, когда кластерный индекс меняет физический порядок строк в таблице.

Уникальный индекс.

Уникальность значений в индексируемом столбце гарантируют уникальные индексы. При их наличии сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в столбце появились два одинаковых значения.

Уникальный индекс является своеобразной надстройкой и может быть реализован как для кластерного, так и для некластерного индекса. В одной таблице может существовать один уникальный кластерный и множество уникальных некластерных индексов.

Уникальные индексы следует определять только тогда, когда это действительно необходимо. Для обеспечения целостности данных в столбце можно определить ограничение целостности UNIQUE или PRIMARY KEY, а не прибегать к уникальным индексам. Их использование только для обеспечения целостности данных является неоправданной тратой пространства в базе данных. Кроме того, на их поддержание тратится и процессорное время.

Средства языка SQL предлагают несколько способов определения индекса:

- автоматическое создание индекса при создании первичного ключа;
- автоматическое создание индекса при определении ограничения целостности UNIQUE;
- создание индекса с помощью команды CREATE INDEX.

Последняя команда имеет следующий формат:

```
<создание_индекса>::=
  CREATE [ UNIQUE ]
    [ CLUSTERED | NONCLUSTERED ]
  INDEX имя_индекса ON имя_таблицы(имя_столбца
    [ASC|DESC][,...n])
  [WITH [PAD_INDEX]
    [[,] FILLFACTOR=фактор_заполнения]
    [[,] IGNORE_DUP_KEY]
    [[,] DROP_EXISTING]
    [[,] STATISTICS_NORECOMPUTE] ]
    [ON имя_группы_файлов ]
```

Рассмотрим некоторые параметры приведенной команды.

Имя индекса должно быть уникальным в пределах таблицы, а сам индекс создается исключительно для таблицы текущей базы данных.

Параметр UNIQUE используется при необходимости ввода в определенное поле только уникальных значений. При указании этого ключевого слова будет создан уникальный индекс. В индексируемом столбце желательно запретить хранение значений NULL, чтобы избежать проблем, связанных с уникальностью значений. После того как для столбца появится уникальный индекс, сервер не разрешит выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующих значений.

Параметр CLUSTERED использует возможность физического индексирования данных и позволяет произвести так называемое кластерное индексирование, в результате чего будут отсортированы данные в

самой таблице согласно порядку этого индекса, а вся добавляемая информация станет приводить к изменению физического порядка данных. Кластерным может быть только один индекс в таблице.

Параметр NONCLASTERED позволяет создавать некластерные индексы.

Параметр FILLFACTOR осуществляет настройку разбиения индекса на страницы и заметно оптимизирует работу SQL-сервера. Коэффициент FILLFACTOR определяет в процентном соотношении размер создаваемых индексных страниц. При этом имеется обратно пропорциональная зависимость частоты работы с таблицей и коэффициента FILLFACTOR.

Параметр PAD_INDEX определяет заполнение внутреннего пространства индекса и применяется совместно с FILLFACTOR.

Параметр DROP_EXISTING при использовании кластерного индекса определяет его повторное создание, что позволяет предотвратить нежелательное обновление кластерных индексов.

Параметр STATISTICS_NORECOMPUTE определяет функции автоматического обновления статистики для таблицы.

Параметр имя_группы_файлов позволяет осуществить выбор файловой группы, в которой будет находиться создаваемый индекс. Использование индекса из другой файловой группы повышает производительность некластерных индексов в связи с параллельностью выполнения процессов ввода/вывода и работы с самим индексом.

Удаление индекса.

Удаление индекса выполняется командой

```
DROP INDEX 'имя_индекса'[,...n]
```

Пример. Создать уникальный кластерный индекс для таблицы Клиент по столбцу Фамилия в первичной группе файлов.

```
CREATE UNIQUE CLUSTERED INDEX index_klient1
  ON Клиент (Фамилия)
  WITH DROP_EXISTING
  ON PRIMARY
```

Пример. Создать уникальный некластерный индекс для таблицы Клиент по столбцам Фамилия и Имя в первичной группе файлов. Кроме того, элементы индекса будут упорядочены по убыванию. Также запретим автоматическое обновление статистики при изменении данных в таблице и установим фактор заполнения индексных страниц на уровне 30%.

```
CREATE UNIQUE NONCLUSTERED INDEX index_klient2
  ON Клиент (Фамилия DESC,Имя DESC)
  WITH FILLFACTOR=30,
```

```
STATISTICS_NORECOMPUTE
ON PRIMARY
```

Вопрос 9. Создание ограничений.

Таблицы с ограничениями в стандарте языка.

При создании баз данных большое внимание должно быть уделено средствам поддержания данных в целостном состоянии. Рассмотрим предусмотренные стандартом языка SQL функции, которые предназначены для поддержания целостности данных. Эта поддержка включает средства задания ограничений, они вводятся с целью защиты базы данных от нарушения согласованности сохраняемых в ней данных. К таким типам поддержки целостности данных относятся:

- обязательные данные;
- ограничения для доменов полей;
- целостность сущностей;
- ссылочная целостность;
- требования конкретного предприятия.

Большая часть перечисленных ограничений задается в операторах CREATE TABLE и ALTER TABLE.

Создание таблицы.

В стандарте SQL дано несколько вариантов определения оператора создания таблицы, однако его базовый формат имеет следующий вид:

```
<определение_таблицы> ::=
CREATE TABLE имя_таблицы
{(имя_столбца тип_данных [ NOT NULL ][ UNIQUE]
[DEFAULT <значение>]
[ CHECK (<условие_выбора>)][,...n]}
[CONSTRAINT имя_ограничения]
[PRIMARY KEY (имя_столбца [,...n])
{[UNIQUE (имя_столбца [,...n])}
[FOREIGN KEY (имя_столбца_внешнего_ключа
   [,...n])
REFERENCES имя_род_таблицы
   [(имя_столбца_род_таблицы [,...n])],
[MATCH {PARTIAL | FULL}
[ON UPDATE {CASCADE| SET NULL |SET DEFAULT
   |NO ACTION}]
[ON DELETE {CASCADE| SET NULL |SET DEFAULT
   |NO ACTION}]
{[CHECK(<условие_выбора>)][,...n]})
```

Ограничения.

Представленная версия оператора создания таблицы включает средства определения требований целостности данных, а также другие конструкции.

Имеются очень большие вариации в наборе функциональных возможностей этого оператора, реализованных в различных диалектах языка SQL. Рассмотрим назначение параметров команды, используемых для поддержания целостности данных.

Обязательные данные.

Для некоторых столбцов требуется наличие в каждой строке таблицы конкретного и допустимого значения, отличного от опущенного значения или значения NULL. Для заданий ограничений подобного типа стандарт SQL предусматривает использование спецификации NOT NULL.

Требования конкретного предприятия.

Обновления данных в таблицах могут быть ограничены существующими в организации требованиями (бизнес-правилами). Стандарт SQL позволяет реализовать бизнес-правила предприятий с помощью предложения CHECK и ключевого слова UNIQUE.

Ограничения для доменов полей.

Каждый столбец имеет собственный домен - некоторый набор допустимых значений. Стандарт SQL предусматривает два различных механизма определения доменов. Первый состоит в использовании предложения CHECK, позволяющего задать требуемые ограничения для столбца или таблицы в целом, а второй предполагает применение оператора CREATE DOMAIN.

Целостность сущностей.

Первичный ключ таблицы должен иметь уникальное непустое значение в каждой строке. Стандарт SQL позволяет задавать подобные требования поддержки целостности данных с помощью фразы PRIMARY KEY. В пределах таблицы она может указываться только один раз. Однако существует возможность гарантировать уникальность значений и для любых альтернативных ключей таблицы, что обеспечивает ключевое слово UNIQUE. Кроме того, при определении альтернативных ключей рекомендуется использовать и спецификаторы NOT NULL.

Ссылочная целостность.

Внешние ключи представляют собой столбцы или наборы столбцов, предназначенные для связывания каждой из строк дочерней таблицы, содержащей этот внешний ключ, со строкой родительской таблицы, содержащей соответствующее значение потенциального ключа. Стандарт SQL предусматривает механизм определения внешних ключей с помощью предложения FOREIGN KEY, а фраза REFERENCES определяет имя родительской таблицы, т.е. таблицы, где находится соответствующий потенциальный ключ. При использовании этого предложения система отклонит выполнение любых операторов INSERT или UPDATE, с помощью которых будет предпринята попытка создать в дочерней таблице значение внешнего ключа, не соответствующее одному из уже существующих значений потенциального ключа родительской таблицы. Когда действия системы выполняются при поступлении операторов UPDATE и DELETE, содержащих попытку обновить или удалить

значение потенциального ключа в родительской таблице, которому соответствует одна или более строк дочерней таблицы, то они зависят от правил поддержки ссылочной целостности, указанных во фразах ON UPDATE и ON DELETE предложения FOREIGN KEY. Если пользователь предпринимает попытку удалить из родительской таблицы строку, на которую ссылается одна или более строк дочерней таблицы, язык SQL предоставляет следующие возможности:

CASCADE - выполняется удаление строки из родительской таблицы, сопровождающееся автоматическим удалением всех ссылающихся на нее строк дочерней таблицы;

SET NULL - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы записывается значение NULL;

SET DEFAULT - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию;

NO ACTION - операция удаления строки из родительской таблицы отменяется. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа фраза ON DELETE опущена.

Те же самые правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется.

Определитель MATCH позволяет уточнить способ обработки значения NULL во внешнем ключе.

При определении таблицы предложение FOREIGN KEY может указываться произвольное количество раз.

В операторе CREATE TABLE используется необязательная фраза DEFAULT, которая предназначена для задания принимаемого по умолчанию значения, когда в операторе INSERT значение в данном столбце будет отсутствовать.

Фраза CONSTRAINT позволяет задать имя ограничению, что позволит впоследствии отменить то или иное ограничение с помощью оператора ALTER TABLE.

Изменение и удаление таблицы.

Для внесения изменений в уже созданные таблицы стандартом SQL предусмотрен оператор ALTER TABLE, предназначенный для выполнения следующих действий:

- добавление в таблицу нового столбца;
- удаление столбца из таблицы;
- добавление в определение таблицы нового ограничения;
- удаление из определения таблицы существующего ограничения;
- задание для столбца значения по умолчанию;
- отмена для столбца значения по умолчанию.

Оператор изменения таблицы имеет следующий обобщенный формат:

```
<изменение_таблицы> ::=
ALTER TABLE имя_таблицы
[ADD [COLUMN]имя_столбца тип_данных
   [ NOT NULL ][UNIQUE]
[DEFAULT <значение>][ CHECK (<условие_выбора>)]]
[DROP [COLUMN] имя_столбца [RESTRICT | CASCADE ]]
[ADD [CONSTRAINT [имя_ограничения]]
[{PRIMARY KEY (имя_столбца [,...n])
   |[UNIQUE (имя_столбца [,...n])}
|[FOREIGN KEY (имя_столбца_внешнего_ключа [,...n])
   REFERENCES имя_род_таблицы
     [(имя_столбца_род_таблицы [,...n])],
[ MATCH {PARTIAL | FULL}
   [ON UPDATE {CASCADE| SET NULL |
      SET DEFAULT | NO ACTION}]
   [ON DELETE {CASCADE| SET NULL |
      SET DEFAULT | NO ACTION}]
   |[CHECK(<условие_выбора>)][,...n]}]
[DROP CONSTRAINT имя_ограничения
    [RESTRICT | CASCADE]]
[ALTER [COLUMN] SET DEFAULT <значение>]
[ALTER [COLUMN] DROP DEFAULT]
```

Здесь параметры имеют то же самое назначение, что и в определении оператора CREATE TABLE.

Оператор ALTER TABLE реализован не во всех диалектах языка SQL. В некоторых диалектах он поддерживается, однако не позволяет удалять из таблицы уже существующие столбцы.

Для удаления таблицы используется команда DROP TABLE.

Таблицы в среде MS SQL Server.

Создание таблицы.

В процессе проектирования таблиц принимается решение о том, какие таблицы должны входить в базу данных, какие у них будут имена (идентификаторы), какие типы данных потребуются для построения таблиц и какие пользователи получат доступ к каждой из них. Кроме того, для эффективного создания таблиц необходимо ответить на следующие вопросы:

Столбцы какого типа и размера будут составлять каждую из таблиц, какие требуется выбрать имена для столбцов таблиц?

Какие столбцы могут содержать значение NULL?

Будут ли использованы ограничения целостности, значения по умолчанию и правила для столбцов?

Необходимо ли индексирование столбцов, какие типы индексов будут применены для конкретных столбцов?

Какие столбцы будут входить в первичные и внешние ключи.

Для создания таблиц в среде MS SQL Server используется команда:

```
<определение_таблицы> ::=
CREATE TABLE [ имя_базы_данных.[владелец].
```

```
| владелец. ]имя_таблицы
(<элемент_таблицы>[,...n])
```

где

```
<элемент_таблицы> ::=
{<определение_столбца>}
| <имя_столбца> AS <выражение>
|>ограничение_таблицы<
```

Обычно владельцем таблицы (dbo) является тот, кто ее создал.

<Выражение> задает значение для вычисляемого столбца. Вычисляемые столбцы - это виртуальные столбцы, т. е. физически в таблице они не хранятся и вычисляются с использованием значений столбцов той же таблицы. В выражении для вычисляемого столбца могут присутствовать имена обычных столбцов, константы и функции, связанные одним или несколькими операторами. Подзапросы в таком выражении участвовать не могут. Вычисляемые столбцы могут быть включены в раздел SELECT при указании списка столбцов, которые должны быть возвращены в результате выполнения запроса. Вычисляемые столбцы не могут входить во внешний ключ, для них не используются значения по умолчанию. Кроме того, вычисляемые столбцы не могут участвовать в операциях INSERT и DELETE.

```
<определение_столбца> ::=
{ имя_столбца <тип_данных>}
[ [ DEFAULT <выражение> ]
| [ IDENTITY (начало, шаг) [NOT FOR REPLICATION]]]]
[ROWGUIDCOL][<ограничение_столбца>][...n]]
```

В определении столбца обратим внимание на параметр IDENTITY, который указывает, что соответствующий столбец будет столбцом-счетчиком. Для таблицы может быть определен только один столбец с таким свойством. Можно дополнительно указать начальное значение и шаг приращения. Если эти значения не указываются, то по умолчанию они оба равны 1. Если с ключевым словом IDENTITY указано NOT FOR REPLICATION, то сервер не будет выполнять автоматического генерирования значений для этого столбца, а разрешит вставку в столбец произвольных значений.

В качестве ограничений используются ограничения столбца и ограничения таблицы. Различие между ними в том, что ограничение столбца применяется только к определенному полю, а ограничение таблицы - к группам из одного или более полей.

```
<ограничение_столбца>::=
[ CONSTRAINT имя_ограничения ]
{ [ NULL | NOT NULL ]
```

```
| [ {PRIMARY KEY | UNIQUE }
[ CLUSTERED | NONCLUSTERED ]
[ WITH FILLFACTOR=фактор_заполнения ]
[ ON {имя_группы_файлов | DEFAULT } ] ] ]
| [ [ FOREIGN KEY ]
REFERENCES имя_род_таблицы
   [(имя_столбца_род_таблицы) ]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
[ NOT FOR REPLICATION ]]
| CHECK [ NOT FOR REPLICATION](<лог_выражение>) }

<ограничение_таблицы>::=
[CONSTRAINT имя_ограничения ]
{ [ {PRIMARY KEY | UNIQUE }
   [ CLUSTERED | NONCLUSTERED ]
{(имя_столбца [ASC | DESC][,...n])}
[WITH FILLFACTOR=фактор_заполнения ]
[ON {имя_группы_файлов | DEFAULT } ]]
|FOREIGN KEY[(имя_столбца [,...n])]
REFERENCES имя_род_таблицы
   [(имя_столбца_род_таблицы [,...n])]
[ ON DELETE { CASCADE | NO ACTION } ]
[ ON UPDATE { CASCADE | NO ACTION } ]
| NOT  FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ] (лог_выражение) }
```

Рассмотрим отдельные параметры представленных конструкций, связанные с ограничениями целостности данных. Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся ограничение первичного ключа PRIMARY KEY, ограничение внешнего ключа FOREIGN KEY, ограничение уникальности UNIQUE, ограничение значения NULL, ограничение на проверку CHECK.

Ограничение первичного ключа (PRIMARY KEY).

Таблица обычно имеет столбец или комбинацию столбцов, значения которых уникально идентифицируют каждую строку в таблице. Этот столбец (или столбцы) называется первичным ключом таблицы и нужен для обеспечения ее целостности. Если в первичный ключ входит более одного столбца, то значения в пределах одного столбца могут дублироваться, но любая комбинация значений всех столбцов первичного ключа должна быть уникальна.

При создании первичного ключа SQL Server автоматически создает уникальный индекс для столбцов, входящих в первичный ключ. Он ускоряет доступ к данным этих столбцов при использовании первичного ключа в запросах.

Таблица может иметь только одно ограничение PRIMARY KEY, причем ни один из включенных в первичный ключ столбцов не может

принимать значение NULL. При попытке использовать в качестве первичного ключа столбец (или группу столбцов), для которого ограничения первичного ключа не выполняются, первичный ключ создан не будет, а система выдаст сообщение об ошибке.

Поскольку ограничение PRIMARY KEY гарантирует уникальность данных, оно часто определяется для столбцов-счетчиков. Создание ограничения целостности PRIMARY KEY возможно как при создании, так и при изменении таблицы. Одним из назначений первичного ключа является обеспечение ссылочной целостности данных нескольких таблиц. Естественно, это может быть реализовано только при определении соответствующих внешних ключей в других таблицах.

Ограничение внешнего ключа (FOREIGN KEY).

Ограничение внешнего ключа - это основной механизм для поддержания ссылочной целостности между таблицами реляционной базы данных. Столбец дочерней таблицы, определенный в качестве внешнего ключа в параметре FOREIGN KEY, применяется для ссылки на столбец родительской таблицы, являющийся в ней первичным ключом. Имя родительской таблицы и столбцы ее первичного ключа указываются в предложении REFERENCES. Данные в столбцах, определенных в качестве внешнего ключа, могут принимать только такие же значения, какие находятся в связанных с ним столбцах первичного ключа родительской таблицы. Совпадение имен столбцов для связи дочерней и родительской таблиц необязательно. Первичный ключ может быть определен для столбца с одним именем, в то время как столбец, на который наложено ограничение FOREIGN KEY, может иметь совершенно другое имя. Единственным требованием остается соответствие столбцов по типу и размеру данных.

На первичный ключ могут ссылаться не только столбцы других таблиц, но и столбцы, расположенные в той же таблице, что и собственно первичный ключ; это позволяет создавать рекурсивные структуры.

Внешний ключ может быть связан не только с первичным ключом другой таблицы. Он может быть определен и для столбцов с ограничением UNIQUE второй таблицы или любых других столбцов, но таблицы должны находиться в одной базе данных.

Столбцы внешнего ключа могут содержать значение NULL, однако проверка на ограничение FOREIGN KEY игнорируется. Внешний ключ может быть проиндексирован, тогда сервер будет быстрее отыскивать нужные данные. Внешний ключ определяется как при создании, так и при изменении таблиц.

Ограничение ссылочной целостности задает требование, согласно которому для каждой записи в дочерней таблице должна иметься запись в родительской таблице. При этом изменение значения столбца связи в записи родительской таблицы при наличии дочерней записи блокируется, равно как и удаление родительской записи (запрет каскадного изменения и удаления), что гарантируется параметрами ON DELETE NO ACTION и ON UPDATE NO

ACTION, принятыми по умолчанию. Для разрешения каскадного воздействия следует использовать параметры ON DELETE CASCADE и ON UPDATE CASCADE.

Ограничение уникального ключа (UNIQUE).

Это ограничение задает требование уникальности значения поля (столбца) или группы полей (столбцов), входящих в уникальный ключ, по отношению к другим записям. Ограничение UNIQUE для столбца таблицы похоже на первичный ключ: для каждой строки данных в нем должны содержаться уникальные значения. Установив для некоторого столбца ограничение первичного ключа, можно одновременно установить для другого столбца ограничение UNIQUE. Отличие в ограничении первичного и уникального ключа заключается в том, что первичный ключ служит как для упорядочения данных в таблице, так и для соединения связанных между собой таблиц. Кроме того, при использовании ограничения UNIQUE допускается существование значения NULL, но лишь единственный раз.

Ограничение на значение (NOT NULL).

Для каждого столбца таблицы можно установить ограничение NOT NULL, запрещающее ввод в этот столбец нулевого значения.

Ограничение проверочное (CHECK) и правила.

Данное ограничение используется для проверки допустимости данных, вводимых в конкретный столбец таблицы, т.е. ограничение CHECK обеспечивает еще один уровень защиты данных.

Ограничения целостности CHECK задают диапазон возможных значений для столбца или столбцов. В основе ограничений целостности CHECK лежит использование логических выражений.

Допускается применение нескольких ограничений CHECK к одному и тому же столбцу. В этом случае они будут применимы в той последовательности, в которой происходило их создание. Возможно применение одного и того же ограничения к разным столбцам и использование в логических выражениях значений других столбцов. Указание параметра NOT FOR REPLICATION предписывает не выполнять проверочных действий, если они выполняются подсистемой репликации.

Проверочные ограничения могут быть реализованы и с помощью правил. Правило представляет собой самостоятельный объект базы данных, который связывается со столбцом таблицы или пользовательским типом данных. Причем одно и то же правило может быть одновременно связано с несколькими столбцами и пользовательскими типами данных, что является неоспоримым преимуществом. Однако существенный недостаток заключается в том, что с каждым столбцом или типом данных может быть связано только одно правило. Разрешается комбинирование ограничений целостности CHECK с правилами. В этом случае выполняется проверка соответствия вводимого значения как ограничениям целостности, так и правилам.

Правило может быть создано командой:

```
CREATE RULE имя_правила AS выражение
```

Чтобы связать правило с тем или иным столбцом какой-либо таблицы, необходимо использовать системную хранимую процедуру:

```
sp_bindrule [@rulename=] 'rule'
[@objname=] 'object_name'
[,[@futureonly=['futureonly_flag']
```

Чтобы отменить правила, следует выполнить следующую процедуру:

```
sp_unbindrule [@objname=] 'object_name'
[,[@futureonly=['futureonly_flag']
```

Удаление правила производится командой

```
DROP RULE {имя_правила} [,...n].
```

Ограничение по умолчанию (DEFAULT).

Столбцу может быть присвоено значение по умолчанию. Оно будет актуальным в том случае, если пользователь не введет в столбец никакого иного значения.

Отдельно необходимо отметить пользу от использования значений по умолчанию при добавлении нового столбца в таблицу. Если для добавляемого столбца не разрешено хранение значений NULL и не определено значение по умолчанию, то операция добавления столбца закончится неудачей.

При определении в таблице столбца с параметром ROWGUIDCOL сервер автоматически определяет для него значение по умолчанию в виде функции NEWID(). Таким образом, для каждой новой строки будет автоматически генерироваться глобальный уникальный идентификатор.

Дополнительным механизмом использования значений по умолчанию являются объекты базы данных, созданные командой:

```
CREATE DEFAULT имя_умолчания AS константа
```

Умолчание связывается с тем или иным столбцом какой-либо таблицы с помощью процедуры:

```
sp_bindefault [@defname=] 'default',
[@objname=]  'object_name'
[,[@futureonly=] 'futureonly_flag'],
```

где

'object_name'

может быть представлен как

'имя_таблицы.имя_столбца'

Удаление ограничения по умолчанию выполняется командой

DROP DEFAULT {имя_умолчания} [,...n]

если предварительно это ограничение было удалено из всех таблиц процедурой

sp_unbindefault [@objname=] 'object_name'
[,[@futureonly=] 'futureonly_flag']

При создании таблицы, кроме рассмотренных приемов, можно указать необязательное ключевое слово CONSTRAINT, чтобы присвоить ограничению имя, уникальное в пределах базы данных.

Ключевые слова CLUSTERED и NONCLUSTERED позволяют создать для столбца кластерный или некластерный индекс. Для ограничения PRIMARY KEY по умолчанию создается кластерный индекс, а для ограничения UNIQUE - некластерный. В каждой таблице может быть создан лишь один кластерный индекс, отличительной особенностью которого является то, что в соответствии с ним изменяется физический порядок строк в таблице. ASC и DESC определяют метод упорядочения данных в индексе.

С помощью параметра WITH FILLFACTOR=фактор_заполнения задается степень заполнения индексных страниц при создании индекса. Значение фактора заполнения указывается в процентах и может изменяться в промежутке от 0 до 100.

Параметр ON имя_группы_файлов обозначает группу, в которой предполагается хранить таблицу.

Изменение таблицы.

Изменения в таблицу можно внести командой:

<изменение_таблицы> ::=
ALTER TABLE имя_таблицы
{[ALTER COLUMN имя_столбца
{ тип_данных [(точность[,масштаб])]
[NULL | NOT NULL]
| {ADD | DROP } ROWGUIDCOL }]
| ADD { [<определение_столбца>]
| имя_столбца AS выражение } [,...n]
| [WITH CHECK | WITH NOCHECK]
ADD { <ограничение-таблицы> } [,...n]

```
| DROP
{ [CONSTRAINT ] имя_ограничения
| COLUMN имя_столбца}[,...n]
| {CHECK | NOCHECK } CONSTRAINT
{ALL | имя_ограничения[,...n]}
| {ENABLE | DISABLE } TRIGGER
{ALL | имя_триггера [,...n]}}
```

В дополнение к уже названным параметрам определим параметр {ENABLE | DISABLE } TRIGGER ALL_, предписывающий задействовать или отключить конкретный триггер или все триггера, связанные с таблицей.

Удаление таблицы.

Удаление таблицы выполняется командой:

```
DROP TABLE имя_таблицы
```

Удалить можно любую таблицу, даже системную. К этому вопросу нужно подходить очень осторожно. Однако удалению не подлежат таблицы, если существуют объекты, ссылающиеся на них. К таким объектам относятся таблицы, связанные с удаляемой таблицей посредством внешнего ключа. Поэтому, прежде чем удалять родительскую таблицу, необходимо удалить либо ограничение внешнего ключа, либо дочерние таблицы. Если с таблицей связано хотя бы одно представление, то таблицу также удалить не удастся. Кроме того, связь с таблицей может быть установлена со стороны функций и процедур. Следовательно, перед удалением таблицы необходимо удалить все объекты базы данных, которые на нее ссылаются, либо изменить их таким образом, чтобы ссылок на удаляемую таблицу не было.

Пример. Создание родительской таблицы Товар с ограничениями.

```
CREATE TABLE Товар
(КодТовара INT IDENTITY(1,1) PRIMARY KEY,
 Название  VARCHAR(50)      NOT NULL UNIQUE,
 Цена      MONEY            NOT NULL,
 Тип       VARCHAR(50)      NOT NULL,
 Сорт      VARCHAR(50)      NOT NULL
   CHECK(сорт in('первый','второй','третий')),
 Город     VARCHAR(50)      NOT NULL,
 Остаток   INT
   CHECK(остаток>=0))
```

Пример. Создание родительской таблицы Клиент с ограничениями.

```
CREATE TABLE  Клиент
(КодКлиента INT IDENTITY(1,1) PRIMARY KEY,
 Фирма         VARCHAR(50)      NOT NULL,
 Фамилия    VARCHAR(50)      NOT NULL,
 Город      VARCHAR(50)      NOT NULL,
```

```
 Телефон    CHAR(10)         NOT NULL
   CHECK(Телефон LIKE
     '[1-9][0-9]-[0-9][0-9]-[0-9][0-9]'))
```

Пример. Создание дочерней таблицы Сделка с ограничениями.

```
CREATE TABLE Сделка
(КодСделки  INT IDENTITY(1,1) PRIMARY KEY,
 КодТовара  INT       NOT NULL,
 КодКлиента INT       NOT NULL,
 Количество INT       NOT NULL DEFAULT 0,
 Дата       DATETIME  NOT NULL DEFAULT
                GETDATE(),
CONSTRAINT  fk_Товар
FOREIGN KEY(КодТовара)  REFERENCES Товар,
CONSTRAINT  fk_Клиент
FOREIGN KEY(КодКлиента) REFERENCES Клиент)
```

Пример. Создание таблицы Склад.

```
CREATE TABLE Склад
(КодТовара     INT PRIMARY KEY,
 Остаток       INT)
```

Пример. Удаление ограничения внешнего ключа.

```
ALTER TABLE Сделка DROP CONSTRAINT fk_Товар
```

Пример. Добавление ограничения внешнего ключа, реализующего декларативную ссылочную целостность.

```
ALTER TABLE Сделка ADD CONSTRAINT fk_Товар
FOREIGN KEY (КодТовара) REFERENCES товар
ON UPDATE NO ACTION ON DELETE NO ACTION
```

Пример. Добавления ограничения внешнего ключа, реализующего каскадные обновления и изменения.

```
ALTER TABLE Сделка ADD CONSTRAINT fk_Товар
FOREIGN KEY (КодТовара) REFERENCES Товар
ON UPDATE CASCADE ON DELETE CASCADE
```

Пример. Пример создания и удаления вычисляемого поля.

```
ALTER TABLE Товар ADD Налог AS Цена*0.05
ALTER TABLE Товар DROP COLUMN Налог
```

Пусть создана таблица без ограничений:

```
CREATE TABLE Товар
(КодТовара     INT,
 Название      VARCHAR(20),
 Тип           VARCHAR(20),
 Дата          DATETIME,
 Цена          MONEY,
 Остаток       INT)
```

Рассмотрим примеры внесения в таблицу всевозможных ограничений.

Пример. Поле КодТовара необходимо сделать первичным ключом. Выполнение следующей команды будет отвергнуто, поскольку поле КодТовара допускает внесение значений NULL.

```
ALTER TABLE Товар ADD CONSTRAINT pk1
PRIMARY KEY(КодТовара)
```

Сначала нужно изменить объявление столбца КодТовара, запретив внесение значений NULL:

```
ALTER TABLE Товар
ALTER COLUMN КодТовара INT NOT NULL
```

И только потом создать ограничение первичного ключа:

```
ALTER TABLE Товар ADD CONSTRAINT pk1
PRIMARY KEY(КодТовара)
```

Пример. Удалить столбец целого типа и добавить столбец-счетчик.

```
ALTER TABLE Товар DROP  COLUMN КодТовара
ALTER TABLE Товар ADD
   КодТовара INT IDENTITY(1,1)
```

Пример. Добавить ограничение первичного ключа.

```
ALTER TABLE Товар ADD CONSTRAINT pk1
PRIMARY KEY(КодТовара)
```

Пример. Изменить столбец, добавив ограничение NOT NULL.

```
ALTER TABLE Товар ALTER COLUMN
   Название VARCHAR(40) NOT NULL
```

Пример. Добавить ограничение уникальности значения.

```
ALTER TABLE Товар ADD CONSTRAINT
   u1 UNIQUE(Название)
```

Пример. Создать умолчание и добавить умолчание столбцу.

```
CREATE DEFAULT df1 AS 0
sp_bindefault 'df1', 'Товар.Остаток'

CREATE DEFAULT df2 AS GETDATE()
sp_bindefault 'df2', 'Товар.Дата'
```

Пример. Создать правило и добавить правило столбцу.

```
CREATE RULE r1 AS @m IN
   ('мебель','бытовая химия','косметика')
sp_bindrule 'r1','Товар.Тип'
```

Вопрос 10. Создание представлений.

Представления, или просмотры (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются. Представление - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. Результат выполнения этого запроса в каждый момент времени становится содержанием представления. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

У СУБД есть две возможности реализации представлений. Если его определение простое, то система формирует каждую запись представления по мере необходимости, постепенно считывая исходные данные из базовых таблиц. В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, т.е. сохранить информацию, из которой состоит представление, во временной таблице. Затем система приступает к выполнению пользовательской команды и формированию ее результатов, после чего временная таблица удаляется.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти. Для хранения представления используется только оперативная память. В отличие от других объектов базы данных представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.

Создания и изменения представлений в стандарте языка и реализации в MS SQL Server совпадают и представлены следующей командой:

```
<определение_просмотра> ::=
   { CREATE| ALTER} VIEW имя_просмотра
   [(имя_столбца [,...n])]
   [WITH ENCRYPTION]
   AS SELECT_оператор
   [WITH CHECK OPTION]
```

Рассмотрим назначение основных параметров.

По умолчанию имена столбцов в представлении соответствуют именам столбцов в исходных таблицах. Явное указание имени столбца требуется для вычисляемых столбцов или при объединении нескольких таблиц, имеющих столбцы с одинаковыми именами. Имена столбцов перечисляются через запятую, в соответствии с порядком их следования в представлении.

Параметр WITH ENCRYPTION предписывает серверу шифровать SQL-код запроса, что гарантирует невозможность его несанкционированного просмотра и использования. Если при определении представления необходимо скрыть имена исходных таблиц и столбцов, а также алгоритм объединения данных, необходимо применить этот аргумент.

Параметр WITH CHECK OPTION предписывает серверу исполнять проверку изменений, производимых через представление, на соответствие критериям, определенным в операторе SELECT. Это означает, что не допускается выполнение изменений, которые приведут к исчезновению строки из представления. Такое случается, если для представления установлен горизонтальный фильтр и изменение данных приводит к несоответствию строки установленным фильтрам. Использование аргумента WITH CHECK OPTION гарантирует, что сделанные изменения будут отображены в представлении. Если пользователь пытается выполнить изменения, приводящие к исключению строки из представления, при заданном аргументе WITH CHECK OPTION сервер выдаст сообщение об ошибке и все изменения будут отклонены.

Пример. Показать в представлении клиентов из Москвы.

Создание представления:

```
CREATE  VIEW view1 AS
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
```

```
WHERE ГородКлиента='Москва'
```

Выборка данных из представления:

```
SELECT * FROM view1
```

Обращение к представлению осуществляется с помощью оператора SELECT как к обычной таблице.

Представление можно использовать в команде так же, как и любую другую таблицу. К представлению можно строить запрос, модифицировать его (если оно отвечает определенным требованиям), соединять с другими таблицами. Содержание представления не фиксировано и обновляется каждый раз, когда на него ссылаются в команде. Представления значительно расширяют возможности управления данными. В частности, это прекрасный способ разрешить доступ к информации в таблице, скрыв часть данных.

Так, в примере представление просто ограничивает доступ пользователя к данным таблицы Клиент, позволяя видеть только часть значений.

Выполним команду:

```
INSERT INTO view1 VALUES (12,'Петров', 'Самара')
```

Это допустимая команда в представлении, и строка будет добавлена с помощью представления view1 в таблицу Клиент. Однако, когда информация будет добавлена, строка исчезнет из представления, поскольку название города отлично от Москвы. Иногда такой подход может стать проблемой, т.к. данные уже находятся в таблице, но пользователь их не видит и не в состоянии выполнить их удаление или модификацию. Для исключения подобных моментов служит WITH CHECK OPTION в определении представления. Фраза размещается в определении представления, и все команды модификации будут подвергаться проверке.

Пример. Создание представления с проверкой команд модификации.

```
ALTER VIEW view1
SELECT КодКлиента, Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва'
WITH CHECK OPTION
```

Для такого представления вышеупомянутая вставка значений будет отклонена системой.

Таким образом, представление может изменяться командами модификации DML, но фактически модификация воздействует не на само представление, а на базовую таблицу.

Представление удаляется командой:

```
DROP VIEW имя_просмотра [,...n]
```

Обновление данных в представлениях.

Не все представления в SQL могут быть модифицированы. Модифицируемое представление определяется следующими критериями:

- основывается только на одной базовой таблице;
- содержит первичный ключ этой таблицы;
- не содержит DISTINCT в своем определении;
- не использует GROUP BY или HAVING в своем определении;
- по возможности не применяет в своем определении подзапросы;
- не использует константы или выражения значений среди выбранных полей вывода;
- в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут NOT NULL;
- оператор SELECT просмотра не использует агрегирующие (итоговые) функции, соединения таблиц, хранимые процедуры и функции, определенные пользователем;
- основывается на одиночном запросе, поэтому объединение UNION не разрешено.

Если просмотр удовлетворяет этим условиям, к нему могут применяться операторы INSERT, UPDATE, DELETE. Различия между модифицируемыми представлениями и представлениями, предназначенными только для чтения, не случайны. Цели, для которых их используют, различны. С модифицируемыми представлениями в основном обходятся точно так же, как и с базовыми таблицами. Фактически, пользователи не могут даже осознать, является ли объект, который они запрашивают, базовой таблицей или представлением, т.е. прежде всего это средство защиты для сокрытия конфиденциальных или не относящихся к потребностям данного пользователя частей таблицы. Представления в режиме <только для чтения> позволяют получать и форматировать данные более рационально. Они создают целый арсенал сложных запросов, которые можно выполнить и повторить снова, сохраняя полученную информацию. Результаты этих запросов могут затем использоваться в других запросах, что позволит избежать сложных предикатов и снизить вероятность ошибочных действий.

Пример. Немодифицируемое представление с данными из разных таблиц.

```
CREATE VIEW view2 AS
SELECT Клиент.Фамилия, Клиент.Фирма,
    Сделка.Количество
FROM Клиент INNER JOIN Сделка
ON Клиент.КодКлиента=Сделка.КодКлиента
```

Пример. Немодифицируемое представление с группировкой и итоговыми функциями.

```
CREATE VIEW view3(Тип, Общ_остаток) AS
SELECT Тип, Sum(Остаток)
FROM Товар
GROUP BY Тип
```

Обычно в представлениях используются имена, полученные непосредственно из имен полей основной таблицы. Однако иногда необходимо дать столбцам новые имена, например, в случае итоговых функций или вычисляемых столбцов.

Пример. Модифицируемое представление с вычислениями.

```
CREATE VIEW view4(Код, Название,
   Тип, Цена, Налог) AS
SELECT КодТовара, Название,
  Тип, Цена, Цена*0.05 FROM Товар
```

Преимущества и недостатки представлений.

Механизм представления - мощное средство СУБД, позволяющее скрыть реальную структуру БД от некоторых пользователей за счет определения представлений. Любая реализация представления должна гарантировать, что состояние представляемого отношения точно соответствует состоянию данных, на которых определено это представление. Обычно вычисление представления производится каждый раз при его использовании. Когда представление создается, информация о нем записывается в каталог БД под собственным именем. Любые изменения в данных адекватно отобразятся в представлении - в этом его отличие от очень похожего на него запроса к БД. В то же время запрос представляет собой как бы <мгновенную фотографию> данных и при изменении последних запрос к БД необходимо повторить. Наличие представлений в БД необходимо для обеспечения логической независимости данных. Если система обеспечивает физическую независимость данных, то изменения в физической структуре БД не влияют на работу пользовательских программ. Логическая независимость подразумевает тот факт, что при изменении логической структуры данных влияние на пользовательские программы также не оказывается, а значит, система должна уметь решать проблемы, связанные с ростом и реструктуризацией БД. Очевидно, что с увеличением количества данных, хранимых в БД, возникает необходимость ее расширения за счет добавления новых атрибутов или отношений - это называется ростом БД. Реструктуризация данных подразумевает сохранение той же самой информации, но изменяется ее расположение, например, за счет перегруппировки атрибутов в отношениях. Предположим, некоторое отношение в силу каких-либо причин необходимо разделить на два.

Соединение полученных отношений в представлении воссоздает исходное отношение, а у пользователя складывается впечатление, что никакой реструктуризации не производилось. Помимо решения проблемы реструктуризации представление можно применять для просмотра одних и тех же данных разными пользователями и в различных вариантах. С помощью представлений пользователь имеет возможность ограничить объем данных для удобства работы. Наконец, механизм представлений позволяет скрыть служебные данные, не интересные пользователям.

В случае выполнения СУБД на отдельно стоящем персональном компьютере использование представлений обычно имеет целью лишь упрощение структуры запросов к базе данных. Однако в случае многопользовательской сетевой СУБД представления играют ключевую роль в определении структуры базы данных и организации защиты информации. Рассмотрим основные преимущества применения представлений в подобной среде.

Независимость от данных.

С помощью представлений можно создать согласованную, неизменную картину структуры базы данных, которая будет оставаться стабильной даже в случае изменения формата исходных таблиц (например, добавления или удаления столбцов, изменения связей, разделения таблиц, их реструктуризации или переименования). Если в таблицу добавляются или из нее удаляются не используемые в представлении столбцы, то изменять определение этого представления не потребуется. Если структура исходной таблицы переупорядочивается или таблица разделяется, можно создать представление, позволяющее работать с виртуальной таблицей прежнего формата. В случае разделения исходной таблицы, прежний формат может быть виртуально воссоздан с помощью представления, построенного на основе соединения вновь созданных таблиц - конечно, если это окажется возможным. Последнее условие можно обеспечить с помощью помещения во все вновь созданные таблицы первичного ключа прежней таблицы.

Актуальность.

Изменения данных в любой из таблиц базы данных, указанных в определяющем запросе, немедленно отображается на содержимом представления.

Повышение защищенности данных.

Права доступа к данным могут быть предоставлены исключительно через ограниченный набор представлений, содержащих только то подмножество данных, которое необходимо пользователю. Подобный подход позволяет существенно ужесточить контроль за доступом отдельных категорий пользователей к информации в базе данных.

Снижение стоимости.

Представления позволяют упростить структуру запросов за счет объединения данных из нескольких таблиц в единственную виртуальную таблицу. В результате многотабличные запросы сводятся к простым запросам к одному представлению.

Дополнительные удобства.

Создание представлений может обеспечивать пользователей дополнительными удобствами - например, возможностью работы только с действительно нужной частью данных. В результате можно добиться максимального упрощения той модели данных, которая понадобится каждому конечному пользователю.

Возможность настройки.

Представления являются удобным средством настройки индивидуального образа базы данных. В результате одни и те же таблицы могут быть предъявлены пользователям в совершенно разном виде.

Обеспечение целостности данных.

Если в операторе CREATE VIEW будет указана фраза WITH CHECK OPTION, то СУБД станет осуществлять контроль за тем, чтобы в исходные таблицы базы данных не была введена ни одна из строк, не удовлетворяющих предложению WHERE в определяющем запросе. Этот механизм гарантирует целостность данных в представлении.

Практика ограничения доступа некоторых пользователей к данным посредством создания специализированных представлений, безусловно, имеет значительные преимущества перед предоставлением им прямого доступа к таблицам базы данных.

Однако использование представлений в среде SQL не лишено недостатков.

Ограниченные возможности обновления.

В некоторых случаях представления не позволяют вносить изменения в содержащиеся в них данные.

Структурные ограничения.

Структура представления устанавливается в момент его создания. Если определяющий запрос представлен в форме SELECT * FROM_, то символ * ссылается на все столбцы, существующие в исходной таблице на момент создания представления. Если впоследствии в исходную таблицу базы данных добавятся новые столбцы, то они не появятся в данном представлении до тех пор, пока это представление не будет удалено и вновь создано.

Снижение производительности.

Использование представлений связано с определенным снижением производительности. В одних случаях влияние этого фактора совершенно незначительно, тогда как в других оно может послужить источником существенных проблем. Например, представление, определенное с помощью сложного многотабличного запроса, может потребовать значительных затрат времени на обработку, поскольку при его разрешении потребуется выполнять соединение таблиц всякий раз, когда понадобится доступ к данному представлению. Выполнение разрешения представлений связано с использованием дополнительных вычислительных ресурсов.

Вопрос 11. Создание Функций.

Понятие функции пользователя.

При реализации на языке SQL сложных алгоритмов, которые могут потребоваться более одного раза, сразу встает вопрос о сохранении разработанного кода для дальнейшего применения. Эту задачу можно было бы реализовать с помощью хранимых процедур, однако их архитектура не позволяет использовать процедуры непосредственно в выражениях, т.к. они требуют промежуточного присвоения возвращенного значения переменной, которая затем и указывается в выражении. Естественно, подобный метод применения программного кода не слишком удобен. Многие разработчики уже давно хотели иметь возможность вызова разработанных алгоритмов непосредственно в выражениях.

Возможность создания пользовательских функций была предоставлена в среде MS SQL Server 2000. В других реализациях SQL в распоряжении пользователя имеются только встроенные функции, которые обеспечивают выполнение наиболее распространенных алгоритмов: поиск максимального или минимального значения и др.

Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как хранимые процедуры или триггеры. Функция пользователя располагается в определенной базе данных и доступна только в ее контексте.

В SQL Server имеются следующие классы функций пользователя:

Scalar – функции возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;

Inline – функции содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения типа данных TABLE;

Multi-statement – функции также возвращают пользователю значение типа данных TABLE, содержащее набор данных, однако в теле функции находится множество команд SQL (INSERT, UPDATE и т.д.). Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения функции.

Пользовательские функции сходны с хранимыми процедурами, но, в отличие от них, могут применяться в запросах так же, как и системные встроенные функции. Пользовательские функции, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а пользовательские функции способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Функции Scalar.

Создание и изменение функции данного типа выполняется с помощью команды:

<определение_скаляр_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]

```
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS скаляр_тип_данных
[WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN скаляр_выражение
END
```

Рассмотрим назначение параметров команды.

Функция может содержать один или несколько входных параметров либо не содержать ни одного. Каждый параметр должен иметь уникальное в пределах создаваемой функции имя и начинаться с символа «@». После имени указывается тип данных параметра. Дополнительно можно указать значение, которое будет автоматически присваиваться параметру (DEFAULT), если пользователь явно не указал значение соответствующего параметра при вызове функции.

С помощью конструкции RETURNS скаляр_тип_данных указывается, какой тип данных будет иметь возвращаемое функцией значение.

Дополнительные параметры, с которыми должна быть создана функция, могут быть указаны посредством ключевого слова WITH. Благодаря ключевому слову ENCRYPTION код команды, используемый для создания функции, будет зашифрован, и никто не сможет просмотреть его. Эта возможность позволяет скрыть логику работы функции. Кроме того, в теле функции может выполняться обращение к различным объектам базы данных, а потому изменение или удаление соответствующих объектов может привести к нарушению работы функции. Чтобы избежать этого, требуется запретить внесение изменений, указав при создании этой функции ключевое слово SCHEMABINDING.

Между ключевыми словами BEGIN...END указывается набор команд, они и будут являться телом функции.

Когда в ходе выполнения кода функции встречается ключевое слово RETURN, выполнение функции завершается и как результат ее вычисления возвращается значение, указанное непосредственно после слова RETURN. Отметим, что в теле функции разрешается использование множества команд RETURN, которые могут возвращать различные значения. В качестве возвращаемого значения допускаются как обычные константы, так и сложные выражения. Единственное условие – тип данных возвращаемого значения должен совпадать с типом данных, указанным после ключевого слова RETURNS.

Пример. Создать и применить функцию скалярного типа для вычисления суммарного количества товара, поступившего за определенную дату. Владелец функции – пользователь с именем user1.

```
CREATE FUNCTION
   user1.sales(@data DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @c INT
SET @c=(SELECT SUM(количество)
       FROM Сделка
       WHERE дата=@data)
RETURN (@c)
END
```

В качестве входного параметра используется дата. Функция возвращает значение целого типа, полученное из оператора SELECT путем суммирования количества товара из таблицы Сделка. Условием отбора записей для суммирования является равенство даты сделки значению входного параметра функции.

Проиллюстрируем обращение к функции пользователя: определим количество товара, поступившего за 02.11.01:

```
DECLARE @kol INT
SET @kol=user1.sales ('02.11.01')
SELECT @kol
```

Функции Inline.

Создание и изменение функции этого типа выполняется с помощью команды:

```
<определение_табл_функции>::=
{CREATE | ALTER } FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS TABLE
[ WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
RETURN [(] SELECT_оператор [)]
```

Основная часть параметров, используемых при создании табличных функций, аналогична параметрам скалярной функции. Тем не менее создание табличных функций имеет свою специфику.

После ключевого слова RETURNS всегда должно указываться ключевое слово TABLE. Таким образом, функция данного типа должна строго возвращать значение типа данных TABLE. Структура возвращаемого значения типа TABLE не указывается явно при описании собственно типа данных. Вместо этого сервер будет автоматически использовать для

возвращаемого значения TABLE структуру, возвращаемую запросом SELECT, который является единственной командой функции.

Особенность функции данного типа заключается в том, что структура значения TABLE создается автоматически в ходе выполнения запроса, а не указывается явно при определении типа после ключевого слова RETURNS.

Возвращаемое функцией значение типа TABLE может быть использовано непосредственно в запросе, т.е. в разделе FROM.

Пример. Создать и применить функцию табличного типа для определения двух наименований товара с наибольшим остатком.

```
CREATE FUNCTION user1.itog()
RETURNS TABLE
AS
RETURN (SELECT TOP 2 Товар.Название
  FROM Товар INNER JOIN Склад
  ON Товар.КодТовара=Склад.КодТовара
  ORDER BY Склад.Остаток DESC)
```

Использовать функцию для получения двух наименований товара с наибольшим остатком можно следующим образом:

```
SELECT Название
FROM user1.itog()
```

Функции Multi-statement.

Создание и изменение функций типа Multi-statement выполняется с помощью следующей команды:

```
<определение_мульти_функции>::=
{CREATE | ALTER }FUNCTION [владелец.]
   имя_функции
( [ { @имя_параметра скаляр_тип_данных
   [=default]}[,...n]])
RETURNS @имя_параметра TABLE
   <определение_таблицы>
[WITH {ENCRYPTION | SCHEMABINDING}
   [,...n] ]
[AS]
BEGIN
<тело_функции>
RETURN
END
```

Использование большей части параметров рассматривалось при описании предыдущих функций.

Отметим, что функции данного типа, как и табличные, возвращают значение типа TABLE. Однако, в отличие от табличных функций, при создании функций Multi-statement необходимо явно задать структуру

возвращаемого значения. Она указывается непосредственно после ключевого слова TABLE и, таким образом, является частью определения возвращаемого типа данных. Синтаксис конструкции <определение_таблицы> полностью соответствует одноименным структурам, используемым при создании обычных таблиц с помощью команды CREATE TABLE.

Набор возвращаемых данных должен формироваться с помощью команд INSERT, выполняемых в теле функции. Кроме того, в теле функции допускается использование различных конструкций языка SQL, которые могут контролировать значения, размещаемые в выходном наборе строк. При работе с командой INSERT требуется явно указать имя того объекта, куда необходимо вставить строки. Поэтому в функциях типа Multi-statement, в отличие от табличных, необходимо присвоить какое-то имя объекту с типом данных TABLE – оно и указывается как возвращаемое значение.

Завершение работы функции происходит в двух случаях: если возникают ошибки выполнения и если появляется ключевое слово RETURN. В отличие от функций скалярного типа, при использовании команды RETURN не нужно указывать возвращаемое значение. Сервер автоматически возвратит набор данных типа TABLE, имя и структура которого была указана после ключевого слова RETURNS. В теле функции может быть указано более одной команды RETURN.

Необходимо отметить, что работа функции завершается только при наличии команды RETURN. Это утверждение верно и в том случае, когда речь идет о достижении конца тела функции – самой последней командой должна быть команда RETURN.

Пример. Создать и применить функцию (типа multi-statement), которая для некоторого сотрудника выводит список всех его подчиненных (подчиненных как непосредственно ему, так и опосредствованно через других сотрудников).

Список сотрудников с указанием каждого руководителя представлен в таблице emp_mgr со следующей структурой:

```
CREATE TABLE emp_mgr
(emp CHAR(2) PRIMARY KEY,-- сотрудник
mgr CHAR(2))          -- руководитель
```

Пример данных в таблице emp_mgr показан ниже. Для упрощения иллюстрации имена сотрудников и их начальников представлены буквами латинского алфавита. У директора организации начальника нет (NULL).

Пример. Создание функции, которая для некоторого сотрудника выводит список всех его подчиненных.

```
emp    mgr
---------
a    NULL
b    a
```

```
c    a
d    a
e    f
f    b
g    b
i    c
k    d

CREATE FUNCTION fn_findReports(@id_emp
   CHAR(2))
RETURNS @report TABLE(empid CHAR(2)
   PRIMARY KEY,
   mgrid CHAR(2))
AS
BEGIN
   DECLARE @r INT
   DECLARE @t TABLE(empid CHAR(2)
      PRIMARY KEY,
      mgrid CHAR(2),
        pr   INT DEFAULT 0)
INSERT @t SELECT emp,mgr,0
      FROM emp_mgr
      WHERE emp=@id_emp
SET @r=@@ROWCOUNT
WHILE @r>0
BEGIN
   UPDATE @t   SET pr=1   WHERE pr=0
   INSERT @t SELECT e.emp, e.mgr,0
      FROM  emp_mgr e, @t t
      WHERE e.mgr=t.empid
         AND t.pr=1
   SET @r=@@ROWCOUNT
   UPDATE @t SET pr=2 WHERE pr=1
END
INSERT @report SELECT empid, mgrid
   FROM @t
RETURN
END
```

Применим созданную функцию для определения списка подчиненных сотрудника ‘b’:

```
SELECT * FROM fn_findReports('b')
```

Оператор возвращает следующие значения:

```
emp    mgr
-----------
b    a
e    f
f    b
g    b
```

Список подчиненных сотрудника ‘a’ создается с помощью оператора

```
SELECT * FROM fn_findReports('a')
```

```
emp     mgr
---------
a     NULL
b     a
c     a
d     a
e     f
f     b
g     b
i     c
k     d
```

Другой оператор формирует список подчиненных сотрудника ‘e’:

```
SELECT * FROM fn_findReports('e')
emp     mgr
--------
e     f
```

Список подчиненных сотрудника ‘c’ создает следующий оператор:

```
SELECT * FROM fn_findReports('c')
emp     mgr
--------
c     a
i     c
```

Удаление любой функции осуществляется командой:

```
DROP FUNCTION {[ владелец.] имя_функции }
   [,...n]
```

Встроенные функции.

Встроенные функции, имеющиеся в распоряжении пользователей при работе с SQL, можно условно разделить на следующие группы:

- математические функции;
- строковые функции;
- функции для работы с датой и временем;
- функции конфигурирования;
- функции системы безопасности;
- функции управления метаданными;
- статистические функции.

Математические функции.

Краткий обзор математических функций представлен в таблице 10.

Таблица 10.

ABS	вычисляет абсолютное значение числа
ACOS	вычисляет арккосинус
ASIN	вычисляет арксинус
ATAN	вычисляет арктангенс
ATN2	вычисляет арктангенс с учетом квадратов
CEILING	выполняет округление вверх
COS	вычисляет косинус угла
COT	возвращает котангенс угла
DEGREES	преобразует значение угла из радиан в градусы
EXP	возвращает экспоненту
FLOOR	выполняет округление вниз
LOG	вычисляет натуральный логарифм
LOG10	вычисляет десятичный логарифм
PI	возвращает значение «пи»
POWER	возводит число в степень
RADIANS	преобразует значение угла из градуса в радианы
RAND	возвращат случайное число
ROUND	выполняет округление с заданной точностью
SIGN	определяет знак числа
SIN	вычисляет синус угла
SQUARE	выполняет возведение числа в квадрат
SQRT	извлекает квадратный корень
TAN	возвращает тангенс угла

Пример. Использование функции округления до одного знака после запятой для расчета налога.

```
SELECT Товар.Название, Сделка.Количество,
Round(Товар.Цена*Сделка.Количество
   *0.05,1)
AS Налог
FROM Товар INNER JOIN Сделка
   ON Товар.КодТовара=
   Сделка.КодТовара
```

Строковые функции.

Краткий обзор строковых функций представлен в таблице 11.

Таблица 11.

ASCII	возвращает код ASCII левого символа строки
CHAR	по коду ASCII возвращает символ
CHARINDEX	определяет порядковый номер символа, с которого начинается вхождение подстроки в

	строку
DIFFERENCE	возвращает показатель совпадения строк
LEFT	возвращает указанное число символов с начала строки
LEN	возвращает длину строки
LOWER	переводит все символы строки в нижний регистр
LTRIM	удаляет пробелы в начале строки
NCHAR	возвращает по коду символ Unicode
PATINDEX	выполняет поиск подстроки в строке по указанному шаблону
REPLACE	заменяет вхождения подстроки на указанное значение
QUOTENAME	конвертирует строку в формат Unicode
REPLICATE	выполняет тиражирование строки определенное число раз
REVERSE	возвращает строку, символы которой записаны в обратном порядке
RIGHT	возвращает указанное число символов с конца строки
RTRIM	удаляет пробелы в конце строки
SOUNDEX	возвращает код звучания строки
SPACE	возвращает указанное число пробелов
STR	выполняет конвертирование значения числового типа в символьный формат
STUFF	удаляет указанное число символов, заменяя новой подстрокой
SUBSTRING	возвращает для строки подстроку указанной длины с заданного символа
UNICODE	возвращает Unicode-код левого символа строки
UPPER	переводит все символы строки в верхний регистр

Пример. Использование функции LEFT для получения инициалов клиентов.

```
SELECT Фирма, [Фамилия]+’’+Left([Имя],1)+’.’+Left([Отчество],1)+’.’ AS ФИО
FROM Клиент
```

Функции для работы с датой и временем.

Краткий обзор основных функций для работы с датой и временем представлен в таблице 12.

Таблица 12.

DATEADD	добавляет к дате указанное значение дней, месяцев, часов и т.д.
DATEDIFF	возвращает разницу между указанными частями двух дат
DATENAME	выделяет из даты указанную часть и возвращает ее в символьном формате
DATEPART	выделяет из даты указанную часть и возвращает ее в числовом формате
DAY	возвращает число из указанной даты
GETDATE	возвращает текущее системное время
ISDATE	проверяет правильность выражения на соответствие одному из возможных форматов ввода даты
MONTH	возвращает значение месяца из указанной даты
YEAR	возвращает значение года из указанной даты

Пример. Использование функций YEAR и MONTH для определения общего количества товара, проданного за каждый месяц каждого года.

```
SELECT Year(Дата) AS Год, Month(Дата)
```

```
  AS Месяц,
  Sum(Количество) AS Общ_Количество
FROM Сделка
GROUP BY Year(Дата), Month(Дата)
```

Пример. Пример выделения из даты значения года.

```
DECLARE @d DATETIME
DECLARE @y INT
SET @d='29.10.03'
SET @y=DATEPART(yy,@d)
SELECT @y
```

Вопрос 12. Хранимые процедуры.

Понятие хранимой процедуры.

Хранимые процедуры представляют собой группы связанных между собой операторов SQL, применение которых делает работу программиста более легкой и гибкой, поскольку выполнить хранимую процедуру часто оказывается гораздо проще, чем последовательность отдельных операторов SQL. Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. Выполнение в базе данных хранимых процедур вместо отдельных операторов SQL дает пользователю следующие преимущества:

- необходимые операторы уже содержатся в базе данных;
- все они прошли этап синтаксического анализа и находятся в исполняемом формате; перед выполнением хранимой процедуры SQL Server генерирует для нее план исполнения, выполняет ее оптимизацию и компиляцию;
- хранимые процедуры поддерживают модульное программирование, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;
- хранимые процедуры могут вызывать другие хранимые процедуры и функции;
- хранимые процедуры могут быть вызваны из прикладных программ других типов;
- как правило, хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;
- хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Хранение процедур в том же месте, где они исполняются, обеспечивает уменьшение объема передаваемых по сети данных и повышает общую

производительность системы. Применение хранимых процедур упрощает сопровождение программных комплексов и внесение изменений в них. Обычно все ограничения целостности в виде правил и алгоритмов обработки данных реализуются на сервере баз данных и доступны конечному приложению в виде набора хранимых процедур, которые и представляют интерфейс обработки данных. Для обеспечения целостности данных, а также в целях безопасности, приложение обычно не получает прямого доступа к данным – вся работа с ними ведется путем вызова тех или иных хранимых процедур.

Подобный подход делает весьма простой модификацию алгоритмов обработки данных, тотчас же становящихся доступными для всех пользователей сети, и обеспечивает возможность расширения системы без внесения изменений в само приложение: достаточно изменить хранимую процедуру на сервере баз данных. Разработчику не нужно перекомпилировать приложение, создавать его копии, а также инструктировать пользователей о необходимости работы с новой версией. Пользователи вообще могут не подозревать о том, что в систему внесены изменения.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

Хранимые процедуры в среде MS SQL Server.

При работе с SQL Server пользователи могут создавать собственные процедуры, реализующие те или иные действия. Хранимые процедуры являются полноценными объектами базы данных, а потому каждая из них хранится в конкретной базе данных. Непосредственный вызов хранимой процедуры возможен, только если он осуществляется в контексте той базы данных, где находится процедура.

Типы хранимых процедур.

В SQL Server имеется несколько типов хранимых процедур.

Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс sp_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.

Пользовательские хранимые процедуры реализуют те или иные действия. Хранимые процедуры – полноценный объект базы данных.

Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.

Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

Создание, изменение и удаление хранимых процедур.

Создание хранимой процедуры предполагает решение следующих задач:

- определение типа создаваемой хранимой процедуры: временная или пользовательская. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом sp_ и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера;
- планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;
- определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами;
- разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

```
<определение_процедуры>::=
{CREATE | ALTER } PROC[EDURE] имя_процедуры
   [;номер]
[{@имя_параметра тип_данных } [VARYING ]
  [=default][OUTPUT] ][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
  ENCRYPTION }]
[FOR REPLICATION]
AS
  sql_оператор [...n]
```

Рассмотрим параметры данной команды.

Используя префиксы sp_, #, ##, создаваемую процедуру можно определить в качестве системной или временной. Как видно из синтаксиса команды, не допускается указывать имя владельца, которому будет принадлежать создаваемая процедура, а также имя базы данных, где она должна быть размещена. Таким образом, чтобы разместить создаваемую хранимую процедуру в конкретной базе данных, необходимо выполнить команду CREATE PROCEDURE в контексте этой базы данных. При обращении из тела хранимой процедуры к объектам той же базы данных можно использовать укороченные имена, т. е. без указания имени базы данных. Когда же требуется обратиться к объектам, расположенным в других базах данных, указание имени базы данных обязательно.

Номер в имени – это идентификационный номер хранимой процедуры, однозначно определяющий ее в группе процедур. Для удобства управления процедурами логически однотипные хранимые процедуры можно группировать, присваивая им одинаковые имена, но разные идентификационные номера.

Для передачи входных и выходных данных в создаваемой хранимой процедуре могут использоваться параметры, имена которых, как и имена локальных переменных, должны начинаться с символа @. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры.

Для определения типа данных, который будет иметь соответствующий параметр хранимой процедуры, годятся любые типы данных SQL, включая определенные пользователем. Однако тип данных CURSOR может быть использован только как выходной параметр хранимой процедуры, т.е. с указанием ключевого слова OUTPUT.

Наличие ключевого слова OUTPUT означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Однако это вовсе не означает, что параметр не подходит для передачи значений в хранимую процедуру. Указание ключевого слова OUTPUT предписывает серверу при выходе из хранимой процедуры присвоить текущее значение параметра локальной переменной, которая была указана при вызове процедуры в качестве значения параметра. Отметим, что при указании ключевого слова OUTPUT значение соответствующего параметра при вызове процедуры может быть задано только с помощью локальной переменной. Не разрешается использование любых выражений или констант, допустимое для обычных параметров.

Ключевое слово VARYING применяется совместно с параметром OUTPUT, имеющим тип CURSOR. Оно определяет, что выходным параметром будет результирующее множество.

Ключевое слово DEFAULT представляет собой значение, которое будет принимать соответствующий параметр по умолчанию. Таким образом,

при вызове процедуры можно не указывать явно значение соответствующего параметра.

Так как сервер кэширует план исполнения запроса и компилированный код, при последующем вызове процедуры будут использоваться уже готовые значения. Однако в некоторых случаях все же требуется выполнять перекомпиляцию кода процедуры. Указание ключевого слова RECOMPILE предписывает системе создавать план выполнения хранимой процедуры при каждом ее вызове.

Параметр FOR REPLICATION востребован при репликации данных и включении создаваемой хранимой процедуры в качестве статьи в публикацию.

Ключевое слово ENCRYPTION предписывает серверу выполнить шифрование кода хранимой процедуры, что может обеспечить защиту от использования авторских алгоритмов, реализующих работу хранимой процедуры.

Ключевое слово AS размещается в начале собственно тела хранимой процедуры, т.е. набора команд SQL, с помощью которых и будет реализовываться то или иное действие. В теле процедуры могут применяться практически все команды SQL, объявляться транзакции, устанавливаться блокировки и вызываться другие хранимые процедуры. Выход из хранимой процедуры можно осуществить посредством команды RETURN.

Удаление хранимой процедуры осуществляется командой:

```
DROP PROCEDURE {имя_процедуры} [,...n]
```

Выполнение хранимой процедуры.

Для выполнения хранимой процедуры используется команда:

```
[[ EXEC [ UTE] имя_процедуры [;номер]
[[@имя_параметра=]{значение | @имя_переменной}
  [OUTPUT ]|[DEFAULT ]][,...n]
```

Если вызов хранимой процедуры не является единственной командой в пакете, то присутствие команды EXECUTE обязательно. Более того, эта команда требуется для вызова процедуры из тела другой процедуры или триггера.

Использование ключевого слова OUTPUT при вызове процедуры разрешается только для параметров, которые были объявлены при создании процедуры с ключевым словом OUTPUT.

Когда же при вызове процедуры для параметра указывается ключевое слово DEFAULT, то будет использовано значение по умолчанию. Естественно, указанное слово DEFAULT разрешается только для тех параметров, для которых определено значение по умолчанию.

Из синтаксиса команды EXECUTE видно, что имена параметров могут быть опущены при вызове процедуры. Однако в этом случае пользователь

должен указывать значения для параметров в том же порядке, в каком они перечислялись при создании процедуры. Присвоить параметру значение по умолчанию, просто пропустив его при перечислении нельзя. Если же требуется опустить параметры, для которых определено значение по умолчанию, достаточно явного указания имен параметров при вызове хранимой процедуры. Более того, таким способом можно перечислять параметры и их значения в произвольном порядке.

Отметим, что при вызове процедуры указываются либо имена параметров со значениями, либо только значения без имени параметра. Их комбинирование не допускается.

Пример. Процедура без параметров. Разработать процедуру для получения названий и стоимости товаров, приобретенных Ивановым.

```
CREATE PROC my_proc1
AS
SELECT Товар.Название,
      Товар.Цена*Сделка.Количество
  AS Стоимость, Клиент.Фамилия
 FROM Клиент INNER JOIN
 (Товар INNER JOIN Сделка
 ON Товар.КодТовара=Сделка.КодТовара)
 ON Клиент.КодКлиента=Сделка.КодКлиента
 WHERE Клиент.Фамилия='Иванов'
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc1 или my_proc1
```

Процедура возвращает набор данных.

Пример. Процедура без параметров. Создать процедуру для уменьшения цены товара первого сорта на 10%.

```
CREATE PROC my_proc2
AS
UPDATE Товар SET Цена=Цена*0.9
  WHERE Сорт='первый'
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc2 или my_proc2
```

Процедура не возвращает никаких данных.

Пример. Процедура с входным параметром. Создать процедуру для получения названий и стоимости товаров, которые приобрел заданный клиент.

```
CREATE PROC my_proc3
```

```
  @k VARCHAR(20)
AS
SELECT Товар.Название,
      Товар.Цена*Сделка.Количество
  AS Стоимость, Клиент.Фамилия
  FROM Клиент INNER JOIN
  (Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара)
  ON Клиент.КодКлиента=Сделка.КодКлиента
  WHERE Клиент.Фамилия=@k
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc3 'Иванов' или
my_proc3 @k='Иванов'
```

Пример. Процедура с входными параметрами. Создать процедуру для уменьшения цены товара заданного типа в соответствии с указанным %.

```
CREATE PROC my_proc4
  @t VARCHAR(20), @p FLOAT
AS
UPDATE Товар SET Цена=Цена*(1-@p)
  WHERE Тип=@t
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc4 'Вафли',0.05 или
EXEC my_proc4 @t='Вафли', @p=0.05
```

Пример. Процедура с входными параметрами и значениями по умолчанию. Создать процедуру для уменьшения цены товара заданного типа в соответствии с указанным %.

```
CREATE PROC my_proc5
  @t VARCHAR(20)='Конфеты',
  @p FLOAT=0.1
AS
UPDATE Товар SET Цена=Цена*(1-@p)
  WHERE Тип=@t
```

Для обращения к процедуре можно использовать команды:

```
EXEC my_proc5 'Вафли',0.05 или
EXEC my_proc5 @t='Вафли', @p=0.05 или
EXEC my_proc5 @p=0.05
```

В этом случае уменьшается цена конфет (значение типа не указано при вызове процедуры и берется по умолчанию).

```
EXEC my_proc5
```

В последнем случае оба параметра (и тип, и проценты) не указаны при вызове процедуры, их значения берутся по умолчанию.

Пример. Процедура с входными и выходными параметрами. Создать процедуру для определения общей стоимости товаров, проданных за конкретный месяц.

```
CREATE PROC my_proc6
  @m INT,
  @s FLOAT OUTPUT
AS
SELECT @s=Sum(Товар.Цена*Сделка.Количество)
  FROM Товар INNER JOIN Сделка
  ON Товар.КодТовара=Сделка.КодТовара
  GROUP BY Month(Сделка.Дата)
  HAVING Month(Сделка.Дата)=@m
```

Для обращения к процедуре можно использовать команды:

```
DECLARE @st FLOAT
EXEC my_proc6 1,@st OUTPUT
SELECT @st
```

Этот блок команд позволяет определить стоимость товаров, проданных в январе (входной параметр месяц указан равным 1).

Создать процедуру для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник.

Сначала разработаем процедуру для определения фирмы, где работает сотрудник.

```
CREATE PROC my_proc7
  @n VARCHAR(20),
  @f VARCHAR(20) OUTPUT
AS
SELECT @f=Фирма
  FROM Клиент
  WHERE Фамилия=@n
```

Пример. Использование вложенных процедур. Создать процедуру для определения общего количества товаров, приобретенных фирмой, в которой работает заданный сотрудник.

Затем создадим процедуру, подсчитывающую общее количество товара, который закуплен интересующей нас фирмой.

```
CREATE PROC my_proc8
  @fam VARCHAR(20),
```

```
  @kol INT OUTPUT
AS
DECLARE @firm VARCHAR(20)
EXEC my_proc7 @fam,@firm OUTPUT
SELECT @kol=Sum(Сделка.Количество)
  FROM Клиент INNER JOIN Сделка
  ON Клиент.КодКлиента=Сделка.КодКлиента
  GROUP BY Клиент.Фирма
  HAVING Клиент.Фирма=@firm
```

Вызов процедуры осуществляется с помощью команды:

```
DECLARE @k INT
EXEC my_proc8 ‘Иванов’,@k OUTPUT
SELECT @k
```

Вопрос 13. Триггеры.

Определение триггера в стандарте языка SQL.

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения значений таблицы, чтобы нужным образом изменить связанные данные. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или

нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:

```
<Определение_триггера>::=
  CREATE TRIGGER имя_триггера
  BEFORE | AFTER <триггерное_событие>
  ON <имя_таблицы>
  [REFERENCING
    <список_старых_или_новых_псевдонимов>]
  [FOR EACH { ROW | STATEMENT}]
  [WHEN(условие_триггера)]
  <тело_триггера>
```

триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанных с ним событий) или AFTER (после их выполнения).

Выполняемые триггером действия задаются для каждой строки (FOR EACH ROW), охваченной данным событием, или только один раз для каждого события (FOR EACH STATEMENT).

Обозначение <список_старых_или_новых_псевдонимов> относится к таким компонентам, как старая или новая строка (OLD / NEW) либо старая или новая таблица (OLD TABLE / NEW TABLE). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри базы данных и согласованно активизируются при каждом ее обновлении. Это может существенно упростить приложения. Тем не менее следует упомянуть и о присущих триггеру недостатках:

- сложность: при перемещении некоторых функций в базу данных усложняются задачи ее проектирования, реализации и администрирования;
- скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
- влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным. Очевидно, что при возрастании количества триггеров увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление «мертвых» блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

Реализация триггеров в среде MS SQL Server.

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения триггера:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
```

```
AS
  sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
 [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
 бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере.

Рассмотрим назначение аргументов из команды CREATE | ALTER TRIGGER.

Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

При указании аргумента WITH ENCRYPTION сервер выполняет шифрование кода триггера, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров.

В SQL Server существует два параметра, определяющих поведение триггеров:

AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.

INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как

для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Триггеры различают по типу команд, на которые они реагируют.

Существует три типа триггеров:

- INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды INSERT.
- UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE.
- DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. Допускается создание триггера, реагирующего на две или на все три команды.

Аргумент WITH APPEND позволяет создавать несколько триггеров каждого типа.

При создании триггера с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске триггера.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как: создание, изменение и удаление базы данных; восстановление резервной копии базы данных или журнала транзакций. Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых триггеров. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

Программирование триггера.

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: inserted и deleted. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ.

В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

- команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни

одной строки; после завершения триггера все строки из таблицы inserted переместятся в исходную таблицу;

- команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;
- команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Если триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Для получения списка столбцов, измененных при выполнении команд INSERT или UPDATE, вызвавших выполнение триггера, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение «1», то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

Для удаления триггера используется команда

```
DROP TRIGGER {имя_триггера} [,...n]
```

Приведем примеры использования триггеров.

Пример. Использование триггера для реализации ограничений на значение. В добавляемой в таблицу Сделка записи количество проданного товара должно быть не меньше, чем его остаток из таблицы Склад.

Команда вставки записи в таблицу Сделка может быть, например, такой:

```
INSERT INTO Сделка
  VALUES (3,1,-299,'01/08/2002')
```

Создаваемый триггер должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Склад величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=3). Во вставляемой записи количество товара указывается со знаком «+», если товар поставляется, и со знаком «-», если он продается. Представленный триггер настроен на обработку только одной добавляемой записи.

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
  IF NOT EXISTS(SELECT *
     FROM inserted
     WHERE -inserted.количество<=ALL(SELECT
       Склад.Остаток
    FROM Склад,Сделка
    WHERE Склад.КодТовара=
     Сделка.КодТовара))
    BEGIN
     ROLLBACK TRAN
    PRINT
     'Отмена поставки: товара на складе нет'
    END
END
```

Пример. Использования триггера для сбора статистических данных.

Создать триггер для обработки операции вставки записи в таблицу Сделка, например, такой команды:

```
INSERT INTO Сделка
  VALUES (3,1,200,'01/08/2002')
```

поставляется товар с кодом 3 от клиента с кодом 1 в количестве 200 единиц.

При продаже или получении товара необходимо соответствующим образом изменить количество его складского запаса. Если товара на складе

еще нет, необходимо добавить соответствующую запись в таблицу Склад. Триггер обрабатывает только одну добавляемую строку.

```
ALTER TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
DECLARE @x INT, @y INT
IF @@ROWCOUNT=1
--в таблицу Сделка добавляется запись
--о поставке товара
BEGIN
--количество проданного товара должно быть не
--меньше, чем его остаток из таблицы Склад
IF NOT EXISTS(SELECT *
         FROM inserted
         WHERE -inserted.количество<
  =ALL(SELECT Склад.Остаток
       FROM Склад,Сделка
       WHERE Склад.КодТовара=
          Сделка.КодТовара))
   BEGIN
     ROLLBACK TRAN
      PRINT 'откат товара нет '
    END
--если записи о поставленном товаре еще нет,
--добавляется соответствующая запись
--в таблицу Склад
  IF NOT EXISTS ( SELECT *
         FROM Склад C, inserted i
         WHERE C.КодТовара=i.КодТовара )
     INSERT INTO Склад (КодТовара,Остаток)
  ELSE
--если запись о товаре уже была в таблице
--Склад, то определяется код и количество
--товара издобавленной в таблицу Сделка записи
  BEGIN
    SELECT @y=i.КодТовара, @x=i.Количество
    FROM Сделка C, inserted i
    WHERE C.КодТовара=i.КодТовара
--и производится изменения количества товара в
--таблице Склад
       UPDATE Склад
       SET Остаток=остаток+@x
       WHERE КодТовара=@y
   END
END
```

Пример. Создать триггер для обработки операции удаления записи из таблицы Сделка, например, такой команды:

```
DELETE FROM Сделка WHERE КодСделки=4
```

Для товара, код которого указан при удалении записи, необходимо откорректировать его остаток на складе. Триггер обрабатывает только одну удаляемую запись.

```
CREATE TRIGGER Триггер_del
ON Сделка FOR DELETE
AS
IF @@ROWCOUNT=1 -- удалена одна запись
BEGIN
  DECLARE @y INT,@x INT
--определяется код и количество товара из
--удаленной из таблицы Склад записи
  SELECT @y=КодТовара, @x=Количество
  FROM deleted
--в таблице Склад корректируется количество
--товара
    UPDATE Склад
    SET Остаток=Остаток-@x
    WHERE КодТовара=@y
END
```

Пример. Создать триггер для обработки операции изменения записи в таблице Сделка, например, такой командой:

```
UPDATE Сделка SET количество=количество-10
WHERE КодТовара=3
```

во всех сделках с товаром, имеющим код, равный 3, уменьшить количество товара на 10 единиц.

Указанная команда может привести к изменению сразу нескольких записей в таблице Сделка. Поэтому покажем, как создать триггер, обрабатывающий не одну запись. Для каждой измененной записи необходимо для старого (до изменения) кода товара уменьшить остаток товара на складе на величину старого (до изменения) количества товара и для нового (после изменения) кода товара увеличить его остаток на складе на величину нового (после изменения) значения. Чтобы обработать все измененные записи, введем курсоры, в которых сохраним все старые (из таблицы deleted) и все новые значения (из таблицы inserted).

```
CREATE TRIGGER Триггер_upd
ON Сделка FOR UPDATE
AS
DECLARE @x INT, @x_old INT, @y INT, @y_old INT
-- курсор с новыми значениями
DECLARE CUR1 CURSOR FOR
   SELECT КодТовара,Количество
   FROM inserted
-- курсор со старыми значениями
```

```
DECLARE CUR2 CURSOR FOR
   SELECT КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
-- перемещаемся параллельно по обоим курсорам
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
--для старого кода товара уменьшается его
--количество на складе
     UPDATE Склад
     SET Остаток=Остаток-@y_old
     WHERE КодТовара=@x_old
--для нового кода товара, если такого товара
--еще нет на складе, вводится новая запись
     IF NOT EXISTS (SELECT * FROM Склад
       WHERE КодТовара=@x)
     INSERT INTO Склад(КодТовара,Остаток)
      VALUES (@x,@y)
     ELSE
--иначе для нового кода товара увеличивается
--его количество на складе
      UPDATE Склад
      SET Остаток=Остаток+@y
      WHERE КодТовара=@x
     FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

В рассмотренном триггере отсутствует сравнение количества товара при изменении записи о сделке с его остатком на складе.

Пример. Исправим этот недостаток. Для генерирования сообщения об ошибке используем в теле триггера команду MS SQL Server RAISERROR, аргументами которой являются текст сообщения, уровень серьезности и статус ошибки.

```
ALTER TRIGGER Триггер_upd
ON Сделка FOR UPDATE
AS
DECLARE @x INT, @x_old INT, @y INT,
      @y_old INT ,@o INT
DECLARE CUR1 CURSOR FOR
   SELECT КодТовара,Количество
   FROM inserted
DECLARE CUR2 CURSOR FOR
```

```
   SELECT КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
     SELECT @o=остаток
     FROM Склад
     WHERE кодтовара=@x
     IF @o<-@y
     BEGIN
      RAISERROR('откат',16,10)
      CLOSE CUR1
      CLOSE CUR2
      DEALLOCATE CUR1
      DEALLOCATE CUR22
      ROLLBACK TRAN
      RETURN
     END
      UPDATE Склад
     SET Остаток=Остаток-@y_old
     WHERE КодТовара=@x_old
     IF NOT EXISTS (SELECT * FROM Склад
      WHERE КодТовара=@x)
     INSERT INTO Склад(КодТовара,Остаток)
      VALUES (@x,@y)
    ELSE
     UPDATE Склад
     SET Остаток=Остаток+@y
     WHERE КодТовара=@x
   FETCH NEXT FROM CUR1 INTO @x, @y
   FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Пример . В примере выше происходит отмена всех изменений при невозможности реализовать хотя бы одно из них. Создадим триггер, позволяющий отменять изменение только некоторых записей и выполнять изменение остальных.

В этом случае триггер выполняется не после изменения записей, а вместо команды изменения.

```
ALTER TRIGGER Триггер_upd
ON Сделка INSTEAD OF UPDATE
AS
DECLARE @k INT, @k_old INT
```

```
DECLARE @x INT, @x_old INT, @y INT
DECLARE @y_old INT ,@o INT
DECLARE CUR1 CURSOR FOR
   SELECT КодСделки, КодТовара,Количество
   FROM inserted
DECLARE CUR2 CURSOR FOR
   SELECT КодСделки, КодТовара,Количество
   FROM deleted
OPEN CUR1
OPEN CUR2
   FETCH NEXT FROM CUR1 INTO @k,@x, @y
   FETCH NEXT FROM CUR2 INTO @k_old,@x_old,
    @y_old
   WHILE @@FETCH_STATUS=0
    BEGIN
     SELECT @o=остаток
     FROM Склад
     WHERE КодТовара=@x
     IF @o>=-@y
     BEGIN
     RAISERROR('изменение',16,10)
     UPDATE Сделка SET количество=@y,
      КодТовара=@x
     WHERE КодСделки=@k

     UPDATE Склад
     SET Остаток=Остаток-@y_old
     WHERE КодТовара=@x_old

     IF NOT EXISTS (SELECT * FROM Склад
       WHERE КодТовара=@x)
     INSERT INTO Склад(КодТовара, Остаток)
       VALUES (@x,@y)
     ELSE
     UPDATE Склад
     SET Остаток=Остаток+@y
     WHERE КодТовара=@x
    END
    ELSE
     RAISERROR('запись не изменена',16,10)
    FETCH NEXT FROM CUR1 INTO @k,@x, @y
    FETCH NEXT FROM CUR2 INTO @k_old,@x_old,
     @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Тема 5. Обеспечение целостности данных в БД[15]

Вопрос 2. Организация процессов обработки данных в файловых системах и СУБД. [16]

Общая структура СУБД.

Для лучшего понимания принципов работы современных СУБД рассмотрим структуру одной из наиболее распространенных клиент-серверных СУБД - Microsoft SQL Server 2008. Несмотря на то, что каждая коммерческая СУБД имеет свои отличительные особенности, информации о том, как устроена какая-то из СУБД, обычно бывает достаточно для быстрого первоначального освоения другой СУБД. Краткий обзор возможностей Microsoft SQL Server - 2008 был приведен в разделе, посвященном краткому обзору современным СУБД. В данном разделе рассмотрим основные моменты, связанные со структурой соответствующей СУБД (архитектурой базы данных и структурой программного обеспечения).

Под архитектурой (структурой) базы данных конкретной СУБД будем понимать основные модели представления данных, используемые в соответствующей СУБД а также взаимосвязи между этими моделями.

В соответствии с рассмотренными выше различными уровнями описания данных различают разные уровни абстракции архитектуры базы данных.

Логический уровеньL (уровень модели данных СУБД) - средство представления концептуальной модели). Здесь каждая СУБД имеет некоторые отличия, но они являются не очень значительными. Отметим, что у разных СУБД существенно отличаются механизмы перехода от логического к физическому уровню представления.

Физический уровень (внутреннее представление данных в памяти ЭВМ - физическая структура базы данных). Данный уровень рассмотрения подразумевает изучение базы данных на уровне файлов, хранящихся на жестком диске. Структура этих файлов – особенность каждой конкретной СУБД, в т.ч. и Microsoft SQL Server.

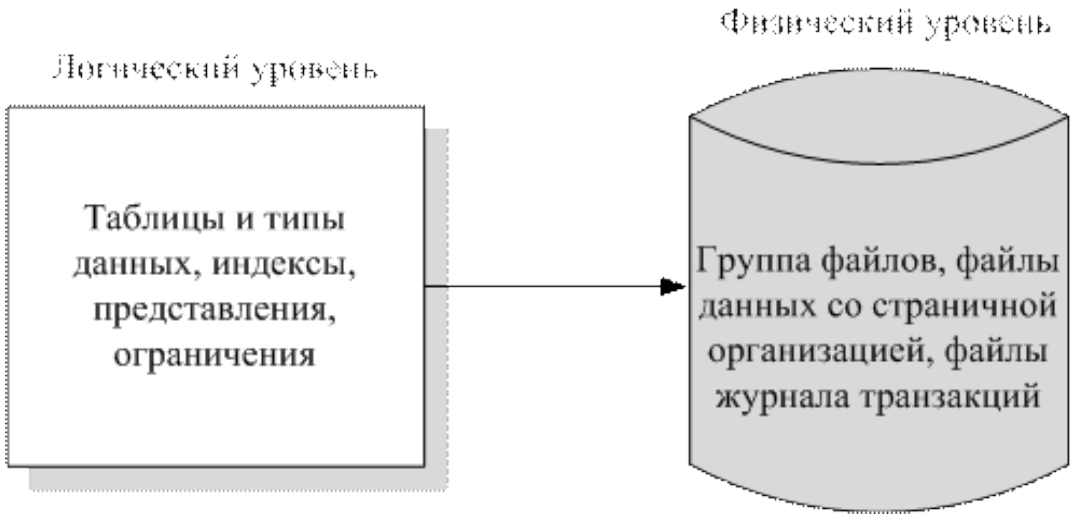


Рис. 36. Архитектура базы данных в Microsoft SQL Server 2008

Архитектура базы данных. Логический уровень.

Рассмотрим логический уровень представления базы данных (http://msdn.microsoft.com). Microsoft SQL Server 2008 представляет собой реляционную СУБД (данные представляются в виде таблиц). Таким образом, основной структурой модели данных этой СУБД являются таблицы.

Таблицы и типы данных.

Таблицы содержат данные о всех сущностях концептуальной модели базы данных. При описании каждого столбца (поля) пользователь должен определить тип соответствующих данных. Microsoft SQL Server 2008 поддерживает как уже ставшие традиционными типы данных (символьная строка с разным представлением, число с плавающей точкой длиной 8 или 4 байта, целое число длины 2 или 4 байта, дата и время, поле примечаний, булево значение и т. д.), так и новые типы данных. Кроме этого Microsoft SQL Server 2008 предоставляет специальный аппарат для создания пользовательских типов данных.

Рассмотрим краткую характеристику некоторых новых типов данных, значительно расширяющих возможности пользователя (www.oszone.net).

Тип данных hierarchyid.

Тип данных hierarchyid позволяет создавать отношения между элементами данных в таблице, для того, чтобы задать позицию в иерархии связей между строками таблицы. В результате использования этого типа данных в таблице строки таблицы могут отображать определенную иерархическую структуру, соответствующую связям между данными этой таблицы.

Пространственные типы данных.

Пространственные данные – это данные, определяющие географические расположения и формы, преимущественно на Земле. Это

могут быть ориентиры, дороги и даже расположение фирмы. В SQL Server 2008 есть географические (geography) и геометрические (geometry) типы данных для работы с этой информацией. Тип данных geography работает с информацией для шарообразной земли. Модель шарообразной земли использует при расчетах кривизну земной поверхности. Информация о положении задается широтой и долготой. Эта модель хорошо годится для приложений, связанных с морскими перевозками, военным планированием и краткосрочными приложениями, имеющими привязку к земной поверхности. Эту модель нужно использовать, если данные хранятся в виде широт и долгот.

Тип данных geometry работает с планарной моделью или моделью плоской земли. В этой модели земля считается плоской проекцией из определенной точки. Модель плоской земли не принимает в расчет кривизну поверхности земли, поэтому используется, в первую очередь, для описания коротких расстояний, например, в базе данных приложения, описывающего внутреннюю часть строения.

Типы geography и geometry создаются из векторных объектов, заданных в форматах Well-Known Text (WKT) или Well-Known Binary (WKB). Это форматы для перенесения пространственных данных, описанные в простых функциях открытого геопространственного консорциума (Open Geospatial Consortium (OGC) Simple Features) для спецификаций SQL (SQL Specification).

Ключи.

Для каждой таблицы должен быть определен первичный ключ – минимальный набор атрибутов, уникально идентифицирующий каждую запись в таблице. Для реализации связи между таблицами в одну из связанных таблиц включается дополнительное поле (несколько полей) – первичный ключ другой таблицы. Дополнительно включенные поле или поля в этом случае называются внешним ключом соответствующей таблицы.

Кроме таблиц, в модель данных Microsoft SQL Server 2008 входит еще целый ряд компонентов. Дадим краткую характеристику основным из них.

Индексы.

Ранее рассматривалось понятие индекса. Здесь понятие индекса вынесено на логический уровень для удобства пользователя. Индексы создаются для ускорения поиска нужной информации и содержат информацию об упорядоченности данных по различным критериям. Индексирование может быть выполнено по одному или нескольким столбцам. Индексирование может быть произведено в любой момент. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в виде структуры сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям в SQL Server.

Представления.

Представление — это виртуальная таблица, содержимое которой определяется запросом. Представление формируется на основе SQL-запроса

SELECT, формируемого по обычным правилам. Таким образом, представление есть поименованный запрос SELECT.

Как и настоящая таблица, представление состоит из совокупности именованных столбцов и строк данных. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений. Строки и столбцы данных извлекаются из таблиц, указанных в определяющем представление запросе и динамически создаваемых при обращениях к представлению. Представление выполняет функцию фильтра базовых таблиц, на которые оно ссылается. Определяющий представление запрос может быть инициирован в одной или нескольких таблицах или в других представлениях текущей или других баз данных. Кроме того, для определения представлений с данными из нескольких разнородных источников можно использовать распределенные запросы. Это полезно, например, если нужно объединить структурированные подобным образом данные, относящиеся к разным серверам, каждый из которых хранит данные конкретного отдела организации.

Сборки.

Сборки являются файлами динамической библиотеки, которые используются в экземпляре SQL Server для развертывания функций, хранимых процедур, триггеров, определяемых пользователем статистических вычислений и определяемых пользователем типов.

Ограничения.

Ограничения позволяют задать метод, с помощью которого компонент СУБД Database Engine автоматически обеспечивает целостность базы данных. Ограничения задают правила допустимости определенных значений в столбцах и представляют собой стандартный механизм обеспечения целостности. Рекомендуется использовать ограничения, а не триггеры , правила и значения по умолчанию. Оптимизатор запросов также использует определения ограничений для построения высокопроизводительных планов выполнения запросов.

Правила.

Правила – еще один специальный механизм, предназначенный для обеспечения целостности базы данных, по функциональности напоминающие некоторые типы ограничений. Microsoft отмечает, что при соответствующей возможности использование ограничений по ряду причин предпочтительнее и, возможно, в будущей версии эта возможность будет удалена.

Значения по умолчанию.

Значения по умолчанию определяют, какими значениями заполнять столбец, если при вставке строки для этого столбца значение не указано. Значение по умолчанию могут быть любым выражением, результат которого — константа, например собственно константой, встроенной функцией или математическим выражением.

Архитектура базы данных. Физический уровень.

Физический уровень это представление данных в памяти ЭВМ. Основными понятиями, используемыми для представления структуры хранения (физического уровня) являются понятия файла (физического) и единицы обмена между внешней и оперативной памятью (физической записи или страницы). Рассмотрим, как представлены соответствующие понятия в СУБД Microsoft SQL Server 2008 (http://msdn.microsoft.com).

Файлы и файловые группы.

На физическом уровне база данных в Microsoft SQL Server 2008 представляется набором файлов операционной системы. Данные и сведения журналов транзакций всегда размещаются в разных файлах. Отдельные файлы используются только одной базой данных. Файловые группы представляют собой именованные коллекции файлов и используются для упрощения размещения данных и выполнения задач администрирования, например резервного копирования и восстановления.

Базы данных SQL Server содержат файлы трех типов:

- *Первичные файлы данных.*

Первичный файл данных является отправной точкой базы данных. Он указывает на остальные файлы базы данных. В каждой базе данных имеется один первичный файл данных. Для имени первичного файла данных рекомендуется использовать расширение MDF.

- *Вторичные файлы данных.*

Ко вторичным файлам данных относятся все файлы данных, за исключением первичного файла данных. Базы данных могут вообще не содержать вторичных файлов данных, или содержать один или несколько вторичных файлов данных. Для имени вторичного файла данных рекомендуется использовать расширение NDF.

- *Файлы журналов.*

Файлы журналов содержат все сведения журналов, используемые для восстановления базы данных. В каждой базе данных должен быть по меньшей мере один файл журнала, но их может быть и больше. Для имен файлов журналов рекомендуется использовать расширение MDF, NDF и LDF. Однако эти расширения помогают пользователю идентифицировать различные виды файлов и правильно их использовать.

В SQL Server расположение всех файлов базы данных записывается в первичный файл базы данных и в специальную служебную структуру СУБД SQL Server, называемою базой данных master. В большинстве случаев при работе с базой данных компонент СУБД (SQL Server Database Engine) использует сведения о размещении файлов, хранимые в базе данных master. Однако в некоторых случаях (например, при восстановлении базы данных master из копии, при определенным образом проводимым присоединении базы данных) компонент Database Engine использует сведения о расположении файлов из первичного файла, чтобы инициализировать записи о расположении файлов в базе данных master.

Файлы SQL Server имеют два имени:

- logical_file_name — имя, используемое для ссылки на физический файл во всех инструкциях Transact-SQL. Логическое имя файла должно соответствовать правилам для идентификаторов SQL Server и быть уникальным среди логических имен файлов в соответствующей базе данных.
- os_file_name — это имя физического файла, включая путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы.

Изначально можно указать максимальный размер каждого файла. Если максимальный размер файла не указан, файлы SQL Server могут автоматически увеличиваться в размерах, превосходя первоначально заданные показатели, пока не займет все доступное место на диске. При определении файла пользователь может указывать требуемый шаг роста. Каждый раз при заполнении файла его размер увеличивается на указанный шаг роста. Если в файловой группе имеется несколько файлов, их автоматический рост начинается лишь по заполнении всех файлов. Затем файлы увеличиваются в размерах по кольцевому списку. Эта функция особенно полезна в случаях, когда SQL Server используется в качестве базы данных, внедренной в приложение, где пользователь не имеет удобного доступа к системному администратору. По мере необходимости пользователь может предоставить файлам возможность увеличиваться в размерах автоматически, тем самым снимая с администратора часть забот по наблюдению за свободным пространством базы данных и по распределению дополнительного пространства вручную.

Из объектов баз данных и файлов можно формировать файловые группы, используемые для решения задач распределения и административного управления. Файлы журналов не могут входить в состав файловых групп. Управление пространством журнала отделено от управления пространством данных. Файл не может входить в состав нескольких файловых групп. Таблицы, индексы и данные больших объектов могут быть ассоциированы с указанной файловой группой. В этом случае все их страницы будут размещены внутри файловой группы; либо таблицы и индексы могут быть секционированы. Данные секционированных таблиц и индексов разделяются на блоки, каждый из которых может быть помещен в отдельную файловую группу базы данных. В каждой базе данных одна файловая группа назначается файловой группой по умолчанию. Если при создании таблицы или индекса файловая группа не указывается, предполагается, что все страницы будут распределяться из файловой группы по умолчанию. В каждый момент времени лишь одна файловая группа может быть файловой группой по умолчанию.

Страницы и экстенты.

Основной единицей хранилища данных и обмена информацией между внешней и оперативной памятью в SQL Server является страница. Место на диске, предоставляемое для размещения файла данных (MDF- или NDF-

файл) в базе данных, логически разделяется на страницы с непрерывным перечислением от 0 до n. Дисковые операции ввода-вывода выполняются на уровне страницы. А именно, SQL Server считывает или записывает целые страницы данных. В SQL Server размер страницы составляет 8 КБ. Это значит, что в одном мегабайте базы данных SQL Server содержится 128 страниц. Каждая страница начинается с 96-байтового заголовка, который используется для хранения системных данных о странице. Эти данные включают номер страницы, тип страницы, объем свободного места на странице и идентификатор единицы распределения объекта, которому принадлежит страница. В файлах данных базы данных SQL Server используется 8 типов страниц (данные с типами данных небольших размеров, данные с типами данных больших размеров, записи индекса, сведения о размещении экстентов, сведения о размещении страниц и доступном на них свободном месте и т. д.).

Для эффективного управления памятью страницы объединяются в экстенты , которые являются основными единицами организации пространства.

Экстент — это коллекция, состоящая из восьми физически непрерывных страниц или 64 Кб; они используются для эффективного управления страницами. Все страницы хранятся в экстентах. Таким образом, в одном мегабайте базы данных SQL Server содержится 16 экстентов.

Чтобы сделать распределение места эффективным, SQL Server не выделяет целые экстенты для таблиц с небольшим объемом данных. SQL Server имеет два типа экстентов:

Однородные экстенты принадлежат одному объекту (определенной таблице, индексу и т. д.); все восемь страниц могут быть использованы только этим владеющим объектом.

Смешанные экстенты могут находиться в общем пользовании у не более восьми объектов. Каждая из восьми страниц в экстенте может находиться во владении разных объектов.

Новая таблица или индекс — это обычно страницы, выделенные из смешанных экстентов. При увеличении размера таблицы или индекса до восьми страниц эти таблица или индекс переходят на использование однородных экстентов для последовательных единиц распределения. При создании индекса для существующей таблицы, в которой содержится достаточно строк, чтобы сформировать восемь страниц в индексе, все единицы распределения для индекса находятся в однородных экстентах. Пример размещения объектов в смешанном и однородном экстентах приводится на рис. 37.

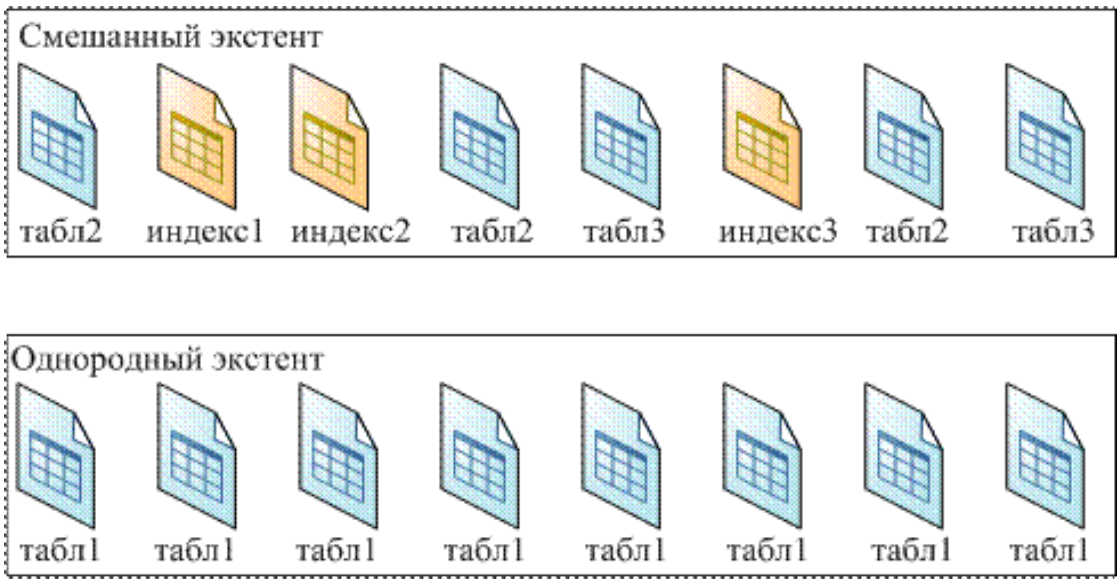


Рис. 37. Размещение объектов в смешанном и однородном экстентах

Страницы файлов данных.

Страницы файлов данных SQL Server нумеруются последовательно; первая страница файла получает нулевой номер (0). Каждый файл базы данных имеет уникальный цифровой идентификатор. Чтобы уникальным образом определить страницу базы данных, необходимо использовать как идентификатор файла, так и номер этой страницы. На рис. 38. показаны номера страниц базы данных, содержащей первичный файл данных объемом в 4 МБ и вторичный файл данных объемом в 1 МБ.

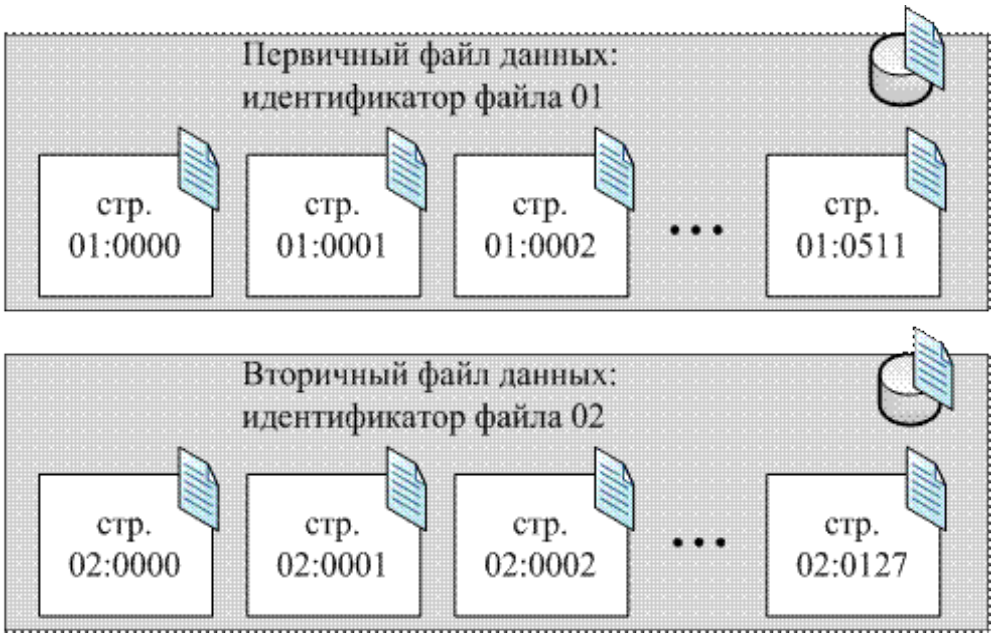


Рис. 38. Пример нумерации страниц файлов базы данных

Первая страница каждого файла (страница с номером 0) — это страница заголовка файла; она содержит сведения об атрибутах данного файла. Страницы с номерами 1,2,3 будут описаны ниже.

Организация таблиц и индексов.

Таблицы и индексы хранятся в виде коллекции страниц размером 8 КБ.

Страницы таблиц и индексов содержатся в одной или нескольких секциях. Секция — это пользовательская единица организации данных. По умолчанию таблица или индекс имеет единственную секцию, которая содержит все страницы таблицы или индекса. Секция располагается в одной файловой группе. Таблица или индекс, имеющие одну секцию, эквивалентны организационной структуре таблиц и индексов предыдущих версий SQL Server.

Если таблица или индекс используют несколько секций, данные секционируются горизонтально, так что группы строк сопоставляются отдельным секциям, основываясь на указанном столбце. Секции могут храниться в одной или нескольких файловых группах в базе данных. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений. Секция состоит из фрагментов одного или нескольких файлов. Данные внутри фрагмента файла представляются в виде кучи (строки данных хранятся без определенного порядка – последовательное размещение) или сбалансированного дерева. Фрагмент файла может иметь один из трех видов: данные с типами небольших размеров (данные IN_ROW_DATA), данные с типами больших размеров (LOB_DATA), данные переменной длины (переполнение строки ROW_OVERFLOW_DATA).

В каждой секции кучи или индекса содержится по крайней мере одна единица распределения IN_ROW_DATA. Кроме того, в зависимости от схемы кучи или индекса, там могут содержаться единицы распределения LOB_DATA или ROW_OVERFLOW_DATA.

Следующая иллюстрация показывает организацию таблицы (рис. 39).

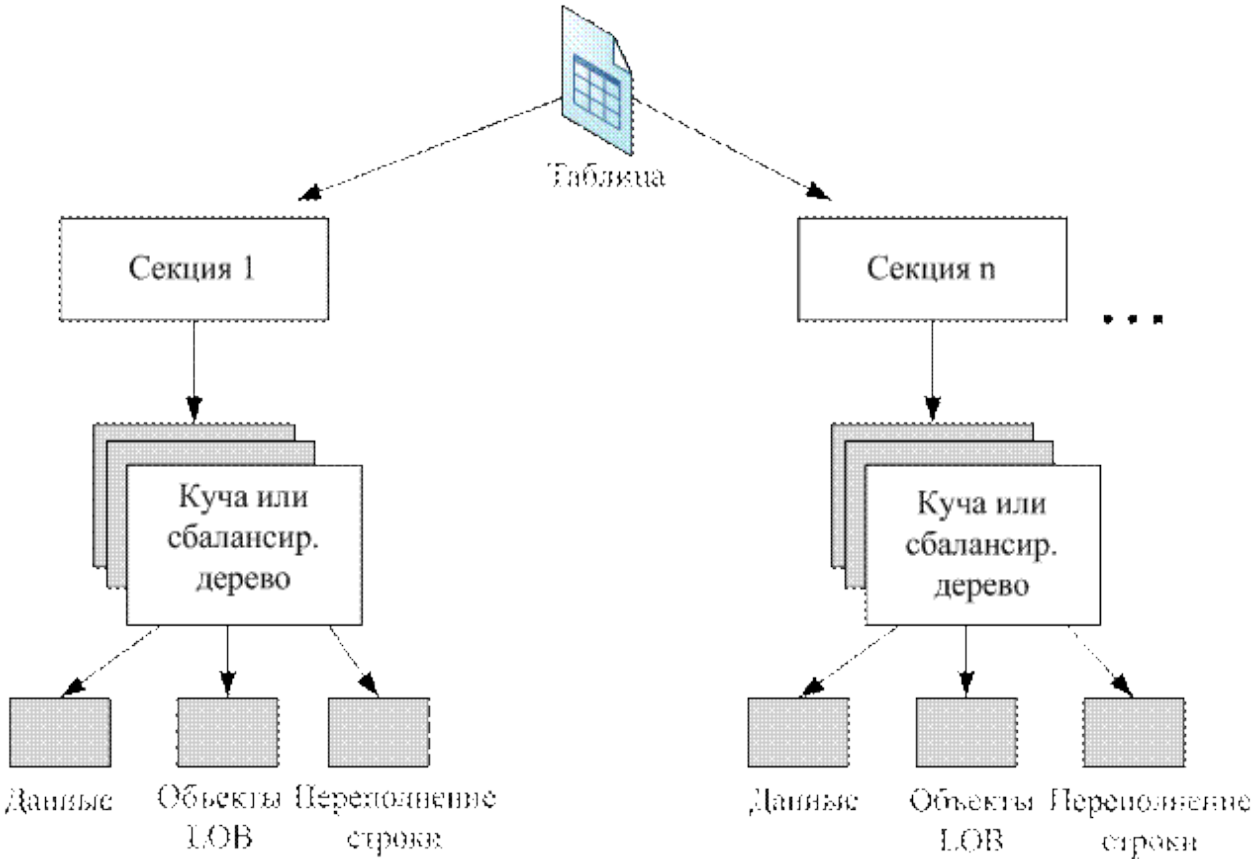


Рис. 39. Физическая структура таблицы в базе данных SQL Server

Каждая секция содержит строки данных либо в куче, либо в структуре кластеризованного индекса. Кластеризованный индекс реализуется в виде структуры индекса сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям. Страницы в каждом уровне индекса, включая страницы данных на конечном уровне, связаны в двунаправленный список. Однако перемещение из одного уровня на другой выполняется при помощи ключевых значений.

Куча — это последовательность строк таблицы, которые не имеют кластеризованного индекса. Строки данных хранятся без определенного порядка, и какой-либо порядок в последовательности страниц данных отсутствует. Страницы данных не связаны в связный список.

Управление работой с экстентами и свободным местом.

Структуры данных SQL Server, управляющие использованием экстента и отслеживанием свободного места, обладают относительно простой структурой. Сведения о свободном месте плотно упакованы, поэтому эти данные содержат относительно небольшое количество страниц. Это приводит к увеличению скорости из-за уменьшения необходимых операций чтения диска для получения сведений о размещении. Также увеличивается вероятность того, что страницы размещения будут оставаться в памяти и повторных операций чтения не потребуется. Большая часть сведений о размещении не связана по цепочке друг с другом. Это упрощает управление сведениями о размещении. Каждое действие по размещению или

освобождению страницы может выполняться быстро. Это сокращает конфликты между одновременными задачами использования и освобождения страниц.

SQL Server использует два типа карт для записи сведений об использовании экстентов:

- *Глобальная карта распределения (GAM)*

На GAM-страницах записано, какие экстенты были задействованы. В каждой карте GAM содержится сведения об использовании 64 000 экстентов или о размещении почти 4 ГБ данных. В карте GAM приходится по одному биту на каждый экстент в покрываемом им интервале. Если бит равен 1, то экстент свободен; если бит равен 0, то экстент задействован.

- *Общая глобальная карта распределения (SGAM)*

На SGAM-страницах записано, какие экстенты в текущий момент используются в качестве смешанных экстентов и имеют как минимум одну неиспользуемую страницу. В каждой карте SGAM содержится сведения об использовании 64 000 экстентов или о размещении почти 4 ГБ данных. В карте SGAM приходится по одному биту на каждый экстент в покрываемом им интервале. Если бит равен 1, то экстент используется как смешанный экстент и имеет свободную страницу. Если бит равен 0, то экстент не используется как смешанный экстент, или он является смешанным экстентом, но все его страницы используются.

Это дает простые алгоритмы управления экстентами страниц. Для использования для хранения объекта однородного экстента компонент СУБД Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0. Для поиска смешанного экстента со свободными страницами компонент Database Engine производит поиск на карте SGAM бита 1. Для размещения смешанного экстента компонент Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0, а затем устанавливает значение соответствующего бита на карте SGAM равным 1. Для освобождения экстента компонент Database Engine устанавливает бит GAM равным 1, а соответствующий бит SGAM равным 0. Внутренние алгоритмы, которые на самом деле используются компонентом Database Engine, более сложны, чем это описано в данном подразделе, так как компонент Database Engine распространяет данные в базе данных равномерно. Однако даже настоящие алгоритмы упрощаются из-за того, что отпадает необходимость управления цепочками сведений о размещении экстентов.

Отслеживание свободного места.

На страницы PFS (Page Free Space) записывается состояние размещения каждой страницы, информация о том, была ли отдельная страница использована или нет, а также количество свободного места на каждой странице. В PFS на каждую страницу приходится по одному байту, хранящему информацию о том, была ли страница использована или нет, а если была — то пустая она, или ее заполнение находится в промежутке от 1

до 50 процентов, от 51 до 80 процентов, от 81 до 95 процентов или от 96 до 100 процентов.

После размещения объекта в экстенте компонент Database Engine использует PFS-страницы для записи информации о том, какие страницы в экстенте использованы, а какие свободны. Эти сведения используются компонентом Database Engine при выборе новой страницы для размещения объектов. Количеством свободного места на странице можно управлять только в случае кучи и страниц с типами данных «Текст» и «Примечание». Это используется при поиске страницы, обладающей свободным местом, достаточным для сохранения в ней новой добавляемой строки. Для индексов не требуется, чтобы отслеживалось свободное место на странице, так как место, в которое будет вставляться новая строка, назначается значениями ключа индекса.

PFS-страница является первой страницей после страницы заголовка файла в файле данных (страница номер 1). Потом следует GAM-страница (страница номер 2), а затем SGAM-страница (страница номер 3). После первой PFS-страницы находится PFS-страница размером примерно 8 000 страниц. После первой GAM-страницы на странице 2 находится другая GAM-страница с 64 000 экстентов и другая SGAM-страница с 64 000 экстентов находится после первой SGAM-страницы на странице номер 3. На рис. 40. показана последовательность страниц, используемая компонентом Database Engine, для размещения и управления экстентами.

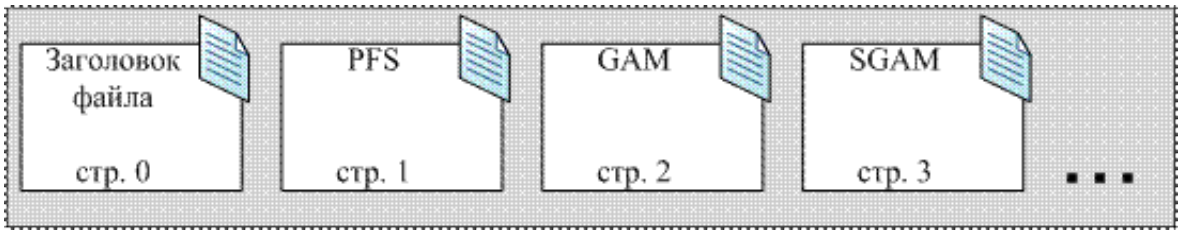


Рис. 40. Страницы файла, используемые для размещения и управления экстентами

Вопрос 3. Транзакции. Свойства транзакций. Журнал транзакций. Технология оперативной обработки транзакции (OLTP–технология).

Введение в транзакции.

Концепция транзакций – неотъемлемая часть любой клиент-серверной базы данных.

Под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), приводящая к одному из двух возможных результатов: либо последовательность выполняется, если все операторы правильные, либо вся транзакция откатывается, если хотя бы один оператор не может быть успешно выполнен. Обработка транзакций гарантирует целостность информации в базе данных. Таким

образом, транзакция переводит базу данных из одного целостного состояния в другое.

Поддержание механизма транзакций – показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности БД. Транзакции также составляют основу изолированности в многопользовательских системах, где с одной БД параллельно могут работать несколько пользователей или прикладных программ. Одна из основных задач СУБД – обеспечение изолированности, т.е. создание такого режима функционирования, при котором каждому пользователю казалось бы, что БД доступна только ему. Такую задачу СУБД принято называть параллелизмом транзакций.

Большинство выполняемых действий производится в теле транзакций. По умолчанию каждая команда выполняется как самостоятельная транзакция. При необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию. В частности, разработано четыре правила, известные как требования ACID, они гарантируют правильность и надежность работы системы.

ACID-свойства транзакций.

Характеристики транзакций описываются в терминах ACID (Atomicity, Consistency, Isolation, Durability – неделимость, согласованность, изолированность, устойчивость).

Транзакция неделима в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной транзакции. Если может быть выполнена лишь часть транзакции, она отклоняется.

Транзакция является согласованной, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.

Транзакция всегда изолированна, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций – это свойство называется сериализуемостью и означает, что транзакции в последовательности независимы.

Транзакция устойчива. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции) состояние, т.е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции.

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных алгоритмов обработки данных. Он решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций. Следует по возможности использовать небольшие транзакции, т.е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

Блокировки.

Повышение эффективности работы при использовании небольших транзакций связано с тем, что при выполнении транзакции сервер накладывает на данные блокировки.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена как на отдельную строку таблицы, так и на всю базу данных. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции и блокировки тесно связаны друг с другом. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции, т.е. происходит блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта. Различают два вида блокировки:

- блокировка записи – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен;
- блокировка чтения – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

В СУБД используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;
- транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;
- если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;

- блокировка записи сохраняется вплоть до конца выполнения транзакции.

Решение проблемы параллельной обработки БД заключается в том, что строки таблиц блокируются, а последующие транзакции, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если каждый сеанс взаимодействия с базой данных реализуется транзакцией, то пользователь начинает с того, что обращается к согласованному состоянию базы данных – состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Если в системе управления базами данных не реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могут возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т.к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема «грязного» чтения возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема неповторяемого чтения является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности;
- проблема чтения фантомов появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть

изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- уровень 0 – запрещение «загрязнения» данных.

Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;

- уровень 1 – запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;
- уровень 2 – запрещение неповторяемого чтения.

Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;

- уровень 3 – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей.

Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Управление транзакциями.

Под управлением транзакциями понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД. Прежде всего, имеется в виду выполнение операторов INSERT, UPDATE и DELETE. Например, после создания таблицы (выполнения оператора CREATE TABLE) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с помощью отмены транзакции не удастся восстановить только что удаленную оператором DROP TABLE таблицу.

После успешного выполнения команд, заключенных в тело одной транзакции, немедленного изменения данных не происходит. Для окончательного завершения транзакции существуют так называемые команды управления транзакциями, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить.

Существуют три команды, которые используются для управления транзакциями:

COMMIT – для сохранения изменений;

ROLLBACK – для отмены изменений;

SAVEPOINT – для установки особых точек возврата.

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не

будет выполнена одна из команд управления транзакциями. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Команда COMMIT предназначена для сохранения в базе данных всех изменений, произошедших в ходе выполнения транзакции. Она сохраняет результаты всех операций, которые имели место после выполнения последней команды COMMIT или ROLLBACK.

Команда ROLLBACK предназначена для отмены транзакций, еще не сохраненных в базе данных. Она отменяет только те транзакции, которые были выполнены с момента выдачи последней команды COMMIT или ROLLBACK.

Команда SAVEPOINT (точка сохранения) предназначена для установки в транзакции особых точек, куда в дальнейшем может быть произведен откат (при этом отката всей транзакции не происходит). Команда имеет следующий вид:

```
SAVEPOINT имя_точки_сохранения
```

Она служит исключительно для создания точек сохранения среди операторов, предназначенных для изменения данных. Имя точки сохранения в связанной с ней группе транзакций должно быть уникальным.

Для отмены действия группы транзакций, ограниченных точками сохранения, используется команда ROLLBACK со следующим синтаксисом:

```
ROLLBACK TO имя_точки_сохранения
```

Поскольку с помощью команды SAVEPOINT крупное число транзакций может быть разбито на меньшие и поэтому более управляемые группы, ее применение является одним из способов управления транзакциями.

Управление транзакциями в среде MS SQL Server.

Определение транзакций.

SQL Server предлагает множество средств управления поведением транзакций. Пользователи в основном должны указывать только начало и конец транзакции, используя команды SQL или API (прикладного интерфейса программирования). Транзакция определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение транзакции, то это ему не удастся. Когда транзакция начинается, все команды, выполненные в соединении, считаются телом одной транзакции, пока не будет достигнут ее конец.

SQL Server поддерживает три вида определения транзакций:

- явное;
- автоматическое;

- подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, когда каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других.

Для установки режима автоматического определения транзакций используется команда:

```
SET IMPLICIT_TRANSACTIONS  OFF
```

При работе в режиме неявного (подразумевающегося) начала транзакций SQL Server автоматически начинает новую транзакцию, как только завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством другой команды:

```
SET IMPLICIT_TRANSACTIONS  ON
```

Явные транзакции.

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

- начало транзакции: в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начала транзакции;

```
BEGIN TRAN[SACTION]
  [имя_транзакции |
  @имя_переменной_транзакции
[WITH MARK [‘описание_транзакции’]]]
```

- конец транзакции: если в теле транзакции не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в транзакции, после чего в журнале транзакций помечается, что изменения зафиксированы и транзакция завершена;

```
COMMIT [TRAN[SACTION]
  [имя_транзакции |
  @имя_переменной_транзакции]]
```

- создание внутри транзакции точки сохранения: СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения;

```
SAVE TRAN[SACTION]
  {имя_точки_сохранеия |
  @имя_переменной_точки_сохранения}
```

- прерывание транзакции; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена. Приведенная ниже команда отменяет все изменения, сделанные в БД после оператора BEGIN TRANSACTION или отменяет изменения, сделанные в БД после точки сохранения, возвращая транзакцию к месту, где был выполнен оператор SAVE TRANSACTION.

```
ROLLBACK [TRAN[SACTION]
  [имя_транзакции |
  @имя_переменной_транзакции
  | имя_точки_сохранения
 |@имя_переменной_точки_сохранения]]
```

Функция @@TRANCOUNT возвращает количество активных транзакций.

Функция @@NESTLEVEL возвращает уровень вложенности транзакций.

Пример. Использование точек сохранения

```
BEGIN TRAN
SAVE TRANSACTION  point1
```

В точке point1 сохраняется первоначальное состояние таблицы Товар

```
DELETE FROM Товар WHERE КодТовара=2
  SAVE TRANSACTION  point2
```

В точке point2 сохраняется состояние таблицы Товар без товаров с кодом 2.

```
DELETE FROM Товар WHERE КодТовара=3
  SAVE TRANSACTION  point3
```

В точке point3 сохраняется состояние таблицы Товар без товаров с кодом 2 и с кодом 3.

```
DELETE FROM Товар WHERE КодТовара<>1
  ROLLBACK TRANSACTION point3
```

Происходит возврат в состояние таблицы без товаров с кодами 2 и 3, отменяется последнее удаление.

```
SELECT * FROM Товар
```

Оператор SELECT покажет таблицу Товар без товаров с кодами 2 и 3.

```
ROLLBACK TRANSACTION point1
```

Происходит возврат в первоначальное состояние таблицы.

```
SELECT * FROM Товар
COMMIT
```

Первоначальное состояние сохраняется.

Вложенные транзакции.

Вложенными называются транзакции, выполнение которых инициируется из тела уже активной транзакции.

Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций. Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с требованиями ACID произойдет откат всех транзакций всех уровней, включая успешно завершенные. Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Каждая команда COMMIT TRANSACTION работает только с последней начатой транзакцией. При завершении вложенной транзакции команда COMMIT применяется к наиболее «глубокой» вложенной транзакции. Даже если в команде COMMIT TRANSACTION указано имя транзакции более высокого уровня, будет завершена транзакция, начатая последней.

Если команда ROLLBACK TRANSACTION используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня. В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда

происходит откат всех транзакций. Если же требуется откатить лишь часть транзакций, можно использовать команду SAVE TRANSACTION , с помощью которой создается точка сохранения.

Пример. Вложенные транзакции.

```
BEGIN TRAN
INSERT Товар (Название, остаток)
VALUES ('v',40)
  BEGIN TRAN
  INSERT Товар (Название, остаток)
  VALUES ('n',50)
  BEGIN TRAN
   INSERT Товар (Название, остаток)
   VALUES ('m',60)
ROLLBACK TRAN
```

Здесь происходит возврат на начальное состояние таблицы, поскольку выполнение команды ROLLBACK TRAN без указания имени транзакции откатывает все транзакции.

Блокировки в среде MS SQL Server.

Управление блокировками.

Пользователю чаще всего не нужно предпринимать никаких действий по управлению блокировками. Всю работу по установке, снятию и разрешению конфликтов выполняет специальный компонент сервера, называемый менеджером блокировок. MS SQL Server поддерживает различные уровни блокирования объектов (или детализацию блокировок), начиная с отдельной строки таблицы и заканчивая базой данных в целом. Менеджер блокировок автоматически оценивает, какое количество данных необходимо блокировать, и устанавливает соответствующий тип блокировки. Это позволяет поддерживать равновесие между производительностью работы системы блокирования и возможностью пользователей получать доступ к данным. Блокирование на уровне строки позволяет наиболее точно управлять таким доступом, поскольку блокируются только действительно изменяемые строки. Множество пользователей могут одновременно работать с данными с минимальными задержками. Платой за это является увеличение числа операций установки и снятия блокировок, а также большое количество служебной информации, которое приходится хранить для отслеживания установленных блокировок. При блокировке на уровне таблицы производительность системы блокирования резко увеличивается, так как необходимо установить лишь одну блокировку и снять ее только после завершения транзакции. Пользователь при этом имеет максимальную скорость доступа к данным. В то же время они не доступны никому другому, потому что вся таблица заблокирована. Приходится ожидать, пока текущий пользователь завершит работу.

Действия, выполняемые пользователями при работе с данными, сводятся к операциям двух типов: их чтению и изменению. В операции по

изменению включаются действия по добавлению, удалению и собственно изменению данных. В зависимости от выполняемых действий сервер накладывает определенный тип блокировки из следующего перечня:

Коллективные блокировки. Они накладываются при выполнении операций чтения данных (например, SELECT). Если сервер установил на ресурс коллективную блокировку, то пользователь может быть уверен, что уже никто не сможет изменить эти данные.

Блокировка обновления. Если на ресурс установлена коллективная блокировка и для этого ресурса устанавливается блокировка обновления, то никакая транзакция не сможет наложить коллективную блокировку или блокировку обновления.

Монопольная блокировка. Этот тип блокировок используется, если транзакция изменяет данные. Когда сервер устанавливает монопольную блокировку на ресурс, то никакая другая транзакция не может прочитать или изменить заблокированные данные. Монопольная блокировка не совместима ни с какими другими блокировками, и ни одна блокировка, включая монопольную, не может быть наложена на ресурс.

Блокировка массивного обновления. Накладывается сервером при выполнении операций массивного копирования в таблицу и запрещает обращение к таблице любым другим процессам. В то же время несколько процессов, выполняющих массивное копирование, могут одновременно вставлять строки в таблицу.

Помимо перечисленных основных типов блокировок SQL Server поддерживает ряд специальных блокировок, предназначенных для повышения производительности и функциональности обработки данных. Они называются блокировками намерений и используются сервером в том случае, если транзакция намеревается получить доступ к данным вниз по иерархии и для других транзакций необходимо установить запрет на наложение блокировок, которые будут конфликтовать с блокировкой, накладываемой первой транзакцией.

Ранее рассмотренные блокировки относятся к данным. Помимо перечисленных в среде SQL Server существует два других типа блокировок: блокировка диапазона ключей и блокировка схемы (метаданных, описывающих структуру объекта).

Блокировка диапазона ключей решает проблему возникновения фантомов и обеспечивает требования сериализуемости транзакции. Блокировки этого типа устанавливаются на диапазон строк, соответствующих определенному логическому условию, с помощью которого осуществляется выборка данных из таблицы.

Блокировка схемы используется при выполнении команд модификации структуры таблиц для обеспечения целостности данных.

«Мертвые» блокировки.

«Мертвые», или тупиковые, блокировки характерны для многопользовательских систем. «Мертвая» блокировка возникает, когда две транзакции блокируют два блока данных и для завершения любой из них

нужен доступ к данным, заблокированным ранее другой транзакцией. Для завершения каждой транзакции необходимо дождаться, пока блокированная другой транзакцией часть данных будет разблокирована. Но это невозможно, так как вторая транзакция ожидает разблокирования ресурсов, используемых первой.

Без применения специальных механизмов обнаружения и снятия «мертвых» блокировок нормальная работа транзакций будет нарушена. Если в системе установлен бесконечный период ожидания завершения транзакции (а это задано по умолчанию), то при возникновении «мертвой» блокировки для двух транзакций вполне возможно, что, ожидая освобождения заблокированных ресурсов, в тупике окажутся и новые транзакции. Чтобы избежать подобных проблем, в среде MS SQL Server реализован специальный механизм разрешения конфликтов тупикового блокирования.

Для этих целей сервер снимает одну из блокировок, вызвавших конфликт, и откатывает инициализировавшую ее транзакцию. При выборе блокировки, которой необходимо пожертвовать, сервер исходит из соображений минимальной стоимости.

Полностью избежать возникновения «мертвых» блокировок нельзя. Хотя сервер и имеет эффективные механизмы снятия таких блокировок, все же при написании приложений следует учитывать вероятность их возникновения и предпринимать все возможные действия для предупреждения этого. «Мертвые» блокировки могут существенно снизить производительность, поскольку системе требуется достаточно много времени для их обнаружения, отката транзакции и повторного ее выполнения.

Для минимизации возможности образования «мертвых» блокировок при разработке кода транзакции следует придерживаться следующих правил:

- выполнять действия по обработке данных в постоянном порядке, чтобы не создавать условия для захвата одних и тех же данных;
- избегать взаимодействия с пользователем в теле транзакции;
- минимизировать длительность транзакции и выполнять ее по возможности в одном пакете;
- применять как можно более низкий уровень изоляции.

Уровни изоляции SQL Server.

Уровень изоляции определяет степень независимости транзакций друг от друга. Наивысшим уровнем изоляции является сериализуемость, обеспечивающая полную независимость транзакций друг от друга. Каждый последующий уровень соответствует требованиям всех предыдущих и обеспечивает дополнительную защиту транзакций.

SQL Server поддерживает все четыре уровня изоляции, определенные стандартом ANSI. Уровень изоляции устанавливается командой:

```
SET TRANSACTION ISOLATION LEVEL
```

```
{ READ COMMITTED
| READ UNCOMMITTED
| REPEATABLE READ
| SERIALIZABLE }
```

READ UNCOMMITED – незавершенное чтение, или допустимо черновое чтение. Низший уровень изоляции, соответствующий уровню 0. Он гарантирует только физическую целостность данных: если несколько пользователей одновременно изменяют одну и ту же строку, то в окончательном варианте строка будет иметь значение, определенное пользователем, последним изменившим запись. По сути, для транзакции не устанавливается никакой блокировки, которая гарантировала бы целостность данных. Для установки этого уровня используется команда:

```
SET TRANSACTION ISOLATION
LEVEL READ UNCOMMITTED
```

READ COMMITTED – завершенное чтение, при котором отсутствует черновое, «грязное» чтение. Тем не менее в процессе работы одной транзакции другая может быть успешно завершена и сделанные ею изменения зафиксированы. В итоге первая транзакция будет работать с другим набором данных. Это проблема неповторяемого чтения. Данный уровень изоляции установлен в SQL Server по умолчанию и устанавливается посредством команды:

```
SET TRANSACTION ISOLATION
LEVEL READ COMMITTED
```

REPEATABLE READ – повторяющееся чтение. Повторное чтение строки возвратит первоначально считанные данные, несмотря на любые обновления, произведенные другими пользователями до завершения транзакции. Тем не менее на этом уровне изоляции возможно возникновение фантомов. Его установка реализуется командой:

```
SET TRANSACTION ISOLATION
LEVEL REPEATABLE READ
```

SERIALIZABLE – сериализуемость. Чтение запрещено до завершения транзакции. Это максимальный уровень изоляции, который обеспечивает полную изоляцию транзакций друг от друга. Он устанавливается командой:

```
SET TRANSACTION ISOLATION
LEVEL SERIALIZABLE
```

В каждый момент времени возможен только один уровень изоляции.

Таблица 13.

Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). В примере шаги 4, 6 и 8 демонстрируют черновое чтение. Шаги 9 и 10 блокируются, потому что данные захвачены конкурирующей транзакцией.	
Пользователь user1 Конкурирующая **транзакция**	Пользователь user2 Текущая **транзакция**
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4	4.SELECT * FROM Товар (читает измененные неподтвержденные данные)
5.DELETE FROM Товар WHERE КодТовара=4	6.SELECT * FROM Товар (читает измененные неподтвержденные данные)
7.INSERT Товар (Название, остаток) VALUES ('SS',999)	8.SELECT * FROM Товар (читает измененные неподтвержденные данные)
12.ROLLBACK TRANSACTION TRA	9.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
10.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)	11.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
13.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT	

Таблица 14.

Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). В примере шаги 4, 6 и 8 демонстрируют блокировку данных, захваченных другой транзакцией, в то время как работа с другими данными разрешается (шаг 10).	
Пользователь user1 Конкурирующая **транзакция**	Пользователь user2 Текущая **транзакция**
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL READ COMMITTED BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар
3.UPDATE Товар SET остаток=остаток+10 (захватывает данные)	4.SELECT * FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
5.DELETE FROM Товар WHERE КодТовара=4	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
7.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется той транзакцией, которая первой захватила данные на изменение или удаление)	8.DELETE FROM Товар WHERE КодТовара=4 (блокируется до окончания конкурирующей транзакции)
9.INSERT Товар (Название, остаток) VALUES ('SS',999)	10.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
11.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	12.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Таблица 15.

Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). На шаге 2 транзакция захватила данные чтением и блокирует работу с ними со стороны конкурирующей транзакции (шаги 3, 5), которая может лишь добавлять записи (шаг 7).	
Пользователь user1 Конкурирующая **транзакция**	Пользователь user2 Текущая **транзакция**
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар (захватывает данные)
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется)	4.SELECT * FROM Товар (блокируется до окончания конкурирующей транзакции)
5.DELETE FROM Товар WHERE КодТовара=4 (блокируется)	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
7.INSERT Товар (Название, остаток) VALUES ('SS',999) (выполняется)	8.DELETE FROM Товар WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
10.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	9.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
	11.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Таблица 16.

Уровень изоляции конкурирующей транзакции принят по умолчанию (READ COMMITTED). Пример демонстрирует, что текущая транзакция захватила данные чтением (шаг 2) и блокирует любые действия с ними со стороны конкурирующей транзакции вплоть до вставки данных (шаг 7).	
Пользователь user1 Конкурирующая **транзакция**	Пользователь user2 Текущая **транзакция**
USE basa_user2 BEGIN TRANSACTION TRA	USE basa_user2 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE BEGIN TRANSACTION TRB
1.SELECT * FROM Товар	2.SELECT * FROM Товар (захватывает данные)
3.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (блокируется)	4.SELECT * FROM Товар (выполняется)
5.DELETE FROM Товар WHERE КодТовара=4 (блокируется)	6.UPDATE Товар SET остаток=остаток+10 WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
7.INSERT Товар (наименование, остаток) VALUES ('SS',999) (блокируется)	8.DELETE FROM Товар WHERE КодТовара=4 (выполняется, т.к. данные захвачены текущей транзакцией)
10.ROLLBACK TRANSACTION TRA SELECT @@TRANCOUNT	9.INSERT Товар(Название, остаток) VALUES ('SS',999) (выполняется)
	11.ROLLBACK TRANSACTION TRB SELECT @@TRANCOUNT

Тема 6. Информационные хранилища и склады данных

Вопрос 1. Хранилища данных.[17]

Хранилище данных (Data Warehouse) - предметно - ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений.

Хранилище данных содержит непротиворечивые консолидированные исторические данные и предоставляет инструментальные средства для их анализа с целью поддержки принятия стратегических решений. Информационные ресурсы хранилища данных формируются на основе фиксируемых на протяжении продолжительного периода времени моментальных снимков баз данных оперативной информационной системы и, возможно, различных внешних источников. В хранилищах данных применяются технологии баз данных, OLAP, глубинного анализа данных, визуализации данных.

Основные характеристики хранилищ данных.

- содержит исторические данные;
- хранит подробные сведения, а также частично и полностью обобщенные данные;
- данные в основном являются статическими;
- нерегламентированный, неструктурированный и эвристический способ обработки данных;
- средняя и низкая интенсивность обработки транзакций;
- непредсказуемый способ использования данных;
- предназначено для проведения анализа;
- ориентировано на предметные области;
- поддержка принятия стратегических решений;
- обслуживает относительно малое количество работников руководящего звена.

Термин OLAP (On-Line Analytical Processing) служит для описания модели представления данных и соответственно технологии их обработки в хранилищах данных. В OLAP применяется многомерное представление агрегированных данных для обеспечения быстрого доступа к стратегически важной информации в целях углубленного анализа. Приложения OLAP должны обладать следующими основными свойствами:

- многомерное представление данных;
- поддержка сложных расчетов;
- правильный учет фактора времени.

Преимущества OLAP:

- повышение производительности производственного персонала, разработчиков прикладных программ. Своевременный доступ к стратегической информации.
- предоставление пользователям достаточных возможностей для внесения собственных изменений в схему.

- приложения OLAP опираются на хранилища данных и системы OLTP, получая от них актуальные данные, что дает сохранение контроля целостности корпоративных данных.
- уменьшение нагрузки на системы OLTP и хранилища данных.

Вопрос 2. OLAP и OLTP. Характеристики и основные отличия.[18]

Таблица 17.

OLAP	OLTP
Хранилище данных должно включать как внутренние корпоративные данные, так и внешние данные	основным источником информации, поступающей в оперативную БД, является деятельность корпорации, а для проведения анализа данных требуется привлечение внешних источников информации (например, статистических отчетов)
Объем аналитических БД как минимум на порядок больше объема оперативных. для проведения достоверных анализа и прогнозирования в хранилище данных нужно иметь информацию о деятельности корпорации и состоянии рынка на протяжении нескольких лет	Для оперативной обработки требуются данные за несколько последних месяцев
Хранилище данных должно содержать единообразно представленную и согласованную информацию, максимально соответствующую содержанию оперативных БД. Необходима компонента для извлечения и «очистки» информации из разных источников. Во многих крупных корпорациях одновременно существуют несколько оперативных ИС с собственными БД (по историческим причинам).	Оперативные БД могут содержать семантически эквивалентную информацию, представленную в разных форматах, с разным указанием времени ее поступления, иногда даже противоречивую
Набор запросов к аналитической базе данных предсказать невозможно. хранилища данных существуют, чтобы отвечать на нерегламентированные запросы аналитиков. Можно рассчитывать только на то, что запросы будут поступать не слишком часто и затрагивать большие объемы информации. Размеры аналитической БД стимулируют использование запросов с агрегатами (сумма, минимальное, максимальное, среднее значение и т.д.)	Системы обработки данных создаются в расчете на решение конкретных задач. Информация из БД выбирается часто и небольшими порциями. Обычно набор запросов к оперативной БД известен уже при проектировании
При малой изменчивости аналитических БД (только при загрузке данных) оказываются разумными упорядоченность массивов, более быстрые методы индексации при массовой выборке, хранение заранее агрегированных данных	Системы обработки данных по своей природе являются сильно изменчивыми, что учитывается в используемых СУБД (нормализованная структура БД, строки хранятся неупорядоченно, B-деревья для индексации, транзакционность)
Информация аналитических БД настолько критична для корпорации, что требуются большая грануляция защиты (индивидуальные права доступа к определенным строкам и/или столбцам таблицы)	Для систем обработки данных обычно хватает защиты информации на уровне таблиц

Правила Кодда для OLAP систем.

В 1993 году Кодд опубликовал труд под названием «OLAP для пользователей-аналитиков: каким он должен быть». В нем он изложил

основные концепции оперативной аналитической обработки и определил 12 правил, которым должны удовлетворять продукты, предоставляющие возможность выполнения оперативной аналитической обработки.

Концептуальное многомерное представление. OLAP-модель должна быть многомерной в своей основе. Многомерная концептуальная схема или пользовательское представление облегчают моделирование и анализ так же, впрочем, как и вычисления.

Прозрачность. Пользователь способен получить все необходимые данные из OLAP-машины, даже не подозревая, откуда они берутся. Вне зависимости от того, является OLAP-продукт частью средств пользователя или нет, этот факт должен быть незаметен для пользователя. Если OLAP предоставляется клиент-серверными вычислениями, то этот факт также, по возможности, должен быть невидим для пользователя. OLAP должен предоставляться в контексте истинно открытой архитектуры, позволяя пользователю, где бы он ни находился, связываться при помощи аналитического инструмента с сервером. В дополнение к этому прозрачность должна достигаться и при взаимодействии аналитического инструмента с гомогенной и гетерогенной средами БД.

Доступность. OLAP должен предоставлять свою собственную логическую схему для доступа в гетерогенной среде БД и выполнять соответствующие преобразования для предоставления данных пользователю. Более того, необходимо заранее позаботиться о том, где и как, и какие типы физической организации данных действительно будут использоваться. OLAP-система должна выполнять доступ только к действительно требующимся данным, а не применять общий принцип «кухонной воронки», который влечет ненужный ввод.

Постоянная производительность при разработке отчетов. Производительность формирования отчетов не должна существенно падать с ростом количества измерений и размеров базы данных.

Клиент-серверная архитектура. Требуется, чтобы продукт был не только клиент-серверным, но и чтобы серверный компонент был бы достаточно интеллектуальным для того, чтобы различные клиенты могли подключаться с минимумом усилий и программирования.

Общая многомерность. Все измерения должны быть равноправны, каждое измерение должно быть эквивалентно и в структуре, и в операционных возможностях. Правда, допускаются дополнительные операционные возможности для отдельных измерений (видимо, подразумевается время), но такие дополнительные функции должны быть предоставлены любому измерению. Не должно быть так, чтобы базовые структуры данных, вычислительные или отчетные форматы были более свойственны какому-то одному измерению.

Динамическое управление разреженными матрицами. OLAP системы должны автоматически настраивать свою физическую схему в зависимости от типа модели, объемов данных и разреженности базы данных.

Многопользовательская поддержка. OLAP-инструмент должен предоставлять возможности совместного доступа (запроса и дополнения), целостности и безопасности.

Неограниченные перекрестные операции. Все виды операций должны быть дозволены для любых измерений.

Интуитивная манипуляция данными. Манипулирование данными осуществлялось посредством прямых действий над ячейками в режиме просмотра без использования меню и множественных операций.

Гибкие возможности получения отчетов. Измерения должны быть размещены в отчете так, как это нужно пользователю.

Неограниченная размерность и число уровней агрегации. Исследование о возможном числе необходимых измерений, требующихся в аналитической модели, показало, что одновременно может использоваться до 19 измерений. Отсюда вытекает настоятельная рекомендация, чтобы аналитический инструмент был способен одновременно предоставить как минимум 15 измерений, а предпочтительнее 20. Более того, каждое из общих измерений не должно быть ограничено по числу определяемых пользователем-аналитиком уровней агрегации и путей консолидации.

Основные элементы и операции OLAP.

В основе OLAP лежит понятие гиперкуба, или многомерного куба данных, в ячейках которого хранятся анализируемые данные.

Факт - это числовая величина которая располагается в ячейках гиперкуба. Один OLAP-куб может обладать одним или несколькими показателями.

Измерение (dimension) - это множество объектов одного или нескольких типов, организованных в виде иерархической структуры и обеспечивающих информационный контекст числового показателя. Измерение принято визуализировать в виде ребра многомерного куба.

Объекты, совокупность которых и образует измерение, называются членами измерений (members). Члены измерений визуализируют как точки или участи, откладываемые на осях гиперкуба.

Ячейка (cell) - атомарная структура куба, соответствующая полному набору конкретный значений измерений.

Иерархия - группировка объектов одного измерения в объекты более высокого уровня. Например - день-месяц-год. Иерархии в измерениях необходимы для возможности агрегации и детализации значений показателей согласно их иерархической структуре. Иерархия целиком основывается на одном измерении и формируется из уровней.

В OLAP-системах поддерживаются следующие базовые операции:

- поворот;
- проекция. При проекции значения в ячейках, лежащих на оси проекции, суммируются по некоторому предопределенному закону;
- раскрытие (drill-down). Одно из значений измерения заменяется совокупностью значений из следующего уровня иерархии измерения; соответственно заменяются значения в ячейках гиперкуба;

- свертка (roll-up/drill-up). Операция, обратная раскрытию;
- сечение (slice-and-dice).

Типы OLAP. Преимущества и недостатки.

Выбор способа хранения данных зависит от объема и структуры детальных данных, требований к скорости выполнения запросов и частоты обновления OLAP-кубов. В настоящее время применяются три способа хранения данных:

MOLAP (Multidimensional OLAP).

Детальные и агрегированные данные хранятся в многомерной базе данных. Хранение данных в многомерных структурах позволяет манипулировать данными как многомерным массивом, благодаря чему скорость вычисления агрегатных значений одинакова для любого из измерений. Однако в этом случае многомерная база данных оказывается избыточной, так как многомерные данные полностью содержат детальные реляционные данные.

Преимущества MOLAP.

Высокая производительность. Поиск и выборка данных осуществляется значительно быстрее, чем при многомерном концептуальном взгляде на реляционную базу данных.

Структура и интерфейсы наилучшим образом соответствуют структуре аналитических запросов.

Многомерные СУБД легко справляются с задачами включения в информационную модель разнообразных встроенных функций.

Недостатки MOLAP.

MOLAP могут работать только со своими собственными многомерными БД и основываются на патентованных технологиях для многомерных СУБД, поэтому являются наиболее дорогими. Эти системы обеспечивают полный цикл OLAP-обработки и либо включают в себя, помимо серверного компонента, собственный интегрированный клиентский интерфейс, либо используют для связи с пользователем внешние программы работы с электронными таблицами.

По сравнению с реляционными, очень неэффективно используют внешнюю память, обладают худшими по сравнению с реляционными БД механизмами транзакций.

Отсутствуют единые стандарты на интерфейс, языки описания и манипулирования данными.

Не поддерживают репликацию данных, часто используемую в качестве механизма загрузки.

ROLAP (Relational OLAP).

ROLAP-системы позволяют представлять данные, хранимые в классической реляционной базе, в многомерной форме или в плоских локальных таблицах на файл-сервере, обеспечивая преобразование информации в многомерную модель через промежуточный слой метаданных.

Агрегаты хранятся в той же БД в специально созданных служебных таблицах. В этом случае гиперкуб эмулируется СУБД на логическом уровне.

Преимущества ROLAP.

Реляционные СУБД имеют реальный опыт работы с очень большими БД и развитые средства администрирования. При использовании ROLAP размер хранилища не является таким критичным параметром, как в случае MOLAP.

При оперативной аналитической обработке содержимого хранилища данных инструменты ROLAP позволяют производить анализ непосредственно над хранилищем (потому что в подавляющем большинстве случаев корпоративные хранилища данных реализуются средствами реляционных СУБД).

В случае переменной размерности задачи, когда изменения в структуру измерений приходится вносить достаточно часто, ROLAP системы с динамическим представлением размерности являются оптимальным решением, так как в них такие модификации не требуют физической реорганизации БД, как в случае MOLAP.

Системы ROLAP могут функционировать на гораздо менее мощных клиентских станциях, чем системы MOLAP, поскольку основная вычислительная нагрузка в них ложится на сервер, где выполняются сложные аналитические SQL-запросы, формируемые системой.

Реляционные СУБД обеспечивают значительно более высокий уровень защиты данных и хорошие возможности разграничения прав доступа.

Недостатки ROLAP.

Ограниченные возможности с точки зрения расчета значений функционального типа.

Меньшая производительность, чем у MOLAP. Для обеспечения сравнимой с MOLAP производительности реляционные системы требуют тщательной проработки схемы БД и специальной настройки индексов. Но в результате этих операций производительность хорошо настроенных реляционных систем при использовании схемы «звезда» сравнима с производительностью систем на основе многомерных БД.

HOLAP (Hybrid OLAP).

Детальные данные остаются в той же реляционной базе данных, где они изначально находились, а агрегатные данные хранятся в многомерной базе данных.

Вопрос 3. Моделирование многомерных кубов на реляционной модели данных.

Схема звезда. Преимущества и недостатки.

Схема типа звезды (Star Schema) - схема реляционной базы данных, служащая для поддержки многомерного представления содержащихся в ней данных.

Особенности ROLAP-схемы типа «звезда»

Одна таблица фактов (fact table), которая сильно денормализована. Является центральной в схеме, может состоять из миллионов строк и содержит суммируемые или фактические данные, с помощью которых можно ответить на различные вопросы.

Несколько денормализованных таблиц измерений (dimensional table). Имеют меньшее количество строк, чем таблицы фактов, и содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации.

Таблица фактов и таблицы размерности связаны идентифицирующими связями, при этом первичные ключи таблицы размерности мигрируют в таблицу фактов в качестве внешних ключей. Первичный ключ таблицы факта целиком состоит из первичных ключей всех таблиц размерности.

Агрегированные данные хранятся совместно с исходными.

Преимущества.

Благодаря денормализации таблиц измерений упрощается восприятие структуры данных пользователем и формулировка запросов, уменьшается количество операций соединения таблиц при обработке запросов. Некоторые промышленные СУБД и инструменты класса OLAP / Reporting умеют использовать преимущества схемы «звезда» для сокращения времени выполнения запросов.

Недостатки.

Денормализация таблиц измерений вносит избыточность данных, возрастает требуемый для их хранения объем памяти. Если агрегаты хранятся совместно с исходными данными, то в измерениях необходимо использовать дополнительный параметр - уровень иерархии.

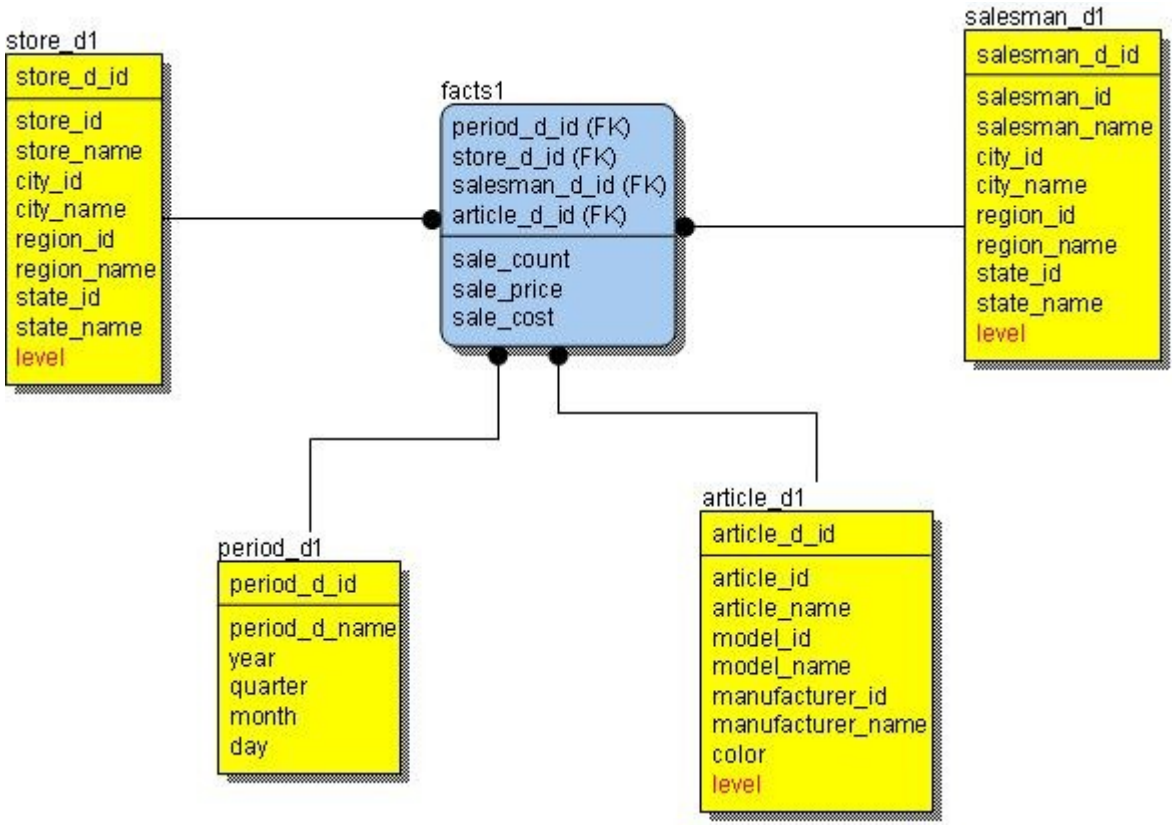


Рис. 41

Схема снежинка. Преимущества и недостатки.

Схема типа снежинки (Snowflake Schema) - схема реляционной базы данных, служащая для поддержки многомерного представления содержащихся в ней данных, является разновидностью схемы типа «звезда» (Star Schema).

Особенности ROLAP-схемы типа «снежинка»

Одна таблица фактов (fact table), которая сильно денормализована. Является центральной в схеме, может состоять из миллионов строк и содержать суммируемые или фактические данные, с помощью которых можно ответить на различные вопросы.

Несколько таблиц измерений (dimensional table), которые нормализованы в отличие от схемы «звезда». Имеют меньшее количество строк, чем таблицы фактов, и содержат описательную информацию. Эти таблицы позволяют пользователю быстро переходить от таблицы фактов к дополнительной информации. Первичные ключи в них состоят из единственного атрибута (соответствуют единственному элементу измерения).

Таблица фактов и таблицы размерности связаны идентифицирующими связями, при этом первичные ключи таблицы размерности мигрируют в таблицу фактов в качестве внешних ключей. Первичный ключ таблицы факта целиком состоит из первичных ключей всех таблиц размерности.

В схеме «снежинка» агрегированные данные могут храниться отдельно от исходных.

Преимущества.

Нормализация таблиц измерений в отличие от схемы «звезда» позволяет минимизировать избыточность данных и более эффективно выполнять запросы, связанные со структурой значений измерений.

Недостатки.

За нормализацию таблиц измерений иногда приходится платить временем выполнения запросов.

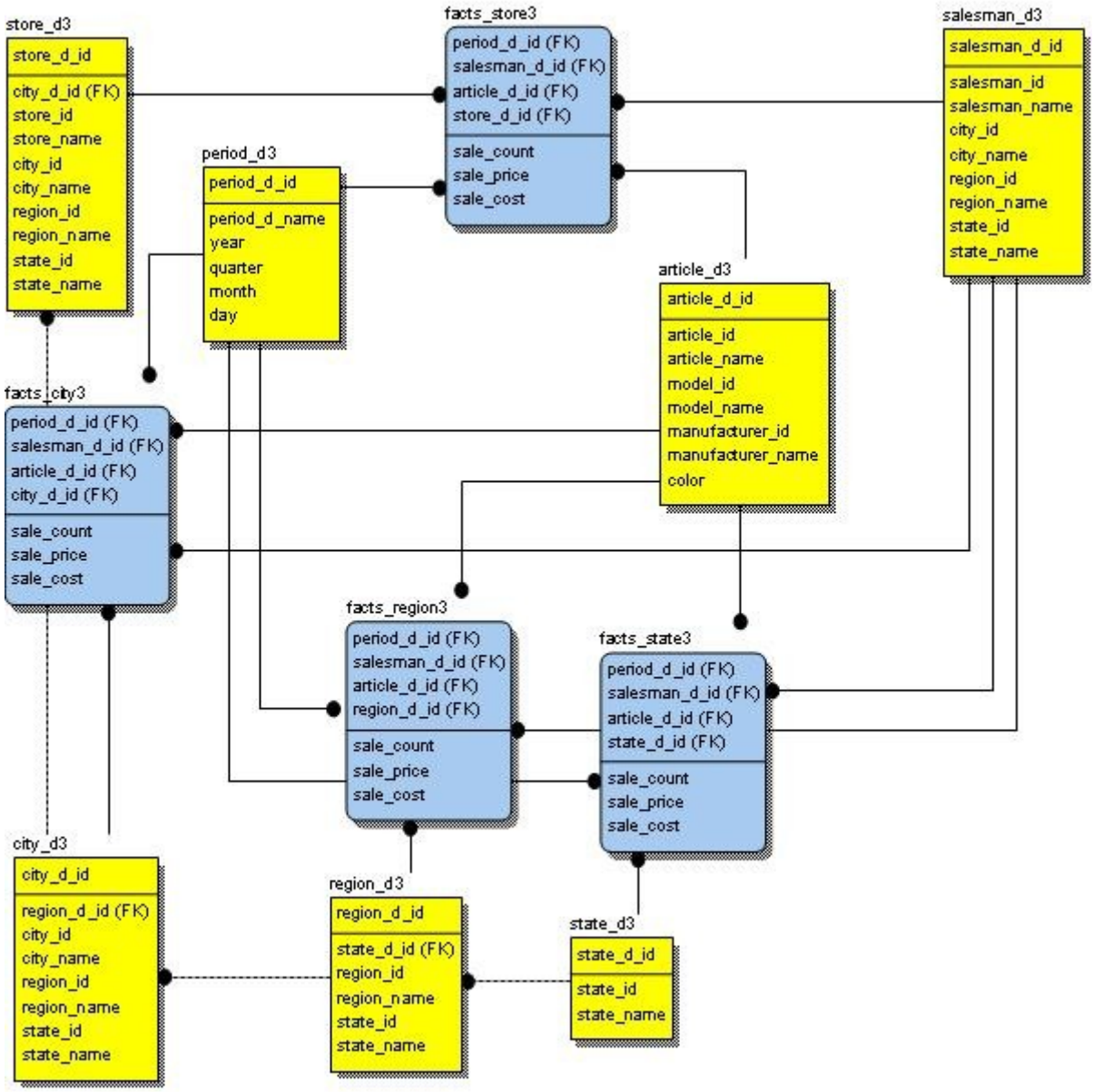


Рис. 42

Вопрос 4. Склады данных. [19]

Склад данных - это логически интегрированный источник данных для систем поддержки принятия решений (DSS) и информационных систем руководителя (EIS). Мы говорим о логической интегрированности, потому что, хотя некоторые простые модели складов данных, которые мы обсудим далее, могут быть централизованными (независимо от того, распределенными или централизованными являются источники данных для них).

Склад данных и база данных – это не одно и то же. Склад данных может быть реализован на основе некоторой СУБД или РаСУБД. Поскольку предназначение склада данных - это информационная поддержка принятия

решений, а не оперативная обработка данных и транзакции, то многие принципы технологий баз данных утрачивают для них свое значение.

Склад данных ориентирован на определенную предметную область и организован на основе некоторого подмножества данных, поступающих из операционных баз данных. Источником информации для них являются различные приложения, которые могут выполняться на разных платформах, следовательно, необходимы средства интеграции. Кроме того, на складе данных хранится не все множество строк данных (как в операционной базе), а в той или иной степени обобщенная информация.

Данные, поступившие на склад, приобретают статус постоянной информации, то есть вносимые изменения носят характер «пополнения» (путем регулярных плановых выборок из операционных баз), а не произвольных поэлементных модификаций, как в операционных базах данных.

Процесс пополнения обычно включает сложные процедуры согласования данных в отношении типов, размеров, кодировок и других свойств данных. Для этих целей пригодны алгоритмы, аналогичные тем, которые применяются в среде разнородных распределенных баз данных, построенных по методике «снизу вверх».

Над складом данных, наполненным исходной информацией, может выполняться множество разнообразных приложений поддержки принятия решений и информационных систем руководителя. В таких приложениях применимы процедуры последовательного уточнения, то есть продвижения на уровни более тонкой детализации. Возможности складов данных полезны в областях, связанных с долговременным управлением информацией, таких как электронные библиотеки и хранилища данных.

Одна из нерешенных критически важных проблем для многих руководителей корпораций и правительственных учреждений, ответственных за принятие решений, - невозможность получения оперативных, консолидированных своевременных и гибких отчетов на основе корпоративных данных. Руководителям необходима точная и актуальная информация, которая, в большинстве случаев, присутствует на предприятии, но ее извлечение из многих разрозненных операционных баз и хранилищ данных сопряжено с трудностями. Технология складов данных дает адекватную основу для создания систем поддержки принятия решений и информационных систем руководителя.

Информация из среды оперативного информационного управления (как правило, из одной или более баз данных) извлекается в соответствии с определенными принципами и помещается в склад данных. Важно понимать, оперативная обработка данных и транзакции не является целью создания складов данных и многие принципы технологий баз данных утрачивают для них свое значение. В частности, в складах данных не поддерживаются операции модификации данных в том смысле, как это понимается в базах данных.

Ниже перечислены четыре основополагающих для организации складов данных принципа.

Предметная ориентация. В операционной базе данных обычно поддерживается несколько предметных областей, каждая из которых может послужить источником данных для склада. Например, для магазина, торгующего видео- и музыкальной продукцией, интерес представляют следующие предметные области: клиенты; видеокассеты; CD-диски и аудиокассеты; сотрудники; поставщики.

Нетрудно провести аналогию между предметными областями склада данных и классами объектов в объектно-ориентированных базах данных. Очевидно, что методы проектирования и моделирования, применяемые в объектно-ориентированных СУБД, могут оказаться полезными и при проектировании предметных областей складов данных.

Средства интеграции. Существует ряд проблем при интеграции данных, возникающие при построении глобальных схем для баз данных. Например, одна и та же сущность в разных базах данных и приложениях может быть представлена совершенно по-разному, и все эти представления должны быть приведены к некоторому общему типу.

Постоянство данных. В складах данных не поддерживаются операции модификации в смысле традиционных баз данных. Для разных окружений баз данных характерные различны степени изменяемости данных. Так, в реляционных базах данных допускаются вставки строк, изменения значений столбцов, удаления строк, выполняемые регулярно в процессе обычной деятельности. В складах данных поддерживается модель «массовых загрузок» данных, производимых в заданные моменты времени согласно установленным правилам. Массовая загрузка может выполняться с централизованной базы данных, находящейся на той же системе, что и склад данных, а может осуществляться и путем одновременных извлечений данных из распределенных операционных баз данных (либо даже из разрозненных баз данных или информационных систем, не объединенных какой-либо глобальной схемой). Таким образом, модель индивидуальных модификаций объектов к складам данных неприменима.

Хронологизм данных. Благодаря средствам интеграции, склад данных - это нечто большее, чем просто изощренная последовательность «моментальных снимков»; она всегда имеет определенный хронологический, временной аспект, присущий ее содержимому. Хронологизм - принцип, гласящий, что время есть ключевой компонент базы данных и ее содержимого, в той же мере можно отнести и к складам данных.

«Мгновенные снимки» операционных данных извлекаются из баз данных или других информационных источников и поступают в склад данных в интегрированном виде. Однако склад данных не поддерживает тот же высокий уровень гранулярности информации, который характерен для баз данных.

Рассмотрим разницу между операционными приложениями и приложениями поддержки принятия решений или информационной

поддержки руководителя. Операционным приложениям, для того чтобы они могли выполнять свои функции, нужен максимально высокий уровень гранулярности. В базе данных должна быть представлена информация о каждом клиенте, сотруднике, поставщике, каждом компакт-диске, каждой видеокассете и т. п.

Пользователей систем DSS или EIS, напротив, вряд ли будет интересовать список всех клиентов или полный отчет о прокате или продажах каждого CD. Менеджеру-аналитику высокого ранга не понадобится даже месячный отчет с подобной информацией. Скорее всего будет необходим отчет о среднем месячном объеме проката и продаж в расчете на одного клиента с детализацией по коду города в пределах того или иного региона, охваченного деятельностью компании. Понадобится также сравнение такого рода информации с данными за прошлый месяц, либо за тот же месяц прошлого и/или позапрошлого года.

Ключ к эффективности приложений склада данных - это обобщение информации. Возможно, приложения DSS или EIS могли бы самостоятельно проводить обобщение подробной информации, выбираемой из операционных баз данных или из складов данных (где она поддерживается с той же высокой степенью гранулярности). Но это абсолютно непрактично, по крайней мере, по одной из двух причин: дополнительная нагрузка на операционную базу данных, создаваемая в результате выполнения многочисленных выборок и обобщений данных, приведет к снижению производительности на текущих операциях; неоправданное увеличение объема необходимой памяти в складе данных для хранения элементов данных, которые никогда не используются индивидуально.

В складе данных, разумеется, можно поддерживать несколько уровней гранулярности (т. е. более или менее обобщенную информацию), что очень желательно для проведения анализа «вглубь» (drill-down analysis). Анализ «вглубь» полезен в тех случаях, когда пользователь обнаруживает интересное для него явление и стремится докопаться до его причин, истоков и подробностей. Если в складе данных поддерживаются связи от каждого уровня обобщения к уровням, «питающим» его, то соответствующие приложения могут производить обход данных по этим ссылкам.

Вопрос 5. Архитектуры хранилищ данных.[20]

Централизованное хранилище данных с ETL.

Виртуальные хранилища данных и независимые витрины показали, что для эффективной работы аналитических систем необходим единый репозитарий данных. Для наполнения этого репозитория необходимо извлечь, согласовать разнородные данные из различных источников и загрузить эти данные в репозиторий.

Средства извлечения, преобразования и загрузки данных (ETL) должны знать все об источниках данных: структуры хранящихся данных и их форматы, различия в алгоритмах обработки данных, смысл хранящихся

данных, график выполнения обработки информации в транзакционных системах. Игнорирование этих данных о данных (метаданных) неизбежно приводит к ухудшению качества информации, загружаемой в хранилище. В результате пользователи теряют доверие к хранилищу данных, стараются получать информацию напрямую из источников, что приводит к неоправданным временным затратам специалистов, эксплуатирующих системы – источники данных.

Таким образом, информация об источниках данных должна использоваться средствами ETL. Поэтому средства ETL должны работать в тесной связке со средствами ведения метаданных.

При обработке извлеченных данных необходимо преобразовать их к единому виду. Поскольку основные данные хранятся в реляционных базах данных, нужно учесть различие в кодировке данных. Даты могут кодироваться в разных формата; адреса могут использовать различные сокращения; кодировка продуктов может следовать различным номенклатурам. Первоначально информация о нормативно справочной информации (НСИ) заносилась в алгоритмы преобразования данных ETL. По мере роста числа источников данных объема обрабатываемых данных (он может достигать терабайтов в сутки), стало ясно, что необходимо отделить средства управления НСИ от средств ETL, и обеспечить их эффективное взаимодействие.

Таким образом, средства ETL извлекают данные из источников, во взаимодействии со средствами ведения метаданных и НСИ преобразуют их к требуемым форматам и загружают в репозиторий данных. В качестве репозитория чаще всего выступает репозиторий хранилища данных, но также может быть и оперативный склад данных (ОСД), и зоны временного хранения, и даже витрины данных. Поэтому одним из ключевых требований к средствам ETL является их способность взаимодействовать с различными системами.

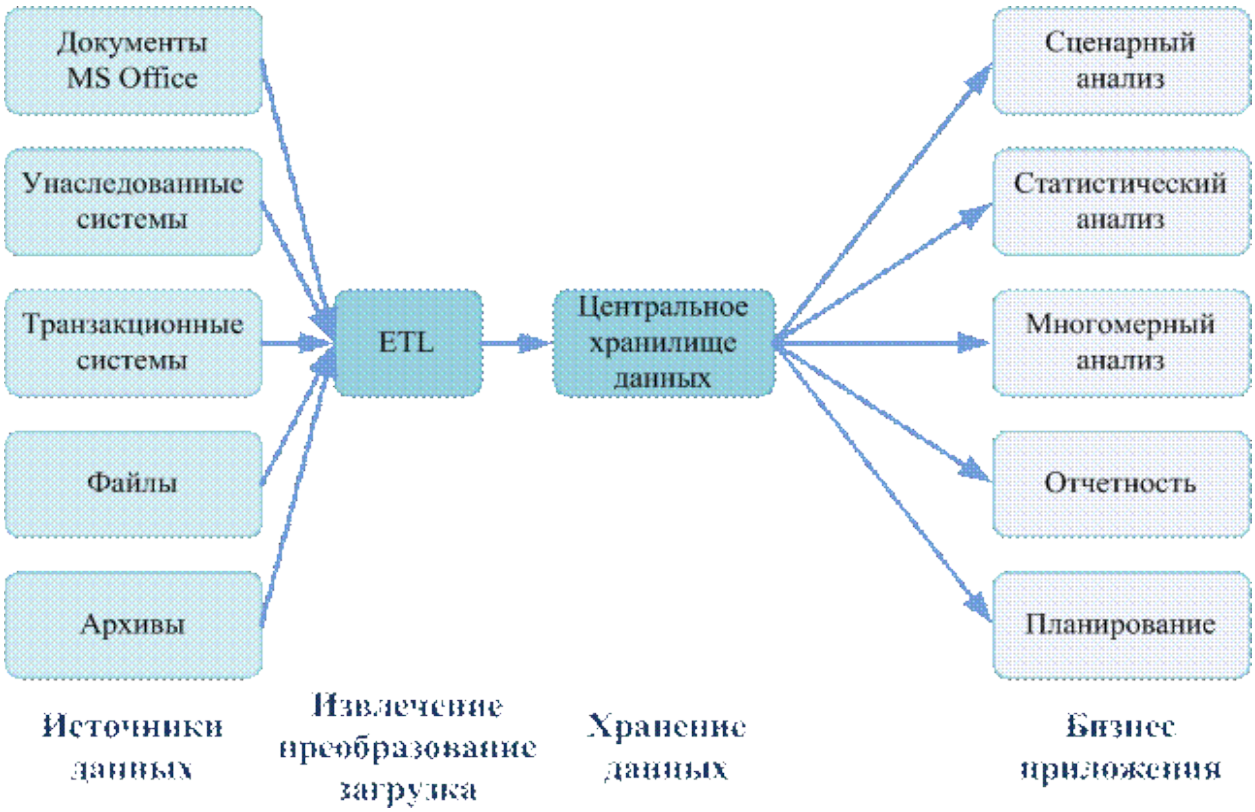


Рис. 43. Централизованное хранилище данных с ETL

Необходимость повышения оперативности предоставляемой аналитической информации и рост объемов обрабатываемых данных выставляют повышенные требования к производительности средств ETL и их масштабируемости. Поэтому средства ETL должны использовать различные схемы параллельных вычислений и уметь работать на высокопроизводительных системах различных архитектур.

Как видно, к средствам ETL предъявляются самые разные требования:

- необходимо собрать данные от разных систем – источников, даже если одна или несколько систем в результате сбоя не смогли в срок завершить свою работу и предоставить необходимые данные;
- полученная информация должна быть распознана и преобразована в соответствии с алгоритмами преобразования, а также с помощью систем ведения НСИ и метаданных;
- преобразованная информация должна быть загружена в зоны временного хранения, в хранилище данных, в ОСД, в витрины данных, как того требует производственный процесс;
- средства ETL должны иметь высокую пропускную способность с тем, чтобы собирать и выгружать все возрастающие объемы данных;
- средства ETL должны обладать высокими вычислительными возможностями и масштабируемостью для сокращения времени обработки данных для уменьшения задержек в предоставлении данных для аналитических работ;

- средства ETL должны предоставлять разнообразные инструменты извлечения данных в различных режимах работы – от пакетного сбора для систем, некритичных к временным задержкам, до инкрементальной обработки в режиме, близком к реальному времени.

В связи с этими, зачастую взаимоисключающими требованиями, проектирование и разработка средств ETL превращается в сложную задачу даже тогда, когда используются решения, предлагаемые на рынке.

Централизованное хранилище данных с ELT.

Традиционную систему извлечения, преобразования и загрузки данных (ETL) нередко упрекают в низкой производительности и высокой стоимости из-за необходимости создания выделенного программно-аппаратного комплекса. В качестве альтернативы предлагаются средства извлечения, загрузки и преобразования данных (ELT), которым приписываются высокая производительность и эффективное использование оборудования.

С тем, чтобы понять, каковы сравнительные преимущества и недостатки систем ETL и ELT, обратимся к трем основным функциям корпоративного хранилища данных (КХД):

- полный и своевременный сбор и обработка информации от источников данных;
- надежное и защищенное хранение данных;
- предоставление данных для аналитических работ.

На вход систем ETL / ELT поступают разнородные данные, которые необходимо сравнить, очистить, привести к единым форматам, обработать по требуемым вычислительным алгоритмам. С одной стороны, в системах ETL / ELT данные практически не задерживаются, с другой – через эти системы в хранилище втекает основной поток информации. Поэтому требования к обеспечению защиты информации могут быть умеренными.

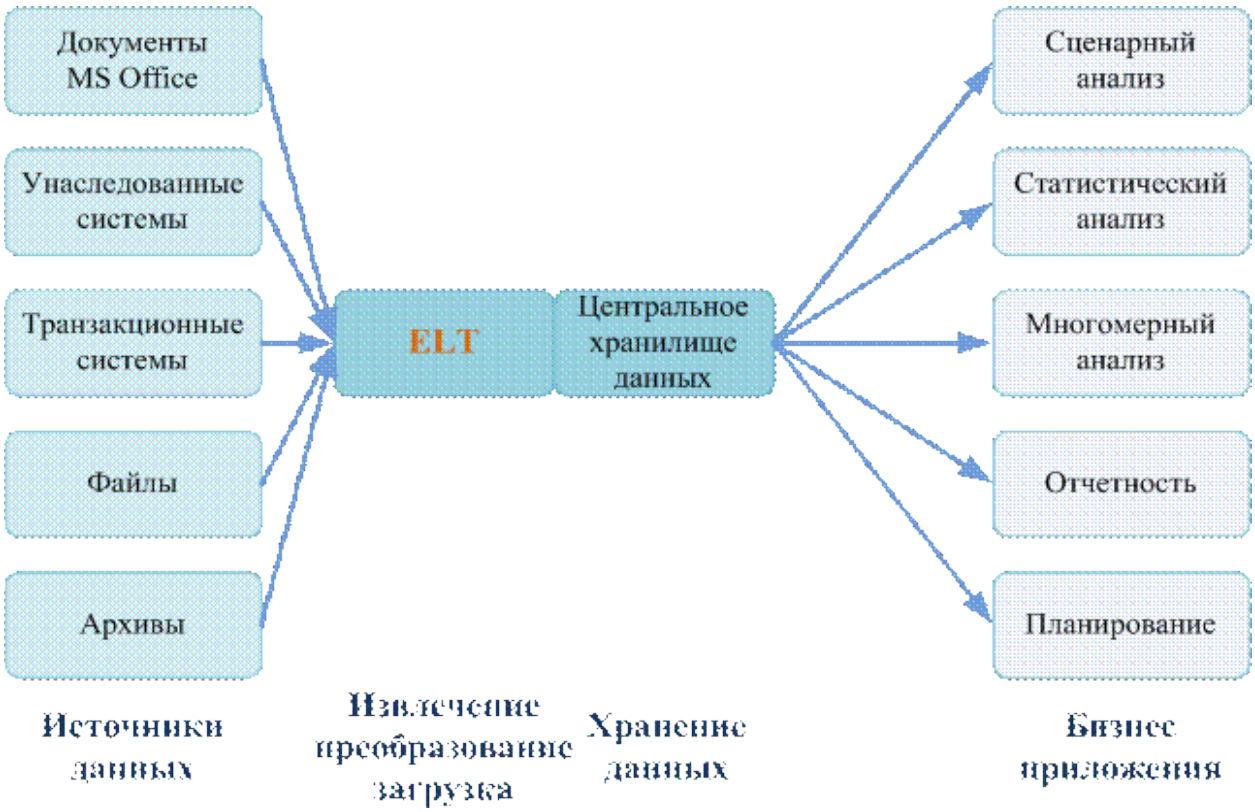


Рис. 44. Централизованное хранилище данных с ELT

Центральное хранилище данных (ЦХД), как правило, содержит такой объем информации, что ее полное раскрытие может привести к серьезным потерям для компании. В этом случае ЦХД требует создания вокруг себя надежного периметра информационной безопасности. Структуры данных в хранилище должны быть оптимизированы под требования долговременного, надежного и защищенного хранения. Применение схемы ELT означает, что ЦХД должно осуществлять и трансформацию данных.

Предоставление данных для аналитических работ требует реорганизации структур данных под каждую специфическую задачу. Многомерный анализу необходимы кубы данных; статистический анализ, как правило, работает с рядами данных; сценарный анализ и моделирование могут использовать файлы MS Excel. В рассматриваемой архитектуре бизнес - приложения используют данные непосредственно из ЦХД. В такой архитектуре в ЦХД должны храниться данные в структурах, оптимизированных как под текущие, так и под будущие бизнес – приложения. Более того, подобный прямой доступ повышает вероятность несанкционированного доступа ко всем данным в хранилище.

Таким образом, мы видим, что в данной архитектуре на ЦХД возложены функции трансформации данных и обслуживания аналитических приложений. Обе эти функции несвойственны ЦХД, которое в таком виде превращается в устройство «все в одном», в котором, как правило, составляющие компоненты хуже, чем если бы они были реализованы отдельно (например, фотоаппарат в мобильном телефоне).

Как решается вопрос разделения функций хранения данных и предоставления данных для аналитических приложений, мы рассмотрим позже.

Применение схемы ETL позволяет полностью разнести функции обработки и хранения данных. Схема ELT нагружает центральное хранилище данных несвойственными ей функциями преобразования данных. В результате переноса функциональности от ETL в ЦХД нам необходимо не только обеспечить ту же вычислительную производительность, но и спроектировать универсальную платформу, способную равно эффективно обрабатывать данные и хранить их. Этот подход, может быть, применим для сегмента SOHO, но для корпоративных решений требуются профессиональные устройства.

Несмотря на декларируемые преимущества производительности схемы ELT, на практике выясняется, что

- Качество данных влияет на время их загрузки. Например, ETL при очистке и преобразовании данных может отбрасывать до 90% повторяющихся данных. ELT в этом случае загрузит все данные в ЦХД, где и будет происходить очистка.
- Скорость преобразования данных в хранилище сильно зависит от алгоритмов обработки и структур данных. В некоторых случаях более эффективна SQL – обработка внутри базы данных хранилища, в других – быстрее будут работать внешние программы, извлекающие данные для обработки и загружающие результаты обработки в хранилище.
- Некоторые алгоритмы очень сложно реализовать, используя средства SQL. Это накладывает ограничения на использование схемы ELT, тогда как ETL может использовать более эффективные инструменты обработки данных
- ETL является единой областью, где сконцентрированы правила извлечения, обработки и загрузки данных, что упрощает эксплуатацию, доработку и тестирование алгоритмов. ELT, напротив, разносит алгоритмы сбора и загрузки с алгоритмами преобразования данных. То есть, для тестирования новых алгоритмов преобразования нужно либо рисковать целостностью данных в хранилище, находящемся в промышленном производстве, либо создавать тестовую копию хранилища, что является весьма дорогостоящим мероприятием.

Таким образом, сравнивая ETL и ELT, мы видим, что преимущества при загрузке и преобразовании данных неочевидны, что ELT сталкивается с ограничениями SQL при преобразовании данных, и что экономия на программно - аппаратном комплексе ELT приводит к финансовым затратам на создание программно-аппаратной тестовой копии ЦХД.

Применение ELT, возможно, оправдано, если:

- нет жестких требований к надежности, производительности и защищенности хранилища;
- бюджетные ограничения вынуждают идти на риск утраты данных;

- хранилище данных и источники данных взаимодействуют через сервисную шину (SOA).

Последний случай наиболее экзотичен, но и он имеет право на существование в определенных условиях. В этом случае на шину возложена интеграция источников с ХД на уровне обмена сообщениями, и минимальное (по меркам хранилища) преобразование данных и их загрузка хранилище.

Централизованное ХД с ОСД.

Процессы извлечения, преобразования и загрузки данных, безусловно, требуют некоторого времени для завершения своей работы. Дополнительная задержка вызвана необходимостью проверки загруженных в хранилище данных на непротиворечивость с уже имеющимися данными, на консолидацию данных, на перевычисления итоговых значений с учетом новых данных.

Оперативный склад данных (ОСД) был предложен в 1998 г. с тем, чтобы сократить время задержки между поступлением информации из ETL и аналитическими системами. Операционный склад данных располагает менее точной информацией из-за отсутствия внутренних проверок, и более детальными данными из-за отсутствия этапа консолидации данных. Поэтому данные из ОСД предназначены для принятия тактических решений, тогда как информация из центрального хранилища данных (ЦХД) лучше подходит для решения стратегических задач.

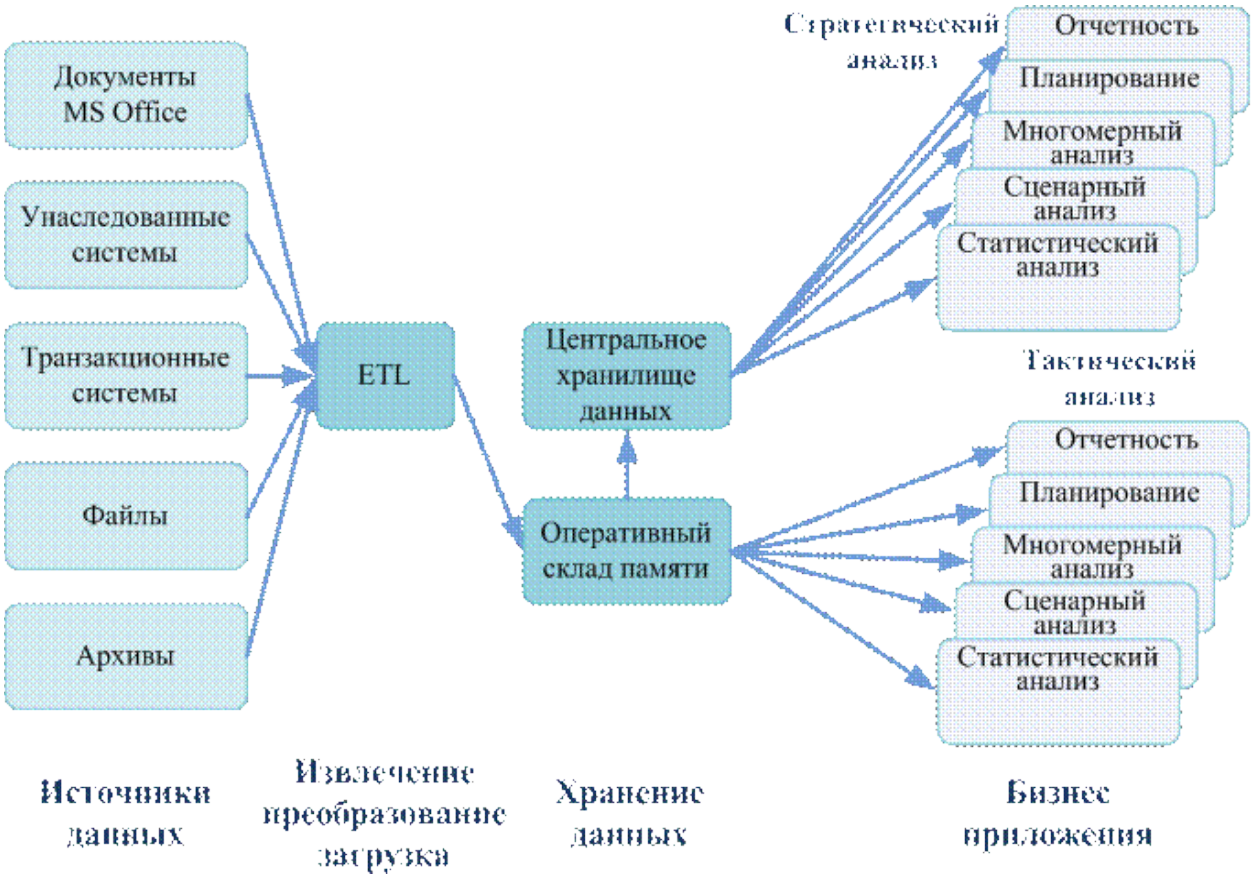


Рис. 45. Централизованное ХД с ОСД

Представим себе компанию, которая занимается продажей напитков и пакетиков с орешками через торговые автоматы на территории всей страны. Простой пустого автомата в течение 15 мин. означает потерю возможной прибыли, поэтому важно своевременно отслеживать состояние автомата и заполнять его отсутствующим товаром. Сбор и обработка всей информации в масштабе всей страны может потребовать несколько часов, тогда как развоз продуктов осуществляется локально – в каждом крупном населенном пункте есть склад, откуда продукты развозятся по ближайшим торговым точкам. С другой стороны, заполнение складов осуществляется через централизованные закупки. Таким образом, есть два различных типа задач тактического (заполнение торговых автоматов) и стратегического планирования (заполнение складов).

Действительно, если в результате неполных и неточных данных, содержащихся в ОСД, будет привезена лишняя бутылка воды, то это не приведет к серьезным потерям. Однако ошибка планирования из-за низкого качества данных в ЦХД может оказать негативное воздействие на принятие решения о номенклатуре и объемах оптовых закупок.

Требования к защите информации в ОСД и ЦХД также различаются. В нашем примере в ОСД размещаются данные с горизонтом времени, не превышающим нескольких часов. В ЦХД может храниться историческая информация, охватывающая период в несколько лет для более надежного прогнозирования необходимых объемов закупок. И эта историческая информация может представлять значительный коммерческий интерес для конкурентов. Поэтому аналитики-тактики могут напрямую работать с ОСД, тогда как аналитики-стратеги должны работать с ЦХД через витрины данных для разграничения ответственности. Отсутствие витрин данных помогает тактикам быстрее получить доступ к данным. Наличие витрин данных не препятствует стратегическому анализу, так как такой анализ осуществляется ежемесячно, или даже ежеквартально.

Архитектура, представленная на рис. 45, предполагает прямую работу бизнес-приложений с ЦХД. Разбор достоинств и ограничений подобного подхода будет выполнен в разделе «Расширенная модель ХД с витринами данных». Сейчас необходимо отметить, что при последовательном перемещении данных ОСД фактически выполняет еще одну роль зоны временного хранения. Аналитики-тактики, работая с данными из ОСД, вольно или невольно выявляют ошибки и противоречия в данных, тем самым, повышая их качество.

Исправленные данные из ОСД в данной схеме перегружаются в ЦХД. Однако возможны и иные схемы, например, когда данные из ETL поступают одновременно в ОСД и ЦХД. После использования в ОСД ненужные данные просто стираются. Эта схема применима в тех случаях, когда человеческое вмешательство в данные может только исказить их, вольно, или невольно.

Расширенная модель ХД с витринами данных.

Прямая работа пользовательских программ с корпоративным хранилищем данных (КХД), допустима, если пользовательские запросы не препятствуют нормальному функционированию КХД, если между пользователями и КХД имеются высокоскоростные линии связи, или если случайный доступ ко всем данным не ведет к серьезным потерям. Администрирование прямого доступа пользователей к КХД представляет собой чрезвычайно сложную задачу. Например, пользователь одного подразделения имеет право доступа к данным другого подразделения только через 10 дней после получения этих данных. Или пользователь может видеть только агрегированные показатели, но не детальные данные. Существуют и другие, еще более запутанные правила доступа. Их ведение, учет и изменение приводит к неизбежным ошибкам, вызванным сочетанием сложных условий доступа.

Витрины данных, содержащие информацию, предназначенную для выделенной группы пользователей, значительно снижают риски нарушения требования информационной безопасности.

До сих пор серьезной проблемой для территориально распределенных организаций является качество линий связи. В случае обрыва или недостаточной пропускной способности удаленные пользователи лишаются доступа к информации, содержащейся в КХД. Решением являются удаленные витрины данных, которые заполняются либо в нерабочее время, либо инкрементально, по мере поступления информации, с использованием транспорта с гарантированной доставкой.

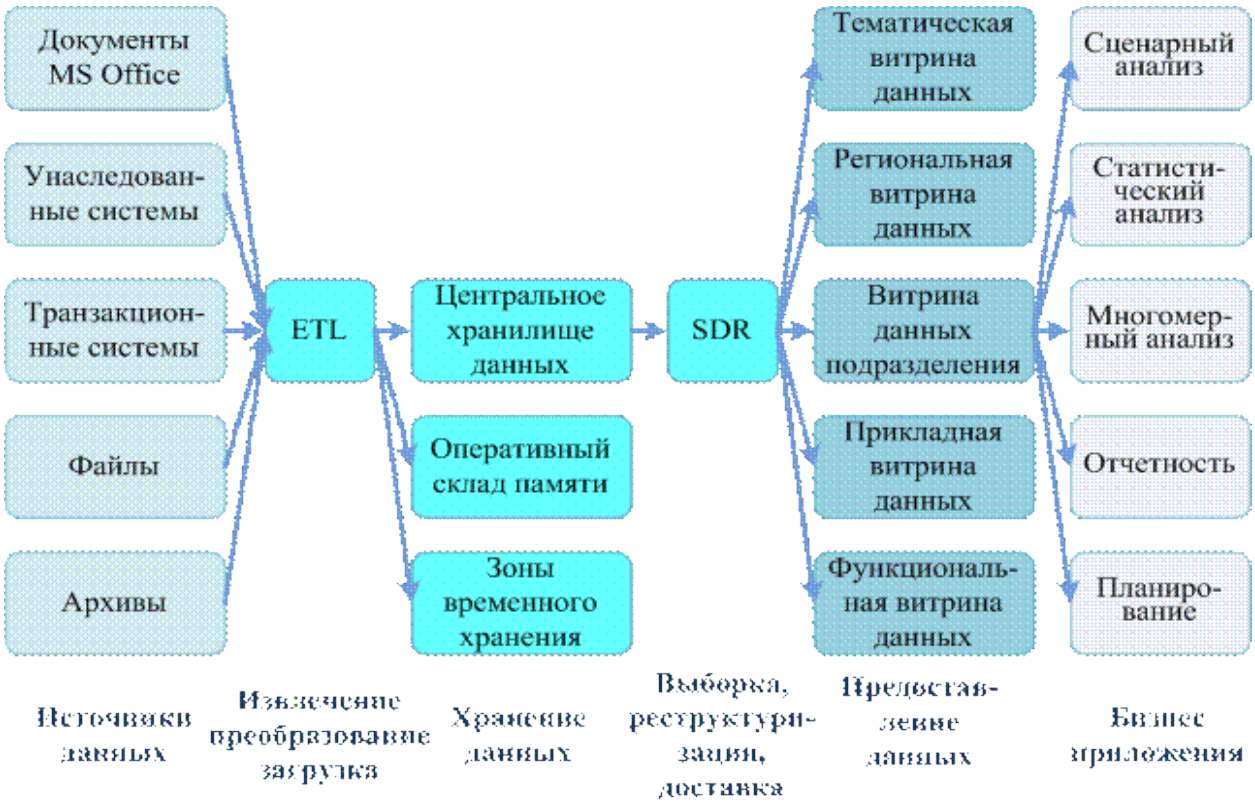


Рис. 46. Расширенная модель с витринами данных

Разные пользовательские приложения нуждаются в различных форматах данных: многомерные кубы, ряды данных, двумерные массивы, реляционные таблицы, файлы в формате MS Excel, текстовые файлы с разделителями, XML-файлы и т.д. Никакая структура данных в КХД не может удовлетворить этим требованиям. Выходом является создание витрин, чьи структуры данных оптимизированы под специфические требования отдельных приложений.

Еще одной причиной необходимости создания витрин данных является требование к надежности КХД, которое часто определяется, как пять или четыре девятки. Это означает, что КХД может простаивать не более 5 минут в год (99,999%) или не более часа в год (99,99%). Создание комплекса с такими характеристиками является сложной и весьма недешевой инженерной задачей. Требования к защите от терактов, саботажа и стихийных бедствий еще более усложняют построение программно-технического комплекса и осуществление соответствующих организационных мероприятий. Чем сложнее такой комплекс, чем больше данных он хранит, тем выше его стоимость и сложнее его поддержка. Наличие витрин данных резко снижает нагрузку на КХД, как по количеству пользователей, так и по объему данных в хранилище, так как эти данные могут быть оптимизированы под хранение, а не под обслуживание запросов.

Если витрины наполняются напрямую из КХД, то фактическое количество пользователей снижается с сотен и тысяч до десятков витрин, которые и являются пользователями КХД. При использовании средств SRD (Sample, Restructure, Delivery - выборка, реструктуризация, доставка) количество пользователей сокращается до 1. В этом случае вся логика информационного снабжения витрин сосредотачивается в SRD. Витрины могут быть оптимизированы под обслуживание пользовательских запросов. Программно-технический комплекс КХД может быть оптимизирован исключительно под надежное, защищенное хранение данных.

Средства SRD также смягчают нагрузку на КХД за счет того, что разные витрины могут обращаться к одним и тем же данным, тогда как SRD извлекает данные один раз, преобразует к различным форматам и доставляет в разные витрины данных.

Централизованная ETL с параллельными хранилищами и витринами данных.

В данном случае система извлечения, преобразования и загрузки данных (ETL) является центром, вокруг которого строится вся архитектура КХД. Информация из разнородных источников поступает в ETL, которая загружает очищенные и согласованные данные в центральное хранилище данных (ЦХД), в оперативный склад данных (ОСД), если он есть, и, при необходимости, в зоны временного хранения. Это обычная практика для КХД. Необычным является загрузка данных из ETL напрямую в витрины данных.

На практике такая архитектура возникает из-за требований скорейшего, без временных задержек, доступа к аналитическим данным. Использование оперативного склада данных не решает задачи, так как пользователи могут находиться в другом регионе, и им требуется территориальная витрина данных. Другой причиной может стать запрет на размещение разнотипной информации в ОСД по соображениям безопасности.

По тем или иным причинам, подобные архитектуры встречаются, и одной из проблем их эксплуатации являются сложности с восстановлением данных после краха витрин, напрямую снабжающихся из ETL. Дело в том, что средства ETL не предназначены для долговременного хранения извлеченных и очищенных данных. Транзакционные системы, как правило, ориентированы на выполнение текущих операций. Поэтому при потере данных в витринах, связанных с ETL, приходится либо поднимать информацию из средств резервного копирования (backup) транзакционных систем, либо организовывать исторические архивы систем - источников данных. Подобные архивы не только требуют средств на свое создание и поддержку в эксплуатации, но и являются, с корпоративной точки зрения, избыточными, так как дублируют функции корпоративного хранилища, но предназначены для ограниченного количества витрин данных.

Еще одним решением является двойное подключение подобных витрин – напрямую к средствам ETL и к хранилищу данных, что приводит к недоразумениям и рассогласованиям результатов аналитических работ. Причина кроется в том, что данные, поступающие в хранилище, как правило, проходят дополнительные проверки на непротиворечивость с уже загруженными данными. Например, может прийти финансовый документ с реквизитами, почти совпадающими с документом, поступившим в ЦХД ранее. Система ETL, не обладая памятью обо всех загруженных данных, не может выявить, является ли новый документ законным исправлением существующего, или это ошибка.

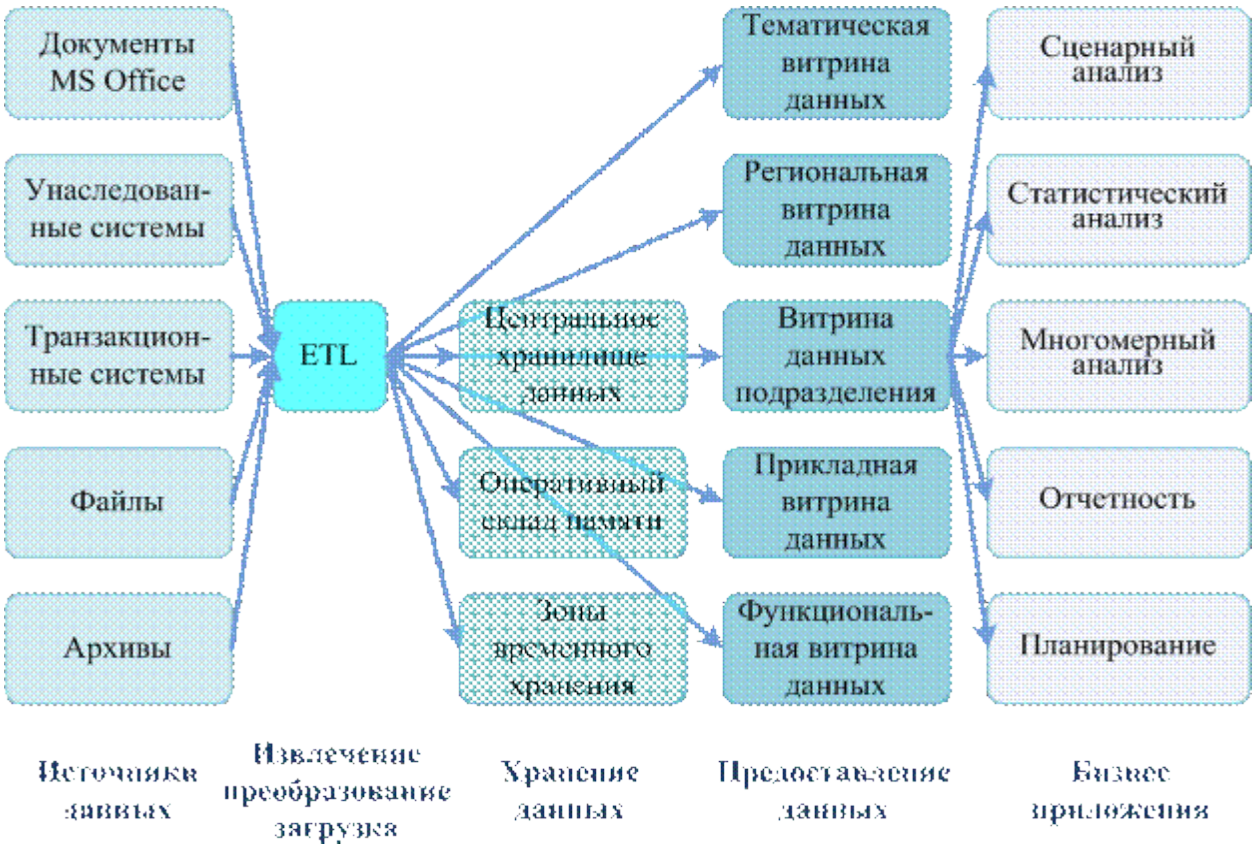


Рис. 47. Централизованная ETL с параллельными ХД И ВД

Средства верификации данных могут выявить подобные ситуации, действуя внутри хранилища данных. В случае выявления ошибки новые данные будут отброшены. Если же это регламентированное исправление, то изменения коснутся не только данных цифр, но и агрегированных показателей, составленных при участии исправляемых данных.

Таким образом, информация, попавшая в витрину данных напрямую из ETL, может противоречить данным, поступившим из ЦХД. В качестве решения иногда в витрине реализуют те же алгоритмы верификации данных, что и в ЦХД. Недостатком является необходимость поддержки и синхронизации одних и тех же алгоритмов в ЦХД и в витринах, питающихся непосредственно от ETL.

Подытоживая, можно сказать, что параллельные витрины данных приводят к повторной обработке данных, к созданию избыточных операционных архивов, к поддержке дублирующих приложений и децентрализации обработки данных, что, как правило, является причиной рассогласования информации.

Тем не менее, параллельные витрины имеют право на существование в тех случаях, когда оперативность доступа к аналитической информации важнее недостатков этой архитектуры.

Хранилище с накоплением данных в витринах.

Основанием для появления этой архитектуры явились следующие предпосылки.

1. Некоторые компании до сих пор внедряют и эксплуатируют разрозненные прикладные витрины данных. Качество данных в этих витринах удовлетворяет аналитиков, работающих с витринами.

2. В некоторых компаниях сложилось мнение, что создание корпоративного хранилища данных (КХД) подобно смертельному трюку с непредсказуемыми последствиями. Несмотря на то, что трудности создания и внедрения КХД, прежде всего, связаны не с технологическими вопросами, а с плохой организаций проекта и недостаточным вовлечением экспертов – будущих пользователей КХД, тем не менее, возникает желание пойти легким путем.

3. Требование быстрых результатов. Необходимость отчитываться ежеквартально вызывает потребность в быстрых осязаемых результатах. В результате появляется стремление сделать и внедрить какое-нибудь ограниченное решение без связи с остальными задачам.

Вольно или невольно следуя этим принципам, компании сначала внедряют разрозненные независимые витрины, в надежде, что содержащиеся в них данные будут легко, просто и быстро объединены. В реальности все гораздо сложнее. Качество данных в витринах может удовлетворять экспертов, работающих с ними, но эти информация не согласована с данными из других витрин, поэтому на стол руководству ложатся отчеты, которые нельзя привести к единому виду.

Одни и те же показатели могут вычисляться по разным алгоритмам, на основании разного набора данных, за разные сроки. Показатели с одинаковыми названиями могут скрывать разные сущности, и наоборот, одинаковые сущности могут иметь разные наименования.

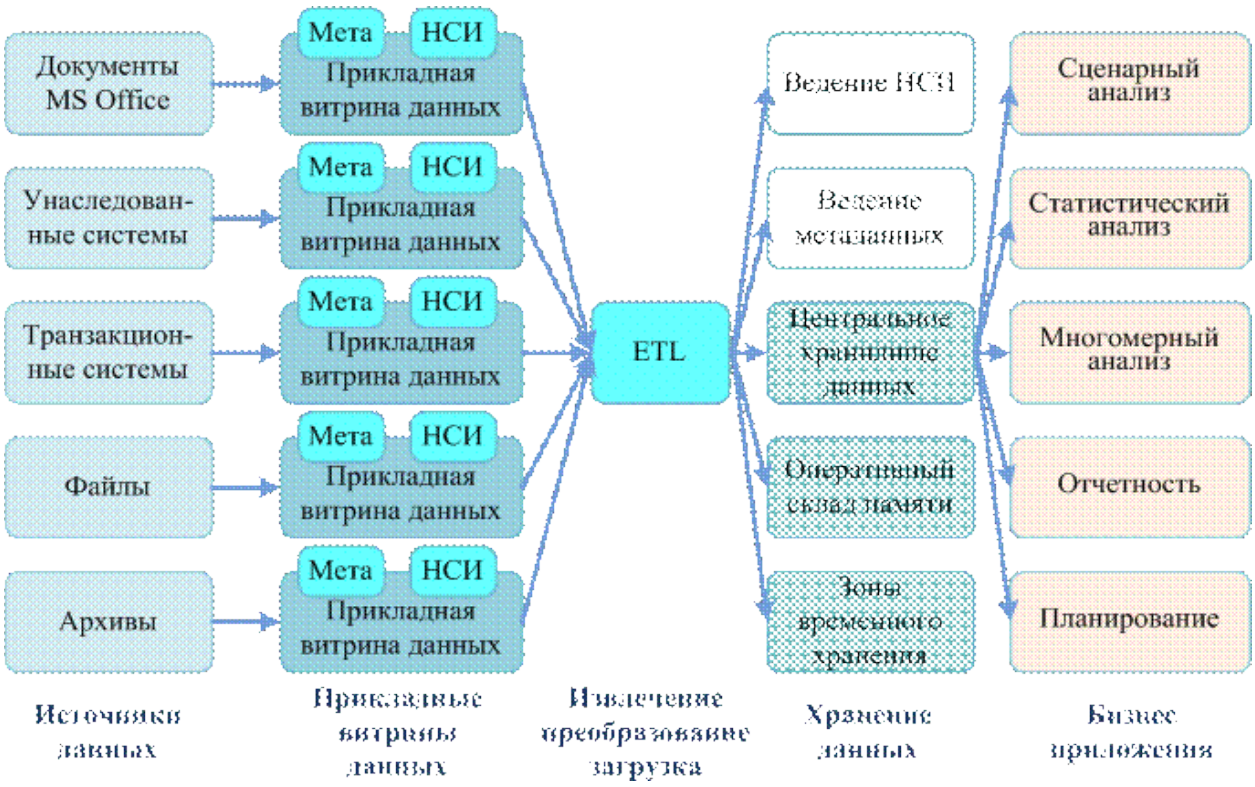


Рис. 48. Хранилище с накоплением данных в витринах

Диагноз – пользователи независимых прикладных витрин говорят на разных языках бизнеса, и каждая витрина содержит собственные метаданные.

Другая проблема заключается в различии нормативно-справочной информации (НСИ), используемых в независимых витринах данных. Разница в кодировке данных, в используемых кодификаторах, справочниках, классификаторах, идентификаторах, нормативах и словарях делает невозможным объединение этих данных без серьезного анализа, проектирования и разработки средств ведения НСИ.

Однако в организации уже существуют планы, бюджет и сроки создания КХД на основе независимых витрин данных. Руководство ожидает получить результат быстро и недорого. Разработчики, обещавшие экономию ресурсов, вынуждены сделать хоть что-нибудь. Так создаются хранилища несогласованных отчетов, что в корне противоречит самой идее создания хранилищ данных как единого и единственного источника очищенных, согласованных и непротиворечивых исторических данных.

Понятно, что ни руководство, ни пользователи подобного хранилища не склонны доверять информации, содержащейся в нем. Поэтому на следующем этапе встает необходимость радикальной переработки, а по сути, создания заново, хранилища, ориентированного на хранение не отчетов, а показателей, из которых будут собираться отчеты.

Эта работа невозможна без использования средств ведения метаданных и НСИ, область действия которых будет распространяться только на центральное хранилище (ЦХД), так как независимые витрины данных содержат свои метаданные и НСИ.

В результате руководство и эксперты могут получить согласованные и непротиворечивые отчеты, но они не смогут проследить происхождение данных сквозным образом, так как между независимыми витринами и ЦХД есть разрыв в ведении метаданных.

Таким образом, стремление к достижению сиюминутных результатов и к демонстрации быстрых успехов приводит к отказу от единого, сквозного управления метаданными и НСИ. Итогом такого подхода является наличие семантических островов, где сотрудники говорят на разных бизнес – языках.

Тем не менее, эта архитектура имеет право на существование, там, где единая модель данных или не нужна, или невозможна, и где в ЦХД передается сравнительно небольшой объем данных без необходимости детализации их происхождения и исходных составляющих. Например, если компания, оперирующая в разных странах, уже внедрила национальные хранилища данных, которые следуют локальным требованиям законодательства и правилам ведения бизнеса и финансового учета. Центральное хранилище данных может забирать из национальных ХД только часть информации для корпоративной отчетности. Создавать единую модель

данных нет необходимости, поскольку она не будет востребована на национальном уровне.

Естественно, что такая схема требует высокой степени доверия к национальным данным, и может быть использована, если умышленное или неумышленное искажение этих данных не приведет к тяжелым финансовым последствиям для всей организации.

Хранилище данных с интеграционной шиной.

Широкое распространение сервис - ориентированной архитектуры (СОА) 3 привело к желанию использовать ее в решениях для корпоративных хранилищ данных (КХД) вместо средств извлечения, преобразования и загрузки данных (ETL) в центральное хранилище (ЦХД) и вместо средств выборки, реструктуризации и доставки данных (SRD) в витрины данных.

Интеграционная шина, которая лежит в основе СОА, предназначена для интеграции веб - сервисов и приложений и выполняет следующие задачи:

- определяет сервис, соответствующий запросу от источника, и направляет запрос к сервису;
- преобразует транспортные протоколы между источником запроса и сервисом;
- преобразует форматы сообщений между источником запроса и сервисом;
- управляет бизнес - событиями различных источников.

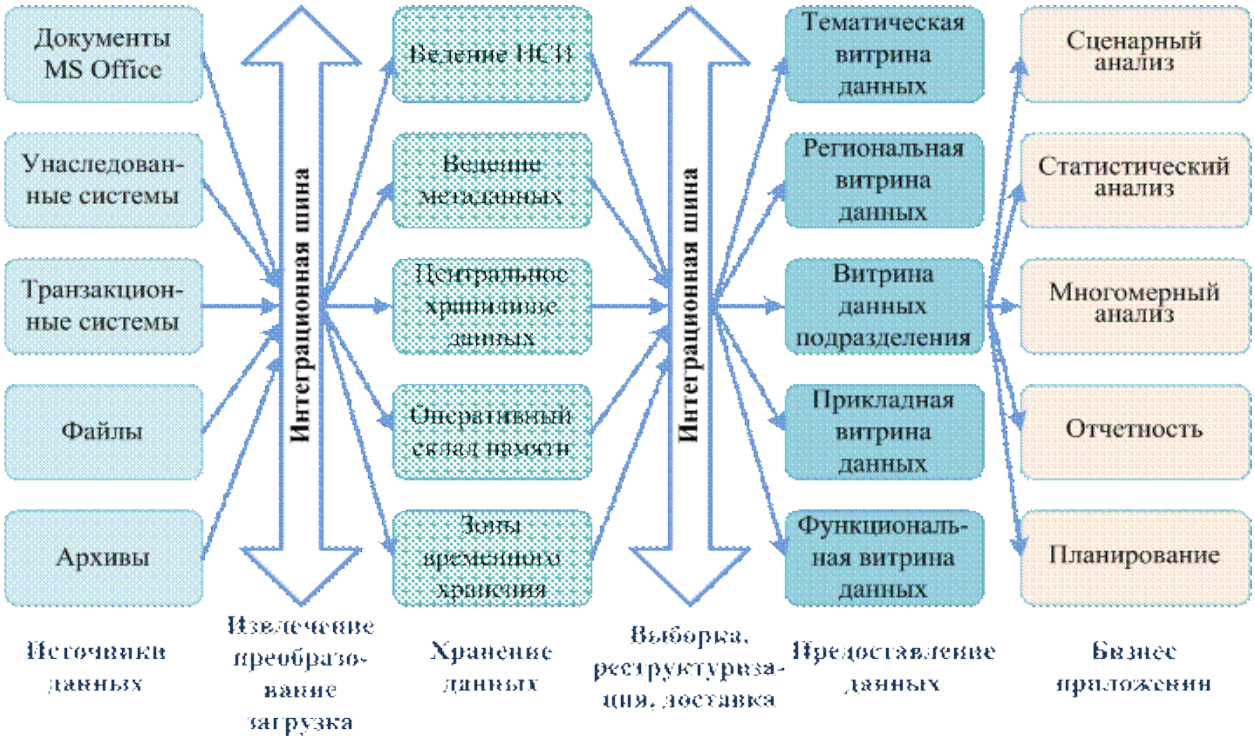


Рис. 49. Хранилище данных с интеграционной шиной

На первый взгляд функциональные возможности СОА позволяют применить ее для замены ETL и SRD. Действительно, ETL выполняет посреднические функции между ЦХД и источниками данных, а SRD

является посредником между ЦХД и витринами данных. Если заменить ETL и SRD на интеграционную шину, то, казалось бы, можно воспользоваться гибкостью, предоставляемой шиной для интеграции приложений. Представим себе, что ЦХД, оперативный склад данных (ОСД), зоны временного хранения, системы ведения метаданных и НСИ обращаются к шине как независимые приложения с запросами к источникам данных на обновление данных.

Прежде всего, в разы возрастет нагрузка на системы-источники данных, так как одна и та же информация будет многократно передаваться по запросам в ЦХД, ОСД, зоны временного хранения и системы управления метаданными и НСИ. Очевидное решение – создать собственное хранилище данных при шине для кеширования запросов.

Во-вторых, регламент сбора информации, ранее централизованный в ETL, теперь рассеян по приложениям, запрашивающим данных. Рано или поздно возникнут рассогласования в регламентах сбора данных для ЦХД, ОСД, систем ведения НСИ и метаданных. Данные, собранные по разным методикам, в разные отрезки времени, обработанные по разным алгоритмам, будут несогласованны друг с другом. Тем самым будет разрушена основная цель создания ЦХД как единого источника согласованных непротиворечивых данных.

В случае замены SRD на интеграционную шину последствия не столь драматичны. Но для того, чтобы ЦХД могло отвечать на запросы витрин данных, направленных через шину, оно должно быть преобразовано в сервис. Это значит, что хранилище должно соответствовать наиболее распространенному стилю web – сервисов, и поддерживать протоколы HTTP/ HTTPS и SOAP и XML – формат сообщений. Такой подход работает для коротких сообщений, но в витрины необходимо передавать большой объем данных, что может быть решено с помощью передачи двоичных объектов. Необходимая реструктуризация данных не может быть возложена на шину, и должна выполняться либо в ЦХД, либо в витрине. Эта функция может быть решена с помощью сервиса-посредника, принимающего данные, и передающего их в витрины данных после реструктуризации. То есть, мы возвращаемся к идее средства SRD с шинным интерфейсом.

Таким образом, интеграционная шина может быть использована в архитектуре КХД как транспортная среда между источниками данных и ETL и между SRD и витринами данных в тех случаях, когда компоненты КХД территориально разнесены и находятся за межсетевыми экранами в соответствии со строгими требованиями к защите информации. В этом случае для обеспечения взаимодействия достаточно, чтобы был разрешен обмен по протоколам HTTP/ HTTPS. Вся логика сбора и преобразования информации должна быть по-прежнему сосредоточена в ETL и SRD.

Рекомендованная архитектура КХД.

Архитектура корпоративного хранилища данных (КХД) должна удовлетворять многим функциональным и нефункциональным требованиям, которые зависят от конкретных задач, решаемых КХД. Как нет

универсального банка, авиакомпании, или нефтяного концерна, так нет и единого решения КХД, пригодного на все случаи жизни. Но основные принципы, которым должно следовать КХД, все же можно сформулировать.

Прежде всего, это качество данных, которое можно понимать, как полные, точные и воспроизводимые данные, доставленные в срок туда, где они нужны. Качество данных трудно измерить напрямую, но о нем можно судить по принимаемым решениям. То есть, качество данных требует инвестиций, но и само способно приносить прибыль.

Во-вторых, это защищенность и надежность хранения данных. Ценность информации, накопленной в КХД, может быть сравнима с рыночной стоимостью компании. Несанкционированный доступ к КХД чреват серьезными последствиями, поэтому должны быть приняты меры, адекватные ценности данных.

В-третьих, данные должны быть доступны сотрудникам в объеме, необходимом и достаточном для выполнения своих функциональных обязанностей.

В-четвертых, сотрудники должны иметь единое понимание данных, то есть должно быть установлено единое смысловое пространство.

В-пятых, необходимо, по возможности, устранить конфликты в кодировках данных в системах источниках.

Рис. 50. Рекомендованная архитектура КХД

Предлагаемая архитектура следует проверенным принципам модульного конструирования «непотопляемых отсеков». Стратегия «Разделяй и властвуй» применима не только в политике. Разделяя архитектуру на модули, мы одновременно концентрируем в них

определенную функциональность, получая власть над неуправляемой ИТ стихией. Средства ETL обеспечивают полный, надежный, точный сбор информации из источников данных благодаря сосредоточенной в ETL логике сбора, обработки и преобразования данных и взаимодействию с системами ведения метаданных и НСИ.

Система ведения метаданных является главным «хранителем мудрости», к которому можно обратиться за советом. Система ведения метаданных поддерживает актуальность бизнес-метаданных, технических, операционных и проектных метаданных.

Система ведения НСИ является третейским судьей при разрешении конфликтов кодировок данных.

Центральное хранилище данных (ЦХД) несет только нагрузку по надежному защищенному хранению данных. В зависимости от поставленных задач, надежность программно-технического комплекса (ПТК) ЦХД может достигать 99,999%, то есть обеспечивать бесперебойную работу с простоем не более 5 мин в год. ПТК ЦХД может обеспечивать защиту данных от несанкционированного доступа, саботажа и стихийных бедствий. Структура данных в ЦХД оптимизирована исключительно с целью обеспечения эффективного хранения данных.

Средства выборки, реструктуризации и доставки данных (SRD) в такой архитектуре являются единственным пользователем ЦХД, беря на себя всю работу по заполнению витрин данных и, тем самым, снижая нагрузку на ЦХД по обслуживанию запросов пользователей.

Витрины данных содержат данные в структурах и форматах, оптимальных для решения задач пользователей данной витрины. В настоящее время, когда даже ноутбук может быть оснащен терабайтным диском, проблемы, связанные с многократным повторением данных в витринах, не имеют значения. Главное преимущество этой архитектуры – предоставление доступа для удобной работы пользователей с необходимым объемом данных, возможность быстрого восстановления содержимого витрин из ЦХД при сбое витрины, обеспечение работы пользователей при отсутствии связи с ЦХД.

Достоинство этой архитектуры заключается в возможности раздельного проектирования, создания, эксплуатации и доработки отдельных компонентов без радикальной перестройки всей системы. Это означает, что начало работ по созданию КХД не требует сверхусилий или сверхинвестиций. Достаточно начать с ограниченного по своим возможностям программно-технического комплекса, и следуя предложенным принципам, создать работающий и действительно полезный для пользователей прототип. Далее необходимо выявить узкие места и развивать соответствующие компоненты.

Применение этой архитектуры вместе с тройной стратегией интеграции данных, метаданных и НСИ, позволяет сократить сроки и бюджет проекта внедрения КХД и развивать его в соответствии с изменяющимися требованиями бизнеса.

Вопрос 6. Фрактальные методы в архивации.[21]

История фрактального сжатия.

Рождение фрактальной геометрии обычно связывают с выходом в 1977 году книги Б. Мандельброта «Фрактальная геометрия природы». Одна из основных идей книги заключалась в том, что средствами традиционной геометрии (то есть используя линии и поверхности), чрезвычайно сложно представить природные объекты. Фрактальная геометрия задает их очень просто.

В 1981 году Джон Хатчинсон опубликовал статью «Фракталы и самоподобие», в которой была представлена теория построения фракталов с помощью системы итерируемых функций (IFS, Iterated Function System). Четыре года спустя появилась статья Майкла Барнсли и Стефана Демко, в которой приводилась уже достаточно стройная теория IFS. В 1987 году Барнсли основал Iterated Systems, компанию, основной деятельностью которой является создание новых алгоритмов и ПО с использованием фракталов. Всего через год, в 1988 году, он выпустил фундаментальный труд «Фракталы повсюду». Помимо описания IFS, в ней был получен результат, известный сейчас как Collage Theorem, который лежит в основе математического обоснования идеи фрактальной компрессии.

Если построение изображений с помощью фрактальной математики можно назвать прямой задачей, то построение по изображению IFS – это обратная задача. Довольно долго она считалась неразрешимой, однако Барнсли, используя Collage Theorem, построил соответствующий алгоритм. (В 1990 и 1991 годах эта идея была защищена патентами.) Если коэффициенты занимают меньше места, чем исходное изображение, то алгоритм является алгоритмом архивации.

Первая статья об успехах Барнсли в области компрессии появилась в журнале BYTE в январе 1988 года. В ней не описывалось решение обратной задачи, но приводилось несколько изображений, сжатых с коэффициентом 1:10000, что было совершенно ошеломительно. Но практически сразу было отмечено, что несмотря на броские названия («Темный лес», «Побережье Монтере», «Поле подсолнухов») изображения в действительности имели искусственную природу. Это, вызвало массу скептических замечаний, подогреваемых еще и заявлением Барнсли о том, что «среднее изображение требует для сжатия порядка 100 часов работы на мощной двухпроцессорной рабочей станции, причем с участием человека».

Отношение к новому методу изменилось в 1992 году, когда Арнауд Джеквин, один из сотрудников Барнсли, при защите диссертации описал практический алгоритм и опубликовал его. Этот алгоритм был крайне медленным и не претендовал на компрессию в 10000 раз (полноцветное 24-разрядное изображение с его помощью могло быть сжато без существенных потерь с коэффициентом 1:8 - 1:50); но его несомненным достоинством было то, что вмешательство человека удалось полностью

исключить. Сегодня все известные программы фрактальной компрессии базируются на алгоритме Джеквина. В 1993 году вышел первый коммерческий продукт компании Iterated Systems. Ему было посвящено достаточно много публикаций, но о коммерческом успехе речь не шла, продукт был достаточно «сырой», компания не предпринимала никаких рекламных шагов, и приобрести программу было тяжело.

В 1994 году Ювал Фишер были предоставил во всеобщее пользование исходные тексты исследовательской программы, в которой использовалось разложение изображения в квадродерево и были реализованы алгоритмы оптимизации поиска. Позднее появилось еще несколько исследовательских проектов, которые в качестве начального варианта программы использовали программу Фишера. В июле 1995 года в Тронхейме (Швеция) состоялась первая школа-конференция, посвященная фрактальной компрессии.

Идея метода.

Фрактальная архивация основана на том, что мы представляем изображение в более компактной форме - с помощью коэффициентов системы итерируемых функций (Iterated Function System - далее по тексту как IFS). Прежде, чем рассматривать сам процесс архивации, разберем, как IFS строит изображение, т.е. процесс декомпрессии.

Строго говоря, IFS представляет собой набор трехмерных аффинных преобразований, в нашем случае переводящих одно изображение в другое. Преобразованию подвергаются точки в трехмерном пространстве (x_координата, y_координата, яркость).

Наиболее наглядно этот процесс продемонстрировал Барнсли в своей книге «Fractal Image Compression». Там введено понятие Фотокопировальной Машины, состоящей из экрана, на котором изображена исходная картинка, и системы линз, проецирующих изображение на другой экран: (рис. 51):

- линзы могут проецировать часть изображения произвольной формы в любое другое место нового изображения;
- области, в которые проецируются изображения, не пересекаются;
- линза может менять яркость и уменьшать контрастность;
- линза может зеркально отражать и поворачивать свой фрагмент изображения;
- линза должна масштабировать (причем только уменьшая) свой фрагмент изображения.

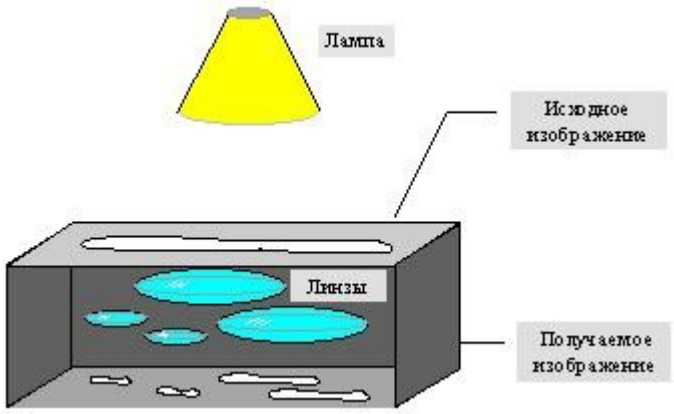


Рис. 51. Машина Барнсли

Расставляя линзы и меняя их характеристики, мы можем управлять получаемым изображением. Одна итерация работы Машины заключается в том, что по исходному изображению с помощью проектирования строится новое, после чего новое берется в качестве исходного. Утверждается, что в процессе итераций мы получим изображение, которое перестанет изменяться. Оно будет зависеть только от расположения и характеристик линз, и не будет зависеть от исходной картинки. Это изображение называется «неподвижной точкой» или аттрактором данной IFS. Соответствующая теория гарантирует наличие ровно одной неподвижной точки для каждой IFS.

Поскольку отображение линз является сжимающим, каждая линза в явном виде задает самоподобные области в нашем изображении. Благодаря самоподобию мы получаем сложную структуру изображения при любом увеличении. Таким образом, интуитивно понятно, что система итерируемых функций задает фрактал (нестрого - самоподобный математический объект).

Наиболее известны два изображения, полученных с помощью IFS: «треугольник Серпинского» (рис. 52) и «папоротник Барнсли» рис. 53.

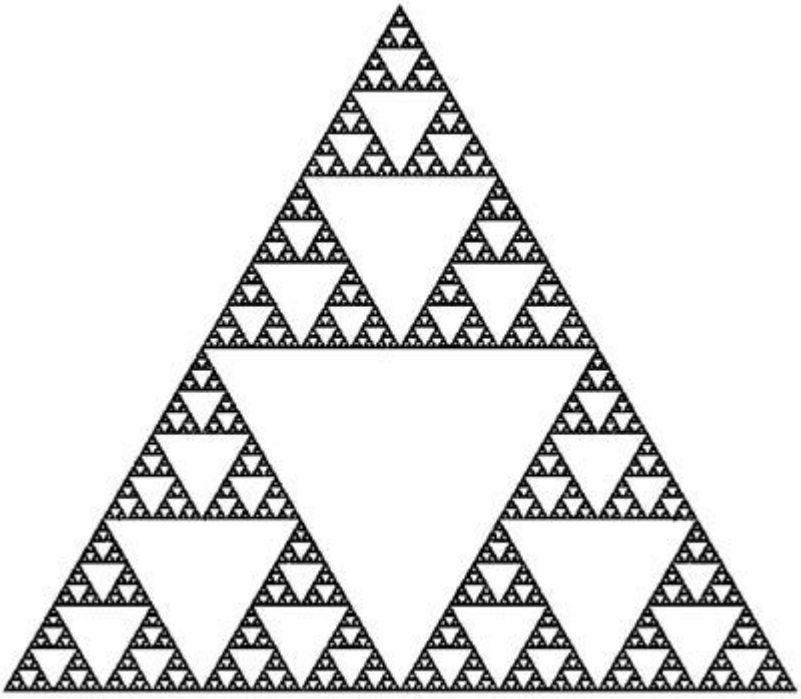


Рис. 52. Треугольник Серпинского

Рис. 53. Папоротник Барнсли

«Треугольник Серпинского» задается тремя, а «папоротник Барнсли» четырьмя аффинными преобразованиями (или, «линзами»). Каждое преобразование кодируется считанными байтами, в то время как изображение, построенное с их помощью, может занимать и несколько мегабайт.

Из вышесказанного становится понятно, как работает архиватор, и почему ему требуется так много времени. Фактически, фрактальная компрессия – это поиск самоподобных областей в изображении и определение для них параметров аффинных преобразований.

В худшем случае, если не будет применяться оптимизирующий алгоритм, потребуется перебор и сравнение всех возможных фрагментов изображения разного размера. Даже для небольших изображений при учете дискретности мы получим астрономическое число перебираемых вариантов. Причем, даже резкое сужение классов преобразований, например, за счет масштабирования только в определенное количество раз, не дает заметного выигрыша во времени. Кроме того, при этом теряется качество изображения. Подавляющее большинство исследований в области фрактальной компрессии сейчас направлены на уменьшение времени архивации, необходимого для получения качественного изображения.

Тема 7. Классификация БД и СУБД[22]

Вопрос 1. Классификация БД.

По форме представляемой информации можно выделить фактографические, документальные и мультимедийные БД.

Особенностью фактографической информации является практическая очевидность (минимальная неопределённость, не требующая использования сложных или нечётких процедур) идентификации и интерпретации факта, как его имени, так и состояния. То есть, в этом случае контекст (содержание) в достаточной степени определяется однозначно понимаемым объявлением о назначении базы данных и таким именованием полей данных, когда в качестве имени используется общепринятое, не зависящее от прикладных задач, имя свойства (и таким образом определяются характеристические признаки).

Документальная информация отличается неопределённостью или переменной структурой данных (документов).

По типу хранимой информации (исключая мультимедийную) можно выделить фактографические, документальные, лексикографические БД.

Лексикографические базы – это классификаторы, кодификаторы, словари основ слов, тезаурусы, рубрикаторы и т.д., обычно используемые в качестве справочных совместно с документальными или фактографическими БД.

Документальные базы подразделяются по уровню представления информации на полнотекстовые (обрабатывающие «первичные» документы) и библиографическо-реферативные (обрабатывающие «вторичные» документы, отражающие на адресном и содержательном уровне первичный документ).

По типу используемой модели данных традиционно выделяют три класса БД: иерархические, сетевые, реляционные. Иерархические и сетевые модели данных называют ещё навигационными.

Иерархическая модель вообще реализовывалась средствами древовидных структур с корневыми сегментами, имеющими физический указатель на другие сегменты. Преимущество таких моделей БД заключалось в том, что они уменьшали избыточность данных. Одно из неудобств такой модели данных заключается в том, что реальный мир не может быть легко представлен в виде древовидной структуры с единственным корневым сегментом. Иерархические базы данных обеспечивали указатели между различными деревьями баз данных, но обработка данных с использованием таких связей иногда могла оказаться неудобной. Примером такой модели является файловая структура.

Сетевая модель данных включала язык определения данных (Data Definition Language, DDL) и язык манипулирования данными (Data Manipulation Language, DML) – формальные языки, предназначенные для определения и манипулирования содержимым базы данных. Предложенное разграничение функций между различными типами языков в системах управления базами данных привело к выделению языков управления транзакциями, языков манипулирования схемой и других групп языков. В сетевых БД, в отличие от иерархических, нет необходимости в корневой записи, поскольку между типами записей могут быть созданы наборы без искусственных ограничений, свойственных иерархии. Однако здесь, как и в иерархических БД, ассоциации поддерживаются с помощью физических указателей. Примером этой модели является Интернет.

Реляционная модель данных обеспечивает ряд важных возможностей, которые делают управление БД и их использование относительно легким, устойчивым по отношению к ошибкам и предсказуемым. Она описывает данные с их естественной структурой, не добавляя каких-либо дополнительных структур, необходимых для машинного представления или для целей реализации; обеспечивает математическую основу для интерпретации выводимости, избыточности и непротиворечивости отношений; обеспечивает независимость данных от их физического представления, от связей между данными и от соображений реализации, связанных с эффективностью и подобными заботами. Главным элементом в реляционной модели является отношение. Для большинства людей обычной визуализацией отношения служит «таблица». Таблица, как известно, имеет строки и столбцы. Столбцы отношения соответствуют «элементам данных» каждой записи, которая представляется строкой отношения. Важное различие между отношением и таблицей (в том виде, как она реализована в большинстве поставляемых реляционных СУБД) заключается в том, что отношение не может иметь дубликатов кортежей (т.е. записей в файле), в то время как для таблиц допускается возможность содержать дубликаты строк.

Реляционная модель данных (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При

создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними.

Разработчики приложений на основе иерархической или сетевой БД обладают контролем над тем, каким образом были определены связи и как осуществляется навигация по ним. При создании систем реляционных БД большая часть этой обработки заключается в саму реляционную СУБД. Поэтому оптимизация запросов стала важной функцией (отсутствие которой первоначально ограничивало производительность) коммерческих реляционных СУБД.

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц, что в ряде случаев мешает эффективной реализации приложений. Развитие технологий обработки данных привело к появлению постреляционных, объектно-ориентированных, многомерных БД, которые в той или иной степени соответствуют упомянутым классическим моделям.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Она допускает многозначные поля – поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

Первые публикации, связанные с объектно-ориентированными базами данных (ООБД) появились в середине 1980-х годов. В общей классической постановке объектно-ориентированный подход базируется на концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Объектно-ориентированные базы данных (ООБД) строятся из объектов. Объекты хранятся физически как строки или столбцы таблицы. В этом случае реляционную модель можно рассматривать как «таблично-ориентированную» модель данных, так как соответствующие ей БД строятся из таблиц. В ООБД важнейшее место отводится объектам, на основе которых могут определяться другие объекты благодаря использованию концепции, называемой наследованием. При этом некоторые или все атрибуты (либо свойства) определяющего объекта наследуются каким-либо другим объектом, одни атрибуты и (или) свойства добавляются, а другие могут удаляться.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом во все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение.

Состояние объекта - набор значений его атрибутов.

Поведение объекта - набор методов (программный код), оперирующих над состоянием объекта.

Значение атрибута объекта - это тоже некоторый объект или множество объектов.

Состояние и поведение объекта инкапсулированы в объекте; взаимодействие между объектами производится на основе передачи сообщений и выполнении соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования).

По типологии доступа и характеру использования хранимой информации выделяют специализированные и интегрированные БД.

По топологии хранения данных различают локальные и распределённые БД.

По степени доступности можно выделить общедоступные и с ограниченным доступом пользователей к БД.

Представленная классификация не является полной. Она в большей степени отражает исторически сложившееся состояние дел в сфере деятельности, связанной с разработкой и применением баз данных.

Вопрос 2. Классификация СУБД.

Рассмотрим ряд классификационных признаков, относящихся к СУБД.

По языкам общения СУБД делятся на открытые, замкнутые и смешанные. *Открытые системы* – это системы, в которых для обращения к базам данных используются универсальные языки программирования. Замкнутые системы имеют собственные языки общения с пользователями БД. Открытые системы в настоящее время используются редко.

По числу уровней в архитектуре различают одноуровневые, двухуровневые, трехуровневые системы. В принципе возможно выделение и большего числа уровней.

Под архитектурным уровнем СУБД понимают функциональный компонент, механизмы которого служат для поддержки некоторого уровня абстракции данных (логический и физический уровень, а также «взгляд» пользователя — внешний уровень).

По выполняемым функциям СУБД делятся на информационные и операционные. Информационные СУБД позволяют организовать хранение информации и доступ к ней. Для выполнения более сложной обработки необходимо писать специальные программы. Операционные СУБД выполняют достаточно сложную об-работку, например, автоматически позволяют получать агрегированные показатели, не хранящиеся непосредственно в базе данных, могут изменять алгоритмы обработки и т. д.

По сфере возможного применения различают универсальные и специализированные, обычно проблемно-ориентированные СУБД. Системы

управления базами данных поддерживают разные типы данных. Набор типов данных, допустимых в разных СУБД, различен. В настоящее время наблюдается тенденция к расширению числа используемых типов данных. Кроме того, ряд СУБД позволяет разработчику (прикладному программисту или администратору БД) добавлять новые типы данных и новые операции над этими данными. Такие системы называются расширяемыми системами баз данных (РСБД).

Дальнейшим развитием концепции РСБД являются объектно-ориентированные системы баз данных, обладающие достаточно мощными выразительными возможностями, чтобы непосредственно моделировать сложные объекты.

СУБД в зависимости от архитектуры делятся на локальные и распределённые. В *локальной СУБД* её компоненты размещаются на одном компьютере, а в *распределённой* – на нескольких, которые могут находиться на любом удалении друг от друга.

По языкам общения СУБД делятся на открытые, замкнутые и смешанные.

Открытые системы – это системы, в которых для обращения к базам данных используются универсальные языки программирования. Замкнутые системы имеют собственные языки общения с пользователями БД. Открытые системы в настоящее время используются редко.

По сфере возможного применения различают универсальные и специализированные, обычно проблемно-ориентированные СУБД.

Набор типов данных, допустимых в разных СУБД, различен. Ряд СУБД позволяет разработчику добавлять новые типы данных и новые операции над этими данными. Такие системы называются *расширяемыми системами баз данных* (РСБД). СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Если компьютер и ОС поддерживают многопользовательский режим работы, то в такой среде может функционировать многопользовательская СУБД. В общем случае она позволяет одновременно обслуживать нескольких пользователей, работающих непосредственно с СУБД или с приложениями.

При обслуживании нескольких параллельных источников запросов (от пользователей и приложений) СУБД планирует использование своих ресурсов и ресурсов ЭВМ таким образом, чтобы обеспечивать независимое или почти независимое выполнение порождаемых запросами операций. Многопользовательские СУБД часто применяются на больших и средних ЭВМ, где основным режимом использования ресурсов является коллективный доступ.

Дальнейшим развитием концепции РСБД являются объектно-ориентированные системы баз данных, обладающие достаточно мощными выразительными возможностями, чтобы непосредственно моделировать сложные объекты.

Обычно СУБД, как и БД, различают по используемой модели данных. К основным типам СУБД относят три базовые модели данных: иерархические, сетевые и реляционные.

Реляционная модель данных обеспечивает ряд важных возможностей, которые делают управление БД и их использование относительно легким, устойчивым по отношению к ошибкам и предсказуемым. СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Краткий обзор СУБД.

Многие авторы классифицируют СУБД на две большие категории: так называемые «настольные» и «серверные».

Настольные СУБД.

Настольные СУБД используются для сравнительно небольших задач (небольшой объем обрабатываемых данных, малое количество пользователей). С учетом этого, указанные СУБД имеют относительно упрощенную архитектуру, в частности, функционируют в режиме файл-сервер, поддерживают не все возможные функции СУБД (например, не ведется журнал транзакций, отсутствует возможность автоматического восстановления базы данных после сбоев и т. п.). Тем не менее, такие системы имеют достаточно обширную область применения. Прежде всего, это государственные (муниципальные) учреждения, сфера образования, сфера обслуживания, малый и средний бизнес. Специфика возникающих там задач заключается в том, что объемы данных не являются катастрофически большими, частота обновлений не бывает слишком высокой, организация территориально обычно расположена в одном небольшом здании, количество пользователей колеблется от одного до 10–15 человек. В подобных условиях использование настольных СУБД для управления информационными системами является вполне оправданным, и они с успехом применяются.

Одними из первых СУБД были так называемые dBase-совместимые программные системы, разработанные разными фирмами. Первой широко распространенной системой такого рода была система dBase III – PLUS (фирма Achton-Tate). Развитый язык программирования, удобный интерфейс, доступный для массового пользователя, способствовали широкому распространению системы. В то же время работа системы в режиме интерпретации обусловливала низкую производительность на стадии выполнения. Это привело к появлению новых систем-компиляторов, близких к системе dBase III – PLUS: Clipper (фирма Nantucket Inc.), FoxPro (фирма Fox Software), FoxBase+ (фирма Fox Software), Visual FoxPro (фирма Microsoft). Одно время достаточно широко использовалась СУБД PARADOX (фирма Borland International).

В последние годы очень широкое распространение получила система управления базами данных Microsoft Access, которая входит в целый ряд версий пакета Microsoft Office(фирма Microsoft).

Серверные СУБД.

Для крупных организаций ситуация принципиально меняется. Там использование файл-серверных технологий является неудовлетворительным по описанным выше причинам. Поэтому на передний край борьбы за автоматизацию выходят так называемые серверные СУБД.

Основными производителями таких систем обработки и хранения данных являются 3 корпорации: Oracle, Microsoft и IBM. Наиболее распространенными клиент-серверными системами здесь соответственно являются системы Oracle (разработчик компания Oracle), MS SQL Server (разработчик компания Microsoft), DB2 , Informix Dynamic Server (компания IBM).

Дадим краткую характеристику этим системам.

MS SQL Server.

К настоящему времени разработано несколько версий систем: MS SQL Server-2000, MS SQL Server -2005, MS SQL Server-2008. Приведем информацию о системе MS SQL Server-2008 с сервера Microsoft (http://www.microsoft.com/rus/SQL/2008/default.mspx)

Microsoft® SQL Server™ 2008 - это законченное предложение в области баз данных и анализа данных для быстрого создания масштабируемых решений электронной коммерции, бизнес-приложений и хранилищ данных. Оно позволяет значительно сократить время выхода этих решений на рынок, одновременно обеспечивая масштабируемость, отвечающую самым высоким требованиям. В SQL Server включена поддержка языка XML и протокола HTTP, средства повышения быстродействия и доступности, позволяющие распределить нагрузку и обеспечить бесперебойную работу, функции для улучшения управления и настройки, снижающие совокупную стоимость владения.

Платформа бизнес-анализа SQL Server 2008, тесно интегрированная с Microsoft Office, предоставляет развитую маштабируемую инфраструктуру для внедрения мощных возможностей бизнес-анализа в рабочий процесс всех бизнес-подразделений вашей компании, открывая доступ к нужной бизнес-информации через знакомый интерфейс MS Excel и MS Word.

MS SQL Server-2008 поддерживает создание и работу с корпоративным хранилищем данных, объединяющим информацию со всех систем и приложений, позволяющим получить единую комплексную картину бизнеса вашей компании.

MS SQL Server-2008 предоставляет масштабируемый и высокопроизводительный «процессор данных» - для самых ответственных и требовательных бизнес-приложений, тем, кому необходим высочайший уровень надежности и защиты, позволяя при этом снизить совокупную стоимость владения за счет расширенных возможностей по управлению серверной инфраструктурой.

MS SQL Server-2008 предлагает разработчикам развитую, удобную и функциональную среду программирования, включая средства работы с веб службами, инновационные технологии доступа к данным – все, что необходимо для эффективной работы с данными любых типов и форматов.

Oracle.

К настоящему времени разработано несколько версий систем, каждая из которых включает целую линейку продуктов, например Oracle 8, Oracle 9i, Oracle 10g.

Соответствующие линейки продуктов включают как собственно СУБД (например Oracle Database 10g, Oracle Database 11g) , так и средства разработки и анализа данных.

Приведем информацию о системе с сервера Oracle http://www.oracle.com/global/ru/mid/oracle_products/database.html).

Oracle предлагает комплексные, открытые, доступные и удобные в использовании технологические решения. Готовые пакетируемые решения автоматически включают в свою стоимость базу данных, сервер приложений, интеграционную платформу, инструменты аналитики и управления неструктурированными данными. Масштабируемые бизнес-приложения Oracle могут быть легко интегрированы с ИТ-инфраструктурой предприятия без потери уже вложенных в IT инвестиций.

СУБД Oracle Database 11g обеспечивает улучшенные характеристики за счет автоматизации задач администрирования и обеспечения лучших в отрасли возможностей по безопасности и соответствию нормативно-правовым актам в области защиты информации. Появилось больше функций автоматизации, самодиагностики и управления. Среди характеристик системы можно отметить управление большими объемами данных с использованием распределенных таблиц и компрессии, эффективную защиту данных, возможность полного восстановления, возможность интеграции геофизических данных медиа-контента в бизнес-процесс и т.д.

Серверы баз данных компании IBM.

К настоящему времени разработаны линейки продуктов DB2 и Informix, включающая как собственно СУБД так и средства разработки и анализа данных (DB2 Universal Database DB2 Personal Edition, DB2 Enterprise 9 и др., а также Informix Dynamic Server, Informix Dynamic Server Express, Informix Extended Parallel Server и др.

Приведем информацию о части таких систем с сервера (http://www-01.ibm.com/software/ru/data/?pgel=ibmhzn)

Универсальный сервер баз данных DB2 Universal Database - это масштабируемая, обьектно-реляционная система управления базами данных с интегрированной поддержкой мультимедиа и Web, работающая на системах от персональных компьютеров и серверов на процессорах Intel до Unix, от однопроцессорных систем до симметричных многопроцессорных систем (SMP) и систем с массовым параллелизмом (MPP), на хостах AS/400 и мейнфреймах. DB2 Universal Database объединяет в себе высокую производительность систем обработки транзакций в режиме on-line, объектно-реляционные расширения, усовершенствованные средства оптимизации с возможностями параллельной обработки и поддержкой очень больших баз данных. DB2 Universal Database также имеет новые встроенные средства для облегчения переноса на свою базу приложений, разработанных

на других системах управления базами данных, таких как Oracle, Microsoft, Sybase и Informix. Помимо этого, DB2 Universal Database включает в себя дополнительные средства поддержки систем аналитической обработки в реальном времени (OLAP) и систем поддержки принятия решений, множество простых в использовании расширений (DB2 extenders). DB2 Universal Database доступна на абсолютном большинстве ключевых платформ, что дает заказчикам ту гибкость, которая им необходима.

Вопрос 3. Тенденции развития СУБД. Объектно-ориентированные СУБД.[23]

Объектно-ориентированный подход к организации баз данных.

В начале 90-х годов XX века начались активные попытки по внедрению объектно-ориентированных технологий в отрасль проектирования и разработки баз данных. Бытовала точка зрения о том, что соответствующие технологии быстро вытеснят все остальные, так же как и во многих других программистских отраслях, но ничего подобного не произошло.

Объектно-ориентированное программирование.

Рассмотрим термин «объектно-ориентированное программирование». Заметим, что это термин, принятый преимущественно в российской литературе. В литературе под этим понимается сразу три аспекта:

Объектно-ориентированный анализ – OOA, object-oriented analysis. Объектно-ориентированный анализ – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.

Объектно-ориентированное проектирование – OOD, object-oriented design. Объектно-ориентированное проектирование – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.

Объектно-ориентированное программирование – OOP, object-oriented programming. Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Здесь и далее по тексту условимся не отступать от традиций и понимать под объектно-ориентированным программированием (ООП) сразу три указанных выше аспекта.

Основой объектно-ориентированной технологии является так называемая объектная модель, которая возникает как результат объектно-ориентированной декомпозиции. Она выделяет основные абстракции предметной области, определяет классы абстракций и выясняет, какими данными (атрибутами) описывается каждая абстракция, какую функциональность эти абстракции должны обеспечивать. В отличие от традиционных технологий программирования объектно-ориентированная

технология представляет программу как совокупность классов и объектов, взаимодействующих друг с другом.

Объект – конкретная материализация абстракции; сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение.

Объект ООП – инкапсулированная структура, имеющая атрибуты и методы.

Термин «инкапсулированная структура» означает, что объект является самодостаточным, программы, внешние по отношению к объекту, ничего «не знают» о его структуре и такое «знание» им не требуется. «Внешний» вид объекта называется его интерфейсом.

В таком понимании объект – это черный ящик, нам неизвестно, что у него внутри, мы лишь можем вызвать его методы и только через них взаимодействовать с ним. Кроме этого, объекты могут принадлежать иерархии «от общего к частному», которая реализуется путем наследования. Инкапсулированные состояния объекта могут быть как простыми типами данных, так и другими объектами, или даже массивами объектов. Каждый объект содержит определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений. Объекты идентифицируются с помощью специальных указателей – дескрипторов. Методы объектов ООП представляют собой последовательности инструкций, выполняемых объектом. Например, у объекта может быть метод, отображающий данный объект, создающий данный объект и изменяющий его.

Предметная область моделируется как множество классов взаимодействующих объектов. Объект характеризуется набором свойств, которые являются как бы его пассивными характеристиками, и набором методов работы с этим объектом. Работать с объектом можно только с использованием его методов. Атрибуты объекта могут принимать множество допустимых значений, набор конкретных значений атрибутов определяет состояние объекта. Используя методы работы с объектом можно изменять значение его атрибутов и тем самым как бы изменить состояние самого объекта. Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

К числу основных идей объектно-ориентированной технологии, как правило, относят: **абстрагирование, инкапсуляцию, модульность, иерархичность, типизацию, полиморфизм, наследование**.

Инкапсуляция ограничивает область видимости имени атрибута пределами того объекта, в котором оно определено. Смысл этого атрибута будет определяться тем объектом, в котором оно инкапсулировано.

Полиморфизм – способность одного и того же программного кода работать с разнообразными данными. Другими словами, он допускает возможность в объектах разных типов иметь методы (процедуры или

функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента.

Наследование. Допускается порождение нового класса на основе уже существующего класса, и этот процесс называется наследованием. В этом случае новый класс, называемый подклассом существующего класса, наследует все атрибуты и методы класса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различают случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного класса, во втором случае – на основе нескольких классов. Набор классов образует иерархическую структуру.

Объектно-ориентированные базы данных.

К настоящему моменту терминология еще не устоялась, существует много разных определений и трактовок. Представляется, что объектно-ориентированная база данных (ООБД) – база данных, основанная на принципах объектно-ориентированной технологии. К основным описательным моментам, связанным с ООБД, относят:

- объекты (в ООБД любая сущность – объект и обрабатывается как объект); отметим, что здесь используется понятие «объект» объектно-ориентированного программирования, которое отличается от понятия «объект», рассматриваемого в ранее в данном учебном пособии.
- классы (понятие «тип данных» реляционной модели заменяется понятиями «класс» и «подкласс»);
- наследование (классы образуют иерархию наследования, заимствуя свойства друг друга);
- атрибуты (характеристики объекта моделируются его атрибутами);
- сообщения и методы (каждый класс имеет определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений);
- инкапсуляция (внутренняя структура объектов скрыта);
- идентификаторы объектов – дескрипторы.

Схема представления объекта приводится на рис. 54.

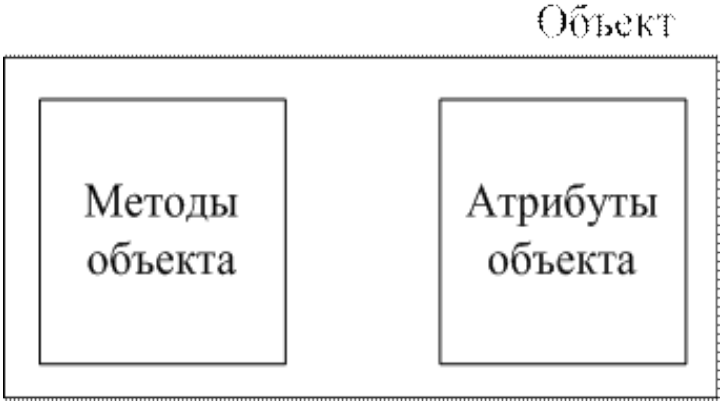


Рис. 54. Схема представления объекта

Система управления объектно-ориентированной базой данных называется объектно-ориентированной СУБД (ООСУБД). Цель ООСУБД – обеспечение постоянного хранения объектов, причем в отличие от традиционной СУБД ООСУБД должна хранить в составе объекта данные и программы.

Поскольку каждый объект данного класса имеет один и тот же набор методов, методы сохраняются только один раз – как методы класса (данные каждого экземпляра объекта хранятся отдельно).

Схема представления класса объектов приводится на рис. 55.

Рис. 55. Схема представления класса объектов

Используя наследование, всем объектам ПОДРАЗДЕЛЕНИЕ можно приписать свойство объекта-родителя (ФАКУЛЬТЕТ) – название факультета, номер факультета. Схема представления объектов ФАКУЛЬТЕТ и ПОДРАЗДЕЛЕНИЕ приводится на рис. 56.

Объект **Факультет**

Методы	**Атрибуты**
Создать объект	Название
Модифицировать	Номер
Удалить	Декан
Вывести на экран	Подразделение - класс
Выдать значения атрибутов _____ у объекта _____	

Объект **Подразделение**

Методы	**Атрибуты**
Создать объект	Название
Модифицировать	Зав. кафедрой
Удалить	Декан
Вывести на экран	Сотрудник - класс
Выдать значения атрибутов _____ у объекта _____	

Рис. 56. Фрагменты представления конкретных объектов

Сравнивая объектно-ориентированный и реляционный подходы к БД, можно отметить следующие особенности. В реляционных БД (РБД) реальные объекты представляются как структуры, состоящие из набора элементарных типов данных. Такое представление имеет понятную интерпретацию – строка в плоской таблице. В том случае, когда специфика предметной области позволяет работать с такого рода приближением реальных объектов, РБД отлично справляются со своей задачей. Довольно часто реляционная модель и ее способ описания предметной области в виде набора плоских таблиц не отражают внутренней структуры для многих предметных областей, являются искусственными и становятся совершенно непонятными при увеличении количества таблиц. Основная причина несостоятельности реляционного подхода заключается в слишком сильной абстракции реального объекта, что ведет к потере семантики.

В отличие от реляционных баз данных объектно-ориентированные базы данных обладают простой и естественной связью с предметной областью, представляя ее структуру и состав, что облегчает проектирование и положительно сказывается на понимании принципов функционирования программ. Так, в сложных неоднородных предметных областях использование ООБД (в частности, там, где разные объекты имеют разные

методы) должно действительно упростить процесс проектирования и разработки.

К сожалению, в ООБД существуют свои проблемы. В ООБД отсутствует универсальная модель данных, и соответственно, отсутствует мощная математическая база, как, например, в реляционной модели. В связи с этим у ООБД нет языка запросов высокого уровня, аналогичного SQL, и при доступе к данным используется мало эффективный навигационный подход. ООСУБД отличаются от реляционных СУБД тем, что программный интерфейс создания приложения либо очень слаб, либо вообще отсутствует. Это означает, для написании приложения, работающего с ООБД не существует мастеров и конструкторов (не считая, например, конструктора создания списка полей в объекте, который поставляется вместе с ООСУБД ObjectStore). Поэтому разработчик создает приложения на одном из алгоритмических языков.

По нашему мнению, существенным ограничением развития объектно-ориентированного подхода к созданию баз данных является то, что методы объекта содержатся внутри объекта и неразрывно связаны с ним. Это делает, по сути, невозможным создание для объектно-ориентированной базы данных соответствующей системы управления базой данных в традиционном понимании СУБД, функциями которой, в частности, является реализация операций обработки данных. Поэтому ООСУБД часто является не системой управления базами данных, а библиотекой программ, с помощью которой можно построить объектно-ориентированную базу данных. Примером такой библиотеки является ООСУБД ObjectStore. В связи с этим, возникает проблема реализации непредвиденных запросов.

Для перехода к объектно-ориентированным БД стандарт объектного программирования был дополнен стандартизованными средствами доступа к базам данных (стандарт ODMG 93; Object Database Management Group – группа управления объектно-ориентированными базами данных). К настоящему времени этот стандарт не реализован. Отметим только, что ООБД используются, но пока не стали реальной альтернативой реляционным базам данных.

Объектно-ориентированные возможности появляются в ведущих современных СУБД, таких, как, например, Oracle. Предпринимаются попытки внесения изменений в стандарты языка SQL с целью его частичной адаптации к ООБД. Так, новый стандарт SQL-3 включает большой раздел, посвященный этому вопросу.

Объектно-реляционные СУБД.

В настоящее время реляционные СУБД доминируют среди систем управления данными. Преимущества объектно-ориентированного подхода для создания сложных специализированных приложений с одной стороны, и стремление разработчиков систем управления базами данных с другой стороны расширить границы применения соответствующих СУБД обусловили включение объектно-ориентированных компонент (расширяемая пользователем система типов, инкапсуляция, наследование, полиморфизм и

т. п.) в модель данных реляционной СУБД. Соответствующие СУБД, называемые объектно-реляционными, соединяют в себе лучшие качества реляционных и объектно-ориентированных баз данных. Отметим, что в разных СУБД реализован разный набор из перечисленных объектно-ориентированных компонент. Таким образом, не существует общепринятой объектно-реляционной модели, а скорее имеется несколько таких моделей, поддерживающих определенный набор объектно-ориентированных компонент. Однако, основой всех таких моделей являются реляционные таблицы, используется язык запросов, включено понятие объекта, а в некоторых дополнительно реализована возможность сохранения методов в базе данных.

Соответствующие изменения реляционной модели обусловили необходимость расширения стандарта языка запросов SQL. Первый вариант такого стандарта получил название SQL3. Работа над стандартом продолжается и в настоящее время.

В качестве примера в максимальной степени объектно-ориентированной СУБД можно указать исследовательскую СУБД Postgres.

Отметим считающиеся объектными расширениями элементы СУБД Microsoft Server 2008.

Пользовательские расширения. Пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые пользовательские типы данных.

Хранение больших объемов данных. Наряду с теми данными, которые хранились в БД традиционно, Microsoft SQL Server 2008 позволяет хранить в столбцах таблицы данные больших размеров (поддерживаются соответствующие типы данных).

Новые, ориентированные на определенные классы объектов, типы данных. В системе определены новые типы данных (geometry, geography), характерные для тех направлений, в которых объектно-ориентированный подход весьма эффективен и часто используется (картография и соответствующие приложения, геометрическое представление объектов самой разной природы).

Хранимые процедуры. В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (стандартный для ООП процедурный подход).

Распределенные базы данных.

База данных – интегрированная совокупность данных, с которой работают много пользователей. Изложение всех предыдущих разделов предполагало единую базу данных, размещаемую на одном компьютере. Напомним основные принципы, положенные в основу теории баз данных:

- централизованное хранение данных;
- централизованное обслуживание данных (ввод, корректировка, чтение, контроль целостности).

Заметим, что базы данных появились в период господства больших ЭВМ. База данных велась на одной ЭВМ, все пользователи работали именно на ЭВМ. Других вариантов использования вычислительной техники в то время просто не существовало. Если проанализировать работу пользователей с данными в компаниях, организациях, предприятиях в «докомпьютерное» время, то нетрудно заметить, что на отдельных участках пользователи работали со «своими» данными (осуществляли сбор определенных данных, их хранение, обработку, передачу обработанных данных на другие участки или уровни управления).

У такой технологии были существенные недостатки, которые уже отмечались в предыдущих разделах: дублирование некоторых данных, отсутствие возможности сравнительного анализа данных всех участков. Однако у этой технологии были и существенные достоинства: данные вводились и хранились в местах их порождения; с этими данными работал пользователь, являющийся специалистом именно по этим данным, что позволяло ему вести эффективный контроль правильности данных на всех стадиях обработки; данные находились непосредственно у пользователя, что давало возможность их оперативной обработки. Централизация данных на одной ЭВМ, несомненно, дающая эффективные возможности хранения и обработки данных, не позволяла реализовывать вышеназванные достоинства.

Развитие вычислительных компьютерных сетей обусловило новые возможности в организации и ведении баз данных, позволяющие каждому пользователю иметь на своем компьютере свои данные и работать с ними и в то же время позволяющие работать всем пользователям со всей совокупностью данных как с единой централизованной базой данных. Соответствующая совокупность данных называется распределенной базой данных.

Термин «**распределенная база данных**» достаточно часто встречается в литературе , однако в разных источниках под этим термином понимаются совершенно разные вещи. Часть авторов понимают под распределенной базой данных то, что имеется удаленный сервер, на котором расположены данные, а также клиентские компьютеры, расположенные территориально в другом месте. Такая трактовка нам представляется неправильной. Настоящая **распределенная база данных** располагается на нескольких компьютерах. При этом часть файлов расположена на одном компьютере, часть на другом и т.д. Более того, возможна и даже часто встречается ситуация, когда информация на этих компьютерах пересекается, дублируется.

Распределенная база данных – совокупность логически взаимосвязанных разделяемых данных (и описаний их структур), физически распределенных в компьютерной сети.

Система управления распределенной базой данных – программная система, обеспечивающая работу с распределенной базой данных и позволяющая пользователю работать как с его локальными данными, так и со всей базой данных в целом.

Система управления распределенной базой данных (РаСУБД) является распределенной системой. Каждый фрагмент базы данных работает под управлением отдельной СУБД, которая осуществляет доступ к данным этого фрагмента. Пользователи взаимодействуют с распределенной базой данных через локальные и глобальные приложения. Локальные приложения дают пользователю возможность работать со своими локальными данными и не требуют доступа к другим фрагментам. Глобальные приложения дают пользователю возможность работать с другими фрагментами базы данных, расположенными на других компьютерах сети. Общая схема распределенной базы данных представлена на рис. 57.

Объединение данных организуется виртуально. Соответствующий подход, по сути, отражает организационную структуру предприятия (и даже общества в целом), состоящего из отдельных подразделений. Причем, хотя каждое подразделение обрабатывает свой набор данных (эти наборы, как правило, пересекаются), существует необходимость доступа к этим данным как к единому целому (в частности, для управления всем предприятием).

Одним из примеров реализации такой модели может служить сеть Интернет: данные вводятся и хранятся на разных компьютерах по всему миру, любой пользователь может получить доступ к этим данным, не задумываясь о том, где они физически расположены.

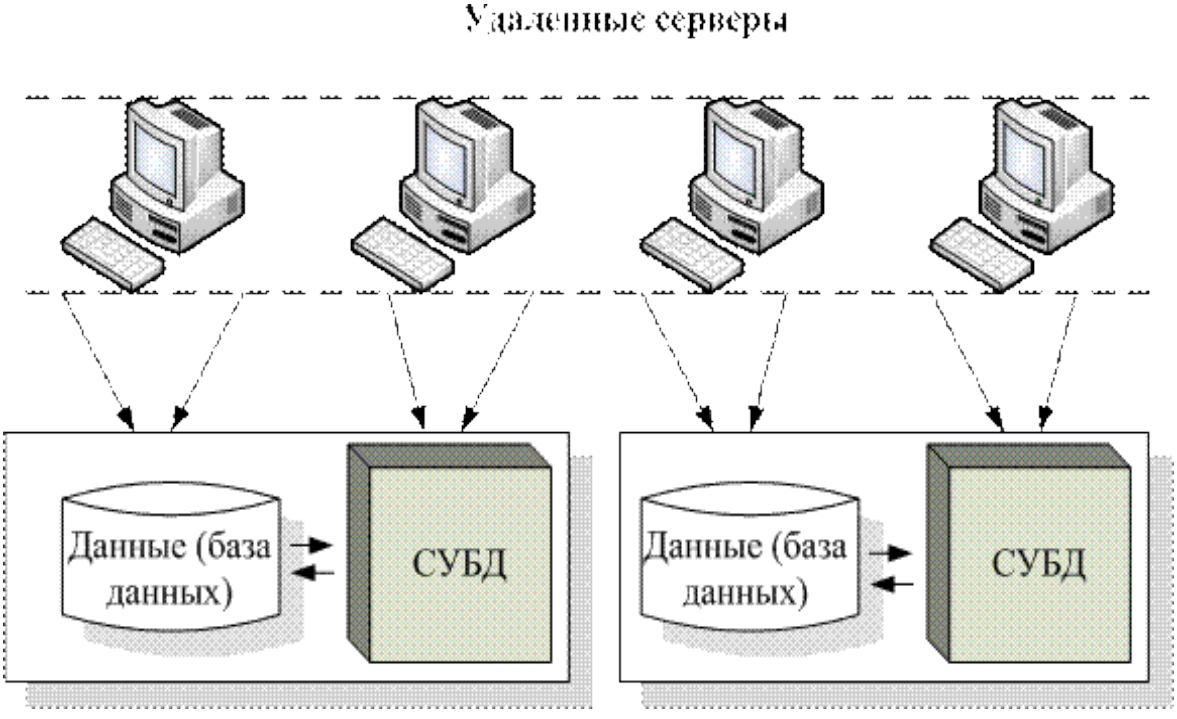


Рис. 57. Распределенная база данных

К.Дж. Дейт провозглашает следующий фундаментальный принцип распределенной базы данных. Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная. Из этого принципа следует ряд правил:

- Локальная автономия.
- Независимость от центрального узла.
- Непрерывное функционирование.
- Независимость от расположения.
- Независимость от фрагментации.
- Независимость от репликации.
- Обработка распределенных запросов.
- Управление распределенными транзакциями.
- Независимость от аппаратного обеспечения.
- Независимость от операционной системы.
- Независимость от сети.
- Независимость от СУБД.

Заметим, что понятие распределенной базы данных можно интерпретировать как следующий шаг в развитии понятий о данных, обусловленный распределенностью данных в реальных предметных областях, а также новым этапом развития средств вычислительной техники – широким использованием вычислительных сетей.

В этой интерпретации распределенную базу данных можно понимать как совокупность логически взаимосвязанных распределенных по разным компьютерам баз данных.

Перечислим основные проблемы создания распределенной базы данных.

Фрагментация данных и распределение по компьютерам.

Составление глобального каталога, содержащего информацию о каждом фрагменте БД и его местоположении в сети. (Каталог может храниться на одном узле или быть распределенным)

Организация обработки запросов (синхронизация нескольких запросов к одним и тем же данным, исключение аномалий удаления и обновления одних и тех же данных, расположенных на различных узлах, оптимизация последовательности шагов при обработке запроса и т.д.).

Значительным достоинством этой модели является приближение данных к месту их порождения, что позволяет существенно повысить их достоверность, недостатком – достаточно высокая сложность управления данными как единым целым.

К сожалению, процесс создания и обслуживания распределенных баз данных связан и с техническими трудностями, среди которых можно выделить жесткие требования к пропускной способности каналов связи, а также низкую производительность, обусловленную значительными затратами коммуникационных и вычислительных ресурсов при их синхронизации во время выполнения транзакций (особенно при интенсивных обращениях из разных узлов к одному фрагменту).

Технология, связанная с использованием распределенных баз данных, в наибольшей степени соответствует организационной человеческой деятельности (информация распределена по месту деятельности людей, и они

обмениваются ей в процессе работы) и позволяет наиболее успешно решать важнейшие проблемы, ведения баз данных:

- повысить достоверность информации (информация вводится в месте ее порождения лицом, которое лучше всех понимает ее смысловое значение);
- повысить оперативность локальной обработки информации (соответствующие вопросы решаются на локальном компьютере с фрагментом базы данных).

Поэтому очевидно, что задача проектирования, создания и функционирования распределенных баз данных является весьма существенной, активно изучается в настоящее время и будет решаться и далее.

Хранилища данных.

Как уже неоднократно отмечалось, технологии баз данных предназначены, как правило, для решения текущих задач обработки данных организации. В базу данных постоянно вносятся изменения, то есть база данных отражает моментальный снимок определенной области деятельности предприятия. Для эффективного принятия решений руководством при управлении организацией важно не только знать текущее положение дел, но и иметь возможность анализировать динамику (изменение во времени) основных показателей, причем, зачастую из разных баз данных. Такую возможность дает технология так называемых хранилищ данных.

Приведем определение хранилища данных (Bill Inmon).

Хранилище данных – предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений.

Под **предметной ориентированностью** здесь понимается ориентированность на предметы (определенные группы данных), а не на конкретные приложения. Например, ориентация на данные о сотрудниках, а не только о расчете их заработной платы. Под **интегрированностью** здесь понимается возможное объединение данных из разных источников (баз данных), имеющих разный формат и несогласованных. **Привязка ко времени** предполагает, что для всех данных указан момент или промежуток времени, в который они корректны. Данные в хранилище не изменяются, они лишь регулярно пополняются из оперативных баз данных. Общая схема взаимодействия информационного хранилища и баз данных приводится на рис. 58.

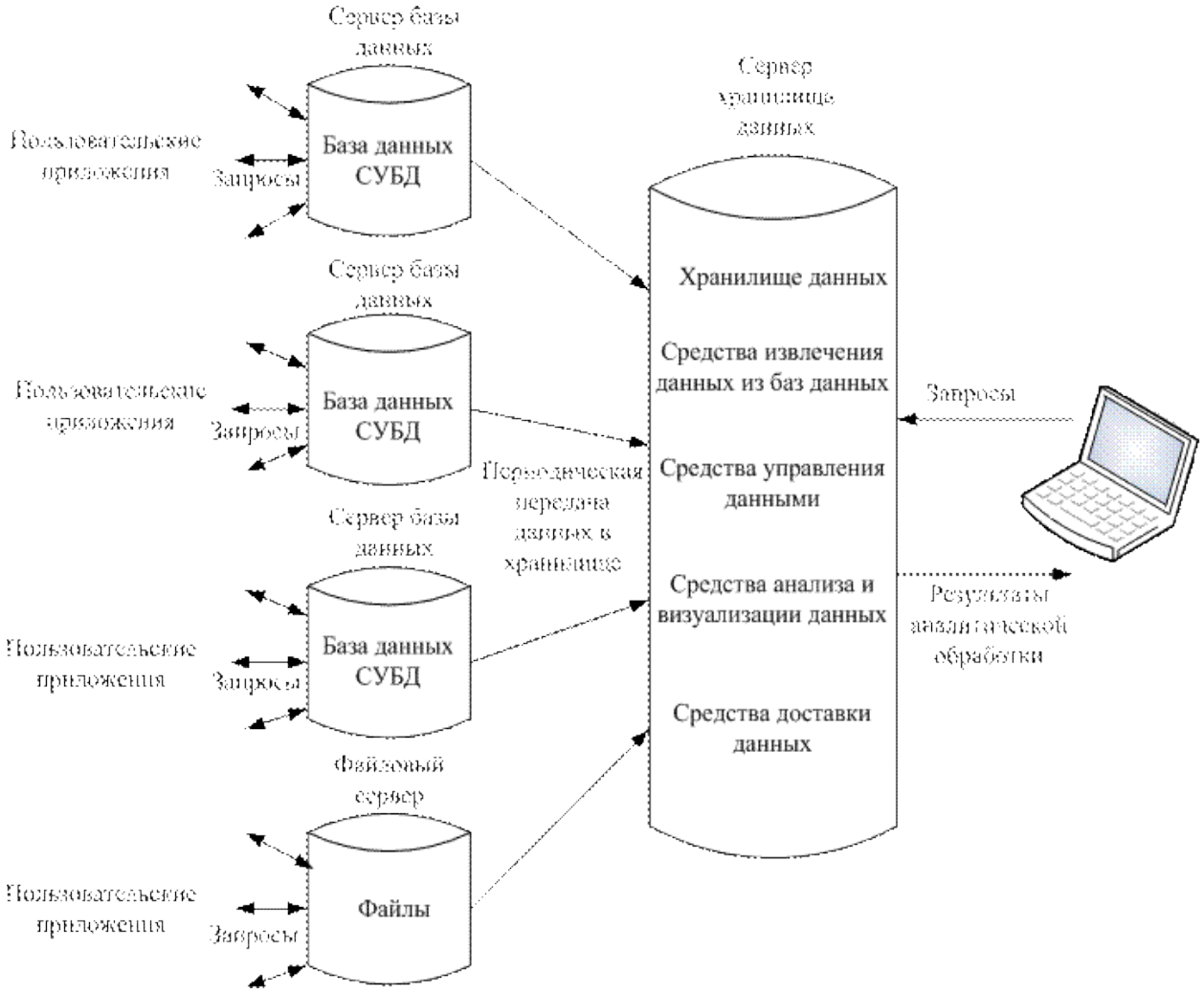


Рис. 58. Схема организации работы хранилища данных

Еще раз подчеркнем, что основной целью хранилищ данных является бизнес-анализ или информационная поддержка принятия управленческих решений.

Для реализации всей необходимой обработки информации в соответствии с этой схемой необходимы следующие программные средства:

- средства извлечения данных из баз данных;
- средства управления данными хранилища (система управления базой данных хранилища);
- средства анализа данных хранилища (используется OLAP-технология):
- средства доставки данных;
- средства визуализации результатов обработки для конечных пользователей.

Для работы соответствующих программных средств необходимо описание структуры содержимого информационного хранилища (метаописание).

Для самого общего случая, если данные берутся из баз данных, управляемых разными СУБД, из файлов разных типов, а данные разнородны, средства управления данными хранилища пока не созданы. Однако, если данные в информационное хранилище выбираются только из реляционных баз данных, то в качестве средств управления данными хранилища может быть взята мощная реляционная СУБД. Поэтому разработчики современных СУБД включают в состав программного обеспечения СУБД средства организации работы с хранилищами данных.

Рассмотрим в качестве **примера** возможности СУБД Microsoft SQL Server 2008 для организации хранилищ данных.

Microsoft SQL Server 2008 содержит в своем составе средства извлечения, преобразования и загрузки данных (SQL Server 2008 Integration Services), способные интегрировать данные из различных источников, проверять данные на допустимость и преобразовывать перед загрузкой в хранилище. Эти средства также способствуют перемещению данных, поддерживают текстовый анализ и нечеткий поиск. Нужно отметить также среду визуальной разработки (Business Intelligence Development Studio) для создания многомерных кубов, отчетов, пакетов извлечения, преобразования и загрузки данных.

Существенной особенностью хранилищ данных является их очень большой объем. Microsoft SQL Server 2008 как средство управления данными хранилища позволяет работать с большими объемами данных, причем для сокращения времени обработки предусмотрена поддержка параллельных вычислений (путем разделения таблиц и индексов на секции о обеспечение параллельной обработки секций). В системе предусмотрена возможность сжатия данных (таблиц), что позволяет уменьшить физический размер таблиц и существенно сокращает время обмена между оперативной и внешней памятью.

В качестве средств анализа данных хранилища используется SQL Server 2008 Analysis Services, применяемый для построения многомерных кубов (многомерных моделей данных). Это средство содержит семь эффективных алгоритмов анализа данных с целью поддержки принятия управленческих решений, в том числе анализ тенденций и статистический анализ данных.

В качестве средств представления аналитических данных пользователям предлагается использовать средство генерации отчетов SQL Server 2008 Reporting Services.

Таким образом, Microsoft SQL Server 2008 является эффективным средством реализации хранилищ данных на основе реляционных баз данных.

Литература

Основная литература:

1. Дейт К. Дж. Введение в системы баз данных – Москва – Санкт-Петербург - Киев: Вильямс, 2006 г. - 1328 с.

2. Кузнецов С. Д. Базы данных.Модели и языки – М.: Бином-Пресс, 2008 г. - 720 с.

Дополнительная литература:

1. Дженнифер Уидом, Гектор Гарсиа-Молина, Джеффри Ульман. Системы баз данных. Полный курс. (Database Systems. The Complete Book). – Москва – Санкт-Петербург - Киев: Вильямс, 2003 г. - 1088 с.

2. Крейг С. Маллинс Администрирование баз данных. Полное справочное руководство по методам и процедурам / Пер. с англ. - М.: КУДИЦ-Образ, 2003.-752 с.

3. Крёнке Д. Теория и практика построения баз данных (9-е издание). – СПб.: Питер, 2004 г. - 864 с.

4. Кузнецов С. Д. Основы баз данных. – М.: Интернет-университет информационных технологий, 2007 г. - 488 с.

5. Никандрова Ю.А. Основы SQL: Учебное пособие для студентов ВУЗов. - Саратов, «Наука», 2009. - 84с.

6. Никандрова Ю.А. Основы семантического моделирования: Учебно-методическое пособие для студентов ВУЗов. - Саратов, «Наука», 2010. - 28с.

7. Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006 г. - 352 с.

Интернет-ссылки:

№	Наименование портала (издания, курса, документа)	ссылка
Основные учебные материалы		
1.	Кузнецов С.Д. Информационно-аналитические материалы. «Основы современных баз данных»	http://citforum.ru/database/osbd/contents.shtml
2.	Кузнецов С.Д. «Базы данных. Вводный курс»	http://citforum.ru/database/advanced_intro/
3	Библиотека MSDN	http://msdn.microsoft.com/ru-ru/library/

Контрольные вопросы и задания

Теоретические вопросы на знание базовых понятий предметной области курса.

Дать определение понятию:

- «данные»;
- «структура данных»;
- «тип данных»;
- «модель»;
- «объект»;
- «класс объектов»;
- «сущность»;
- «связь»;

- «атрибут»;
- «отношение»;
- «предметная область»;
- «база данных»;
- «система управления базами данных»;
- «атрибут»;
- «домен»;
- «кортеж»;
- «кардинальное число отношения»;
- «степени отношения»;
- «первичный ключ»;
- «внешний ключ»;
- «репликация».

Теоретические вопросы, позволяющие оценить степень владения студента терминологией, основными понятиями и принципами предметной области курса, понимание их особенностей и взаимосвязей между ними..

1. Охарактеризовать...
- объектные базы данных;
- объектно-реляционные базы данных;
- реляционные базы данных;
- линейные структуры;
- нелинейные структуры;
- сетевые структуры;
- логическую модель данных;
- концептуальную модель данных;
- физическую модель данных.

2. Описать:
- ER-модель, основные понятия и правила преобразования ER-диаграмм в реляционные таблицы;
- основные требования к модели данных;
- функциональную зависимость атрибутов;
- неполную функциональную зависимость атрибутов;
- транзитивную функциональную зависимость атрибутов;
- метод декомпозиции;
- 1НФ отношения;
- 2НФ отношения;
- 3НФ отношения;
- НФБК отношения;
- 4НФ отношения.

3. Охарактеризовать и привести примеры на такие операции реляционной алгебры, как …

- выборка;
- проекция;
- объединение;
- разность;
- произведение;
- пересечение;
- соединение;
- деление.

Задания на анализ ситуации из предметной области курса с применением соответствующих принципов и методов решения практических проблем, близких к профессиональной деятельности.

1. Проанализировать…

- многоуровневые модели предметной области;
- атрибутивный и опосредованный способы идентификации объектов;
- свойства реляционной структуры данных;
- реляционные операции умножения и деления в точки зрения обратимости;
- двенадцать правил Кодда;
- свойства транзакции и варианты ее завершения;
- модель файлового сервера в архитектуре «клиент-сервер»;
- модель удаленного доступа к данным в архитектуре «клиент-сервер»;
- модель активного сервера в архитектуре «клиент-сервер»;
- модель сервера приложений в архитектуре «клиент-сервер.

2. Дать сравнительную характеристику…

- понятий «структура данных», «структура записи», «структура информации»;
- понятий «модель данных» и «модель базы данных»;
- основных типов нелинейных структур.

Задания на проверку умений и навыков, полученных в результате освоения курса.

1. Построить и описать

- ER- диаграмму на заданную тему;
- реляционную модель на заданную тему;
- примеры для восьми основных операций реляционной алгебры.

2. Определить через примитивные операции реляционной алгебры операцию…

- соединения;

- пересечения;
- деления.

3. Построить SQL-запрос…
- многотабличный на выборку данных;
- с условием (сравнение);
- с условием (диапазон);
- с условием (принадлежность множеству);
- с условием (соответствие шаблону);
- с построением вычисляемых полей;
- с использованием итоговых (агрегатных) функций;
- с использованием итоговых функций и GROUP BY;
- с использованием итоговых функций и HAVING;
- включающий подзапрос, возвращающий единичное значение;
- включающий подзапрос, возвращающий множество значений (IN и NOT IN);
- включающий подзапрос, возвращающий множество значений (ANY и ALL);
- включающий подзапрос, возвращающий множество значений (EXISTS и NOT EXISTS);
- на обновление строк;
- на добавление строк;
- на удаление строк;
- на объединение таблиц.

Перечень вопросов и типовых заданий для промежуточной аттестации.

Вопросы:

1. Базы данных. Банки данных. СУБД. Базы знаний.
2. Виды баз данных.
3. Жизненный цикл баз данных.
4. Характеристика рынка СУБД.
5. Сравнительная характеристика персональных и серверных СУБД.
6. Работа СУБД в архитектуре «клиент-сервер».
7. Характеристика серверных СУБД.
8. Выбор СУБД.
9. Сущности. Атрибуты. Связи. Концептуальная схема.
10. Порядок построения концептуальной схемы.
11. Построение концептуальной модели «Сущность-связь».
12. Логический уровень проектирования. Исходные данные и результат.
13. Понятие структуры данных. Логическая и физическая структуры данных
14. Типовые структуры (модели) данных, линейная структура.

15. Типовые структуры (модели) данных, иерархическая структура.
16. Типовые структуры (модели) данных, сетевая структура.
17. Требования к эксплуатационным характеристикам.
18. Иерархическая модель данных.
19. Сетевая модель данных.
20. Свойства двумерных таблиц. Пример реляционной модели.
21. Реляционная модель данных. Терминология.
22. Понятие функциональной зависимости.
23. Нормальные формы отношений.
24. Целостность по сущностям.
25. Целостность по ссылкам.
26. Математическое описание реляционной модели.
27. Реляционная алгебра. Теоретико-множественные операции.
28. Реляционная алгебра. Выборка.
29. Реляционная алгебра. Проекция.
30. Реляционная алгебра. Соединение.
31. Реляционная алгебра. Деление.
32. Реляционное исчисление.
33. Нормализация. Цель нормализации.
34. Приведение отношений к 3НФ.
35. Метод декомпозиции.
36. Реализация бинарной связи 1:1.
37. Реализация бинарной связи 1:m.
38. Реализация бинарной связи n:m.
39. Реализация n-арных связей.
40. Возможности команды SELECT.
41. SQL. Модификация данных.
42. SQL. Добавление записей.
43. SQL.Обеспечение целостности данных.
44. Распределённая обработка данных.
45. Распределённые базы данных. Технологии файл-сервер и клиент-сервер.

Лабораторный практикум по дисциплине. Типовые задания.

Поскольку дисциплина является практикоориентированной, то сдача экзамена по данной дисциплине предусмотрена с обязательной защитой лабораторного практикума.

Лабораторный практикум выполняется по одной из ниже перечисленных тем. При выборе темы следует учесть, что именно по этой теме будет осуществляться проектирование персональной базы данных и выполнение всего лабораторного практикума. В отдельных случаях, когда ни одна из тем не заинтересовала студента, он может определить ее индивидуально и, согласовав с преподавателем, выполнять лабораторный практикум в рамках инициативной темы.

Темы для проектирования базы данных.

1. Вуз.
2. Деканат.
3. Дисциплина (содержание, сопровождение, контингент и др.).
4. Преподаватели (учет должностей, званий, преподаваемых дисциплин и т.д.).
5. Учебная группа (учет студентов и оценок по дисциплинам и т.д.).
6. Факультет.
7. Государства (характеристика, виды, транспорт и т.д.).
8. Туризм (перечень предоставляемых услуг, заказ туров и др.).
9. Туристическое агентство (учет туров и предоставляемых услуг и др.).
10. Библиотека (формирование фонда, учет выдачи и возврата документов и др.).
11. Издательство.
12. Канцтовары.
13. Книжный магазин (учет поступления, реализации товаров и др.).
14. Бухучет (учет наличия, поступления и прохождения средств).
15. Документооборот предприятия (учет поступления, прохождения документов, их исполнения и др.).
16. Домашняя бухгалтерия.
17. Заработная плата (учет, выдача и др.) .
18. Кадры (учет сотрудников и др.).
19. Автосервис.
20. Автошкола.
21. Транспорт. Авиакасса.
22. Транспорт. Автопарк (виды, свойства, характеристики).
23. Транспорт. Автосалон (услуги, менеджеры, клиенты и др.).
24. Аудио и видеопродукция (коллекция, учет поступления, реализации товаров и т.д.).
25. Видеосъемки на заказ.
26. Интернет-кафе.
27. Организация экскурсий.
28. Ресторан (столовая, кафе и т.п.; учет продуктов, меню, персонал, заказ столиков и т.д.).
29. Устройство праздников.
30. Склад (учет поступления, реализации товаров и др.).
31. Магазин бытовой техники.
32. Магазин велосипедов и аксессуаров .
33. Магазин музыкальных инструментов.
34. Магазин спорттоваров.
35. Магазин стройматериалов.
36. Магазин фототехники. Печать фотографий.
37. Мебельный салон.
38. Производство и продажа алкоголя.

39. Мода. Магазин обуви (образцы, характеристика, модельеры и т.д.).
40. Мода. Сеть магазинов одежды (образцы, характеристика, модельеры и т.д.).
41. Народные промыслы (образцы, характеристика и др.).
42. Торговля рыбацким и охотничьим снаряжением.
43. Радио (каналы, частоты, программы, ведущие и др.).
44. Телевидение (каналы, частоты, программы, ведущие и др.).
45. Программное и техническое обеспечение (виды, характеристика и т.д.).
46. Продажа готовых компьютеров и сборка под заказ.
47. Провайдеры интернет услуг (виды, характеристика и др.).
48. Салон сотовой связи.
49. Парикмахерская.
50. Салон красоты.
51. Аптека.
52. Зал тренажеров.
53. Клиника пластической хирургии.
54. Санаторий.
55. Стоматологическая клиника.
56. Агентство недвижимости.
57. Агентство ритуальных услуг.
58. Банк.
59. Страховая компания.
60. Строительная компания.

Лабораторный практикум по дисциплине разделен на шесть тематических лабораторных практикума, перечисленных в табл. 18. Каждый практикум оценивается исходя из уровня сложности и объема заданий, что в общей сумме дает 100 баллов, отведенных на дисциплину.

Таблица 18.

Лабораторный практикум. Детализация

№	Содержание	Балл
Л./пр. №1	Анализ предметной области и построение ER-диаграммы	10
1	Анализ предметной области	5
2	Построение ER-модели (MS VISIO 2007 или 2010)	5
Л./пр. №2	Создание БД в MS SQL Server 2008 (в 3НФ, с наполнением)	10
1	Построение ER-модели (MS SQL Server 2008) в 3-й нормальной форме	5
2	Внесение данных в БД, удовлетворяющих выбранной тематике	5
Л./пр. №3	Построение запросов на выборку (по два запроса на каждый тип)	42
1	Простой запрос на выборку	2
2	Многотабличный запрос на выборку	2
3	Запрос с условием (сравнение)	2

4	Запрос с условием (диапазон)	2
5	Запрос с условием (принадлежность множеству)	2
6	Запрос с условием (соответствие шаблону)	2
7	Запрос с построением вычисляемых полей	2
8	Запрос с использованием итоговых (агрегатных функций)	4
9	Запрос с использованием итоговых функций (+ GROUP BY)	4
10	Запрос с использованием итоговых функций (+ HAVING)	4
11	Подзапросы, возвращающие единичное значение	4
12	Подзапросы, возвращающие множество значений (IN и NOT IN)	4
13	Подзапросы, возвращающие множество значений (ANY и ALL)	4
14	Подзапросы, возвращающие множество значений (EXISTS и NOT EXISTS)	4
Л./пр. №4	Построение запросов на модифицирование данных (по два запроса на каждый тип)	10
1	Запрос на обновление (UPDATE)	2
2	Запрос на добавление (INSERT INTO)	2
3	Запрос на удаление (DELETE)	2
4	Запрос на объединение (UNION)	4
Л./пр. №5	Определение ограничений целостности (по два запроса на каждый тип)	18
1	Запрос на создание таблицы	2
2	Запрос на удаление таблицы	1
3	Запрос на добавление столбца в таблицу (с атрибутом NULL и NOT NULL)	2
4	Запрос на удаление столбца таблицы	1
5	Запрос на задание для столбца значения по умолчанию	3
6	Запрос на отмену для столбца значения по умолчанию	3
7	Запрос на добавление в определение таблицы нового ограничения	3
8	Запрос на удаление из определения таблицы существующего ограничения	3
Л./пр. №6	Описание проекта БД (по л/пр. 1-5)	10
		100

В лабораторном практикуме №1 осуществляется первый этап проектирования любой базы данных: семантическое моделирование данных – моделирование предметной области с учетом смысла данных, результатом которого является структура проектируемой базы данных. В качестве инструмента семантического моделирования применяется наиболее часто используемая на практике модель «Сущность-Связь» (Entity-Relationship), сокращенно ER-модель или ER-диаграмма.

Процесс моделирования разделен на две задачи: анализ предметной области и построение ER-модели. Семантическое моделирование осуществляется в MS Office Visio (2007 или 2010).

Ниже перечислены условия для успешной сдачи лабораторного практикума №1.

1. Провести анализ предметной области: дать краткую словесную характеристику по выбранной теме, выделить сущности (не менее 5) и их

атрибуты, выявить отношения между сущностями (письменно, в формате MS Office Word).

2. Создать ER-модель с обязательным определением типов отношений между сущностями, идентификационных, обязательных и необязательных, множественных атрибутов.

3. Предоставить на проверку ER-модель (MS Office Visio) и описание предметной области (в формате MS Office Word).

В лабораторном практикуме №2 необходимо создать базу данных в MS SQL Server 2008, используя ER-модель, построенную в ходе выполнения лабораторного практикума №1. Процесс создания разделён на две задачи: реализация ER-модели в MS SQL Server 2008 с приведением к 3-й нормальной форме с указанием типов связей между отношениями и внесение данных (не менее 10 записей для каждого отношения). Вносимые данные должны иметь смысл.

Ниже перечислены условия для успешной сдачи лабораторного практикума №2.

1. Создать ER-модель в 3-й нормальной форме (MS SQL Server 2008) с обязательным определением типов связей между отношениями.

2. Внести данные по выбранной теме.

3. Предоставить на проверку базу данных и доказать, что она находится в 3-й нормальной форме.

Лабораторные практикумы №3, 4, 5 направлены на выработку практических навыков работы с базой данных и формирования умения составления запросов на выборку, на модифицирование данных, определения ограничений целостности.

В лабораторных практикумах использованы стандартные конструкции SQL, которые остаются неизменными в той или иной СУБД, что делает их универсальными и независимыми от используемого программного продукта.

Ниже перечислены условия для успешной сдачи лабораторных практикумов №3, 4, 5.

1. Реализовать все указанные запросы (см. табл. 18) в разработанной базе данных (см. л./пр. №2) со смысловым значением для рассматриваемой предметной области.

2. Продемонстрировать работоспособность всех запросов, входящих в практикум и уметь написать любой тип запроса по заданным преподавателем условиям.

В лабораторном практикуме №6 необходимо составить отчет - словесное описание созданной базы данных в MS SQL Server 2008 с приведением скриншотов (ER-модели, содержания таблиц), реализованных в ней запросов с приведением полного текста запроса и пояснением того, что делает каждый из них.

Отчет должен состоять из введения, описания предметной области, описания построения запросов, заключения и списка литературы. Наличие ссылок в тексте на использованную литературу обязательно. Отчет должен представлять собой связанное описание пройденного лабораторного практикума, начиная с характеристики предварительно созданной базы данных (по выбранной теме) и разработанных запросов и заканчивая анализом проделанной работы.

Ниже приведена структура отчета с краткими комментариями к каждой части.

1. Введение.

Краткая постановка задачи (формулирование цели создания БД и подзадач, реализуемых в БД).

2. Описание предметной области.

Обязательно должны быть представлены описания всех сущностей и их взаимосвязи, ER-модель (MS Office Visio)[24], описания таблиц (с указанием типов данных и ограничений) и ER-модель в 3-й нормальной форме (MS SQL Server 2008)[25].

3. Проектирование запросов.[26]

1) Однотабличные запросы.
2) Многотабличные запросы на выборку.
3) Запросы с условием (сравнение).
4) Запросы с условием (диапазон).
5) Запросы с условием (принадлежность множеству).
6) Запросы с условием (соответствие шаблону).
7) Запросы с построением вычисляемых полей.
8) Запросы с использованием итоговых (агрегатных функций).
9) Запросы с использованием итоговых функций (+ GROUP BY).
10) Запросы с использованием итоговых функций (+ HAVING).
11) Подзапросы, возвращающие единичное значение.
12) Подзапросы, возвращающие множество значений (IN и NOT IN).
13) Подзапросы, возвращающие множество значений (ANY и ALL).
14) Подзапросы, возвращающие множество значений (EXISTS и NOT EXISTS).
15) Запросы на обновление (UPDATE).
16) Запросы на добавление (INSERT INTO).
17) Запросы на удаление (DELETE).
18) Запросы на объединение (UNION).
19) Запросы на создание таблицы.
20) Запросы на удаление таблицы.
21) Запросы на добавление столбца в таблицу (с атрибутом NULL и NOT NULL).
22) Запросы на удаление столбца таблицы.
23) Запросы на задание для столбца значения по умолчанию.

24) Запросы на отмену для столбца значения по умолчанию.
25) Запросы на добавление в определение таблицы нового ограничения.
26) Запросы на удаление из определения таблицы существующего ограничения.

На каждый пункт необходимо создать два различных запроса, имеющих смысловую нагрузку. Привести для каждого запроса формулировку, текст запроса, пояснения (если необходимо), скриншот выполненного запроса.

Пример описания запроса № 2 из л./пр. №3:
Запрос 3.2. Вывести ФИО клиентов, сделавших заказ.
Информация о клиентах хранится в таблице «Клиент», а информация о заказе – в таблице «Заказ», данные таблицы имеют один общий по смыслу столбец «КодКлиента» (названия столбцов в обеих таблицах совпадает, это допустимо). Составляем запрос, объединяя данные таблицы по столбцу «КодКлиента»:

```
SELECT Клиент.Фамилия, Клиент.Имя, Клиент.Отчество
FROM Клиент INNER JOIN Заказ ON Клиент.КодКлиента = Заказ.КодКлиента;
```

В результате выполнения запроса (рис. 59) получили таблицу, содержащую повторяющиеся строки.

Активные клиенты : запрос на выборку

Фамилия	Имя	Отчество
Иванов	Иван	Петрович
Петров	Иван	Иванович
Петров	Иван	Иванович
Сидоров	Петр	Петрович
Борисова	Ольга	Максимовна
Борисова	Ольга	Максимовна
Максимова	Анастасия	Павловна
Голубева	Алена	Анатольевна
Голубева	Алена	Анатольевна
Голубева	Алена	Анатольевна
Макарова	Татьяна	Николаевна
Макаров	Владимир	Сергеевич
Сергеев	Михаил	Яковлевич
Михайлов	Владимир	Петрович
Михайлова	Наталья	Юрьева
Матвеев	Дмитрий	Михайлович

Рис. 59. Клиенты, сделавшие заказ

4. Заключение.
Краткая сводка всех полученных результатов. Вывод. Дальнейшее развитие базы данных.

5. Литература.
Список использованной литературы.

[1] Определения понятий «Первичный ключ», «Внешний ключ», «Домен» приведены по учебнику: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[2] Материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/databases/2/1.html.
[3] Теоретические материалы главы приведены из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[4] Материал параграфа приведен по учебнику: Грекул В. И. Проектирование информационных систем. [Электронный ресурс] // Интернет-университет информационных технологий - ИНТУИТ.ру, 2005.– Режим доступа: http://www.intuit.ru/department/se/devis/.
[5] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[6] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[7] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[8] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[9] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[10] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[11] В математических дисциплинах понятию «таблица» соответствует понятие «отношение» (relation). Отсюда и произошло название модели – реляционная. Т.е., применительно к базам данных понятия «реляционная БД» и «табличная БД» по существу являются синонимами.
[12] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[13] Теоретический материал параграфа приведен из учебника: Попов И. И., Максимов Н. В., Голицына О. Л. Базы данных. – М.: ФОРУМ: Инфра-М, 2006. – 352 с.
[14] Теоретический материал главы приведен из учебника: Полякова Л. Н. Основы SQL. [Электронный ресурс] // Интернет-университет информационных технологий - ИНТУИТ.ру, 2004.– Режим доступа: http://www.intuit.ru/department/database/sql/.
[15] Теоретический материал параграфа приведен из учебника: Полякова Л. Н. Основы SQL. [Электронный ресурс] // Интернет-университет информационных технологий - ИНТУИТ.ру, 2004.– Режим доступа: http://www.intuit.ru/department/database/sql/.
[16] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/databases/10/.
[17] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/olap/2/.
[18] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/olap/2/.
[19] Теоретический материал параграфа приведен по материалам интернет-источника: http://www.eduhmao.ru/info/1/3611/22384/.
[20] Теоретический материал параграфа приведен по материалам интернет-источников: http://www.ibm.com/developerworks/ru/library/sabir/axd_2/index.html?S_TACT=105AGX99&S_CMP=GR01 и http://www.ibm.com/developerworks/ru/library/sabir/axd_3/index.html.
[21] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/graphics/compression/6/3.html.
[22] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/databases/.
[23] Теоретический материал параграфа приведен из интернет-источника: http://www.intuit.ru/department/database/databases/14/.
[24] По лабораторному практикуму №1.
[25] По лабораторному практикуму №2.
[26] По лабораторным практикумам №3, 4, 5.