Тема 29. Физическая организация данных

На физическом уровне БД реализуется в виде совокупности файлов. Будем считать, что каждый файл состоит из записей одного формата. Связи представляются с помощью указателей на записи. Если на некоторую запись в БД есть указатель, то эта запись считается закрепленной на месте. В противном случае запись не закреплена. Над файлом могут быть выполнены операции включения, удаления, модификации записи и поиска записи с указанными значениями полей.

Распространенными способами организации файлов являются хеширование, индексирование и представление в виде сбалансированных деревьев.

Хешированные файлы

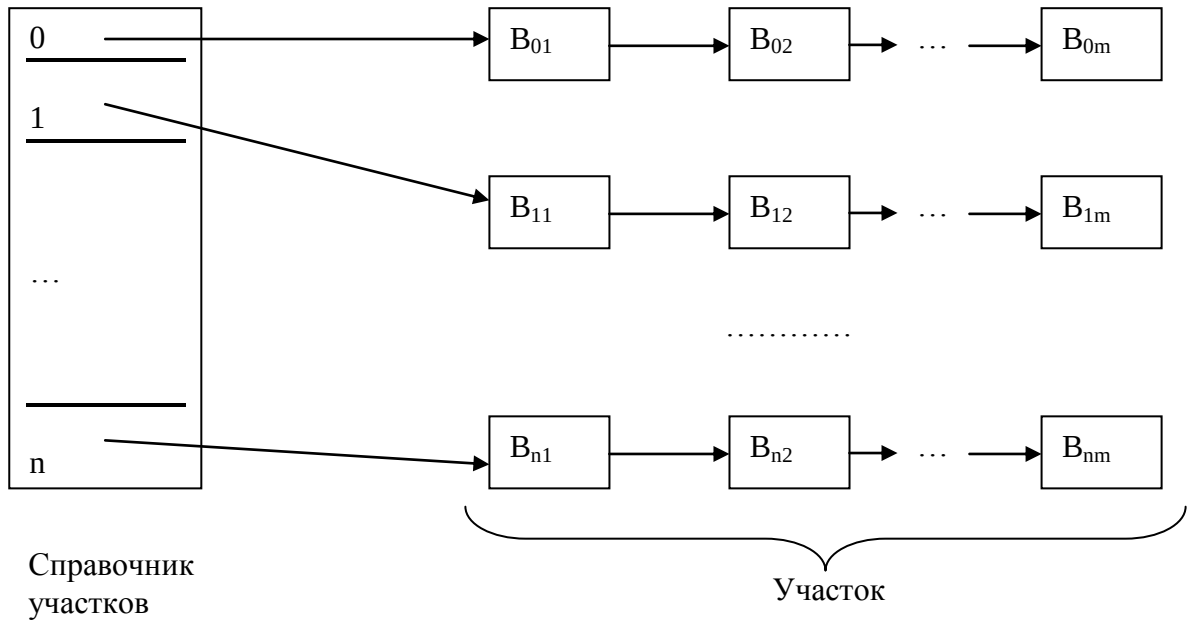

Рис. 29.1. Хешированный файл.

Хешированный файл (рис. 29.1) состоит из справочника и набора участков, участок состоит из одного или более блоков. Записи файла распределяются по участкам с помощью хеш-функции $h:V \rightarrow N$ (V – домен значений ключа, N – домен номеров участков). Для значения v ключа величина h(v) есть номер участка, в котором находится запись с данным значением

ключа. Желательно, чтобы функция h хешировала ключ V, т.е. равномерно отображала значения ключа в номера участков.

Один из простейших алгоритмов хеширования:

1. Ключ рассматривается как последовательность битов.
2. Биты ключа делятся на группы фиксированной длины.
3. Группы складываются как целые числа.
4. Номер участка вычисляется как остаток от деления полученной суммы на число участков.

Для поиска записи с заданным значением v ключа вычисляется h(v) (номер участка), из справочника извлекается ссылка на первый блок участка, последовательно просматриваются блоки участка.

Индексированные файлы

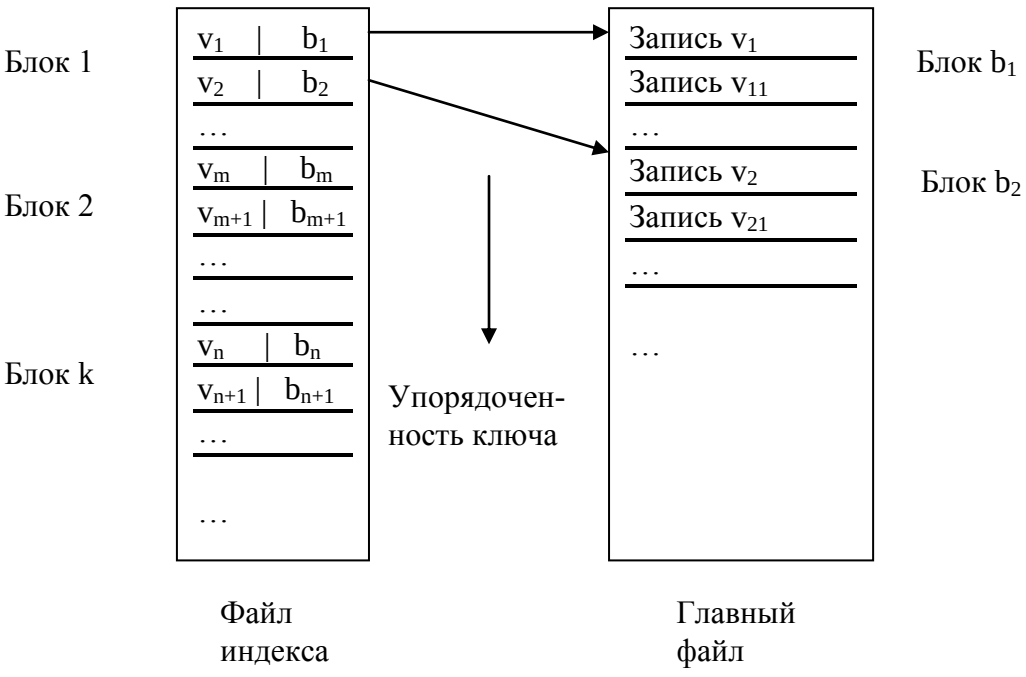

Рис. 29.2. Индексированный файл.

Индексированный файл (рис.29.2) состоит из главного файла, содержащего записи БД, и файла индекса, записи которого хранят пары (ключ записи, указатель блока). Оба файла упорядочены по возрастанию ключа.

Индексная запись (v,b) состоит из указателя блока главного файла и ключа первой записи блока (разреженный индекс).

Над файлом индекса выполняются те же операции, что и над главным файлом. Дополнительно выполняется следующая операция: для заданного значения ключа v_1 найти в индексе такую запись (v_2,b), что $v_1>=v_2$ и либо (v_2,b) – последняя запись в индексе, либо следующая запись (v_3,b) такова, что $v_1<v_3$. В этом случае говорят, что значение v_2 покрывает v_1. Таким способом находят блок главного файла, содержащий запись со значением ключа v_1.

Поиск в индексе

Индекс хранится в известной совокупности блоков. Требуется найти индексную запись (v_2,b), покрывающую заданное значение v_1 ключа.

Простейший вид поиска – **линейный поиск.**

Двоичный поиск. Если задано значение v_1 ключа и индекс, включающий блоки B_1, …B_n, то рассматривается средний блок $B_{\lceil n/2 \rceil}$ и v_1 сравнивается со значением v_2 ключа в первой записи этого блока ($\lceil x \rceil$ - округление до ближайшего целого сверху). При $v_1<v_2$ процесс повторяется над блоками B_1, … $B_{\lceil n/2 \rceil -1}$. Если $v_1>=v_2$, то процесс повторяется над блоками $B_{\lceil n/2 \rceil}$, …B_n. В конце концов, останется единственный блок, который просматривается линейно. Двоичный поиск требует примерно $\log_2 n$ чтений блоков индекса.

Интерполяция. Метод основан на знании распределения значений ключа.

Пусть $v_2<=v_1<=v_3$ и имеется функция $f(v_1,v_2,v_3)=0\div1$, которая указывает ожидаемое положение v_1 на пути от v_2 к v_3. Пусть индекс размещается в блоках B_1, … B_n и v_2 - первый ключ в B_1, а v_3 - последний в B_n. Рассмотрим блок B_i, где $I = \lceil nf(v_1,v_2,v_3) \rceil$, и сравним его первый ключ с v_1. Далее как и в двоичном поиске, повторяем процесс либо над B_1, … B_{i-1}, либо над B_i, … B_n в зависимости от результатов сравнения и т.д.

Можно показать, что при знании распределения значений ключа потребуется около $1+\log_2\log_2 n$ доступов к блокам индекса.

Сбалансированные деревья (B – деревья)

B-дерево (рис.29.3) является многоуровневым индексом, в котором каждый путь от корня (индекса первого уровня) до листа (блока главного файла) имеет одну и ту же длину. Число уровней не фиксировано.

Пусть блок главного файла содержит 2e-1 записей, а блок индекса 2d-1 записей. Всего записей m.

Поиск. Ищется путь от корня к листу с требуемой записью. Пусть в некоторый момент достигнут узел B. Пусть B – лист, тогда проверяем, имеется ли в этом блоке запись со значением ключа v. Если B не лист, тогда он – блок индекса. Определяем, какое значение ключа в блоке покрывает v. В индексной записи с покрывающим значением находим указатель на блок нижнего уровня. С найденным блоком повторяются все рассмотренные шаги. Число обменов $\approx 2+\log_d(m/e)$.

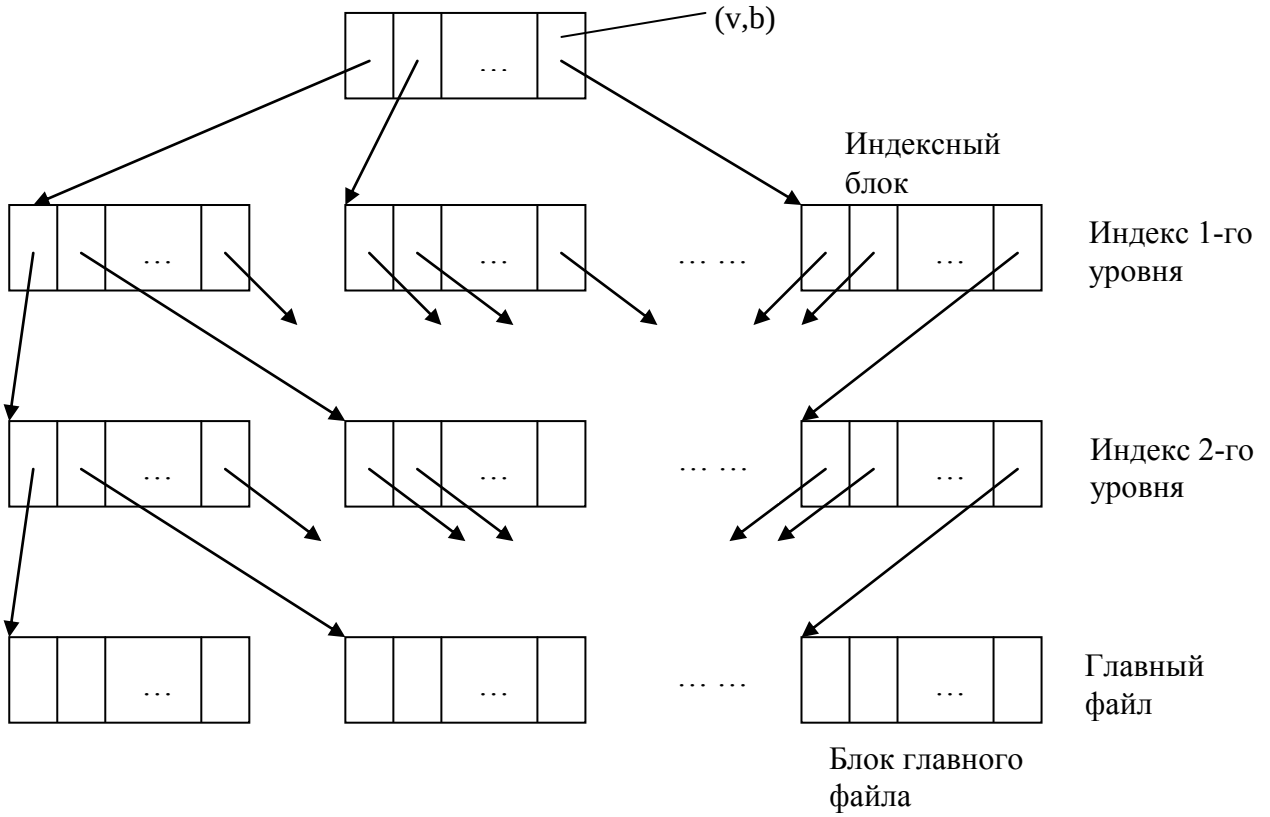

Рис. 29.3. Сбалансированное дерево.

Модификация записи и включение новой.

Включение. Находим нужный блок главного файла. Если в блоке менее 2e-1 записей, то запись включается в блок. Если в блоке уже имеется 2e-1 записей, то создается новый блок B_1, и данные из B частично переписываются в B_1. Пусть P – родитель B (P – индексный блок). Применим процедуру включения рекурсивно, чтобы включить в блок P индекса запись для блока B_1. Процесс включения может затронуть несколько уровней.

Удаление. Находим блок B главного файла с удаляемой записью. Если после удаления в B осталось e или более записей, операция завершена. Если после удаления осталось e-1 записей, то возможна пересылка части записей из соседних блоков в B или слияние двух блоков в один. В последнем случае происходит удаление блока, которое реализуется рекурсивным применением процедуры удаления.

Файлы с плотным индексом

Пусть главный файл не сортирован. Для поиска записи по ключу в несортированном файле используется плотный индекс. Плотный индекс состоит из пар (v,p) для каждого значения ключа v в главном файле (p – указатель записи главного файла).

Чтобы найти, модифицировать или удалить некоторую запись главного файла, нужно найти в плотном индексе запись с таким же значением ключа и затем прочитать блок главного файла с требуемой записью. Далее выполняется требуемое действие. При включении запись помещается в конец главного файла, а указатель на нее - в файл плотного индекса.

Показатели разных способов организации файлов приведены в табл. 4.

Файлы с записями переменной длины

Записи переменной длины содержат в своем составе повторяющиеся однотипные группы значений. С помощью таких записей можно хранить отображения «многие ко многим». Например, для хранения связей объектов набора E_1 с объектами набора E_2 (рис. 29.4) можно использовать файл записей переменной длины, в котором каждая запись состоит из группы полей,

представляющих объект из E_1, и повторяющейся группы полей, в которой каждый экземпляр группы представляет объект типа E_2. Число повторений группы - переменное.

Таблица № 4. Сравнение способов организации файлов.

Организация	Время выполнения	Достоинства и недостатки
Хеширование	>=3	Самый быстрый Нет сортировки
Разреженный Индекс	≈2+logn – двоичн. поиск ≈2+loglogn – вычисл. адреса	Быстрый Сортировка
B – дерево	≈2+$\log_d$(m/e)	Быстрый Сортировка
Плотный Индекс	<=2+ время плотн. индекса	Медленный Сортировка

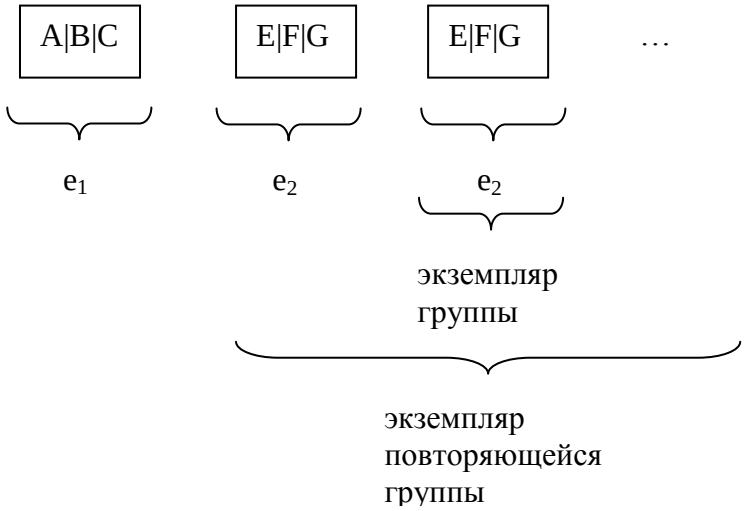

Рис. 29.4. Запись переменной длины.

Формат записи (группы) – это список ее полей. Формат повторяющейся группы записывается так: формат_группы*. Экземпляр записи, группы, повторяющейся группы – совокупность значений полей заданного формата.

Обычно записи переменной длины хранятся в одной или нескольких записях фиксированной длины. Используются три метода хранения.

Метод зарезервированного пространства. Повторяющаяся группа представляется фиксированным числом записей постоянной длины, способной вместить максимальное число экземпляров группы.

Метод указателей. Повторяющуюся группу можно реализовать в виде цепочки блоков, ссылка на которую помещается в запись, содержащую повторяющуюся группу.

Комбинированный метод.

Поиск по неключевым полям

Пусть требуется найти запись со значением v_1 в поле F_1, v_2 в поле F_2, … v_k в поле F_k. Если S_i - множество записей со значением v_i в поле F_i (i = 1..k), то требуемое множество $S=S_1 \cap S_2 \cap \ldots \cap S_k$. Существуют два метода решения этой задачи: использование множественных вторичных индексов, использование функций раздельного хеширования.

Вторичные индексы

Пусть записи файла содержат поле F, не обязательно являющееся ключом и принимающее значения из множества D. Вторичный индекс по полю F есть связь между доменом D и множеством записей файла. Файл с вторичным индексом по полю F называется инвертированным по полю F. Вторичный индекс имеет формат ЗНАЧЕНИЕ ЗАПИСЬ*, где ЗАПИСЬ – ключ записи с требуемым значением в поле F.

В процессе поиска по неключевым полям находится пересечение множеств S_i, поддерживаемых вторичными индексами. В найденном пересечении просматриваются все записи. Для ускорения поиска следует начинать с наименьшего множества S_i. Для этого выбираются поля с наибольшим множеством различных значений.

Функции раздельного хеширования

Пусть каждое поле записи имеет свою функцию хеширования. Полный номер участка в этом случае функционально зависит от хеш-значений всех полей записи. Такая организация адресации позволяет вычислять частично определенные номера участков (часть битов определена, часть - нет).

Пусть число участков 2^B. Логический адрес участка в этом случае есть последовательность B битов. Разделим биты адреса на группы – по одной группе для каждого поля. Некоторым полям может быть назначено по 0 бит.

Если имеются поля $F_1,\ldots F_k$ и для поля F_i назначено b_i бит, то адрес записи $(v_1,v_2,\ldots v_k)$ определяется путем вычисления $h_i(v_i)$ (i = 1..k). В качестве адреса участка принимается последовательность B бит $h_1(v_1)\ h_2(v_2)\ldots\ h_k(v_k)$. Так как все хеш-функции полей вычисляются независимо от значения других полей, знание значения F_i сокращает число просматриваемых участков в 2^{bi} раз.

Бесфайловая физическая организация данных

Использование файловых структур для хранения данных освобождает СУБД от функций управления внешней памятью, которые берет на себя ОС. Это в известной мере ограничивает возможности СУБД, т.к. требования к управлению у СУБД и ОС различаются. По этой причине некоторые СУБД взяли на себя непосредственное управление памятью.

Рассмотрим некоторые принципы бесфайловой физической организации данных.

Имеются две классификации объектов физической модели: физическая (строки, страницы, чанки, экстенты, страницы Blob-объектов) и логическая (область базы данных, область таблиц, область Blob-объектов).

Чанк – часть диска, ассоциированная одному процессу.

Экстент – непрерывная область дисковой памяти. Совокупность экстентов представляет логическую единицу.

Экстенты состоят из четырех типов страниц: страницы данных, страницы индексов, битовые страницы и страницы Blob–объектов (неструктурированные данные). Основная единица операций обмена – страница.

Структура страницы имеет вид:

Слоты характеризуют размещение строк данных на странице. Каждая строка имеет однозначный идентификатор вида (номер экстента/ номер страницы/ номер строки). Страницы индексов организованы в

виде B-деревьев. Страницы Blob-объектов хранят слабоструктурированную информацию, содержащую тексты большого объема, графику, двоичные коды. В страницах данных делаются ссылки на эти объекты. Битовые страницы служат для трассировки других типов страниц.

СУБД организует специальные структуры в оперативной памяти (разделяемая память) и журналы транзакций во внешней памяти. Разделяемая память (рис. 29.5) служит для кэширования данных и для поддержки параллельной обработки данных. Она содержит буферы страниц данных, журнала транзакций, системного каталога.

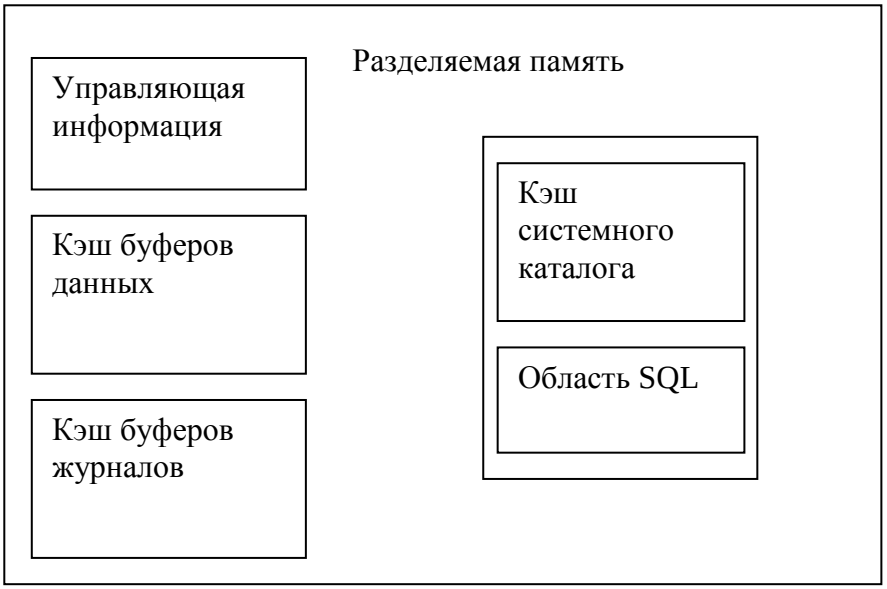

Рис. 29.5. Разделяемая память.

Обобщенная архитектура СУБД.

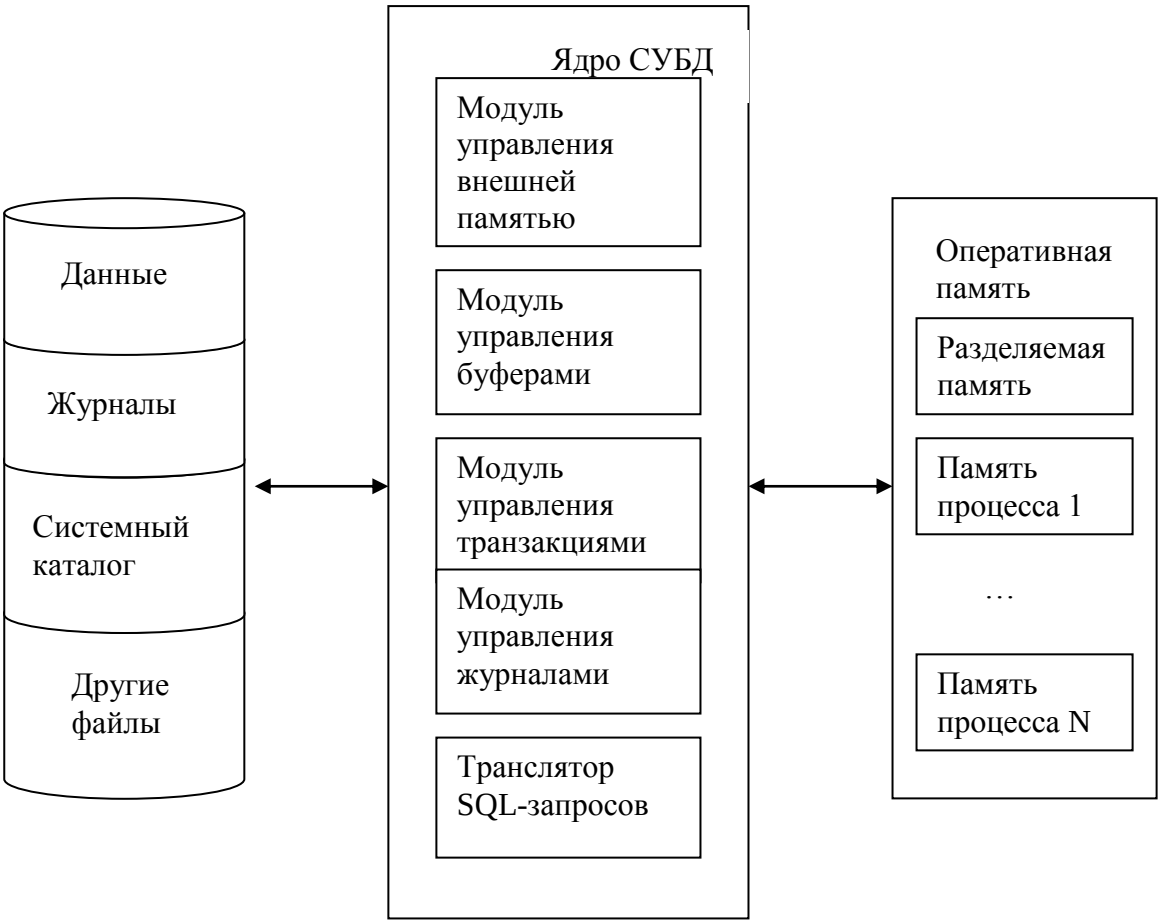

Рис. 29.6. Архитектура СУБД.

Архитектуру СУБД условно можно представить в виде рисунка 29.6.

Внешняя память содержит файлы данных, журналов, системного каталога и др.

Ядро СУБД управляет данными во внешней памяти, буферами оперативной памяти, транзакциями, журналами, содержит транслятор SQL.

В оперативной памяти СУБД размещаются буферы данных, журналов, управляющая информация, кэш словаря данных, область SQL.

Системный каталог является совокупностью специальных таблиц с ограниченным доступом. Таблицы каталога содержат информацию о структурных элементах БД. В SQL 2 определены следующие системные таблицы: USERS, SCHEMA, DATA_TYPE_DESCRIPTION, TABLES, VIEWS, TABLE CONSTRAINS, TABLE_PRIVILEGES и др.