Министерство образования Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерные системы в управлении и проектировании (КСУП)

Т.Д. Карминская

БАЗЫ ДАННЫХ В САПР

Учебное пособие

2000

Рецензент:
ведущий специалист отдела информатизации
Администрации Томской области Омельченко М.В.

Карминская Т.Д.
Базы данных в САПР: Учебное пособие. - Томск: Томский межвузовский центр дистанционного образования, 2000. - 42 с.

Учебное пособие содержит теоретический материал по вопросам принципов проектирования физической базы данных. Рассмотрены технологии клиент-сервер и распределенные системы для организации информационного обеспечения автоматизированных систем и систем управления. Является логическим продолжением учебного курса «Информационное обеспечение».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сегодня трудно себе представить реальную большую автоматизированную систему обработки информации, которая не содержала бы в качестве основной компоненты автоматизированный банк данных. Автоматизированные системы управления различными объектами, системы автоматизированного проектирования и проведения научных исследований, системы информационного обслуживания и многие другие связаны с выполнением основных функций банков данных: восприятия сведений о реальных объектах, формирование больших и сложных баз данных, их ведения и представления данных для обработки и принятия управленческих решений.

1. МОДЕЛИРОВАНИЕ ЛОКАЛЬНЫХ ПРЕДСТАВЛЕНИЙ

При создании описания предметной области проектировщик часто использует прием разбивки предметной области на ряд *локальных областей*, что позволяет учесть все связи между данными и более регулярно структурировать предметную область. В дальнейшем эта информация используется для построения модели «сущность - связь».

Для представления информации в модели «сущность-связь» конструктивными элементами являются *сущность*, *атрибут*, *связь*.

Основным элементом локального представления некоторого явления, процесса или объекта, о котором необходимо собрать информацию, является *сущность*. Необходимо различать такие понятия, как ТИП И ЭКЗЕМПЛЯР СУЩНОСТИ.

Понятие *тип сущности* относится к набору однородных предметов или явлений, выступающему как целое (группа студентов, телевизор).

Экземпляр сущности – конкретный элемент набора (телевизор Рубин).

На концептуальном этапе проектирования необходимо сформулировать сущности, требуемые для описания локального представления.

При этом возникает проблема ее выделения в качестве **конструктивного элемента**, т.к. некоторая информация может быть представлена, как атрибут, сущность, связь.

Например, информация студент учится в ТУСУРе, может быть выражена сущностью – СТУДЕНТ; связью – УЧИТЬСЯ (между сущностями СТУДЕНТ-ТУСУР). Атрибут сущности – N группы.

При возникновении такой необходимости следует руководствоваться следующими правилами:

1) необходимо выбирать вариант более гибкий с точки зрения представления информации, т.е. позволяющий представить не только всю часть некоторой информации, но и ее отдельные фрагменты;

2) для моделирования порции информации должна использоваться одна и только одна конструкция, т.е. следует избегать избыточности в использовании конструктивных элементов;

3) необходимо ответственно подходить к вопросу выбора **наименования сущностей** (т.е. формулирование сущностей), т.к. это имеет важное значение для стадии объединения локальных представлений.

1.1. Выбор атрибутов сущностей

Свойства сущностей определяются с помощью атрибутов.

Атрибут – это характеристика сущности, имеющая имя и используется для определения того, какая информация должна быть собрана о сущности (например, сущность – СТУДЕНТ, номер зачетной книжки, номер группы и другое).

Атрибут, значение которого единственным образом определяет экземпляр сущности, называют **ключом**.

Таким образом *атрибуты делятся на два класса*:

- те, которые служат для идентификации экземпляра сущности, т.е. являются ключами (например, Фамилия, Имя, Отчество).
- те, которые описывают свойства сущностей.

В любом случае, в процессе выбора атрибутов, необходимо каждому ставить в соответствие следующие характеристики:

- наименование, т.е. уникальное обозначение атрибута;
- описание – словесное изложение смысла атрибута;
- роль – т.е. конкретное использование атрибута.

1.2. Спецификация связей

После выделения сущностей и соответствующих атрибутов локальное представление дополняется информацией, раскрывающей зависимости между экземплярами сущностей.

Одна из неформальных процедур – попарное объединение между собой всех экземпляров сущностей и установление существования некоторой связи для каждой пары.

После их выявления определяются связи необходимые и избыточные. Каждой необходимой связи присваивается имя и определяются ее характеристики:

один – к - одному (1:1);

один – ко – многим (1:M);

многие - ко - многим (M : M).

2. ОБЪЕДИНЕНИЕ МОДЕЛЕЙ ЛОКАЛЬНЫХ ПРЕДСТАВЛЕНИЙ

В результате объединения локальных представлений получается единая глобальная информационная структура. *Объединение может быть осуществлено на базе трех основополагающих подходов:*

- идентичности;
- агрегации;
- обобщение.

2.1. Метод идентичности

Идентичность позволяет объединять несколько сущностей путем объединения двух или более элементов **синонимами**.

Для проверки согласованности результата объединения локальных представлений на основе понятия идентичности существует следующее правило:

Если объект из одного локального представления идентичен объекту из другого представления, ни один из этих объектов ***не должен в дальнейшем принимать участие в каком-либо другом объединении идентичности*** *между этими двумя представлениями.*

Сущность применения

В нескольких локальных представлениях рассматривается один и тотже объект, но его отдельные составляющие могут различаться.

В результате объединения идентичности вместо отдельных локальных представлений будет построено новое.

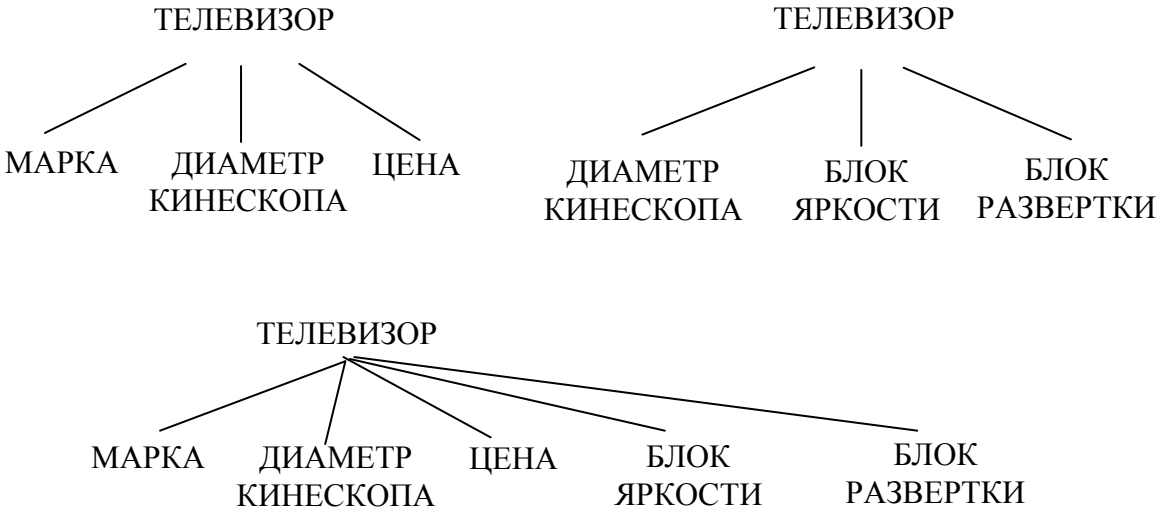


Рис. 2.1. Объединение методом идентичности

2.2. Метод агрегации

Агрегация позволяет рассматривать связь между элементами модели как новый элемент. Например, сущность ФАКУЛЬТЕТ может быть рассмотрена как агрегация сущностей КАФЕДРА и ДЕКАНАТ.

При объединении представлений агрегации встречаются в двух формах.

Простое:

1. В одном представлении агрегатный объект определяется как целое, а в другом – в виде составных частей.

Например, в одном локальном представлении определены в качестве сущности объект - ТЕЛЕВИЗОР, а в другом – КИНЕСКОП, БЛОК ЯРКОСТИ, БЛОК РАЗВЕРТКИ, являющиеся составными частями объекта ТЕЛЕВИЗОР.

Причем, во втором представлении не указан явно тот факт, что вышеперечисленные блоки – составные части телевизора.

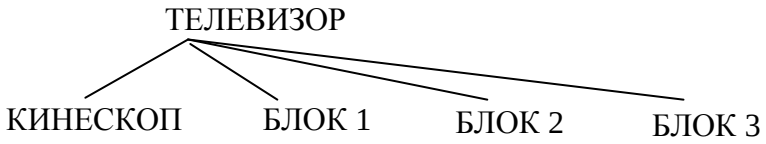
Простое объединение позволяет слить эти два локальных представления, не выражая явным образом то, что ТЕЛЕВИЗОР является агрегацией частей КИНЕСКОП, БЛОК ЯРКОСТИ, БЛОК РАЗВЕРТКИ.

Чтобы включить эту информацию в модель объединенного представления необходимо выполнить объединение с использованием агрегации.

2. Агрегатный объект в одном локальном представлении до конца, как единое целое не объединен.

Например, в одном представлении определены КИНЕСКОП и БЛОК ЯРКОСТИ; в другом – БЛОК РАЗВЕРТКИ, БЛОК ЦВЕТНОСТИ и другие, являющиеся составными частями объекта ТЕЛЕВИЗОР, который не назван ни в одном представлении.

Для повышения возможностей совместного использования данных можно ввести в рассмотрение агрегат.



2.3. Метод обобщение

Понятие **обобщения** близко к понятию агрегации, но в отличие от последней, которая может быть представлена в виде составных частей, образующих некоторое «целое», ***обобщение связано только с «целыми».***

Например, учащийся может быть как учащийся школы, ВУЗа, Техникума.

Обобщение бывает двух типов.

1. В одном локальном представлении определено некоторое множество объектов, которое может быть объединено общим для этих объектов **родовым понятием**, а само оно указано в другом локальном представлении.

Например,

I представление	II представление
цветные телевизоры	телевизоры
черно-белые телевизоры	

Здесь *родовое понятие* – ТЕЛЕВИЗОР.

2. Ни одно из объединенных локальных представлений не содержит родового понятия.

I представление	II представление
цветные	переносные
черно-белые	

В этом случае установить наличие родовой связи между специфичными типами объектов можно только в процессе сопоставления объектов из различных локальных областей (представлений).

Использование объединения обобщением позволяет повысить эффективность доступа пользователя к данным, хранящимся в базе.

3. ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ ФИЗИЧЕСКОЙ БАЗЫ ДАННЫХ

Физическая организация данных оказывает существенное влияние на такие эксплуатационные характеристики проектируемой базы данных (БД), как:

- объем занимаемой памяти;
- время отклика базы на запрос пользователя и другое.

Проектирование физической БД – процесс создания эффективной ее структуры на выбранной логической структуре.

Физическая БД - совокупность совместно хранимых взаимосвязанных данных, состоящих из одного или нескольких типов хранимых записей.

Понятие СТРУКТУРА физических БД включает:

- формат (формы) хранимой записи;
- структура путей доступа к данным;
- размещение записей на физических устройствах.

3.1. Формы (формат) хранения записей (данных). Организация файлов и способы адресации

Наиболее простой формой хранения данных в памяти ЭВМ является **линейный список**.

Линейный список – представляет собой конечное и упорядоченное множество объектов {x[1], x[2], …, x[n]}, структурные свойства которого связаны только с линейным относительным расположением элементов данных. Порядковый номер, указанный в квадратных скобках, указывает на относительное положение элементов в списке.

Линейные списки используются в тех случаях, когда встречаются упорядоченные множества данных переменного размера и где операции включения, поиска и удаления элемента данных должны выполняться в произвольных местах.

Одномерный линейный список, используемый для хранения данных в памяти ЭВМ, называют **вектором данных.**

Для линейного списка возможны два способа представления в ЭВМ:

- последовательное;
- связанное.

а) **Последовательное представление** – такое представление данных в памяти ЭВМ, при котором элементы списка размещаются в последовательных элементах памяти ЭВМ.

Например.

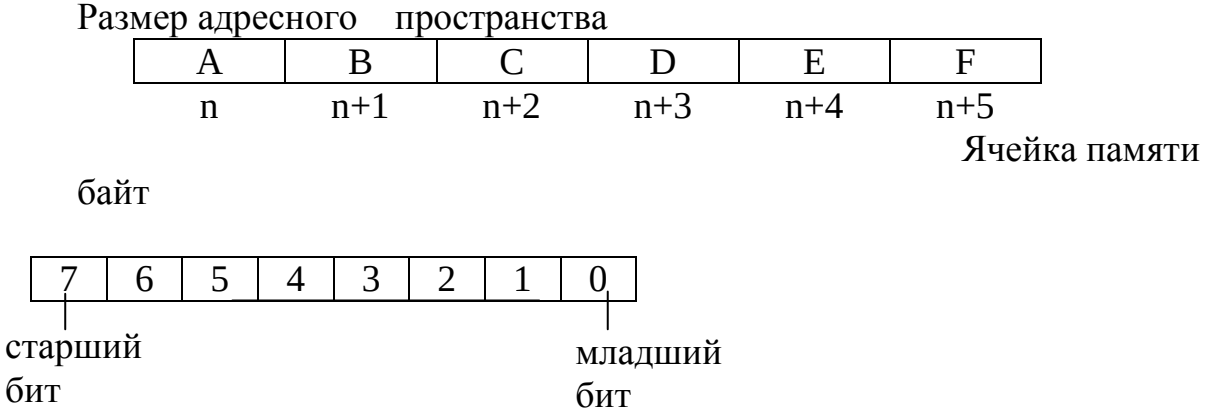


Рис. 3.1. Последовательное распределение памяти для представления линейного списка

В 16-ти разрядных машинах адрес любого бита памяти формируется в 16-битовом регистре и имеет значение от 0 до 2^{16} –1 целых чисел.

Недостатки:

При таком представлении списка возникают определенные сложности в реализации *вставки нового элемента* в середину. Например, чтобы включить в список между элементами D и E новый элемент K, необходимо изменить место элементов E и F.

Аналогично, удаление элемента из списка ведет к появлению пустой ячейки и для его уплотнения необходимо осуществлять смещение оставшихся

элементов. Одним словом, *при таком способе организации данных затруднено «обновление по месту», поэтому такой способ редко используется в СУБД. (Используется для ведения журналов, протоколов).*

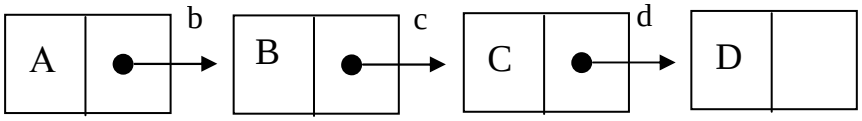
б) **Связное представление –** оно предусматривает задание для каждого элемента списка отношений следования и предшествования с помощью указателей, задающих связь между данными.

При таком представлении каждая ячейка содержит элемент данных и указатель (адрес) на последующий элемент списка.

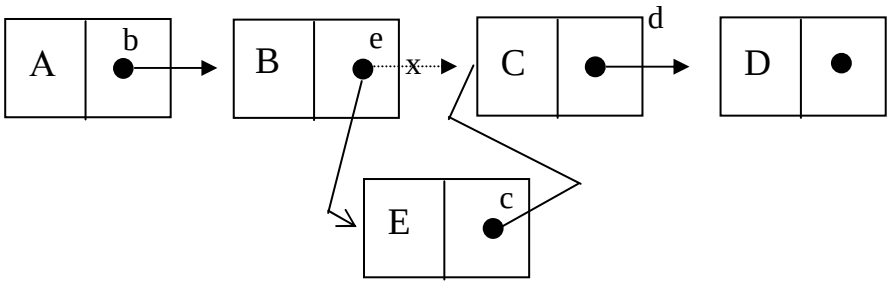
При связанном распределении не требуется, чтобы список хранился в последовательных элементах памяти.

Добавление или исключение данных в этом случае, может выполняться с помощью изменения значения указателя.

1) Пример связанного списка данных.



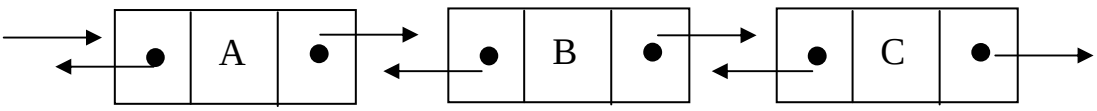
2) Пример включения (или выключения) элемента из связанного списка.



Таким образом, использование связанных списков более удобно в случае динамически изменяющихся линейных структур.

Списки могут быть и **двусвязанными** – когда при задании списка необходимо кроме прямого указателя для каждого элемента вводить в рассмотрение и обратный.

Например,



Структура линейного списка, представленная с помощью связанного распределения, называется также **цепью или цепной структурой**.

Физическая последовательность и связанная структура являются основными для большинства числа методов доступа к данным.

Приложения к конкретным примерам:

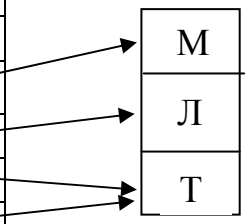
А) **последовательный список** – в приложении …. Поставщики

S#	SNAME	Область	ГОРОД
S1			
:			
S5			

единственным хранимым файлом, содержащим пять экземпляров хранимых записей. *Преимуществом данного варианта является простота*. Однако, если, предположим, что мы имеем 10000 поставщиков вместо 5 и все они размещены только в 10 городах, то в данном случае предпочтительно использовать **связанное представление**, т.е. УКАЗАТЕЛЬ на атрибут (в нашем случае он занимает меньше памяти, чем название города).

В этом случае имеем два хранимых файла: файл поставщиков и файл городов с указателями из первого файла во второй.

Файл	Поставщики	Файл	Город
S#	SИМЯ	SОбласть	УКАЗАТЕЛЬ
S1			
S2			М
S3			Л
S4			Т
S5			Т



Преимуществом данного способа является экономия памяти.

Примером **многосписочной организации (или цепной структуры)** может служить вариант представления данных, когда бы соединили вместе всех поставщиков одного и того же города. Другими словами мы имеем для каждого города список соответствующих поставщиков.

4. МЕТОДЫ ДОСТУПА К ДАННЫМ

Проектирование физической БД включает также решение вопроса методов доступа к данным.

Под **методом доступа** понимается совокупность технических и программных средств, обеспечивающих возможность хранения и выборки данных, расположенных на физических устройствах ЭВМ.

В методе доступа выделяют два компонента:

- структура памяти;
- механизм поиска.

Наиболее широко используемыми методами доступа являются:

1) последовательный;
2) прямой и произвольный;
3) индексно-последовательный;
4) индексно-прямой;
5) использование явных древовидных структур.

4.1. Последовательный доступ

Последовательный метод реализует доступ к данным базы данных путем последовательного просмотра записей. (Обычно каждая запись располагается в отдельном блоке). Причем записи могут быть упорядочены или неупорядочены по значениям первичного ключа. Примером может служить создание индексного файла в системе, например, FoxPro:

Создание индексного файла.

USE [имя файла.расширение]

INDEX ON [имя поля] TO USE MS1.DBF [имя файла]

Например, INDEX NAME TO NAMIDX.

Среднее количество физических блоков Ncp., к которым осуществляется доступ при поиске произвольной записи

$$Ncp=(1+N)/2.$$

Данное выражение справедливо как для упорядоченных, так и неупорядоченных записей, при условии, что искомая запись существует.

Если искомая запись отсутствует, то для неупорядоченного файла

Ncp=N, т.е. будут проверены все записи.

Таким образом, неупорядоченный файл является неэффективным, если приходится часто обращаться к поиску отсутствующей записи.

Кроме того, поскольку внесение изменений в произвольном порядке в последовательную структуру требует большого количества операций по перемещению записей, режим внесения изменений строго ограничен.

4.2. Прямой доступ

В случае, когда имеется возможность выделить в памяти для каждой записи место, определяемое уникальным значением ее первичного ключа, можно построить простую функцию преобразования ключа в адрес, обеспечивающую запоминание и выборку каждой записи в точности за один произвольный доступ к блоку.

Прямой доступ является эффективным с точки зрения временных затрат способом поиска данных в базе.

Методы прямого доступа подразделяют на две группы:

1) доступ с помощью ключа, эквивалентного адресу;
2) хэширование (метод произвольного доступа или рассеянной памяти).

Использование первой группы методов доступа возможно в тех случаях, когда в качестве атрибута в запись включается адрес памяти, в котором будет размещена запись. При работе с БД этот атрибут будет использоваться в качестве ключа.

Методы второй группы используются для обеспечения быстрой выборки и обновления записей по заданному значению первичного ключа.

Основная идея хэширования заключается в том, что каждый экземпляр записи размещается в памяти по адресу, вычисляемому с помощью хэш-функции (это специальная функция, вычисляемая от значения некоторого поля в экземпляре записи – обычно значения первичного ключа).

В этом случае экземпляры записей хранятся не в последовательных ячейках, выделенного блока памяти, а случайным образом рассеиваются по всему блоку.

Причем, каждая новая запись помещается в блок памяти таким образом, что ее присутствие или отсутствие может быть установлено без поиска по всему блоку.

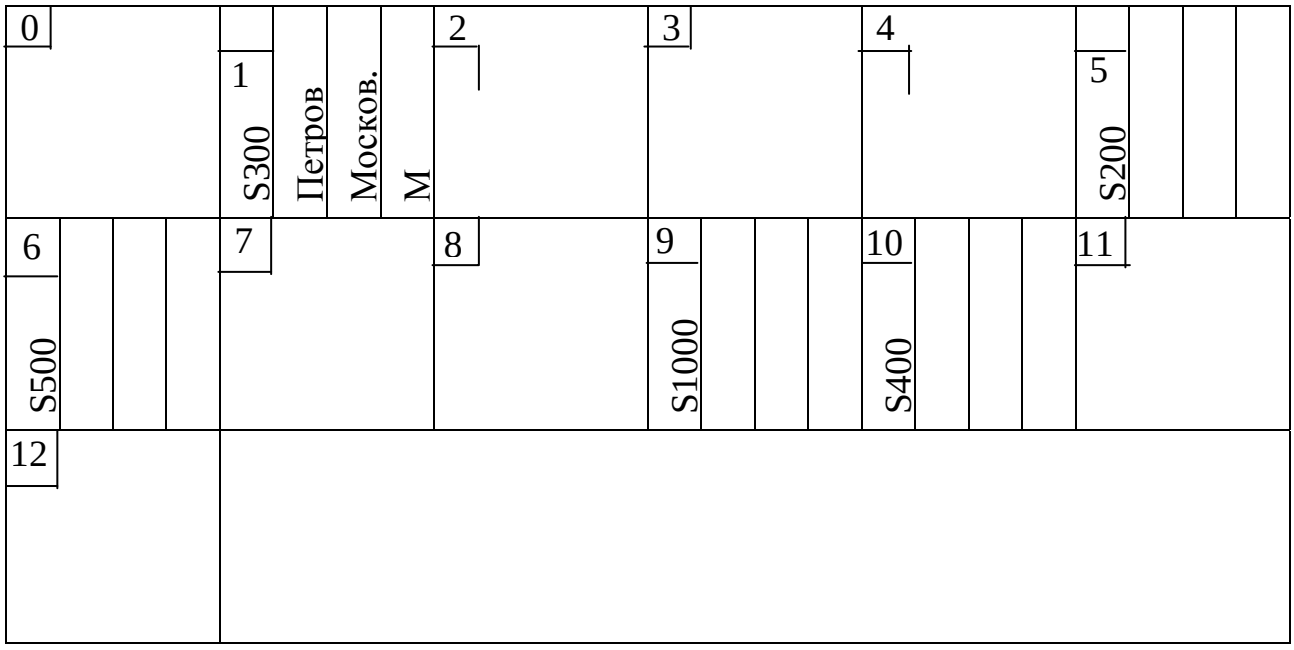
Другими словами, *в методе хэширования строится отображение множества ключевых значений экземпляров записей во множество физических адресов.*

В качестве хэш-функции выбирается **ПРАВИЛО**, преобразующее значение типа записи в адрес.

Простой пример общего класса хэш-функций: **адрес хранимой записи = остаток от деления на простое число «деление/остаток»**.

Рассмотрим пример хэш-адресации.

Предположим, что значениями S# является S100, S200, S300, S400, S500 (вместо S1,… S5). Адрес хранимых записей, например, равен остатку от деления S# на 13. Адресами хранимых записей в данном случае будут: 9,5,1,10,6, соответственно.



Для чего необходима хеш-функция? Теоретически возможно использовать (числовое) значение первичного ключа в качестве адреса хранимой записи.

Однако, этот вариант неприемлем из-за того, что диапазон значений первичного ключа обычно много шире диапазона доступных адресов хранимой записи. Например, предположим, что максимальный номер поставщика может быть 1000 (теоретически). На практике же их может, например, существовать 10.

Чтобы избежать колоссальных издержек памяти, в идеальном случае необходимо, чтобы хэш-функция переводила значение в диапазоне от 0-999 в диапазон от 0-9.

Чтобы учесть возможное расширение в будущем, последний диапазон обычно увеличен на 20% (в нашем примере 12).

Иногда, может возникнуть ситуация, когда, например, для различных записей хэш-функция определит одинаковый адрес, т.е. имеется возможность возникновения подобных коллизий. Для решения коллизий имеется много способов:

- метод последовательного сканирования;
- метод цепочки.

Метод последовательного сканирования состоит в том, что при возникновении коллизии просматриваются последовательно (сканируются) участки памяти, отведенные для хранения записей, до тех пор, пока не будет найдена свободная позиция, куда и помещается запись.

Метод цепочки

Вместо хранения самих элементов блока можно хранить в нем указатели на связанные списки экземпляров записей, имеющих одинаковый хэш-адрес.

После хэширования экземпляра записи, если участок памяти по вычисленному адресу занят, то происходит обращение по указанию к следующему участку памяти (элементу списка), на который и помещается запись.

При поиске записей действия выполняются в той же последовательности. В начале проверяется участок памяти по вычисленному адресу. Если там находится запись с другим значением ключа, то по указателю обращаются к следующей записи и так до тех пор, пока не будет найдена необходимая запись.

Память, выделяемую для организации списков, называют областью переполнения.

4.3. Индексно-последовательный метод доступа

Он строится на основе упорядоченного физически последовательного файла и иерархической структуры индексов блоков, каждый из которых упорядочен по значениям первичных ключей (подобно записям в файле данных).

Данный метод позволяет обеспечивать как последовательный, так и произвольный доступ к данным.

С этой целью в рассмотрение вводится новый параметр. **Индекс блока** – это упорядоченная таблица значений первичных ключей, в которой каждый элемент блока содержит наибольшее значение ключа среди всех записей в указанном блоке. С каждым значением ключа в индексе связан указатель соответствующего блока.

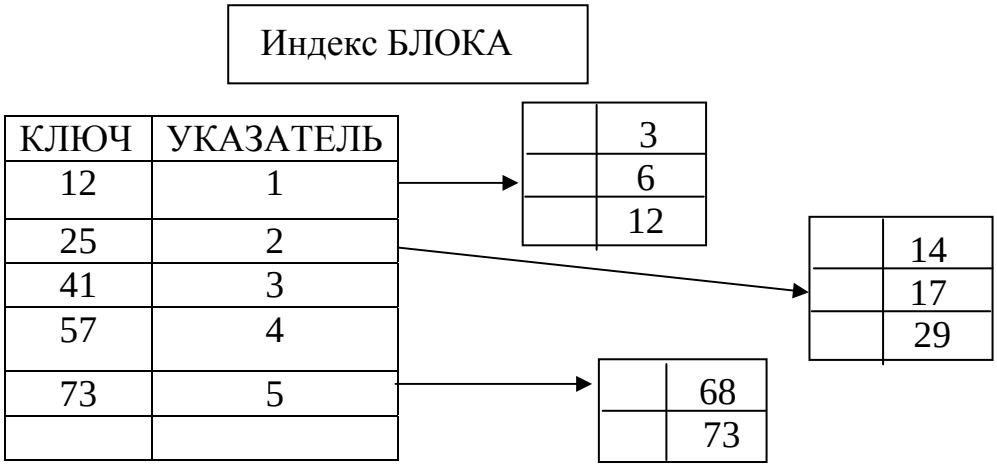


Рис. 4.1. Организация индекса блока

Смысловые организации

Записи организованы последовательно, но имеется дополнительный индекс (указатель), позволяющий обращаться к записям в произвольном порядке.

Механизм поиска

Поиск экземпляра записи осуществляется посредством первоначального установления номера блока, в котором может находится запись, а затем последовательного поиска в этом блоке, пока не будет установлено местонахождение искомой записи или ее отсутствие.

Данный метод позволяет обеспечивать быстрый доступ к записям баз данных и файлов большой размерности, в которых поиск по одному только индексу (ключу) приводит к значительным затратам времени.

Недостатки:

• Хотя индексно-последовательный доступ (или последовательная организация данных) наглядно и легко воспринимается (а кроме того требует меньший индекс). Этот способ более сложен для эксплуатации.

• При включении новых записей возникает необходимость пересортировки файла (если мы рассматриваем файл), либо внесения новых записей в отдельную область (область переполнения) или организации указателей на эти записи. (Пример фамилии в алфавитном порядке).

относительный адрес указатель на область переполнения

Эти записи помещаются в область **переполнения**.

4.4. Индексно-произвольный метод доступа

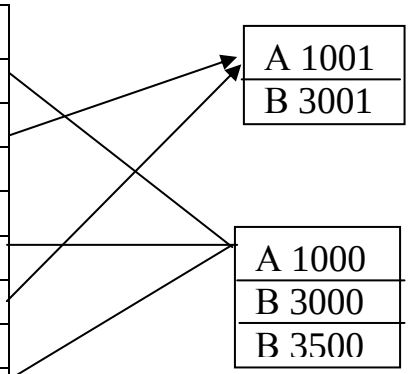
Данный метод обеспечивает доступ к экземплярам записей на основе использования индекса. Поэтому метод называется также «**метод доступа с полным индексом**».

Полный индекс представляет собой такую организацию файла, при которой для каждого конкретного экземпляра записи предусмотрен соответствующий индекс. Этот индекс составляется из значения первичного ключа и указателя экземпляра записи, содержащий это значение.

Обычно, для ускорения поиска индексы упорядочиваются и физически близкое размещение хранимых записей не требуется.

Полный индекс.

Значение ключа	Значение указателя
А 1000	2
А 1001	1
…..	…..
А 1999	3
В 3000	2
В 3001	1
В ….	….
В 3500	2



После того, как в индексе обнаружено искомое значение ключа, доступ к записи можно осуществить с помощью указателя, который хранится в индексе рядом со значением ключа (в этом случае, если искомое значение ключа в индексе не найдено, поиск завершается на уровне индекса).

Допускается произвольный доступ к индексу посредством хэширования. Метод доступа с полным индексом обеспечивает эффективную выборку единичных записей, а также простоту операций обновления(т.е. новые записи просто включаются в конец файла и при этом не требуется вводить указатели на область переполнения).

4.5. Метод доступа, основанный на использовании древовидных структур

Имеют место методы представления в памяти ЭВМ данных в виде древовидных сетевых структур и соответствующие методы доступа к ним.

Эти методы стали конкурентами классических (перечисленных выше) методов. К основным древовидным структурам относятся:

- бинарное дерево;
- B- дерево.

Бинарным (двоичным) деревом называется древовидный граф с двоичным ветвлением, когда из каждой вершины (кроме концевых) выходят две дуги.

Бинарные деревья (двоичные) представляют собой широко используемый вид структур баз данных, *обеспечивающий как произвольный, так и последовательный выбор данных.*

Важную роль проектирования древовидных структур данных играет понятие сбалансированного дерева.

Сбалансированным, называется дерево, если разница уровней любых двух конечных вершин не превышает единицы.

Уровень вершины (i) – определяется длиной пути от корневой вершины (T) до вершины (i).

Корневая вершина (T) имеет нулевой уровень.

Построение сбалансированного дерева делает равномерным обращение к любой из его вершин, что позволяет минимизировать среднюю длину доступа.

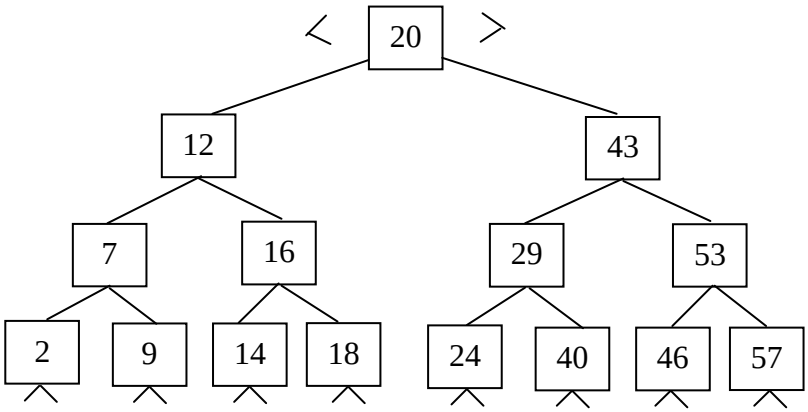
Механизм поиска по бинарному дереву основан на том, что каждая вершина дерева помечена отдельным ключом. Ключи упорядочены следующим образом: $k_{i1} \le k_i \le k_{i2}$, где k_i – ключ i-ой вершины; k_{i1} – ключ i_1-ой вершины, соответствующей левой дуге i-ой вершины; k_{i2} – ключ вершины, лежащей на правой дуге.

При поиске некоторого ключа k вначале просматривается корневая вершина дерева T и сравнивается ключ k с ключом $k_т$ корневой вершины:

а) в случае $k=k_т$ – поиск завершен успешно;

б) если $k<k_т$ – поиск продолжается в левом поддереве;

в) если $k>k_т$ – поиск продолжается в правом поддереве.



B – деревья.

B-дерево представляет собой обобщение понятия бинарного дерева.

В нем из каждой вершины могут выходить более двух ветвей. Обычно B-деревья используются только для организации индекса. Записи данных располагаются в отдельной области, для которой возможен произвольный доступ.

Каждая вершина B-дерева состоит из совокупности значений первичного ключа, указателей индексов и ассоциированных данных.

Указатели индексов используются для перехода на следующий, более низкий уровень в B-дереве.

Ассоциированные данные фактически представляют собой совокупность указателей данных и служат для определения физического местоположения данных, ключевые значения которых хранятся в этой вершине индекса.

Сжатие данных.

Для сокращения объема памяти, занимаемой данными, используются разнообразные методы сжатия.

Во многих случаях сжатие данных дает огромную экономию (размер может быть уменьшен в 2 раза), а иногда и до 80%.

Методы сжатия можно разбить на 2 группы (класса):

1) методы ориентированные на конкретную структуру или содержание записей и разрабатываемые в расчете на конкретное приложение;

2) методы, которые используются во многих приложениях и поэтому могут быть встроены в общее программное обеспечение, аппаратуру или микропрограммы.

Существуют следующие способы:

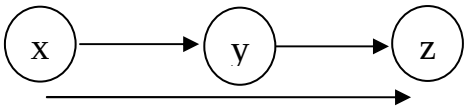
1) исключение избыточных элементов;
2) переход от естественных обозначений к более компактным;
3) подавление повторяющихся символов;
4) ликвидация пустых мест в файле;
5) кодирование часто используемых элементов данных;
6) сжатие упорядоченных данных.

Исключение избыточности элементов.

Является важным методом сокращения расходов памяти при организации базы данных.

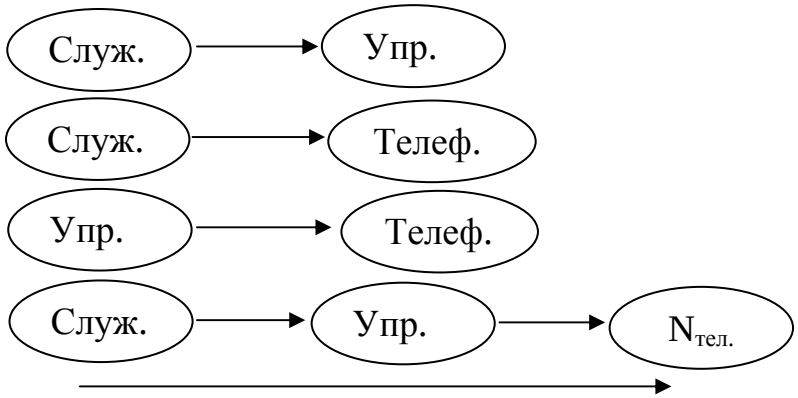
Этот способ состоит в исключении одних и тех же элементов данных в составе нескольких файлов и устранение избыточных связующих элементов.

Например,



Если использовать для изображения связей одиночные стрелки, можно установить те связи, которые избыточны.

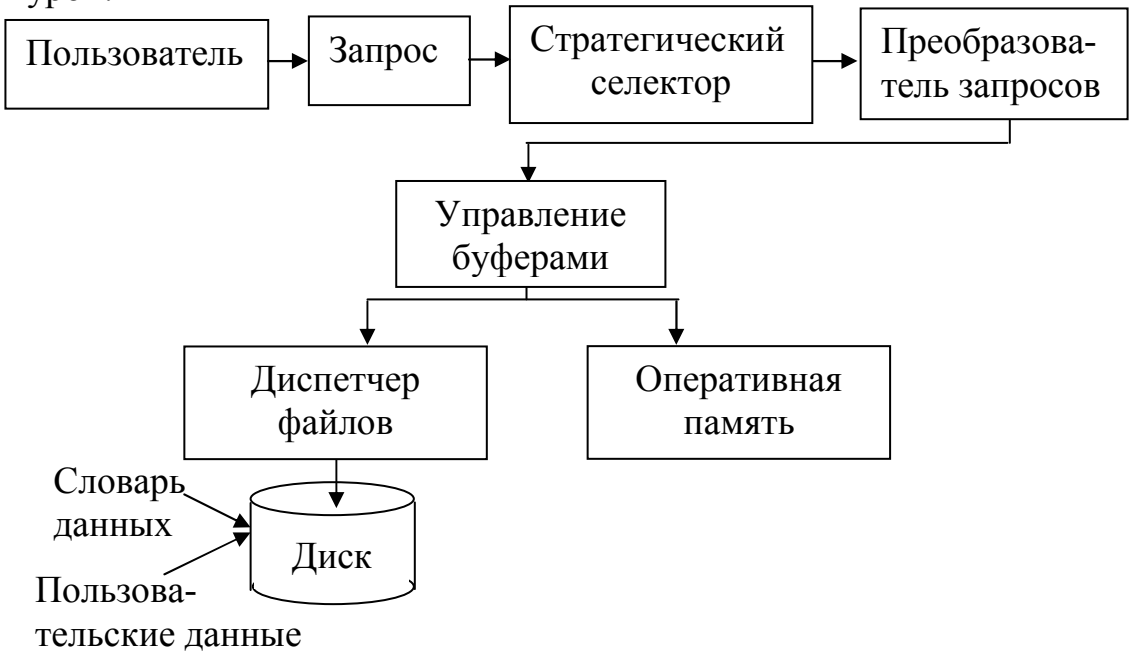
Но прежде, чем удалять связующий элемент необходимо оценить смысл ассоциаций элементов.



Ассоциация элементов не дает однозначного определения для исключения избыточных связующих данных.

4.6. Схема физического доступа к базе данных

Схему физического доступа к данных можно представить следующей структурой:



Итак, пользователь взаимодействует с системой баз данных, запуская *инструкции*.

Стратегический селектор – программное обеспечение, преобразование запроса в эффективную для исполнения форму.

Преобразованное требование активизирует **буферный диспетчер** – программное обеспечение, контролирующее перемещение данных между ОЗУ и диском.

Диспетчер файлов - поддерживает буферный диспетчер, управляя размещением данных на диске и связанными с ним структурами данных.

Кроме пользовательских данных диск содержит **словарь данных** – определяет структуру пользовательских данных и возможности их использования.

Пользовательские данные хранятся в виде физической базы данных или совокупности данных.

5. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

Основные вопросы, включаемые в рассмотрение распределенных баз данных:

1) различные формы прозрачности;
2) выполнение запросов;
3) параллельные обновления информации.

На уровне пользователя **распределенная база данных** представляет собой логическую совокупность локальных и удаленных (т.е. распределенных данных).

Строгое физическое определение следующее: **распределенная база данных** – это набор файлов (отношений), хранящихся в разных узлах информационной сети и логически связанных таким образом, чтобы составлять единую совокупность данных (связь может быть функциональной или через копии одного и того же файла). (ИПС – информационно-поисковые системы базируются на распределенных СУБД и др.)

Итак, СУБД и централизованная обработка информации, позволяют устранять такие недостатки традиционных файловых структур:

- несвязанность;
- несогласованность;
- избыточность данных.

По мере роста баз данных и их использовании в территориально разделенных организациях появляются другие проблемы. Так, для *централизованной СУБД*, находящейся в узле телекоммуникационной сети, с помощью которой различные подразделения организации получают доступ к данным, с ростом объема информации и количество транзакций (запросов) появляются следующие проблемы:

- большой поток обменов данными;

- низкая надежность;
- низкая общая производительность;
- большие затраты на разработку.

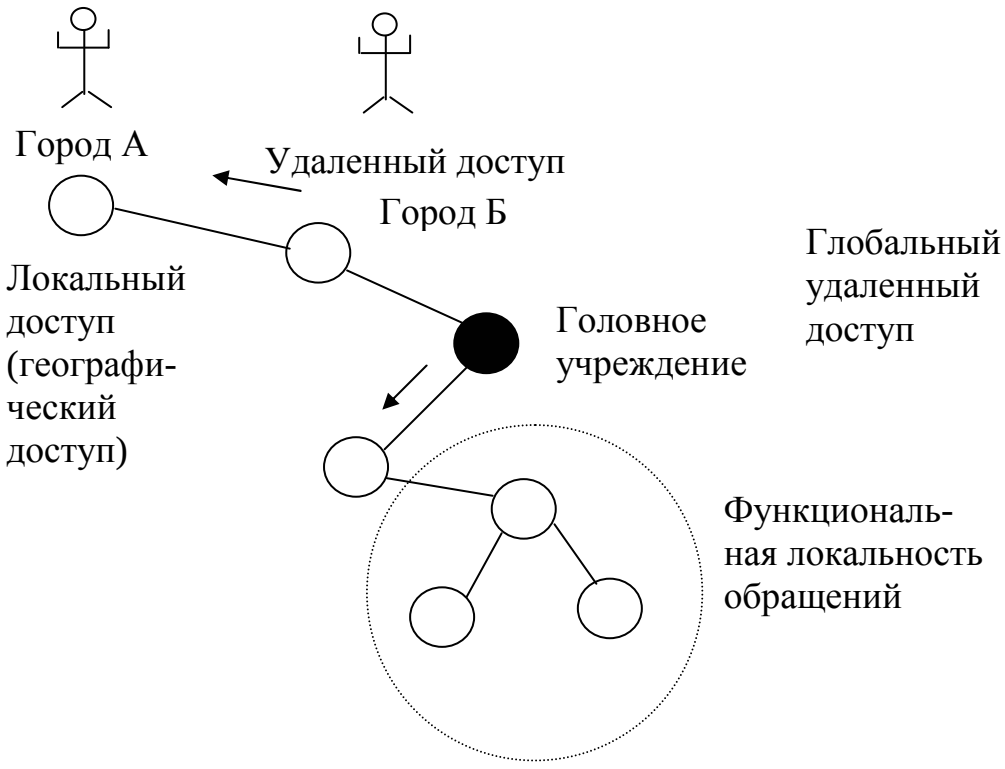
Достоинства централизованных БД: они обеспечивают безопасность, целостность и непротиворечивость информации (при обновлении особенно).

Для решения перечисленных проблем необходимо провести **децентрализацию данных**.

При этом достигается:

- более высокая степень одновременности обработки вследствие распределения нагрузки;
- улучшенное использование данных на местах при выполнении удаленных (дистанционных) запросов;
- меньшие затраты;
- простота управления.

Чем обусловлена децентрализация? Затраты на создание сети, в узлах которой находятся малые ЭВМ, гораздо ниже, чем затраты на создание аналогичной системы с использованием большой ЭВМ.



Модели доступа в распределенных базах данных

Допустим, существует система обработки банковских операций на основе сети.

Клиент А – работает с системой с помощью локальных запросов (географический принцип).

Если он в городе Б – например, имеет счет в банке этого города и хочет получить вклад, для проверки его счета в банке будет выдан **удаленный запрос**.

Глобальные удаленные запросы – запросы головного учреждения.

Функциональная локальность обращений – обмены информацией на основе локальных запросов (например, кредитные операции на уровне одного района).

Итак, запросы бывают: локальные, удаленные и глобальные удаленные.

5.1. Проблемы распределенных баз данных

Основная задача распределенной БД – распределение данных по сети. Существуют следующие способы решения этой задачи:

а) **Разделенное распределение**, т.е. такое при котором в каждом узле хранится и используется собственный набор данных, доступный для удаленных запросов.

б) **Частично-дублированное распределение**, т.е. некоторые данные, часто используемые на удаленных узлах, могут дублироваться.

в) **Полностью дублированное распределение** – когда все данные дублируются в каждом узле.

г) **Расщепленное (фрагментированное)** распределение, когда некоторые файлы могут быть расщеплены горизонтально (выделено подмножество записей) или вертикально (выделено подмножество полей (атрибутов)), при этом, выделенные подмножества хранятся в различных файлах вместе с нерасщепленными данными.

д) Выделенные подмножества могут дублироваться, как в п)п) п.п. б) и в).

Проблемы и принципы создания распределенных баз данных. Какова общая модель данных распределенной системы?

1. Распределенная база данных должна иметь единую концептуальную схему всей сети. Это обеспечивает **логическую прозрачность данных** для пользователя, в результате чего он сможет формировать запрос ко всей базе, находясь за отдельным терминалом. То есть, пользователь работает как бы с централизованной Базой Данных.

2. Необходима схема, определяющая местонахождения данных в сети, что обеспечивает **прозрачность размещения данных**, благодаря которой пользователь может не указывать, куда переслать запрос, чтобы получить требуемые данные.

3. Распределенные базы данных могут быть **однородными** или **неоднородными** в смысле аппаратных и программных средств (СУБД).

Проблему неоднородности легко решить, если распределенная база данных является неоднородной в смысле аппаратных средств, но однородной в смысле программных средств (одинаковые СУБД в узлах).

Если же в узлах распределенной базы данных используются разные СУБД, необходимы средства преобразования структур данных и языков, то есть, необходима **прозрачность преобразования в узлах распределенной базы данных.**

4. Управление словарями. Кроме того, для обеспечения всех видов прозрачности в распределенной базе данных нужны программы, управляющие многочисленными справочниками или словарями.

5. Методы выполнения запросов в распределенной базе данных должны быть скоординированы: так как отдельные части запроса нужно выполнять на месте расположения соответствующих данных и передавать частичные результаты на другие узлы.

6. В распределенной базе данных нужен сложный механизм управления одновременной обработкой, который должен обеспечивать синхронизацию при обновлениях информации, что гарантирует **непротиворечивость** данных.

7. Развитая методология распределения и размещения данных, включая расщепление, является одним из основных требований к распределенной базе данных.

5.2. Выполнение запросов в распределенной базе данных

При выполнении запроса в распределенной базе данных выполняется две задачи:

- декомпозиция запроса;
- оптимизация запроса.

Декомпозиция запроса. Так как данные распределены, а также могут быть расщеплены, запросы, составленные пользователем, который рассматривает данные целиком (как если бы база данных была централизованной), должны быть приведены к виду, учитывающему распределение и расщепление данных.

Декомпозиция обусловлена тем, что программа запроса не может быть выполнена полностью в одном узле, если там не хранится вся требуемая информация.

Оптимизация вызвана тем, что для завершения выполнения программы необходимо перемещать целые отношения (файлы) или результаты промежуточных вычислений.

Пример: рассмотрим базу данных распределенную по трем узлам, схема которой состоит из трех отношений:

(С) СЛУЖАЩИЕ (СЛ#, ОТД#, ИМЯ#, ЗАРПЛАТА, РУКОВОДИТЕЛЬ)

(О) ОТД (ПОДРАЗДЕЛЕНИЕ#, ОТД#, ПРОЕКТ#)

(Р) РАЗМЕЩЕНИЕ (ПОДРАЗДЕЛЕНИЕ#, ГОРОД).

Рассмотрим запрос: как зовут служащих, работающих в отделах подразделения, расположенного в Нью-Йорке.

1) Пусть все отношения распределены по разным узлам.

2) Прием решения задачи «в лоб»: переместим все отношения в узел запроса и обработаем запрос, как для централизованной базы данных (это дорогой способ).

Представим этот запрос в терминах реляционного исчисления:

Из области Область узла запроса

RANGE OF (C,O,P) IS (СЛУЖАЩИЕ, ОТД, РАЗМЕЩЕНИЕ)

отыскать

RETRIEVE W (С.ИМЯ) WHERE(Р.ГОРОД=’НЬЮ-ЙОРК’)
AND(Р.ПОДРАЗД#=О.ПОДРАЗДЕЛЕНИЕ#)
AND(О.ОТД#=С.ОТД#).

1) в этом запросе 3 картежные переменные (СЛУЖАЩИЕ, ОТДЕЛ, РАЗМЕЩЕНИЕ).

Для упрощения запроса отделим в нем часть, содержащую одну переменную: (операция селекция):

Q – исходный запрос → Q_1, Q_2, Q_3

→ Q_1 RANGE OF P IS РАЗМЕЩЕНИЕ
RETRIEVE ВРЕМЕН. (Р.ПОДРАЗДЕЛЕН#)
Создаем временный кортеж
WHERE Р.ГОРОД=’НЬЮ-ЙОРК’

→ Q_2 RANGE OF (C,O,P) IS (СЛ., ВРЕМ.)
RETRIEVE W (С.ИМЯ) WHERE
(Р.ПОДРАЗДЕЛЕНИЕ#=О.ПОДРАЗДЕЛЕНИЕ#)

→ Q_3 AND (О.ОТД#=С.ОТД#)

Аналогично можно отделить еще один запрос с одной переменной от запроса Q_2 и повторять этот запрос, пока не останется запрос с одной переменной.

Такое отделение возможно, так как значения соединяемых атрибутов из кортежей, выбранных во время начальной редукции, становятся предикатами соединения. Этот процесс называется **подстановкой кортежей**.

Процесс рекурсивно повторяется для всех кортежей исходных отношений, а результат получается путем объединения отдельных редукций (соединений), полученных в процессе подстановки кортежей.

Для распределенной базы данных это означает: над каждым отношением (селекция либо проекция) редукция может быть произведена на месте его хранения, а выбранные атрибуты (или кортежи, если требуется соединение) пересылаются в узел, где производится очередное соединение.

При подходе «снизу вверх» нужно передавать атрибут СЛУЖАЩИЕ. ИМЯ при выполнении всех соединений (т.е. методика обработки запросов).

Оптимизация запроса

Оптимизация запроса основана:

- на перестановке операций в пределах запроса;
- на идентификации общих подвыражений и однократном их выполнении;
- на трансляции и оптимизации запроса относительно фрагментов.

Существуют различные способы выполнения запросов: «сверху вниз» ($Q_1 \rightarrow Q_2 \rightarrow Q_3$), «снизу вверх» ($Q_3 \rightarrow Q_2 \rightarrow Q_1$). Способ “сверху вниз” более экономичен как с точки зрения времени выполнения, так и объема перемещаемых данных. Это объясняется тем, что мы начинаем с редукции (т.е. селекции), а затем может использоваться полусоединение, для которого нужно передавать только значения соединяемых атрибутов.

Методика оптимизации запроса

Оптимизация запроса начинается с построения дерева разбора для выражения запроса.

В этом дереве:

листья – соответствуют отношениям

внутренние узлы (включая корень) – операциям реляционной алгебры.

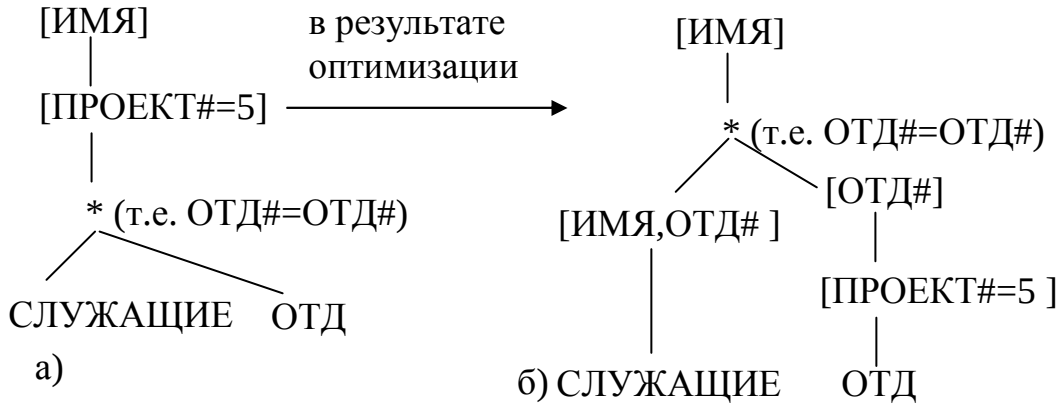
Например,

«Как зовут служащих, работающих над проектом 5#».

Соответствующее выражение реляционной алгебры

((служащие * отдел)[ПРОЕКТ#=5])[ИМЯ].

Итак, используем отношения: СЛУЖАЩИЕ, ОТДЕЛ в [ ] – имена рассмотренных полей. Порядок операций – слева направо;



а) указанный порядок соответствует обходу дерева “снизу вверх”, т.е. не является наилучшим с точки зрения затрат на обработку на местах, т.е. выполнения операций в узлах и передачи данных.

Рассматриваемый запрос можно оптимизировать с учетом двух *отношений эквивалентности*:

- каскадность унарных операций;
- дистрибутивность унарных операций.

Унарные операции – это операции селекции и проекции, уменьшающие размеры отношения по горизонтали и вертикали соответственно).

Каскадность унарных операций показывает, что заданная унарная операция может быть разбита на части, выполняемые последовательно.

Например, $R[A_1\theta c_1 \text{ AND } A_2\theta c_2] \equiv (R[A_1\theta c_1])\,[A_2\theta c_2]$, где S,R – отношения; A – атрибуты, c – операнды, θ – оператор сравнения.

Дистрибутивность унарных операций относительно бинарных позволяет распространить их на операнды бинарных операций.

Например, $(R*S)[A\theta c]=(R[A\theta c]*S[A\theta c])$.

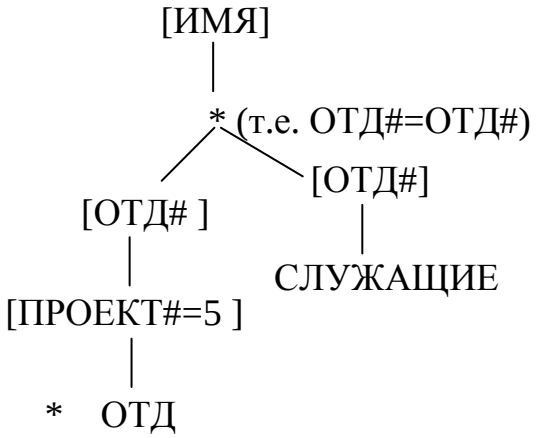
То есть ОПТИМИЗАЦИЯ заключается возможно более ранним применением операций редукции, что приводит к оптимальному выполнению запроса.

Это соответствует перемещению унарных операций по направлению к листьям дерева.

Эквивалентности позволяют:

1) распределить селекцию относительно естественного соединения, последовательно выполнить проекции на нужные компоненты, а затем распределить их также как для операции селекции;

2) проекция теперь начинается снизу:



Одновременная обработка и обновление

1) Одновременная работа в СУБД.

а) в распределенных базах данных может одновременно выполняться несколько транзакций (запросов); при этом могут использоваться одни и те же данные, возможно продублированные в разных узлах;

б) при чтении для получения нужной информации достаточно одной копии, если система обеспечивает непротиворечивость всех копий;

в) при обновлениях, для сохранения непротиворечивости копий нужно их своевременно модифицировать.

Для этого обновления нужно производить в соответствии с **последовательными протоколами.** Что это такое? Введем основные понятия, характеризующие одновременную работу СУБД.

Логическая транзакция – относится к прикладной программе и представляет совокупность системных транзакций.

Расписание – это порядок выполнения транзакций во времени.

Примеры расписаний

1)	2)	3)	4)
T1:R-A	T1:R-A	T1:W-A	T1:W-A
T1:W-A	T2:R-A	T2:W-A	T2:R-A
T2:R-A	T1:W-A	T1:ОТМЕНИТЬ	T1:ПРЕРВАТЬ
T2:W-A	T2:W-A		

↓

модификация

Где R-A – считывание поля; W-A – запись (модификация входных переменных); T1, T2 … - транзакции.

1) Рассогласования не возникает, т.к. транзакции T1 и T2 выполняются последовательно (одна за другой).

2) Модификация транзакции T1 затирается транзакцией T2, следовательно модификация транзакции T1 – теряется.

3) Происходит потеря модификации транзакции T2, вызванная тем, что после записи T2 следует отмена транзакции T1.

4) Называется **неправильное считывание**, так как выбранное транзакцией T2 значение впоследствии удаляется из базы данных.

Для ограничения порядка выполнения транзакций необходимы протоколы, сложность которых зависит от того, на каком уровне требуется поддерживать целостность при работе с базой данных.

Согласованные состояния базы данных обеспечивают:

- последовательность;
- приводимые к последовательным расписания.

Расписание называется **последовательным**, если все действия транзакции выполняются друг за другом.

Расписание называется **приводимым к последовательному**, если результат применения этого расписания к базе данных эквивалентен результату последовательного расписания.

Если учитывать одновременное выполнение транзакций над многочисленными копиями, последовательность обновлений требует развитых средств синхронизации.

Существуют различные решения (варианты) при управлении обновлениями в распределенной базе данных.

Например,

1) использование схемы **с основным узлом** (или основной копией):

а) для каждого отношения выбирается основной узел, в котором производится обновление информации;

б) после успешного завершения обновления в этом узле начинается обновление вторичных копий для полного завершения данной операции обновления.

В этом случае: основная проблема при выполнении транзакции – обеспечить соответствие данных, уже обновленных и еще не обновленных. Кроме того, кроме первичного (центрального) узла необходимо выбрать дополнительные узлы на случай сбоев (свойство надежности).

2) При запросах, требующих только считывания из базы данных, удобным средством обеспечения одновременного обслуживания являются «**моментальные съемки**» основных отношений. Их можно продублировать там, где это необходимо. Такие съемки делаются (т.е. их данные засылаются в разные узлы) тогда, когда удобнее всего производить обновление базы данных (т.е. в менее напряженные часы).

До следующего обновления эти снимки обслуживают пользователей (хотя данные в них старые), и они не подвержены влиянию обновления основных отношений.

6. ПРИНЦИПЫ РАБОТЫ СИСТЕМЫ КЛИЕНТ/СЕРВЕР. МОДЕЛИ ТЕХНОЛОГИИ КЛИЕНТ/СЕРВЕР

В результате изучения материала данной главы Вы:

а) сможете определить схему базы данных на языке СУБД сервера;

б) сможете пользоваться языком манипуляции данными СУБД сервера для написания программ, используя возможности реляционного подхода;

в) познакомитесь со средствами создания приложений клиентских систем.

Платформа клиент/сервер – это компьютерная сеть; один или несколько компьютеров сети обслуживают остальные и потому называются **серверами.** (Serve – прислуживать, служить.)

Остальные компьютерные сети – **клиенты** этих серверов.

Каждый тип машины приспособлен для выполнения своих специфических функций в сетевой системе:

Сервер – приспособлен к обслуживанию большого количества клиентов. **Клиентские системы** – приспособлены для ввода и представления данных: они включают в себя графические пользовательские интерфейсы (ГПИ).

Графический пользовательский интерфейс (ГПИ) - окна и управляющие структуры, представляющие конечному пользователю графические средства доступа к компьютерной системе.

В модели клиент/сервер компьютеры в сети не являются равноправными

- владеют и распоряжаются информационно-вычислительными ресурсами (процессор, файловая система, почтовая служба, базы данных);
- другие имеют возможность обращаться к этим службам.

Компьютер, управляющий теми или иным ресурсом, называют сервером этого ресурса (конкретный сервер определяется видом ресурса, которым он владеет). В сети один компьютер может выполнять роль как клиента, так и сервера.

Этот же принцип распространяется и на взаимодействие программ:

- если одна из них выполняет некоторые функции, предоставляя другим соответствующий набор услуг – программа выступает в качестве сервера.

Например,

- ядро реляционной SQL – ориентированной СУБД называется сервером баз данных;
- программа, обращающаяся к серверу за услугами по обработке данных - SQL – клиент.

Принципы технологии клиент/сервер заключается в разделении функций стандартного интерактивного приложения (4 группы).

1. Объединяет функции ввода и отображения данных.
2. Объединяет прикладные (конкретная предметная область) функции (открытие счета, перевод денег и т.д.).
3. Обеспечивает распределение логических компонент между компьютерами в сети.
4. Реализация механизмов связи компонентов между собой

По способу реализации этих функций существуют модели технологии клиент/сервер:

1 – модель файлового сервера (File Server –FS);

2 – модель доступа к удаленным данным (Remote Data Acces – RDA);

3 – модель сервера баз данных (DataBase Server – DBS);

4 – модель сервера приложений (Application Server – AS);

6.1. Модель файл-сервер (FS)

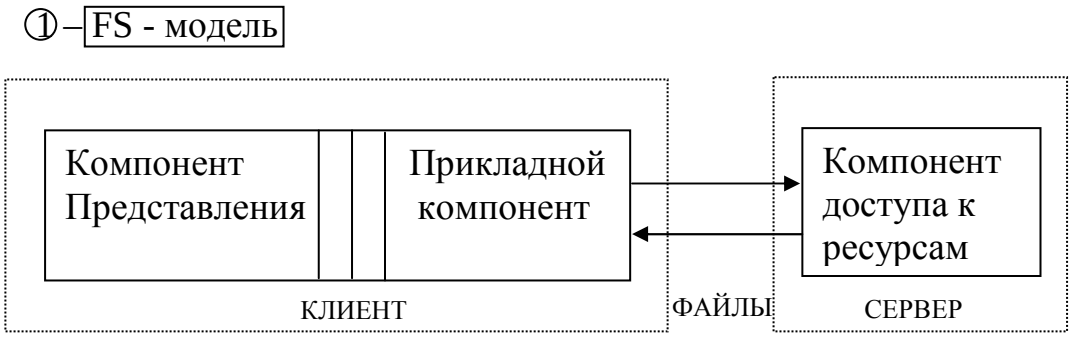


Рис. 6.1

*FS – модель является базовой для локальных сетей ПК.

а) Один из компонентов (наиболее мощный) в сети считается файловым сервером и представляет услуги по обработке файлов другим компьютерам.

б) Файловый сервер работает под управлением сетевой операционной системы (например, Novell NetWare, Windows и др.).

в) На других компьютерах в сети функционирует приложение, в кодах которого совмещены компонент представления и прикладной компонент (рис. 6.1).

г) протокол обмена представляет собой набор низкоуровневых вызовов, обеспечивающих приложению доступ к файловой системе на файл-сервере.

Применительно к базам данных. FS – модель позволила расширить возможности персональных СУБД.

В таком случае на нескольких ПК выполняется как прикладная программа, так и копия СУБД, а базы данных хранятся в разделяемых файлах, которые находятся на файловом сервере. Когда прикладная программа обращается к базе данных, СУБД направляем запрос на файловый сервер. В этом запросе указаны файлы, где находятся запрашиваемые данные.

В ответ на запрос файловый сервер представляет требуемый блок данных, над которыми СУБД выполняет действия.

Технологические недостатки FS-модели:

1) высокий сетевой трафик (передача множества файлов);

2) узкий спектр операций манипуляции с данными (данные – это файлы);

3) отсутствие эффективных средств безопасности доступа к данным (защита только на уровне файловой системы).

6.2. Модель RDA (Remote Data Access)

Основное отличие заключается в характере компоненты доступа к информационным, это как правило, SQL – сервер. В RDA – модели коды компонента представления и прикладной компонент совмещены и выполняются на клиенте (он поддерживает как функции ввода и отображения данных, так и чисто прикладные функции).

Доступ к информационным ресурсам обеспечивается либо операторами специального языка (например, SQL), либо **вызовами функций специальной библиотеки** (если имеется соответствующий интерфейс прикладного программирования – API).

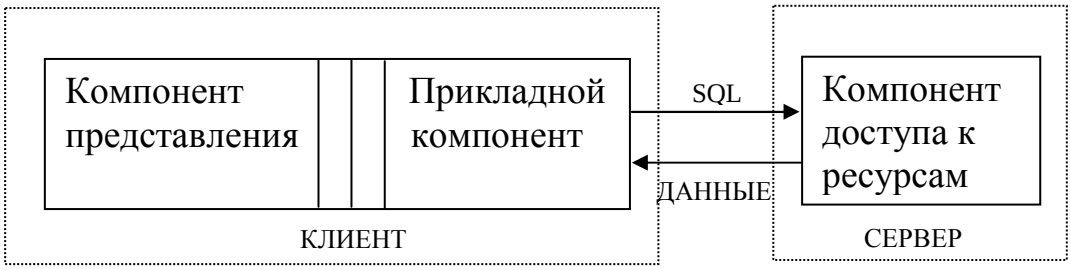


Рис. 6.2

Клиент направляет запросы к информационным ресурсам (например, к базе данных) по сети удаленному компьютеру. На нем функционирует ядро СУБД, которое обрабатывает запросы, выполняя предписанные в них действия и возвращает клиенту результат, оформленный как блок данных (рис. 6.2).

При этом, инициатором манипуляций с данными выступают программы компьютеров-клиентов;

Ядро СУБД – обслуживает запросы и обрабатывают данные.

Преимущества:

1) перенос компонент представления и прикладной на клиент – сводит к минимуму число процессов ОС (разгружает сервер). Сервер, т.е. процессор загружен операциями обработки данных, запросов (увеличивается быстродействие);

2) увеличивается загрузка сети (передаются на запросы на ввод-вывод, а запросы на языке SQL, объем который существенно меньше);

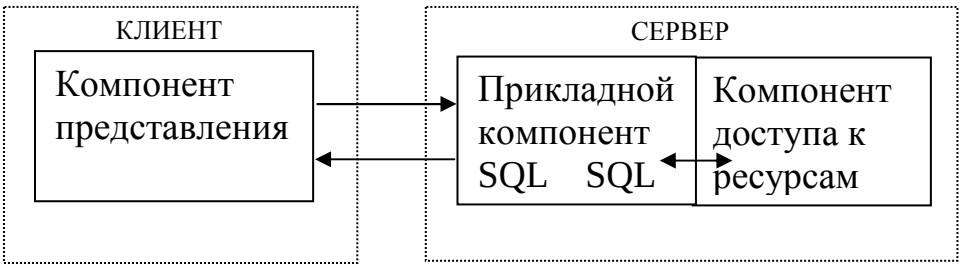
3) унификация интерфейса клиент-сервер в виде языка SOL, который используется не только в качестве средства доступа к данным, но и стандарта общения клиента и сервера.

Недостатки RDA-модели

1. Взаимодействие клиент-сервер посредством языка SQL-запросов при большом числе клиентов имеет большое время отклика (сеть загружается).

2. Затруднено администрирование приложений из-за совмещения в одной программе различных по своей природе функций (представления и прикладных).

6.3. DBS-модель (Data Base Server)/



Ее основу составляет механизм **хранимых процедур** – средство программирования SQL-сервера.

- Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, где функционирует SQL-сервер.

- Язык, на котором разрабатываются хранимые процедуры, представляет собой процедурное расширение языка SQL и уникален для каждой конкретной СУБД.

Преимущества DBS

1. Возможность централизованного администрирования прикладных функций, снижение трафика (вместо SQL-запросов по сети направляются вызовы хранимых процедур).

2. Разделение процедур между несколькими приложениями и экономия ресурсов компьютера за счет использования единожды созданного плана выполнения процедур.

Недостатки DBS

1. Ограниченность средств для написания хранимых процедур (особенно в сравнении с C++, Pascal и др.).

2. Отсутствие в большинстве СУБД возможностей отладки и тестирования разработанных хранимых процедур.

На практике часто используются смешанные модели, когда поддержка целостности базы данных и некоторые простейшие прикладные программы

поддерживаются хранимыми процедурами (DBS), а более сложные функции выполняются на клиенте непосредственно в прикладной программе (RDA модель).

RDA и DBS – имеют в своей основе **двухзвенную схему разделения функции.**

AS-модели (Aplication Server) – трехзвенная схема разделения функций. Здесь прикладной компонент выделен как важный изолированный элемент приложения, для его определения используются универсальные механизмы многозадачной операционной системы и стандартизированы интерфейсы с двумя другими компонентами.

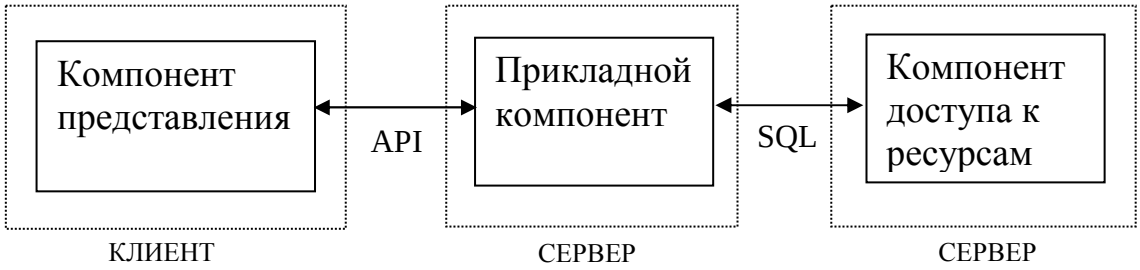
а) В AS-модели процесс, выполняющийся на компьютере – клиенте, отвечает за интерфейс с пользователем.

Обращаясь за выполнением услуг к прикладному компоненту, этот процесс играет роль клиента – приложения (AC – aplication client).

б) Прикладной компонент – реализован как группа процессов, выполняющих прикладные функции, и называется сервером-приложением (AS).

в) Все операции над информационными ресурсами выполняются соответствующим компонентом, по отношению к которому AS играет роль клиента. Из прикладных компонентов доступны ресурсы различных типов:

- базы данных;
- очереди;
- почтовые службы и т.п.



7. СУБД КЛИЕНТ-СЕРВЕР

Системы управления базами данных, использующие платформу клиент-сервер должны быть рассмотрены со стороны сервера (или базы данных) и со стороны клиента (представленных данных).

Рассмотрим серверные системы СУБД Oracle 7 и SQL сервер 4.2.

- Обе системы включают в себя текстовые командные языки, а также пользовательские интерфейсы.
- Это мощные СУБД, содержащие язык определения данных (ЯОД), язык манипулирования (оперирования) данными (ЯМД).

Приведем основные приемы работы:

1. *Определение таблиц.*
   1.1. Создание пользовательских типов данных.
   1.2. Определение отдельных таблиц:
   а) для каждого столбца определить имя, тип данных, и, возможно, ограничения и значения по умолчанию;
   б) определить первичные и внешние ключи;
   в) определить ограничения на вводимые данные.
2. *Создание триггеров.*

7.1. Определение таблиц

7.1.1. Создание пользовательского типа данных

Пользовательский тип данных – это подтип системного типа данных, приспособленный к конкретным требованиям схемы базы данных.

*Введем схему базы данных строительной компании «Премьер»

WORKER (worker_in, worker_name, hrly_rate, skill_type, supv_id);
работник тип специ- менеджер почасовая
альн. ставка
assignment (worker_in, bldg_in, start_date, num_days);
работа
building (bldg_in, adress, type, glty_level, status)
здание

- для связи таблиц в них включены одинаковые столбцы;
- типы данных этих двух столбцов (worker_in, bldg_in) аналогичны и являются числовыми идентификаторами (пусть это положительные числа ≤ 100000 и не содержат пустых значений) т.е. это пользовательский тип данных SQL – сервер дает возможность создавать пользовательские типы помимо системных типов (char, int, date) …

Пользовательский тип – системный подтип, на который наложены ограничения.

В SQL – сервере определение данных осуществляется с помощью SQL Enterprise Manager (Менеджер сервера базы данных).

*Пользовательский тип создается в окне «Manage User Defined Datatypes» (пользовательский тип данных).

Можно определить следующие характеристики типа данных:

- Name (имя);
- Owner (владелец);
- Base Datetype (базовый тип);
- Length (длина);

- Nulls (пустое значение);
- Default (значение по умолчанию);
- Rule (правило).

Типы данных в SQL

Точные числовые
- Integer – целое число;
- Small integer – короткое число;
- Numeric (p,s) – число;
- Decimal (p,s) – десятичное число p – общее количество; s – количество после запятой.

Приблизительные числовые
- Real – действительное число;
- Double precision – числовое с двойной точностью;
- Float – с плавающей запятой (для научных и инженерных расчетов).

Символьные строки
- Character (n) – символьная строка;
- Character varying (n) символьная строка переменной длины;

Тип **character** – всегда хранятся n символов, даже если введенное значение нужно дополнять справа пробелами.

Character varying – хранится столько символов, сколько введено.

Двоичные строки:
- Bit (n) – двоичная строка;
- Bit varying (n) – двоичная строка переменной длины (используется для задания флагов и масок).
- Date – дата;
- Time –время;
- Timestamp – дата и время;
- Time with time zone – время и часовой пояс;

INTERVAL:
- Year – month – год и месяц;
- Date – time – день и время.

ЗАПОЛНЕНИЕ
1) Name – вводим имя id_type;
2) Owner – автоматически устанавливается имя пользователя;
3) Base Datetype – по умолчанию целый «int»;

4) Nulls – не заполняем, что означает запрет на пустые значения;
5) Rule (задается при помощи окна диалога «Managing Rules», в области «Description» вводим:

Create RULE valid_id AS@id>0
And @id <100000

Правило – ограничение значений, принимаемых полем, сформулированное в виде условного выражения.

@id – заменяется действительным значением столбца везде, где этот тип данных используется в определении поля.

В общем случае, при создании правила после AS может стоять любое выражение, допустимое в условии WHERE языка SQL, не содержащее подзапроса и не ссылающиеся на другие поля. Его единственный параметр идентифицируется символом @.

То есть мы создали пользовательский тип с ограничениями;

id_type под таким именем запомним → ПОЛЬЗОВАТЕЛЬСКИЙ ТИП на базе системного «int».

7.1.2. Определение отдельных таблиц

Определяется также в окне «Enterprise Manager» (Таблицы).

Существует следующий порядок:

1. В таблице, расположенной ниже наименования таблица база данных определяются поля новой таблицы (поля таблиц и их описание в одну строку).

Имя таблицы базы данных заносится в окно.

Table

2. Каждая определяющая поле строка состоит из столбцов

Column Name (Имя столбца)

Data type (Тип данных)

Length (Длина)

И индикаторы возможных ограничений:

Nulls (Пустые значения) (если может содержать, то ставим флажок)

Default (Значение по умолчанию)

Rule (Правило)

3. Определяя таблицу, задаются все поля, присваивается имя, выполняется команда Create Table.

Например,

Поле <<skill_type>> (тип_специальный) – имеет символьный тип, ‘длина 10’ и для него надо определить и значение по умолчанию и правило; оно будет автоматически помещаться в столбец, если пользователь не вводит значение.

Например, если не вводим телефон служащего, то система может вводить номер центрального коммутатора.

Значение по умолчанию – это значение, которое автоматически вводится системой, если пользователь опускает значение. (Причем, ввод в столбец Nulls аннулируется любым вводом в столбец Default).

Значение по умолчанию определяется в окне <<Managing Rulles>>

CREATE DEFAULT skill_deflt AS.

‘Плотник’ – теперь в окне определения таблицы в столбце Default строки skill_type мы вводим: «skill_deflt» - значение по умолчанию должно быть константой, т.е. оно не может содержать переменных, имен таблиц или столбцов, а только постоянные значения, возможно, соединенные арифметическими операциями.

Правила (Rule) – ограничивают вводимые значения пользователем.

CREATE RULE valid_skill_type AS

@ skill_type in (‘Электрик’, ‘Штукатур’, ‘Плотник’, ‘Кровельщик’).

7.1.3. Определение первичных и вторичных ключей

Для того чтобы СУБД автоматически поддерживала правила целостности на уровне ссылок реляционной модели, необходимо определить **первичные и внешние ключи**. Первичные и внешние ключи определяются как **ограничительные условия**, но возможные значения кортежей и атрибутов в таблицах баз данных.

Oracle.

В Oracle первичные ключи определяются в контексте определения таблицы, например, в таблице WORKER.

CREAT TABLE worker

(worker_id id_type CONSTRAINT pk_wid PRIMARY KEY), …

Мы выделили определение worker_id как первичного ключа;

CONSTRAINT означает, что далее следует ограничительное условие на поле pk_wid – имя, которое присвоено ограничению.

Если делается попытка нарушить ограничительное условие, то система выдает сообщение об ошибке, указав имя ограничения.

PRIMARY KEY – означает, что поле worker_id будет **внешним** ключом. Это означает, что его значение в таблице worker

1) не должно повторяться;
2) не может содержать пустое значение.

Такое ограничение называется ограничение на поле – является частью определения поля.

Например, в таблице assignment первичный ключ состоит из двух полей: worker_id и bldg_id поэтому он не может быть определен как ограничение на поле. Его нужно определять как **ограничение на таблицу**. Это также определяется в контексте определения таблицы:

CREATE TABLE assignment
(worker_id id_type,
bldg_id id_type,
……
CONSTRAINT pk_’ wkblid PRIMARY KEY (worker_id, bldg_id)

Аналогично, значения пары полей worker_id и bldg_in не должно повторяться в таблице и никакое из полей не должно иметь пустых значений.

Внешние ключи.

Например,

CREATE TABLE assignment
(work_id id_type REFERENCES worker,
bldg_id id_type REFERENCES building,
…….

(Таблица assignment ссылается на поле из таблицы worker и building).

REFERENCES – обозначает определение ВНЕШНЕГО КЛЮЧА. Столбец является внешним ключом, ссылающимся на таблицу, стоящую после слова REFERENCES.

Это означает: его значение должно соответствовать ключевому значению в таблице, на которую внешний ключ ссылается.

Например, в таблице assignment есть кортеж: {1235,515, ….}, то обязательно есть worker_id=1235 и bldg_id=515; которые стоят в ключевом поле некоторого кортежа одноименной таблицы.

Если такие кортежи не существуют в момент ввода строки значения, тогда СУБД не позволит добавить такую строку в таблице assignment.

Если строка успешно введена, но позднее соответствующий кортеж таблицы **worker** удаляется.

В этом случае удаление кортежа **worker** будет запрещено до тех пор, пока не будут удалены все ссылающиеся на него строки таблицы **assignment**.

DELETE RESTRICT – она устанавливается по умолчанию.
(удаление ограничивать)

ON DELETE CASCADE – при удалении строки другой таблицы, на
(удаление каскадное) которую ссылается кортеж таблицы **assignment** – он будет автоматически удален.

Например,

CREATE TABLE assignment
(work_id id_type REFERENCES worker ON DELETE CASCADE,
bldg_id id_type REFERENCES building ON DELETE CASCADE,
…….

То есть в Oracle, все внешние и первичные ключи определяются ЯВНО и СУБД Oracle автоматически поддерживает их. (То есть при вводе данных СУБД проверяет, что первичные ключи не пусты и не повторяются, а значениям внешних ключей соответствуют значения первичных ключей, на которые они ссылаются).

Внешний ключ – набор атрибутов в одной реляционной таблице, составляющих ключ другой реляционной таблицы (или возможно той же самой таблице); применяется для задания логических связей между реляционными таблицами.

Первичный ключ – потенциальный ключ, выбранный в качестве основного средства однозначного определения строк реляционной таблицы.

В SQL-Server(e) поддержка первичных и вторичных ключей выполняется с помощью **триггеров** – программа, которая автоматически исполняется, когда предпринимается попытка обновления определенного типа заданной таблицы.

CHEK – ограничения

CHEK – ограничения – общее ограничение на поле и таблицу, сформулированное в виде условного выражения.

1) CHEK – ограничения, как правило, используются для задания *общих* правил формирования данных.

Например, таблица employee (emp_id, emp_name, emp_adress, salary, bonus).

Пусть политика компании такова, что размер премии не должен превышать 10% от зарплаты:

(*) CHEK (bonus<=.10 * salary)/

2) CHEK – ограничения могут быть частью определения поля или табличным ограничениям.

3) CHEK – ограничения могут влиять на любое поле таблицы, а не только на то, частью определения которого оно является.

4) Часто это ограничение является табличным, так как обычно касается целого кортежа, а не поля.

5) CHEK – ограничения Oracle, аналогичны правилам в SQL Server;

CREAT RULE Valid_skill_type AS

@skill_type IN (‘Электрик’, ‘Штукатур’, ‘Плотник’, ‘Кровельщик’) в Oracle тоже самое:

CHECK (skill_type IN (‘Электрик’, ‘Штукатур’, ‘Плотник’, ‘Кровельщик’))

CHECK – ограничения дают больше возможностей, чем правило (RULE), так как применяются сразу к нескольким столбцам, но в отличие от триггеров CHECK не может содержать подзапрос, ссылающийся на значения других кортежей той же таблицы или кортежи любой другой таблицы.

То есть ограничение (*) всегда будет сравнивать только значение атрибута bonus кортежа со значением атрибута salary того же кортежа.

7.2. Создание триггеров

Триггер – программа, которая автоматически выполняется при попытке изменения содержимого заданной таблицы.

Существует три типа триггеров:

- ввода;
- добавления;
- удаления.

Предположим мы хотим поддерживать в базе данных столбец, полученный в результате вычислений над другими столбцами.

Например, имея почасовую ставку работника и полагая рабочий день восьмичасовым, мы можем подсчитать зарплату работника за определенное количество дней.

Тогда в таблице Worker, появится поле «comulative_pay».

Технология работы триггеров.

1. Когда над таблицей выполняются операции добавления, изменения или удаления данных, создаются новые версии управляемых системой **триггерных таблиц**.

2. Эти **триггерные таблицы** называются inserted (введено) и deleted (удалено).

- Если в таблице введены строки, то они помещаются в таблицу inserted, а таблица deleted остается пустой.
- Если из таблицы удаляются строки, то они помещаются в таблице deleted, а таблица inserted остается пустой.
- Если таблица обновляется, то deleted состоит из старых версий строк, а таблица inserted состоит из новых версий тех же строк.
- SQL сервер запускает триггер после того, как обновление файла произошло. Таким образом, когда триггер запускается, файлы inserted и deleted уже существуют.

Например:

предположим, что таблица worker была расширена: в нее включено поле «comulative_pay».

1. Worker (Worker_id, worker name, hrly_rate, skill_type, supv_id, comulative_pay).

2. Формула подсчета comulative_pay:

comulative_pay = total_name_days*8* hrly_rate)

total_name_days вычисляется по таблице.

3. Каждый раз, когда к таблице assignment добавляется новая строка, или же существующая строка обновляется, мы хотим обновлять и поле comulative_pay для соответствующего работника.

Реализация:

```
Create trigger update_assignment
on assignment
for insert, update, deleted
as
update worker
set comulative_pay = comulative_pay+ 8 *
hrly_rate*
(select sum (num_days) from inserted
where inserted. Worker_id = worker. worker_id)
update worker
set comulative_pay = comulative_pay-8 * hrly_rate*
(select sum (num_days) from deleted
where deleted.worker_id = worker. worker_id)
```

Разберем этот триггер:

1. Первая строка дает имя триггеру update_assignment/
2. Вторая означает, что он применяется к таблице assignment.
3. Третья – триггер будет запускаться от каждой операции – ввода, обновления или удаления.
4. Строка с AS открывает программную часть триггера (все, что следует далее выполняется системой при запуске триггера).
5. Программная часть состоит из двух команд обновления, каждая из которых применяется к таблице WORKER;

- первая команда прибавляет к значению столбца comulative_pay значение, вычисленное по таблице inserted;
- вторая вычитает значение, вычисленное по таблице deleted.

Эти две команды заставляют систему проходить таблицу worker дважды

- первая команда update рассматривает строки, которые были добавлены к таблице assignment;

Если к-п строки были добавлены (т.е. вводились данные) атрибут num_days добавленных кортежей учитывается в соответствие кортеже таблице worker.

7.3. Описание с помощью SQL

```
CREATE SHEMA PREMIER
AUTORIZATION JON {имя владельца не обязательно}
CREATE TABLE WORKER(
WORKER_ID ITEM_IDENTIFIER PRIMARY’ KEY,
WORKER_NAME CHARACTER (12),
```

HRLY_RATE NUMERIC (5,2), (_ _ _ _ _)
SKILL_TYPE CHARACTER (8),
SUPV_ID NUMERIC (4),
FOREIGN KEY SUPV_ID REFERENCES WORKER
CREATE TABLE …..
ON DELETE SET NULL)
……

Означает (если кортеж, на который указывает внешний ключ, удаляется, то значение внешнего ключа, должно быть пустым).

То есть рекурсивные или внешние ключи существуют для поддержания целостности данных (то есть, чтобы не было ситуации, когда осуществляется связь с несуществующими данными).