# 

# Курсовая работа по дисциплине "Базы данных"

на тему " Структура языка SQL"

# Содержание

# Введение

1 Понятие базы данных и СУБД

## 1.1 Предметная область

## 1.2 Концепция баз данных

### 1.2.1 Независимость приложений от организации данных во внешней памяти

### 1.2.2 Эффективность организации данных

### 1.2.3 Интеграция данных

### 1.2.4 Что такое база данных

2. Типы данных SQL

# 2.1 Таблицы SQL

# 2.2 Структура языка SQL

# 2.3 Операторы SQL

# Заключение

Глоссарий

Список использованных источников

# Приложение А

# Введение

Одним из основных преимуществ реляционного подхода к организации баз данных (БД) является то, что пользователи реляционных БД получают возможность эффективной работы в терминах простых и наглядных понятий таблиц, их строк и столбцов без потребности знания реальной организации данных во внешней памяти.

Реляционная модель данных, содержащая набор четких предписаний к базовой организации любой реляционной системы управления базами данных (СУБД), позволяет пользователям работать в ненавигационной манере, т.е. для выборки информации из БД человек должен всего лишь указать список интересующих его таблиц и те условия, которым должны удовлетворять выбираемые данные. СУБД скрывает от пользователя выполняемые ей последовательные просмотры таблиц, выполняя их наиболее эффективным образом. Очень важная особенность реляционных систем состоит в том, что результатом выполнения любого запроса к таблицам БД является также таблица, которую можно сохранить в БД и/или по отношению к которой можно выполнять новые запросы.

Базовым требованием к реляционным СУБД является наличие мощного и в тоже время простого языка, позволяющего выполнять все необходимые пользователям операции. В последние годы таким повсеместно принятым языком стал язык реляционных БД SQL - Structured Query Language (теперь все чаще название языка понимается как Standard Query Language) .

До появления SQL в СУБД (независимо от того, на какой модели они основывались) приходилось поддерживать по крайней мере три языка, которые обычно имели мало общего: язык определения данных (ЯОД), служащий для спецификации структур БД (обычно общую структуру БД называют схемой БД); язык манипулирования данными (ЯМД), позволяющий создавать прикладные программы, взаимодействующие с БД; и язык администрирования БД (ЯАДБ), с помощью которого можно было выполнять служебные действия (например, изменять структуру БД или производить ее настройку с целью повышения эффективности). Кроме того, если требовалось предоставить пользователям СУБД интерактивный доступ к БД, приходилось вводить еще один язык, операторы которого выполняются в диалоговом режиме. Язык SQL позволяет решать все эти задачи.

Следует отметить, что к достоинствам языка SQL относится наличие международных стандартов. Первый международный стандарт был принят в 1989 г., и соответствующая версия языка называется SQL-89. Этот стандарт полностью поддерживается практически во всех современных коммерческих реляционных СУБД (например, в Informix, Sybase, Ingres, DB2 и т.д.). Стандарт SQL-89 во многих частях имеет чрезвычайно общий характер и допускает очень широкое толкование. В этом стандарте полностью отсутствуют такие важные разделы, как манипулирование схемой БД и динамический SQL. Многие важные аспекты языка в соответствии со стандартом определяются в реализации.

Возможно, наиболее важными достижениями стандарта SQL-89 являются четкая стандартизация синтаксиса и семантики операторов выборки и манипулирования данными и фиксация средств ограничения целостности БД, включающих возможности определения первичного и внешних ключей отношений и так называемых проверочных ограничений целостности, позволяющих сформулировать условие для каждой отдельной строки таблицы. Средства определения внешних ключей позволяют легко формулировать требования так называемой целостности БД по ссылкам. Формулировка ограничений целостности на основе понятия внешнего ключа проста и понятна.

Осознавая неполноту стандарта SQL-89, на фоне завершения разработки этого стандарта специалисты различных фирм начали работу над стандартом SQL2. Эта работа также длилась много лет, было выпущено несколько проектов стандарта, пока, наконец, в марте 1992 г. не был выработан окончательный проект стандарта (после чего стандарт и соответствующий язык стали называть SQL-92). Этот стандарт существенно более полный и охватывает практически все необходимые для реализации аспекты: манипулирование схемой БД, управление транзакциями и сессиями (сессия - это последовательность транзакций, в пределах которой сохраняются временные отношения), подключение к БД, динамический SQL. Наконец стандартизованы отношения-каталоги БД, что вообще-то не связано с языком непосредственно, но очень сильно влияет на реализацию. Заметим, что в стандарте представлены три уровня языка - базовый, промежуточный и полный. В течение нескольких лет после принятия стандарта производители СУБД, утверждавшие совместимость своих продуктов со стандартом, на самом деле в лучшем случае поддерживали промежуточный уровень языка SQL-92 (естественно, с собственными расширениями). Только в последних выпусках СУБД ведущих производителей обеспечивается совместимость с полным вариантом языка. Наконец, одновременно с завершением работ по определению стандарта SQL-92 была начата разработка стандарта SQL3. Общей точкой зрения ведущих производителей СУБД является то, что будущие продукты, обладая более развитыми возможностями, должны быть совместимы с предыдущими выпусками. Хотя многие разработчики и пользователи реляционных СУБД осознают наличие многих неустранимых недостатков языка SQL, от него теперь уже невозможно отказаться (как невозможно отказаться от использования языка Си в процедурном программировании). Следовательно, нужен новый стандарт языка, обеспечивающий такие очевидно необходимые возможности как определяемые пользователями типы данных, более развитые средства определения таблиц, наличие полного механизма триггеров и т.д. Нужен именно стандарт, а не наличие развитых частных версий языка, поскольку это выгодно и производителям и пользователям СУБД.

**1 Понятие базы данных и СУБД**

## 1.1 Предметная область

Основным назначением информационных систем является хранение сведений об окружающем мире и процессах происходящих в нем, которые в конечном итоге предоставляются пользователям. Поскольку для различных групп людей интерес представляют только определенные части реального мира, то и данные каждой информационной системы будут относится к определенной области. Часть реальной системы, подлежащая исследованию с целью ее описания называется предметной областью.

Различают полную предметную область и ее фрагменты, при этом каждый фрагмент может представлять свою предметную область. Например, для университета можно выделить следующие фрагменты: учебный отдел, бухгалтерия, отдел кадров, бюро расписаний и т. д.

Информация, необходимая для описания предметной области, может включать сведения о людях, предметах, документах, событиях, понятиях и т.д.

Каждая предметная область характеризуется множеством объектов – элементов реальных систем и процессов, использующих объекты, а также множеством пользователей, характеризуемых единым взглядом на предметную область. В частности, для бухгалтерии объекты – всевозможные документы. Процессы бухгалтерии – расчет заработной платы, материальный учет, учет банковских операций и др. Наконец пользователи этого фрагмента сотрудники бухгалтерии, работники финансовых органов, руководители предприятия и т. д.

Каждый объект обладает определенным набором свойств, которые запоминаются в информационной системе. При обработке данных часто приходится иметь дело с совокупностью однородных объектов , например, таких, как студенты или факультеты, и записывать информацию об одних и тех же свойствах для каждого из них. Совокупность объектов, обладающих одинаковым набором свойств, называется классом объектов. Для объектов одного класса набор свойств будет одинаков, хотя значения этих свойств для каждого объекта могут быть разными.

Часто класс объектов называют сущностью. Каждая сущность обладает атрибутами. Атрибут – это свойство объекта, характеризующее его экземпляр. Сущность "студент" может иметь атрибуты "имя" , "год рождения", " дата поступления" и т. д. Таким образом сущность можно определить, как множество индивидуальных объектов одного типа (экземпляров), причем все эти объекты различны, т. е. набор атрибутов одинаков, а их значения различны.

## 1.2 Концепция баз данных

### 1.2.1 Независимость приложений от организации данных во внешней памяти

Основой любой информационной системы являются хранимые в ней данные. В общем случае данные – это информация, подготовленная для определенных целей, при этом часто подразумевается определенный формат представления.

Во все времена люди фиксировали данные на том или ином материальном носителе (бумага, камень и т. д.) Обычно данные фиксируются совместно с их интерпретацией (семантикой), так как системы письменности естественных языков позволяет это делать достаточно гибко. Например, запись на бумаге "Заработная плата – 1000" содержит данное – "1000" и его семантику (смысл) – "Заработная плата".

Довольно часто данные и их интерпретация бывают разделены. Они могут быть отражены в различных частях носителя (например, таблицы, в которых смысл записывается в верхних строках, а сами данные в последующих) и более того могут находиться на разных носителях. Такое разделение существенно затрудняет работу с данными.

Применение ЭВМ для обработки и хранения данных привело к еще большему разделению данных и интерпретирующей информации. Так как на ранних этапах развития вычислительной техники ее возможности были весьма ограничены, память использовалась только для хранения самих данных. Описание же данных (порядок, тип данных, длина) находились в программах, которые таким образом "знали", например, что с начиная определенного байта адресного пространства внешней памяти четыре байта содержат данное в числовом формате, связанное с заработной платой, а последующие 30 байтов в текстовом формате – содержат фамилию сотрудника.

Такая зависимость между данными и программой, существенно ограничивает возможности и эффективность информационных систем.

В первую очередь это проявляется, когда возникает необходимость в модификации информационной системы. Представьте себе ситуацию, когда к некоторому файлу обращаются несколько программ. В определенный момент времени в предметной области произошли изменения, которые в свою очередь потребовали изменения структуры записей файла, например добавление новых полей в запись файла или изменение длины некоторых полей. В этом случае придется переделывать все программы, даже если некоторым из них, для выполнения своих функций новые или измененные поля не требуются.

По этой причине часто для отдельных приложений создавались свои наборы данных, не смотря на то, что характер информации хранившейся в них был частично или полностью одинаковым. Такое многократное дублирование приводило или к несоответствию данных состоянию предметной области или к противоречию одних данных другим.

Современный подход требует, чтобы описание данных было независимым от программ пользователей, а некоторая система управления данными, которая используя эти описания, размещает их во внешней памяти и впоследствии "знает" где и как они хранятся, обеспечивала бы автоматический интерфейс между данными и приложениями. В этом случае становится возможным, в программах задавать только имена необходимых для обработки данных и форматы их представления, в связи с чем изменения в организации данных существенно не отражаются на прикладном программном обеспечении.

Описания данных часто хранятся вместе с самими данными и называются метаданными. В ряде современных систем метаданные, содержащие так же информацию о пользователях, права доступа, статистику обращения к данным и другие сведения, хранятся в словаре данных.

Такой подход позволяет манипулировать данными достаточно гибко и не требует значительных усилий при расширении и модификации информационных систем.

### 

### 1.2.2 Эффективность организации данных

Предположим, что мы хотим реализовать информационную систему, поддерживающую учет студентов университета. Система должна выполнять следующие действия: выдавать списки студентов по факультетам и группам, поддерживать возможность перевода студента из одной группы в другую, приема новых студентов и исключения учащихся. Для каждого факультета должна поддерживаться возможность получения имени декана этого факультета, а для группы имя куратора, общей численности факультета, и т.д.

Допустим, мы решили создавать эту информационную систему используя некоторую файловую систему, в которой пользователи представляют файл как последовательность записей. Каждая запись – это последовательность байтов постоянного или переменного размера. Записи можно читать или записывать последовательно, позиционировать файл на запись с указанным номером, структурировать на поля и объявлять некоторые поля ключами записи. Кроме того, имеется возможность выборки записи из файла по ее заданному ключу. Естественно, что в этом случае файловая система поддерживает в том же (или другом, служебном) базовом файле дополнительные, невидимые пользователю, служебные структуры данных. Распространенные способы организации ключевых файлов основываются на технике хеширования и B-деревьев.

Для хранения данных будем использовать один файл СТУДЕНТЫ. Поскольку объектом предметной области в нашем случае является студент, определяем, чтобы в этом файле содержалась одна запись для каждого студента. Исходя из требований к информационной системе, обозначим набор свойств, описывающий объект и отразим его следующими полями в файле:

ФИО\_СТУДЕНТА,

АДРЕС,

ДАТА\_РОЖДЕНИЯ,

НАИМНОВАНИЕ\_ФАКУЛЬТЕА,

ФИО\_ДЕКАНА,

НОМЕР\_ГРУППЫ,

ФИО\_КУРАТОРА.

Функции нашей информационной системы требуют, чтобы обеспечивалась возможность многоключевого доступа к записям этого файла по значениям уникального ключа ФИО\_СТУДЕНТА. Кроме того, должна обеспечиваться возможность выбора всех записей с общим значением НАИМЕНОВАНИЕ\_ФАКУЛЬТЕТА и НОМЕР\_ГРУППЫ т. е. доступ по неуникальному ключу.

Не трудно заметить, что такая организация данных вызывает существенную избыточность хранения данных (для каждого студента одной группы повторяется имя куратора, а для студентов одного факультета наименование факультета и имя декана). Кроме того, если в ходе эксплуатации системы на каком-либо факультете поменяется декан, придется вносить изменения в нескольких сотнях записей, а для того чтобы получить численность факультета, каждый раз при выполнении такой функции надо будет выбрать все записи с заданным значением наименования факультета и посчитать их количество.

Анализ предметной области показывает, что можно выделить еще два класса объектов ФАКУЛЬТЕТЫ и ГРУППЫ с присущими только им свойствами (для факультетов – деканы, а для групп – кураторы). Естественно предположить, что информация о каждом объекте должна находиться в отдельном файле. Поэтому наша систем будет состоять из трех файлов СТУДЕНТЫ, ФАКУЛЬТЕТЫ, ГРУППЫ, со следующей организацией записей:

СТУДЕНТЫ :

СТУД\_ФИО\_СТУДЕНТА,

СТУД\_АДРЕС,

СТУД\_ДАТА\_РОЖДЕНИЯ,

СТУД\_НОМЕР\_ГРУППЫ.

ГРУППЫ :

ГРУП\_НОМЕР\_ГРУППЫ,

ГРУП\_ФИО\_КУРАТОРА,

ГРУП\_ИДЕНТИФИКАТОР\_ФАКУЛЬТЕА,

ГРУП\_КОЛИЧЕСТВО\_СТУДЕНТОВ.

ФАКУЛЬТЕТЫ :

ФАК\_ИДЕНТИФИКАТОР\_ФАКУЛЬТЕА,

ФАК\_НАИМНОВАНИЕ\_ФАКУЛЬТЕА,

ФАК\_ФИО\_ДЕКАНА,

ФАК\_КОЛИЧЕСТВО\_СТУДЕНТОВ;

Поле СТУД\_НОМЕР\_ГРУППЫ добавлено в файл СТУДЕНТЫ, а ГРУП\_ИДЕНТИФИКАТОР\_ФАКУЛЬТЕТА в файл ГРУППЫ для установки связей между файлами. Так например, по значению поля ГРУП\_ИДЕНТИФИКАТОР\_ФАКУЛЬТЕТА в файле ГРУППЫ можно определить к какому факультету относится та или иная группа. Иначе говоря, один файл ссылается на другой, поля же, использующееся для ссылок называются ключами.

Такой механизм установки информационных связей между файлами называется механизмом дублирования ключей. Таким образом, каждый из файлов будет содержать только не дублированную информацию, необходимости в динамических вычислениях суммарной информации не возникает.

Однако теперь система должна теперь "знать", что она работает с тремя информационно связанными файлами, должна знать структуру и смысл каждого поля (например, что ГРУП\_ ДЕНТИФИКАТОР\_ ФАКУЛЬТЕТА и ФАК\_ ИДЕНТИФИКАТОР\_ФАКУЛЬТЕТА означают одно и то же), а также понимать, что в ряде случаев изменение информации в одном файле должно автоматически вызывать модификацию в других, чтобы их общее содержимое было согласованным.

Например, если на факультет принимается новый студент, то необходимо добавить запись в файл СТУДЕНТ, а также соответствующим образом изменить поля ФАК\_КОЛИЧЕСТВО\_СТУДЕНТОВ и ГРУП\_КОЛИЧЕСТВО\_СТУДЕНТОВ в файлах ФАКУЛЬТЕТЫ и ГРУППЫ. Иначе говоря данные во всех файлах должны быть согласованными.

Понятие согласованности данных является ключевым понятием баз данных. Фактически, если информационная система (даже такая простая, как в нашем примере) поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает базу данных.

Понятно, что даже для такой простой системы ее реализация традиционными методами программирования, требует от разработчиков создания достаточно сложных процедур, которые связывают отдельные файлы в единое целое.

Для поддержания согласованности данных в нескольких файлах недостаточно библиотеки процедур, необходимо, чтобы такая система имела некоторые собственные данные (метаданные) и даже знания, определяющие целостность данных.

Если некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее системой управления базами данных.

### 1.2.3 Интеграция данных

В информационных системах реализованных на множестве локальных приложений, каждое из которых использует свой набор данных, имеет место избыточность (дублирование) данных. Вследствие этого синхронное поддержание данных для всех приложений становится весьма проблематичным, что в может привести к неправильному отражению состояния предметной области и вызвать весьма неприятные последствия.

Рассмотрим следующий пример для двух приложений "Учет кадров" и "Расчет заработной платы" для которых необходимы следующие сведения:

|  |  |
| --- | --- |
| Учет кадров | Расчет заработной платы |
| Табельный номер | Табельный номер |
| Ф. И. О. Сотрудника | Ф. И. О. Сотрудника |
| Отдел | Отдел |
| Должность | Должностной оклад |
| Стаж работы | Стаж работы |
| Год рождения |  |

По правилам предметной области первоначально информация о переводе сотрудника на новую должность или в другой отдел, а так же об увольнении поступает в отдел кадров. Если же по каким-либо причинам об этом не будет своевременно известно в бухгалтерии, то в лучшем случае фамилия этого сотрудника окажется не в той ведомости или же возникнет совсем курьезное положение, когда уволенному сотруднику будет начислена заработная плата.

Поэтому существует необходимость в создании системы с единым хранилищем данных, в которой все данные накапливаются и хранятся централизованно, а информация внесенная одним пользователем становится доступной для других. В памяти ЭВМ должна быть создана динамически обновляемая модель предметной области это значит, что соответствие базы данных текущему состоянию предметной области обеспечивается не периодически, а в режиме реального времени. Благодаря интеграции устраняется дублирование данных, повышается уровень их достоверности, упрощаются и становятся эффективнее процедуры обновления данных. Интеграция данных порождает ряд проблем, которые мы рассмотрим ниже.

### 

### 1.2.4 Что такое база данных

В общем случае понятие базы данных можно определить как совокупность файлов или файл, состоящий из некоторого числа записей, каждая из которых формируется из полей определенного типа, вместе с набором операций поиска, сортировки, рекомбинации и др.

Однако традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем. Существует несколько потребностей, которые не покрываются возможностями систем управления файлами:

поддержание логически согласованного набора файлов;

обеспечение языка манипулирования данными;

восстановление информации после разного рода сбоев;

параллельная работа в режиме реального времени нескольких пользователей.

Можно считать, что если прикладная информационная система опирается на некоторую систему управления данными, обладающую этими свойствами, то эта система управления данными является системой управления базами данных. Таким образом, база данных – это совокупность взаимосвязанных данных, используемых несколькими приложениями под управлением СУБД. Система управления базой данных – система программного обеспечения, имеющая средства обработки на языке базы данных, позволяющие обрабатывать обращения к базе данных, которые поступают от прикладных программ и (или) конечных пользователей, и поддерживать целостность базы данных.

**2. Типы данных SQL**

В SQL используются следующие основные типы данных, форматы которых могут несколько различаться для разных СУБД:

INTEGER

- целое число (обычно до 10 значащих цифр и знак);

SMALLINT

- "короткое целое" (обычно до 5 значащих цифр и знак);

DECIMAL(p,q)

- десятичное число, имеющее p цифр (0 < p < 16) и знак; с помощью q задается число цифр справа от десятичной точки (q < p, если q = 0, оно может быть опущено);

FLOAT

- вещественное число с 15 значащими цифрами и целочисленным порядком, определяемым типом СУБД;

CHAR(n)

- символьная строка фиксированной длины из n символов (0 < n < 256);

VARCHAR(n)

- символьная строка переменной длины, не превышающей n символов (n > 0 и разное в разных СУБД, но не меньше 4096);

DATE

- дата в формате, определяемом специальной командой (по умолчанию mm/dd/yy); поля даты могут содержать только реальные даты, начинающиеся за несколько тысячелетий до н.э. и ограниченные пятым-десятым тысячелетием н.э.;

TIME

- время в формате, определяемом специальной командой, (по умолчанию hh.mm.ss);

DATETIME

- комбинация даты и времени;

MONEY

- деньги в формате, определяющем символ денежной единицы ($, руб, ...) и его расположение (суффикс или префикс), точность дробной части и условие для показа денежного значения.

В некоторых СУБД еще существует тип данных LOGICAL, DOUBLE и ряд других. СУБД INGRES предоставляет пользователю возможность самостоятельного определения новых типов данных, например, плоскостные или пространственные координаты, единицы различных метрик, пяти- или шестидневные недели (рабочая неделя, где сразу после пятницы или субботы следует понедельник), дроби, графика, большие целые числа (что стало очень актуальным для российских банков) и т.п.

# 

# 2.1 Таблицы SQL

До сих пор понятие "таблица", как правило, связывалось с реальной или базовой таблицей, т.е. c таблицей, для каждой строки которой в действительности имеется некоторый двойник, хранящийся в физической памяти машины (Приложение А). Однако SQL использует и создает ряд виртуальных (как будто существующих) таблиц: представлений, курсоров и неименованных рабочих таблиц, в которых формируются результаты запросов на получение данных из базовых таблиц и, возможно, представлений. Это таблицы, которые не существуют в базе данных, но как бы существуют с точки зрения пользователя.

Базовые таблицы создаются с помощью предложения CREATE TABLE (создать таблицу). Здесь же приведем пример предложения для создания описания таблицы Блюда:

CREATE TABLE Блюда

(БЛ SMALLINT,

Блюдо CHAR (70),

В CHAR (1),

Основа CHAR (10),

Выход FLOAT,

Труд SMALLINT);

Предложение CREAT TABLE специфицирует имя базовой таблицы, которая должна быть создана, имена ее столбцов и типы данных для этих столбцов (а также, возможно, некоторую дополнительную информацию, не иллюстрируемую данным примером). CREAT TABLE - выполняемое предложение. Если его ввести с терминала, система тотчас построит таблицу Блюда, которая сначала будет пустой: она будет содержать только строку заголовков столбцов, но не будет еще содержать никаких строк с данными.

# 

# 2.2 Структура языка SQL

Нужно заметить, что в настоящее время, ни одна система не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект - это надмножество некоторого подмножества стандарта SQL. Это затрудняет переносимость приложений, разработанных для одних СУБД в другие СУБД.

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее. Например, отношение в реляционной модели данных не допускает наличия одинаковых кортежей, а таблицы в терминологии SQL могут иметь одинаковые строки. Имеются и другие отличия.

Язык SQL является реляционно-полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором SQL.

# 2.3 Операторы SQL

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям.

Можно выделить следующие группы операторов (перечислены не все операторы SQL):

Операторы DDL (Data Definition Language) - операторы определения объектов базы данных

CREATE SCHEMA - создать схему базы данных

DROP SHEMA - удалить схему базы данных

CREATE TABLE - создать таблицу

ALTER TABLE - изменить таблицу

DROP TABLE - удалить таблицу

CREATE DOMAIN - создать домен

ALTER DOMAIN - изменить домен

DROP DOMAIN - удалить домен

CREATE COLLATION - создать последовательность

DROP COLLATION - удалить последовательность

CREATE VIEW - создать представление

DROP VIEW - удалить представление

Операторы DML (Data Manipulation Language) - операторы манипулирования данными

SELECT - отобрать строки из таблиц

INSERT - добавить строки в таблицу

UPDATE - изменить строки в таблице

DELETE - удалить строки в таблице

COMMIT - зафиксировать внесенные изменения

ROLLBACK - откатить внесенные изменения

Операторы защиты и управления данными

CREATE ASSERTION - создать ограничение

DROP ASSERTION - удалить ограничение

GRANT - предоставить привилегии пользователю или приложению на манипулирование объектами

REVOKE - отменить привилегии пользователя или приложения

Кроме того, есть группы операторов установки параметров сеанса, получения информации о базе данных, операторы статического SQL, операторы динамического SQL.

Наиболее важными для пользователя являются операторы манипулирования данными (DML).

Примеры использования операторов манипулирования данными

INSERT - вставка строк в таблицу

Пример 1. Вставка одной строки в таблицу:

INSERT INTO

P (PNUM, PNAME)

VALUES (4, "Иванов");

Оператор SELECT является фактически самым важным для пользователя и самым сложным оператором SQL. Он предназначен для выборки данных из таблиц, т.е. он, собственно, и реализует одно их основных назначение базы данных - предоставлять информацию пользователю.

Оператор SELECT всегда выполняется над некоторыми таблицами, входящими в базу данных.

На самом деле в базах данных могут быть не только постоянно хранимые таблицы, а также временные таблицы и так называемые представления. Представления - это просто хранящиеся в базе данные SELECT-выражения. С точки зрения пользователей представления - это таблица, которая не хранится постоянно в базе данных, а "возникает" в момент обращения к ней. С точки зрения оператора SELECT и постоянно хранимые таблицы, и временные таблицы и представления выглядят совершенно одинаково. Конечно, при реальном выполнении оператора SELECT системой учитываются различия между хранимыми таблицами и представлениями, но эти различия скрыты от пользователя.

Результатом выполнения оператора SELECT всегда является таблица. Таким образом, по результатам действий оператор SELECT похож на операторы реляционной алгебры. Любой оператор реляционной алгебры может быть выражен подходящим образом сформулированным оператором SELECT. Сложность оператора SELECT определяется тем, что он содержит в себе все возможности реляционной алгебры, а также дополнительные возможности, которых в реляционной алгебре нет.

Отбор данных из одной таблицы

Пример. Выбрать все данные из таблицы поставщиков (ключевые слова SELECT… FROM…):

SELECT \*

FROM P;

Иногда приходится выполнять запросы, в которых таблица соединяется сама с собой, или одна таблица соединяется дважды с другой таблицей. При этом используются имена корреляции (алиасы, псевдонимы), которые позволяют различать соединяемые копии таблиц. Имена корреляции вводятся в разделе FROM и идут через пробел после имени таблицы. Имена корреляции должны использоваться в качестве префикса перед именем столбца и отделяются от имени столбца точкой. Если в запросе указываются одни и те же поля из разных экземпляров одной таблицы, они должны быть переименованы для устранения неоднозначности в именованиях колонок результатирующей таблицы. Определение имени корреляции действует только во время выполнения запроса.

Пример. Отобрать все пары поставщиков таким образом, чтобы первый поставщик в паре имел статус, больший статуса второго поставщика:

SELECT

P1.PNAME AS PNAME1,

P1.PSTATUS AS PSTATUS1,

P2.PNAME AS PNAME2,

P2.PSTATUS AS PSTATUS2

FROM

P P1, P P2

WHERE P1.PSTATUS1 > P2.PSTATUS2;

Опишем синтаксис оператора выборки данных (оператора SELECT) более точно. При описании синтаксиса операторов обычно используются условные обозначения, известные как стандартные формы Бэкуса-Наура (BNF).

В BNF обозначениях используются следующие элементы:

Символ "::=" означает равенство по определению. Слева от знака стоит определяемое понятие, справа - собственно определение понятия.

Ключевые слова записываются прописными буквами. Они зарезервированы и составляют часть оператора.

Метки-заполнители конкретных значений элементов и переменных записываются курсивом.

Необязательные элементы оператора заключены в квадратные скобки.

Вертикальная черта | указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка после этой черты.

Фигурные скобки {} указывают на то, что все находящееся внутри них является единым целым.

Троеточие "…" означает, что предшествующая часть оператора может быть повторена любое количество раз.

Многоточие, внутри которого находится запятая ".,.." указывает, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений. Запятую нельзя ставить после последнего элемента. Замечание: данное соглашение не входит в стандарт BNF, но позволяет более точно описать синтаксис операторов SQL.

Круглые скобки являются элементом оператора.

Синтаксис оператора выборки

В довольно сильно упрощенном виде оператор выборки данных имеет следующий синтаксис (для некоторых элементов мы дадим не BNF-определения, а словесное описание):

Оператор выборки ::=

Табличное выражение

[ORDER BY

{{Имя столбца-результата [ASC | DESC]} | {Положительное целое [ASC | DESC]}}.,..];

Табличное выражение ::=

Select-выражение

[

{UNION | INTERSECT | EXCEPT} [ALL]

{Select-выражение | TABLE Имя таблицы | Конструктор значений таблицы}

]

Select-выражение ::=

SELECT [ALL | DISTINCT]

{{{Скалярное выражение | Функция агрегирования | Select-выражение} [AS Имя столбца]}.,..}

| {{Имя таблицы|Имя корреляции}.\*}

| \*

FROM {

{Имя таблицы [AS] [Имя корреляции] [(Имя столбца.,..)]}

| {Select-выражение [AS] Имя корреляции [(Имя столбца.,..)]}

| Соединенная таблица }.,..

[WHERE Условное выражение]

[GROUP BY {[{Имя таблицы|Имя корреляции}.]Имя столбца}.,..]

[HAVING Условное выражение]

Select-выражение в разделе SELECT, используемое в качестве значения для отбираемого столбца, должно возвращать таблицу, состоящую из одной строки и одного столбца, т.е. скалярное выражение. Условное выражение в разделе WHERE должно вычисляться для каждой строки, являющейся кандидатом в результатирующее множество строк. В этом условном выражении можно использовать подзапросы. Синтаксис условных выражений, допустимых в разделе WHERE рассматривается ниже.

Раздел HAVING содержит условное выражение, вычисляемое для каждой группы, определяемой списком группировки в разделе GROUP BY. Это условное выражение может содержать функции агрегирования, вычисляемые для каждой группы. Условное выражение, сформулированное в разделе WHERE, может быть перенесено в раздел HAVING. Перенос условий из раздела HAVING в раздел WHERE невозможен, если условное выражение содержит агрегатные функции. Перенос условий из раздела WHERE в раздел HAVING является плохим стилем программирования - эти разделы предназначены для различных по смыслу условий (условия для строк и условия для групп строк). Если в разделе SELECT присутствуют агрегатные функции, то они вычисляются по-разному в зависимости от наличия раздела GROUP BY. Если раздел GROUP BY отсутствует, то результат запроса возвращает не более одной строки. Агрегатные функции вычисляются по всем строкам, удовлетворяющим условному выражению в разделе WHERE. Если раздел GROUP BY присутствует, то агрегатные функции вычисляются по отдельности для каждой группы, определенной в разделе GROUP BY.

Скалярное выражение - в качестве скалярных выражений в разделе SELECT могут выступать либо имена столбцов таблиц, входящих в раздел FROM, либо простые функции, возвращающие скалярные значения.

Функция агрегирования ::=

COUNT (\*) |

{

{COUNT | MAX | MIN | SUM | AVG} ([ALL | DISTINCT] Скалярное выражение)

}

Конструктор значений таблицы ::=

VALUES Конструктор значений строки.,..

Конструктор значений строки ::=

Элемент конструктора | (Элемент конструктора.,..) | Select-выражение

Замечание. Select-выражение, используемое в конструкторе значений строки, обязано возвращать ровно одну строку.

Элемент конструктора ::=

Выражение для вычисления значения | NULL | DEFAULT

Синтаксис соединенных таблиц

В разделе FROM оператора SELECT можно использовать соединенные таблицы. Пусть в результате некоторых операций мы получаем таблицы A и B. Такими операциями могут быть, например, оператор SELECT или другая соединенная таблица. Тогда синтаксис соединенной таблицы имеет следующий вид:

Соединенная таблица ::=

Перекрестное соединение

| Естественное соединение

| Соединение посредством предиката

| Соединение посредством имен столбцов

| Соединение объединения

Тип соединения ::=

INNER

| LEFT [OUTER]

| RIGTH [OUTER]

| FULL [OUTER]

Перекрестное соединение ::=

Таблица А CROSS JOIN Таблица В

Естественное соединение ::=

Таблица А [NATURAL] [Тип соединения] JOIN Таблица В

Соединение посредством предиката ::=

Таблица А [Тип соединения] JOIN Таблица В ON Предикат

Соединение посредством имен столбцов ::=

Таблица А [Тип соединения] JOIN Таблица В USING (Имя столбца.,..)

Соединение объединения ::=

Таблица А UNION JOIN Таблица В

Опишем используемые термины.

CROSS JOIN - Перекрестное соединение возвращает просто декартово произведение таблиц. Такое соединение в разделе FROM может быть заменено списком таблиц через запятую.

NATURAL JOIN - Естественное соединение производится по всем столбцам таблиц А и В, имеющим одинаковые имена. В результатирующую таблицу одинаковые столбцы вставляются только один раз.

JOIN … ON - Соединение посредством предиката соединяет строки таблиц А и В посредством указанного предиката.

JOIN … USING - Соединение посредством имен столбцов соединяет отношения подобно естественному соединению по тем общим столбцам таблиц А и Б, которые указаны в списке USING.

OUTER - Ключевое слово OUTER (внешний) не является обязательными, оно не используется ни в каких операциях с данными.

INNER - Тип соединения "внутреннее". Внутренний тип соединения используется по умолчанию, когда тип явно не задан. В таблицах А и В соединяются только те строки, для которых найдено совпадение.

LEFT (OUTER) - Тип соединения "левое (внешнее)". Левое соединение таблиц А и В включает в себя все строки из левой таблицы А и те строки из правой таблицы В, для которых обнаружено совпадение. Для строк из таблицы А, для которых не найдено соответствия в таблице В, в столбцы, извлекаемые из таблицы В, заносятся значения NULL.

RIGHT (OUTER) - Тип соединения "правое (внешнее)". Правое соединение таблиц А и В включает в себя все строки из правой таблицы В и те строки из левой таблицы А, для которых обнаружено совпадение. Для строк из таблицы В, для которых не найдено соответствия в таблице А, в столбцы, извлекаемые из таблицы А заносятся значения NULL.

FULL (OUTER) - Тип соединения "полное (внешнее)". Это комбинация левого и правого соединений. В полное соединение включаются все строки из обеих таблиц. Для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений.

UNION JOIN - Соединение объединения является обратным по отношению к внутреннему соединению. Оно включает только те строки из таблиц А и В, для которых не найдено совпадений. В них используются значения NULL для столбцов, полученных из другой таблицы. Если взять полное внешнее соединение и удалить из него строки, полученные в результате внутреннего соединения, то получится соединение объединения.

Использование соединенных таблиц часто облегчает восприятие оператора SELECT, особенно, когда используется естественное соединение. Если не использовать соединенные таблицы, то при выборе данных из нескольких таблиц необходимо явно указывать условия соединения в разделе WHERE. Если при этом пользователь указывает сложные критерии отбора строк, то в разделе WHERE смешиваются семантически различные понятия - как условия связи таблиц, так и условия отбора строк (см. примеры 13, 14, 15 данной главы).

Синтаксис условных выражений раздела WHERE

Условное выражение, используемое в разделе WHERE оператора SELECT должно вычисляться для каждой строки-кандидата, отбираемой оператором SELECT. Условное выражение может возвращать одно из трех значений истинности: TRUE, FALSE или UNKNOUN. Строка-кандидат отбирается в результатирующее множество строк только в том случае, если для нее условное выражение вернуло значение TRUE.

Условные выражения имеют следующий синтаксис (в целях упрощения изложения приведены не все возможные предикаты):

Условное выражение ::=

[ ( ] [NOT]

{Предикат сравнения

| Предикат between

| Предикат in

| Предикат like

| Предикат null

| Предикат количественного сравнения

| Предикат exist

| Предикат unique

| Предикат match

| Предикат overlaps}

[{AND | OR} Условное выражение] [ ) ]

[IS [NOT] {TRUE | FALSE | UNKNOWN}]

Предикат сравнения ::=

Конструктор значений строки {= | < | > | <= | >= | <>} Конструктор значений строки

Предикат LIKE производит поиск строки-поиска в строке-шаблоне. В строке-шаблоне разрешается использовать два трафаретных символа:

Символ подчеркивания "\_" может использоваться вместо любого единичного символа в строке-поиска,

Символ процента "%" может заменять набор любых символов в строке поиска (число символов в наборе может быть от 0 и более).

Предикат null ::=

Конструктор значений строки IS [NOT] NULL Предикат NULL применяется специально для проверки, не равно ли проверяемое выражение null-значению.

Предикат количественного сравнения ::=

Конструктор значений строки {= | < | > | <= | >= | <>}

{ANY | SOME | ALL} (Select-выражение)

Кванторы ANY и SOME являются синонимами и полностью взаимозаменяемы.

Замечание. Если указан один из кванторов ANY и SOME, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает хотя бы с одним значением, возвращаемом в подзапросе (select-выражении). Если указан квантор ALL, то предикат количественного сравнения возвращает TRUE, если сравниваемое значение совпадает с каждым значением, возвращаемом в подзапросе (select-выражении).

Предикат EXIST возвращает значение TRUE, если результат подзапроса (select-выражения) не пуст.

Предикат unique ::=

UNIQUE (Select-выражение)

Предикат UNIQUE возвращает TRUE, если в результате подзапроса (select-выражения) нет совпадающих строк.

Предикат match ::=

Конструктор значений строки MATCH [UNIQUE]

[PARTIAL | FULL] (Select-выражение)

Предикат MATCH проверяет, будет ли значение, определенное в конструкторе строки совпадать со значением любой строки, полученной в результате подзапроса.

Предикат overlaps ::=

Конструктор значений строки OVERLAPS Конструктор значений строки

Предикат OVERLAPS, является специализированным предикатом, позволяющем определить, будет ли указанный период времени перекрывать другой период времени.

Порядок выполнения оператора SELECT

Для того чтобы понять, как получается результат выполнения оператора SELECT, рассмотрим концептуальную схему его выполнения. Эта схема является именно концептуальной, т.к. гарантируется, что результат будет таким, как если бы он выполнялся шаг за шагом в соответствии с этой схемой. На самом деле, реально результат получается более изощренными алгоритмами, которыми "владеет" конкретная СУБД.

Стадия 1. Выполнение одиночного оператора SELECT

Если в операторе присутствуют ключевые слова UNION, EXCEPT и INTERSECT, то запрос разбивается на несколько независимых запросов, каждый из которых выполняется отдельно:

Шаг 1 (FROM). Вычисляется прямое декартовое произведение всех таблиц, указанных в обязательном разделе FROM. В результате шага 1 получаем таблицу A.

Шаг 2 (WHERE). Если в операторе SELECT присутствует раздел WHERE, то сканируется таблица A, полученная при выполнении шага 1. При этом для каждой строки из таблицы A вычисляется условное выражение, приведенное в разделе WHERE. Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат. Если раздел WHERE опущен, то сразу переходим к шагу 3. Если в условном выражении участвуют вложенные подзапросы, то они вычисляются в соответствии с данной концептуальной схемой. В результате шага 2 получаем таблицу B.

Шаг 3 (GROUP BY). Если в операторе SELECT присутствует раздел GROUP BY, то строки таблицы B, полученной на втором шаге, группируются в соответствии со списком группировки, приведенным в разделе GROUP BY. Если раздел GROUP BY опущен, то сразу переходим к шагу 4. В результате шага 3 получаем таблицу С.

Шаг 4 (HAVING). Если в операторе SELECT присутствует раздел HAVING, то группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются. Если раздел HAVING опущен, то сразу переходим к шагу 5. В результате шага 4 получаем таблицу D.

Шаг 5 (SELECT). Каждая группа, полученная на шаге 4, генерирует одну строку результата следующим образом. Вычисляются все скалярные выражения, указанные в разделе SELECT. По правилам использования раздела GROUP BY, такие скалярные выражения должны быть одинаковыми для всех строк внутри каждой группы. Для каждой группы вычисляются значения агрегатных функций, приведенных в разделе SELECT. Если раздел GROUP BY отсутствовал, но в разделе SELECT есть агрегатные функции, то считается, что имеется всего одна группа. Если нет ни раздела GROUP BY, ни агрегатных функций, то считается, что имеется столько групп, сколько строк отобрано к данному моменту. В результате шага 5 получаем таблицу E, содержащую столько колонок, сколько элементов приведено в разделе SELECT и столько строк, сколько отобрано групп.

Стадия 2. Выполнение операций UNION, EXCEPT, INTERSECT

Если в операторе SELECT присутствовали ключевые слова UNION, EXCEPT и INTERSECT, то таблицы, полученные в результате выполнения 1-й стадии, объединяются, вычитаются или пересекаются.

Стадия 3. Упорядочение результата

Если в операторе SELECT присутствует раздел ORDER BY, то строки полученной на предыдущих шагах таблицы упорядочиваются в соответствии со списком упорядочения, приведенном в разделе ORDER BY.

Как на самом деле выполняется оператор SELECT

Если внимательно рассмотреть приведенный выше концептуальный алгоритм вычисления результата оператора SELECT, то сразу понятно, что выполнять его непосредственно в таком виде чрезвычайно накладно. Даже на самом первом шаге, когда вычисляется декартово произведение таблиц, приведенных в разделе FROM, может получиться таблица огромных размеров, причем практически большинство строк и колонок из нее будет отброшено на следующих шагах.

На самом деле в РСУБД имеется оптимизатор, функцией которого является нахождение такого оптимального алгоритма выполнения запроса, который гарантирует получение правильного результата.

Схематично работу оптимизатора можно представить в виде последовательности нескольких шагов:

Шаг 1 (Синтаксический анализ). Поступивший запрос подвергается синтаксическому анализу. На этом шаге определяется, правильно ли вообще (с точки зрения синтаксиса SQL) сформулирован запрос. В ходе синтаксического анализа вырабатывается некоторое внутренне представление запроса, используемое на последующих шагах.

Шаг 2 (Преобразование в каноническую форму). Запрос во внутреннем представлении подвергается преобразованию в некоторую каноническую форму. При преобразовании к канонической форме используются как синтаксические, так и семантические преобразования. Синтаксические преобразования (например, приведения логических выражений к конъюнктивной или дизъюнктивной нормальной форме, замена выражений "x AND NOT x" на "FALSE", и т.п.) позволяют получить новое внутренне представление запроса, синтаксически эквивалентное исходному, но стандартное в некотором смысле. Семантические преобразования используют дополнительные знания, которыми владеет система, например, ограничения целостности. В результате семантических преобразований получается запрос, синтаксически не эквивалентный исходному, но дающий тот же самый результат.

Шаг 3 (Генерация планов выполнения запроса и выбор оптимального плана). На этом шаге оптимизатор генерирует множество возможных планов выполнения запроса. Каждый план строится как комбинация низкоуровневых процедур доступа к данным из таблиц, методам соединения таблиц. Из всех сгенерированных планов выбирается план, обладающий минимальной стоимостью. При этом анализируются данные о наличии индексов у таблиц, статистических данных о распределении значений в таблицах, и т.п. Стоимость плана это, как правило, сумма стоимостей выполнения отдельных низкоуровневых процедур, которые используются для его выполнения. В стоимость выполнения отдельной процедуры могут входить оценки количества обращений к дискам, степень загруженности процессора и другие параметры.

Шаг 4. (Выполнение плана запроса). На этом шаге план, выбранный на предыдущем шаге, передается на реальное выполнение.

Во многом качество конкретной СУБД определяется качеством ее оптимизатора. Хороший оптимизатор может повысить скорость выполнения запроса на несколько порядков. Качество оптимизатора определяется тем, какие методы преобразований он может использовать, какой статистической и иной информацией о таблицах он располагает, какие методы для оценки стоимости выполнения плана он знает.

Реализация реляционной алгебры средствами оператора SELECT (Реляционная полнота SQL)

Для того, чтобы показать, что язык SQL является реляционно полным, нужно показать, что любой реляционный оператор может быть выражен средствами SQL. На самом деле достаточно показать, что средствами SQL можно выразить любой из примитивных реляционных операторов.

Оператор декартового произведения

Реляционная алгебра:



Оператор SQL:

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …

FROM A, B;

или

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …

FROM A CROSS JOIN B;

Оператор проекции

Реляционная алгебра:



Оператор SQL:

SELECT DISTINCT X, Y, …, Z

FROM A;

Оператор выборки

Реляционная алгебра: ,



Оператор SQL:

SELECT \*

FROM A

WHERE c;

Оператор объединения

Реляционная алгебра:



Оператор SQL:

SELECT \*

FROM A

UNION

SELECT \*

FROM B;

Оператор вычитания

Реляционная алгебра:



Оператор SQL:

SELECT \*

FROM A

EXCEPT

SELECT \*

FROM B

Реляционный оператор переименования RENAME выражается при помощи ключевого слова AS в списке отбираемых полей оператора SELECT. Таким образом, язык SQL является реляционно-полным.

Остальные операторы реляционной алгебры (соединение, пересечение, деление) выражаются через примитивные, следовательно, могут быть выражены операторами SQL. Тем не менее, для практических целей приведем их.

Оператор соединения

Реляционная алгебра:



Оператор SQL:

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …

FROM A, B

WHERE c;

или

SELECT A.Поле1, A.Поле2, …, B.Поле1, B.Поле2, …

FROM A CROSS JOIN B

WHERE c;

Оператор пересечения

Реляционная алгебра:



Оператор SQL:

SELECT \*

FROM A

INTERSECT

SELECT \*

FROM B;

Оператор деления

Реляционная алгебра:



Оператор SQL:

SELECT DISTINCT A.X

FROM A

WHERE NOT EXIST

(SELECT \*

FROM B

WHERE NOT EXIST

(SELECT \*

FROM A A1

WHERE

A1.X = A.X AND

A1.Y = B.Y));

Пусть отношение A содержит данные о поставках деталей, отношение B содержит список всех деталей, которые могут поставляться. Атрибут X является номером поставщика, атрибут Y является номером детали.

Разделить отношение A на отношение B означает в данном примере "отобрать номера поставщиков, которые поставляют все детали".

Преобразуем текст выражения:

"Отобрать номера поставщиков, которые поставляют все детали" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых не существует непоставляемых деталей в таблице B" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых не существует тех номеров деталей из таблицы B, которые не поставляются этим поставщиком" эквивалентно

"Отобрать те номера поставщиков из таблицы A, для которых не существует тех номеров деталей из таблицы B, для которых не существует записей о поставках в таблице A для этого поставщика и этой детали".

Последнее выражение дословно переводится на язык SQL. При переводе выражения на язык SQL нужно учесть, что во внутреннем подзапросе таблица A должна быть переименована, для того чтобы отличать ее от экземпляра этой же таблицы, используемой во внешнем запросе.

# Заключение

Фактически стандартным языком доступа к базам данных в настоящее время стал язык SQL (Structured Query Language).

Язык SQL оперирует терминами, несколько отличающимися от терминов реляционной теории, например, вместо "отношений" используются "таблицы", вместо "кортежей" - "строки", вместо "атрибутов" - "колонки" или "столбцы".

Стандарт языка SQL, хотя и основан на реляционной теории, но во многих местах отходит он нее.

Основу языка SQL составляют операторы, условно разбитые не несколько групп по выполняемым функциям:

Операторы DDL (Data Definition Language) - операторы определения объектов базы данных.

Операторы DML (Data Manipulation Language) - операторы манипулирования данными.

Операторы защиты и управления данными, и др.

Одним из основных операторов DML является оператор SELECT, позволяющий извлекать данные из таблиц и получать ответы на различные запросы. Оператор SELECT содержит в себе все возможности реляционной алгебры. Это означает, что любой оператор реляционной алгебры может быть выражен при помощи подходящего оператора SELECT. Этим доказывается реляционная полнота языка SQL.

Различают концептуальную схему выполнения оператора SELECT и фактическую схему его выполнения. Концептуальная схема описывает, в какой логической последовательности должны выполняться операции, чтобы получить результат. При реальном выполнении оператора SELECT на первый план выступает достижение максимальной скорости выполнения запроса. Для этого используется оптимизатор, который, анализируя различные планы выполнения запроса, выбирает наилучший из них.

**Глоссарий**

|  |  |  |
| --- | --- | --- |
| №  п/п | Понятие | Определение |
| 1 | Информационная система | система, реализующая автоматизированный сбор, обработку и манипулирование данными и включающая технические средства обработки данных, программное обеспечение и соответствующий персонал |
| 2 | База данных (БД) | поименованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области |
| 3 | Объект | элемент предметной области, информацию о котором мы сохраняем |
| 4 | Поле | элементарная единица логической организации данных, которая соответствует неделимой единице информации - реквизиту |
| 5 | Запись | совокупность логически связанных полей |
| 6 | Файл (таблица) | совокупность экземпляров записей одной структуры |
| 7 | Модель данных | совокупность структур данных и операций их обработки |
| 8 | Реляционная модель данных | совокупность взаимосвязанных двумерных таблиц - объектов модели |
| 9 | Атрибут | поименованная характеристика объекта. Атрибут показывает, какая информация должна быть собрана об объекте |
| 10 | Связи | соответствия, отношения, возникающие между объектами предметной области |
| 11 | Сущность | основное содержание объекта предметной области, о котором собирают информацию. В качестве сущности могут выступать место, вещь, личность, явление |
| 12 | Конструктор (Builder) | инструмент Access, который облегчает выполнение конкретного задания |

**Список использованных источников**

1. Дейт К.Дж. Введение в системы баз данных. 6-е изд. - М.: Вильямс. 2000. – 317 с.
2. Конноли Т., Бегг Л., Страчан А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. Вильямс 2003. – Таблицы, картинки. Леонтьев В.П. ПК: универсальный справочник пользователя - М.: 2003. – 251 с.
3. Дейт К. Введение в системы баз данных. 6-е изд. – М.: Вильямс, 2000. – 657с
4. В.В. Фаронов Основы программирования в SQL. - М.: Издатель Молгачева С.В., 2002. – 329 с.
5. Самоучитель по языку SQL (SQL DML) [Электронный ресурс] – Режим доступа http://www.sql-ex.ru/help
6. Коннолли Т., Бегг К., Страчан А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 2-е изд. – С-Пб.: Вильямс, 2000. – 1120 с.
7. Корнеев В.В., Гареев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. 2-е изд. – М.: Изд. Молгачева С.В., 2001. – 494 с.
8. Мамаев Е. Microsoft SQL Server 2000 – СПБ.: БХВ-Петербург, 2002.

# Когаловский М.Р. Энциклопедия технологий баз данных. – М.: Финансы и статистика, 2002.

# Приложение А

База данных в восприятии пользователя

