**Тема 1. Введение в базы данных. Язык запросов SQL. Базовые CRUD операции.**

**Предпосылки появления баз данных**

Важным шагом в развитии информационных систем явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы, файл — это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Конечные пользователи видят файл как линейную последовательность записей и могут выполнить над ним ряд стандартных операций:

* создать файл (требуемого типа и размера);
* открыть ранее созданный файл;
* прочитать из файла некоторую запись (текущую, следующую, предыдущую, первую, последнюю);
* записать в файл на место текущей записи новую, добавить новую запись в конец файла.

В разных файловых системах эти операции могли несколько отличаться, но общий смысл их был именно таким. Главное, что следует отметить, это то, что структура записи файла была известна только программе, которая с ним работала, система управления файлами не знала ее. И поэтому для того, чтобы извлечь некоторую информацию из файла, необходимо было точно знать структуру записи файла с точностью до бита. Каждая программа, работающая с файлом, должна была иметь у себя внутри структуру данных, соответствующую структуре этого файла. Поэтому при изменении структуры файла требовалось изменять структуру программы, а это требовало новой компиляции, то есть процесса перевода программы в исполняемые машинные коды. Такая ситуация характеризовалась как зависимость программ от данных. Для информационных систем характерным является наличие большого числа различных пользователей (программ), каждый из которых имеет свои специфические алгоритмы обработки информации, хранящейся в одних и тех же файлах. Изменение структуры файла, которое было необходимо для одной программы, требовало исправления и перекомпиляции и дополнительной отладки всех остальных программ, работающих с этим же файлом. Это было первым существенным недостатком файловых систем, который явился толчком к созданию новых систем хранения и управления информацией.

Поскольку файловые системы являются общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, системы управления файлами должны обеспечивать авторизацию доступа к файлам. В общем виде подход состоит в том, что по отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю. В большинстве современных систем управления файлами применяется подход к защите файлов, впервые реализованный в ОС UNIX. В этой ОС каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: идентификатор группы, к которой относится этот пользователь, и его собственный идентификатор в группе. При каждом файле хранится полный идентификатор пользователя, который создал этот файл, и фиксируется, какие действия с файлом может производить его создатель, какие действия с файлом доступны для других пользователей той же группы и что могут делать с файлом пользователи других групп. Администрирование режимом доступа к файлу в основном выполняется его создателем-владельцем. Для множества файлов, отражающих информационную модель одной предметной области, такой децентрализованный принцип управления доступом вызывал дополнительные трудности. И отсутствие централизованных методов управления доступом к информации послужило еще одной причиной разработки СУБД.

Следующей причиной стала необходимость обеспечения эффективной параллельной работы многих пользователей с одними и теми же файлами. В общем случае системы управления файлами обеспечивали режим многопользовательского доступа. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда два или более пользователя одновременно пытаются работать с одним и тем же файлом. Если все пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этих пользователей требуется взаимная синхронизация их действий по отношению к файлу.

В системах управления файлами обычно применялся следующий подход. В операции открытия файла (первой и обязательной операции, с которой должен начинаться сеанс работы с файлом) среди прочих параметров указывался режим работы (чтение или изменение). Если к моменту выполнения этой операции некоторым пользовательским процессом PR1 файл был уже открыт другим процессом PR2 в режиме изменения, то в зависимости от особенностей системы процессу PR1 либо сообщалось о невозможности открытия файла, либо он блокировался до тех пор, пока в процессе PR2 не выполнялась операция закрытия файла.

При подобном способе организации одновременная работа нескольких пользователей, связанная с модификацией данных в файле, либо вообще не реализовывалась, либо была очень замедлена.

Эти недостатки послужили тем толчком, который заставил разработчиков информационных систем предложить новый подход к управлению информацией. Этот подход был реализован в рамках новых программных систем, названных впоследствии Системами Управления Базами Данных (СУБД), а сами хранилища информации, которые работали под управлением данных систем, назывались базами данных (БД).

**База данных (БД)** — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

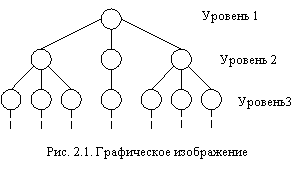
**Система управления базами данных (СУБД)** — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Программы, с помощью которых пользователи работают с базой данных, называются приложениями. В общем случае с одной базой данных могут работать множество различных приложений. Например, если база данных моделирует некоторое предприятие, то для работы с ней может быть создано приложение, которое обслуживает подсистему учета кадров, другое приложение может быть посвящено работе подсистемы расчета заработной платы сотрудников, третье приложение работает как подсистемы складского учета, четвертое приложение посвящено планированию производственного процесса. При рассмотрении приложений, работающих с одной базой данных, предполагается, что они могут работать параллельно и независимо друг от друга, и именно СУБД призвана обеспечить работу множества приложений с единой базой данных таким образом, чтобы каждое из них выполнялось корректно, но учитывало все изменения в базе данных, вносимые другими приложениями.

**Ранние подходы к организации баз данных**

**Иерархическая модель**

Иерархическая структура представляет совокупность элементов, связанных между собой по определенным правилам. Графическим способом представлением иерархической структуры является дерево (см. рис. 2.1).



Дерево представляет собой иерархию элементов, называемых узлами. Под элементами понимается совокупность атрибутов, описывающих объекты. В модели имеется корневой узел (корень дерева), который находится на самом верхнем уровне и не имеет узлов, стоящих выше него. У одного дерева может быть только один корень. Остальные узлы, называемые порожденными, связаны между собой следующим образом: каждый узел имеет только один исходный, находящийся на более высоком уровне, и любое число (один, два или более, либо ни одного) подчиненных узлов на следующем уровне.

Примером простого иерархического представления может служить административная структура высшего учебного заведения: институт – отделение – факультет – студенческая группа (см. рис. 2.2).

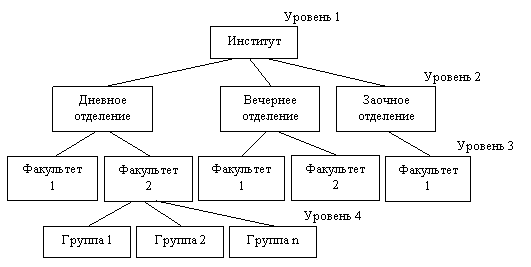


Рис. 2.2. Пример иерархической структуры

К достоинствам иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения операций над данными.

Недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями.

**Сетевая модель данных**

Отличие сетевой структуры от иерархической заключается в том, что каждый элемент в сетевой структуре может быть связан с любым другим элементом (см. рис. 2.3). Пример простой сетевой структуры показан на рис. 2.4.



Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. Недостатком сетевой модели данных являются высокая сложность и жесткость схемы БД, построенной на ее основе.

**Реляционная модель данных**

Реляционная модель данных была предложена Е.Ф. Коддом, известным исследователем в области баз данных, в 1969 году, когда он был сотрудником фирмы IBM. Впервые основные концепции этой модели были опубликованы в 1970.

Реляционная база данных представляет собой хранилище данных, организованных в виде двумерных таблиц (см. рис. 2.5). Любая таблица реляционной базы данных состоит из строк (называемых также записями) и столбцов (называемых также полями).

Строки таблицы содержат сведения о представленных в ней фактах (или документах, или людях, одним словом, - об однотипных объектах). На пересечении столбца и строки находятся конкретные значения содержащихся в таблице данных.

Данные в таблицах удовлетворяют следующим принципам:

* Каждое значение, содержащееся на пересечении строки и столбца, должно быть атомарным.
* Значения данных в одном и том же столбце должны принадлежать к одному и тому же типу, доступному для использования в данной СУБД.
* Каждая запись в таблице уникальна, то есть в таблице не существует двух записей с полностью совпадающим набором значений ее полей.
* Каждое поле имеет уникальное имя.
* Последовательность полей в таблице несущественна.
* Последовательность записей в таблице несущественна.
* Поле или комбинацию полей, значения которых однозначно идентифицируют каждую запись таблицы, называют возможным ключом (или просто ключом ).
* Если таблица имеет более одного возможного ключа, тогда один ключ выделяют в качестве первичного . Первичный ключ любой таблицы обязан содержать уникальные непустые значения для каждой строки.
* Поле, указывающее на запись в другой таблице, связанную с данной записью, называется внешним ключом .
* Подобное взаимоотношение между таблицами называется связью . Связь между двумя таблицами устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой.
* Группа связанных таблиц называется схемой базы данных. Информация о таблицах, их полях, первичных и внешних ключах, а также иных объектах базы данных, называется метаданными.

Достоинство реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной ее широкого использования.

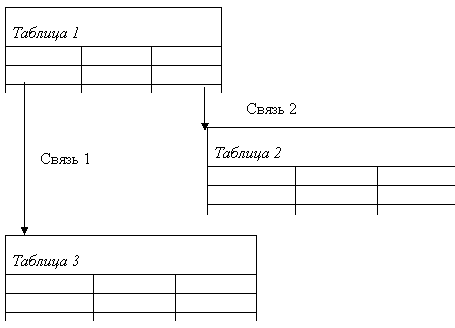


Рис. 2.5. Схема реляционной модели данных

К основным недостаткам реляционной модели относятся отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

**Классификация баз данных**

По **технологии обработки** данных базы данных подразделяются на **централизованные** и **распределенные.**

Централизованная база данных хранится в памяти одной вычислительной системы. Эта вычислительная система может быть мэйнфреймом - тогда доступ к ней организуется с использованием терминалов - или файловым сервером локальной сети ПК.

Распределенная база данных состоит из нескольких, возможно, пересекающихся или даже дублирующих друг друга частей, которые хранятся в различных ЭВМ вычислительной сети. Работа с такой базой осуществляется с помощью системы управления распределенной базой данных (СУБД).

По **способу доступа** к данным базы данных разделяются на базы данных с **локальным** доступом и базы данных с **сетевым** доступом.

Для всех современных баз данных можно организовать сетевой доступ с многопользовательским режимом работы.

Централизованные базы данных с сетевым доступом могут иметь следующую архитектуру:

* файл-сервер ;
* клиент-сервер базы данных;
* "тонкий клиент" - сервер приложений - сервер базы данных (трехуровневая архитектура).

**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.

**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 всего 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

**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 формами (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.

Чтобы познакомиться с синтаксисом языка SQL, для начала давайте рассмотрим, как выглядят базовые CRUD-операторы SQL.

**Оператор SELECT**

Оператор SELECT является наиболее часто используемым оператором SQL (и также называется проекцией — projection). Он позволяет извлекать все или некоторые данные из таблицы на основе определенных критериев.

Проще всего оператор SELECT выглядит, когда применяется для извлечения всех данных таблицы:

SELECT \* FROM employees;

Для извлечения только определенных столбцов необходимо указывать имена этих столбцов после ключевого слова SELECT, например:

SELECT first\_name, last\_name, hiredate FROM employees;

С помощью необязательной конструкции WHERE в операторе SELECT можно задавать различные условия и тем самым обеспечивать возврат только определенных строк. Вот наиболее типичные условия, которые можно указывать в конструкции WHERE: Символ Условие

= Равно

> Больше

< Меньше

<= Меньше или равно

>= Больше или равно

<> Не равно

Ниже приведено несколько примеров применения конструкции WHERE :

SELECT employee\_id WHERE salary = 50000;

SELECT employee\_id WHERE salary < 50000;

SELECT employee\_id WHERE salary > 50000;

SELECT employee\_id WHERE salary <= 50000;

**Оператор INSERT**

Оператор INSERT позволяет добавлять в таблицу новые данные, в том числе и дублированные в случае отсутствия требований уникальности со стороны первичного ключа или индекса. В общем, синтаксис оператора INSERT выглядит следующим образом:

INSERT INTO <таблица> [(<столбец i, ... , столбец j>)]

VALUES (<значение i, ... , значение j>);

Ниже приведен пример применения оператора INSERT:

INSERT INTO employees(employee\_id, last\_name, email, hire\_date, job\_id) VALUES (56789, 'alapati', 'salapati@netbsa.org', sysdate, 98765);

При желании вставить значения во все столбцы таблицы можно использовать более простой оператор INSERT, подобный тому, что показан ниже:

INSERT INTO department VALUES (34567, 'payroll', 'headquarters', 'dallas');

**Оператор DELETE**

Оператор DELETE служит для удаления строк из таблицы. Его синтаксис, в общем случае, выглядит так:

DELETE FROM <таблица> [WHERE <условие>];

Например, для удаления пользователя Fay из таблицы employees потребуется использовать такой оператор DELETE:

DELETE FROM employees WHERE last\_name='Fay';

Если ограничивающее условие WHERE не указано, оператор DELETE удалит все строки из таблицы:

**Оператор UPDATE**

Оператор UPDATE применяется для изменения значения (или значений) одного или нескольких столбцов в строке (или строках) внутри таблицы. Выражение, используемое для установки или изменения значения столбца, может представлять собой константную, арифметическую или строковую операцию или же результат выполнения оператораS ELECT.

В общем, синтаксис оператора UPDATE выглядит следующим образом (элементы в квадратных скобках являются необязательными):

UPDATE <таблица> SET <столбец i> = <выражение i>, ... , <столбец j> = <выражение j> [WHERE <условие>];

При желании изменить или модифицировать значения столбца во всех строках таблицы можно использовать оператор UPDATE без конструкции WHERE:

UPDATE persons SET salary = salary\*0.10;

Для модификации только определенных строк нужно обязательно задавать в операторе UPDATE конструкцию WHERE:

UPDATE persons SET salary = salary \* 0.10 WHERE review\_grade > 5;