1.1.Структурированный язык запросов SQL

1.1.1. Общие положения

Изложенные выше механизмы РМД легли в основу языков манипулирования данными. Заметим, что крайне редко РА или РИ принимаются в качестве полной основы какого-либо языка РБД. Обычно язык основывается на некоторой смеси алгебраических и логических конструкций.

Реализация концепции операций, ориентированных на табличное представление данных, позволило создать компактный язык с небольшим набором предложений – SQL. Этот язык может использоваться как интерактивный для выполнения запросов и как встроенный для построения прикладных программ.

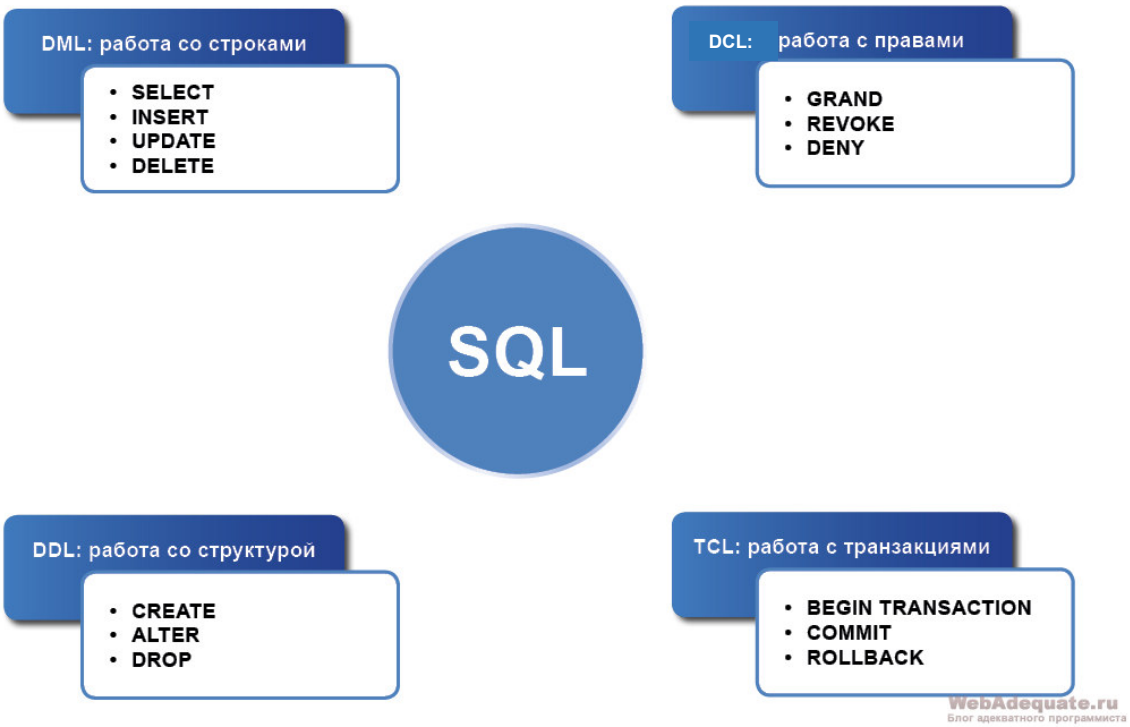
В современных СУБД обычно поддерживается единый интегрированный язык SQL, содержащий разнообразные средства для обеспечения базового пользовательского интерфейса при работе с БД. В нем можно выделить следующие группы команд:

1) язык определения данных (DDL) используется для определения структур данных, хранящихся в базе данных. Операторы DDL позволяют создавать, изменять и удалять отдельные объекты в БД. Допустимые типы объектов зависят от используемой СУБД и обычно включают базы данных, пользователей, таблицы и ряд более мелких вспомогательных объектов, например, роли и индексы.

2) язык манипуляции данными (DML) используется для извлечения и изменения данных в БД. Операторы DML позволяют извлекать, вставлять, изменять и удалять данные в таблицах. Иногда операторы select извлечения данных не рассматриваются как часть DML, поскольку они не изменяют состояние данных. Все операторы DML носят декларативный характер.

3) язык определения доступа к данным (DCL) используется для контроля доступа к данным в БД. Операторы DCL применяются к привилегиям и позволяют выдавать и отбирать права на применение определенных операторов DDL и DML к определенным объектам БД.

4) язык управления транзакциями (TCL) используется для контроля обработки транзакций в БД. Обычно операторы TCL включают commit для подтверждения изменений, сделанных в ходе транзакции, rollback для их отмены и savepoint для разбиения транзакции на несколько меньших частей.



При этом SQL предоставляет и другие возможности, например, выполнение вычислений и преобразований, упорядочение записей и группировку данных.

Особенность команд SQL состоит в том, что они ориентированы в большей степени на конечный результат обработки данных, чем на процедуру этой обработки. SQL сам определяет, где находятся данные, какие индексы и последовательности операций следует использовать для их эффективного выполнения.

Рассмотрим синтаксис основных команд SQL. При изложении материала будем использовать следующие обозначения:

– звездочка (*) означает "все" и употребляется в обычном для программирования смысле, т.е. "все случаи, удовлетворяющие определению";

– квадратные скобки ([]) означают, что конструкции, заключенные в эти скобки, являются необязательными, т.е. могут быть опущены;

– фигурные скобки ({}) означают, что конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы; эти скобки позволяют уточнить порядок разбора синтаксических конструкций, заменяя обычные скобки, используемые в синтаксисе SQL;

– многоточие (...) указывает на то, что непосредственно предшествующая ему синтаксическая единица факультативно может повторяться один или более раз;

– прямая черта (|) означает наличие выбора из двух или более возможностей; например, конструкция [термин_1 | термин_2] означает, что можно выбрать один из двух терминов (или термин_1, или термин_2); при этом термин_1 выбирается по умолчанию; отсутствие всей этой конструкции

будет восприниматься как выбор термин_1;

– точка с запятой (;) завершающий элемент предложений SQL; этот знак должен присутсвовать после каждой команды;

– запятая (,) используется для разделения элементов списков;

– пробелы ( ) могут вводиться для повышения наглядности между любыми синтаксическими конструкциями предложений SQL;

– прописные латинские буквы и символы используются для написания конструкций языка SQL и должны записываться без изменений;

– строчные буквы используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем, причем для определенности отдельные слова этих конструкций связываются между собой символом подчеркивания (_);

– термины таблица, поле, ... заменяют с целью сокращения термины имя_таблицы, имя_поля, ... соответственно,

– сочетание знаков ::= означает, что синтаксис должен иметь указанный вид.

При составлении команд рекомендуется их комментировать. Комментарии в SQL обычно начинаются с двойного дефиса и заканчиваются символом новой строки.

1.1.2. Команды определения данных

Команды определения данных иногда выделяют как отдельный язык DDL (Data Definition Language). Эти команды дают возможность создавать объекты БД (таблицы и индексы), определять их структуру, устанавливать свойства полей и поддержки целостности данных по ссылкам.

Рассмотрим синтаксис **команды создания таблицы**.

CREATE TABLE <имя таблицы> (<эл_табл> [{,<эл_табл>}...])
<Эл_табл> ::= <определение поля>
| <определение ограничения на таблицу>

<Определение поля> ::=<имя поля> <тип данных>
[DEFAULT { <литерал> | NULL }] раздел умолчания
[COMPUTED [BY]] Вычисляется как”
[NOT NULL [UNIQUE | PRIMARY KEY] ограничение на поле
| [FOREIGN KEY(поле)
REFERENCES <ссылочные таблицы и поля>] спецификация ссылок
| [CHECK (<условии поиска>)] проверочное ограничение

Рассмотрим ограничения для таблицы, которые фактически проверяются после выполнения каждого оператора SQL.

<Определение ограничения на таблицу> ::=

[UNIQUE
| PRIMARY KEY (<имя поля> [{,<имя поля>}...])] уникальность
| [FOREIGN KEY (<имя поля> [{,<имя поля>}...]
REFERENCES <имя таблицы> [(<имя поля> [{,<имя поля>}...)]]
ограничения на ссылки
| [CHECK (<условие поиска>)] проверочное ограничение

Указание значений по умолчанию

Значение по умолчанию может быть:

- литералом (символьная строка, числовое значение или дата)
- константой NULL (если значение не введено, полю будет присвоено значение NULL)
- константой USER (в поле будет введено имя пользователя). Длины поля должно хватить для сохранения имени пользователя.

Описатель NOT NULL

Если поле описано как NOT NULL, его значение обязательно должно быть определено. Если пользователь забудет определить такое поле, он не сможет добавить строку в таблицу.

Определение вычисляемых полей

Определение:

<имя столбца> COMPUTED [BY] (<выражение>);

Столбцы, на которые ссылается выражение, должны существовать до определения вычисляемого поля.

Например, в следующей таблице поле «Полное имя» вычисляется с помощью конкатенации полей «Имя» и «Фамилия»:

CREATE TABLE employee (

first_name **VARCHAR**(10) **NOT NULL**,

last_name **VARCHAR**(15) **NOT NULL**,

full_name **COMPUTED BY** (last_name || ', ' || first_name)

);

Ограничение целостности CHECK

Синтаксис:

CHECK (<условие>);

В условии можно пользоваться операторами сравнения, операторами NOT, BETWEEN, LIKE, IN, IS [NOT] NULL и прочими. У столбца может быть только одно условие CHECK.

Назначение имени ограничению

CONSTRAINT Имя ограничения CHECK (Ограничение)

Т.е. определение ограничения состоит из необязательной части, состоящей из служебного слова CONSTRAINT и имени ограничения, и обязательной части, состоящей из собственно определения ограничения. В случае, если первая часть опущена, сервер генерирует имя ограничения автоматически. Посмотреть имя ограничения можно просмотрев системную таблицу RDB$RELATION_CONSTRAINTS.

Пример:

```
CREATE TABLE TestTable (
    id int DEFAULT 1 NOT NULL,
    vcName varchar(50) NOT NULL,
    iApartment int,
    dDate datetime,
    CONSTRAINT check_iApartment CHECK
      (iApartment>0 and iApartment<1000),
    CONSTRAINT check_dDate CHECK
      (dDate<getdate())
)
```

Как видите, в этом примере ограничения описываются через запятую, так же, как и поля. После ключевого слова CONSTRAINT указывается имя ограничения. Из своей практики хочу порекомендовать называть их в виде check_имя, где имя – имя поля, которое проверяется ограничением. После этого указывается само ограничение точно также как и в предыдущем примере.

Обратите внимание, что для ограничения ввода в поле "dDate" мы использовали функцию (getdate). Как и при описании значений по умолчанию, в ограничениях также могут использоваться функции.

При создании ограничения, можно использовать многие операторы сравнения языка SQL. Например, в SQL есть очень удобный оператор IN. С его помощью можно задать возможные значения для поля, которые оно может принимать. Например, вам нужно в таблице ограничить ввода данных в поле содержащую такую информацию как пол человека. В этом случае, можно разрешить ввод букв "М" и "Ж" следующим образом:

```
CREATE TABLE TestTable (
   id int DEFAULT 1 NOT NULL,
   vcName varchar(50) NOT NULL,
   cPol char(1),
   CONSTRAINT check_cPol CHECK
     (cPol IN ('М', 'Ж'))
)
```

В данном случае, ограничение выглядит следующим образом: cPol IN ('М', 'Ж'). Оператор IN означает, что поле может принимать любые значения, перечисленные в круглых скобках. В нашем случае указано две строки 'М' и 'Ж'. Другие буквы вносить в поле нельзя. Конечно же, то же самое можно было бы написать и следующим образом:

```
(cPol = 'М' or cPol = 'Ж')
```

Но это не очень удобно, особенно, если очень много возможных вариантов. Ограничение окажется не очень удобным для чтения. Оператор IN в этом случае намного красивее и читабельнее.

Очень мощных возможностей можно добиться, используя в ограничении оператора LIKE. Например, вы хотите, чтобы поле для хранения телефонного номера содержало номер в формате (XXX) XXX-XX-XX, где X – это любая цифра. Для реализации примера такого ограничения создадим новую таблицу с полем "vcPhonenumber":

```
CREATE TABLE TestTable (
   id int DEFAULT 1 NOT NULL,
   vcPhonenumber varchar(50) NOT NULL
   CONSTRAINT check_vcPhonenumber CHECK
   (vcPhonenumber LIKE
    ('[0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]')
)
```

В данном случае, ограничение с именем check_vcPhonenumber при проверки использует оператор LIKE. С этим оператором мы познакомимся

ближе в следующей главе, когда будем изучать работу с оператором SELECT, но я решила вставить этот пример, чтобы вы заранее знали о его мощи. Если классический знак равенства производит жесткое сравнение, то LIKE позволяет сравнивать строки с определенным шаблоном. В данном случае шаблоном является:

([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]

В квадратных скобках мы указываем возможный диапазон символа. В данном случае, диапазон от 0 до 9. Если заменить все [0-9] на X, то мы получим искомый шаблон (XXX) XXX-XX-XX. Следующий пример запроса добавляет в таблицу номер телефона (085) 880-08-00:

```
INSERT INTO TestTable(vcPhonenumber) VALUES('(085) 880-08-00')
```

При разработке базы данных, в которой формировалась отчетность из списка клиентов организации, оказалось, что в базе есть дубликаты клиентов. В старой базе данных не было проверок на двойственность записей, только не эффективные проверки в пользовательской программе.

Самая простая проверка на двойников проходила выявлением записей, где полностью совпадали ФИО и дата рождение клиента. Вероятность совпадения всех этих параметров внутри одного города стремиться к нулю. Таким образом, необходимо было всего лишь добавить ограничение, при котором сочетание из этих полей было уникальным. Это можно сделать следующим образом:

```
CREATE TABLE Names (
    idName int ,
    vcName varchar(50),
    vcLastName varchar(50),
    vcSurName varchar(50),
    dBirthDay datetime,
    CONSTRAINT cn_unique UNIQUE (vcName, vcLastName,
      vcSurName, dBirthDay) )
```

В данном примере создается таблица из полей: идентификатора, имени, фамилии, отчества и даты рождения. После этого, создается ограничение, при этом оно имеет тип не CHECK, а UNIQUE. В скобках указываются поля, которые должны быть уникальными. В данном примере, я перечислил поля фамилия, имя, отчество и дата рождения. Таким образом, база данных не позволит создать дубликат записи.

Описание ограничений целостности

Первичный ключ есть минимальный набор полей, однозначно идентифицирующих строку. Первичный ключ может быть только один, он описывается ограничением целостности PRIMARY KEY. Кроме первичного ключа, в таблице может быть произвольное количество возможных (альтернативных) ключей. Для описания возможных ключей служит ограничение UNIQUE. Смысл UNIQUE такой же, как и PRIMARY KEY – значение поля (полей), описанных как UNIQUE должно быть уникальным по таблице.

В случае, если первичный или альтернативный ключ состоит из одного поля, его можно описать в той же строке, что и само поле (такое ограничение называется ограничением на уровне столбца). Если первичный или альтернативный ключ состоит из нескольких атрибутов, он описывается после определения всех полей таблицы (такое ограничение называется ограничением на уровне таблицы).

Определение ограничения на уровне таблицы описывается так:
<определение ограничения> = {
UNIQUE | **PRIMARY KEY**|**CHECK** (<условие проверки>)
| **REFERENCES** <ИмяДругойТаблицы> [(<имя столбца>[, <имя столбца> …])]
[**ON DELETE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}]
[**ON UPDATE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}] }

Поле можно определить как первичный ключ, его значение можно сделать уникальным по таблице, либо можно указать, что поле ссылается (REFERENCES) на некоторое поле из другой таблицы. Таблица, на которую ссылается поле, называется родительской. Смысл ограничения REFERENCES следующий: значение в поле в дочерней таблицы можно занести только в том случае, если это значение есть в соответствующем поле родительской таблицы. Например, если у нас есть таблица клиентов, и таблица покупок, поле “Номер клиента” из таблицы покупок должно ссылаться на поле “Номер клиента” из таблицы клиентов. Таким образом в таблицу покупок невозможно будет занести номер несуществующего клиента. Вторая часть описателя REFERENCES описывает, какие действия необходимо предпринять при удалении из родительской таблицы (ON DELETE) и при обновлении родительской таблицы (ON UPDATE). Всего возможно четыре варианта:

- NO ACTION - ничего не предпринимать

- CASCADE – каскадировать. В случае удаления из родительской таблицы будут удалены все связанные записи из дочерней таблицы. Например, удаление клиента повлечет удаление всех его покупок. При модификации записи из родительской таблицы будут модифицированы также записи в дочерней таблице. Например, если клиент поменял номер с пятого на десятый, во всех его покупках поле “Номер клиента” также изменит свое значение с 5 на 10.
- SET DEFAULT – поле получает свое значение по умолчанию
- SET NULL – поле устанавливается в NULL

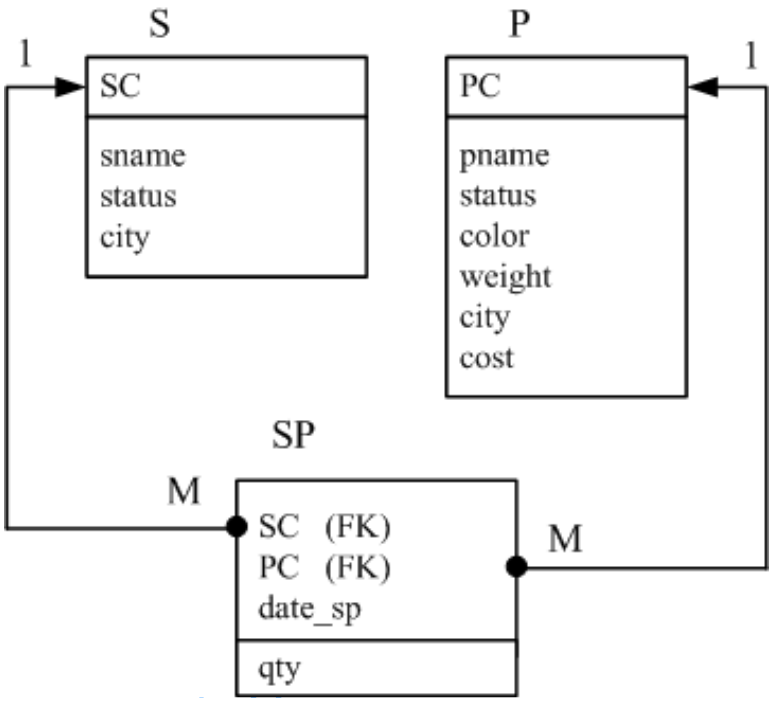
Какое именно действие необходимо предпринимать зависит от смысла таблиц. При удалении клиента удалять все его покупки имеет смысл. Но при удалении желтого цвета из таблицы цветов не стоит удалять все желтые автомобили.

Ограничения целостности на уровне таблицы описывается так же, как и ограничение на уровне столбца, за исключением определения самого ограничения (уф). Оно описывается так:

<ограничение> = {{**PRIMARY KEY** | **UNIQUE**} (<имя столбца> [,<имя столбца> …])
| **FOREIGN KEY** (<имя столбца> [,<имя столбца> …]) **REFERENCES** <ИмяДругойТаблицы>
[**ON DELETE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}]
[**ON UPDATE** {**NO ACTION** | **CASCADE** | **SET DEFAULT** | **SET NULL**}]
| **CHECK** (<условие>)}

Отличие в том, что в PRIMARY KEY, UNIQUE и FOREIGN KEY можно описать более одного столбца.

Рассмотрим ***пример*** БД поставщиков и деталей.



P (Детали)

PC	PNAME	STATUS	COLOR	WEIGHT	CITY	COST
P1	Nut	20	Red	12	London	20.00
P2	Bolt	20	Green	17	Paris	25.30
P3	Screw	50	Blue	17	Rome	40.50
P4	Screw	30	Red	14	London	15.00
P5	Cam	10	Blue	12	Paris	25.50
P6	Cog	20	Red	19	London	40.50

S (Поставщики)

SC	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Black	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP (Поставки)

SC	PC	DATE_SP	QTY
S1	P1	21.01.2010	300
S1	P2	21.01.2010	200
S1	P1	30.03.2010	300
S2	P1	21.01.2010	500
S1	P3	25.03.2010	400
S1	P4	21.01.2010	200
S1	P5	21.01.2010	100
S1	P6	25.03.2010	100
S1	P6	30.04.2010	300
S2	P1	25.06.2010	300
S3	P1	25.06.2010	500
S2	P2	01.09.2010	400
S3	P2	01.09.2010	200
S4	P2	10.10.2010	200
S4	P4	10.10.2010	300
S4	P5	10.10.2010	400

Создание таблицы поставщиков:

```
CREATE TABLE   S (
    sc  varchar(5)  NOT NULL  PRIMARY KEY,
    sname       varchar(20),
    status       integer,
    city          varchar(15),
    CHECK  (status > 0)
   );
```

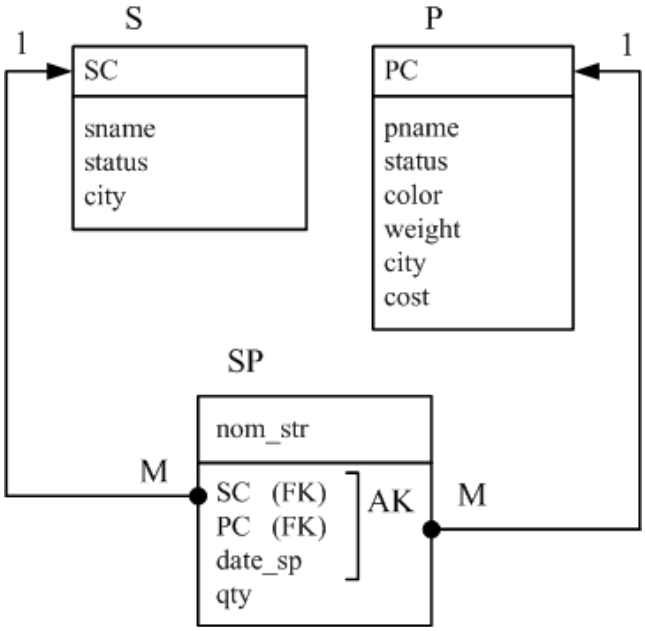
Создание таблицы деталей:

```
CREATE TABLE   P (
    pc varchar (5) NOT NULL PRIMARY KEY,
    pname varchar(20),
    status  integer,
    color varchar (25),
    weight real,
    city varchar(20),
    cost real,
    CHECK  (status > 0)   );
```

Создание таблицы поставок:

```
CREATE TABLE     SP (
    sc   varchar(5) NOT NULL  REFERENCES S (sc)  ON DELETE
    CASCADE ON UPDATE CASCADE,
    pc  varchar(5) NOT NULL REFERENCES P (pc),
    date_sp  date,
    qty  integer  DEFAULT 100,
    PRIMARY KEY (sc, pc, date_sp),
    CHECK (qty BETWEEN 100 AND 1000)
    );
```

Скорректированная схема данных БД



Создание таблицы деталей (скорректированной):

```
CREATE TABLE    SP (
    nom_str  integer NOT NULL PRIMARY KEY,
    sc   varchar(5) NOT NULL  REFERENCES
    S (sc)  ON DELETE CASCADE ON UPDATE CASCADE ,
    pc  varchar(5) NOT NULL REFERENCES P (pc),
    date_sp  DATE,
    qty  integer  DEFAULT 100,
    UNIQUE (sc, pc, date_sp),
    CHECK (qty BETWEEN 100 AND 1000)
);
```

Команда создания представления CREATE VIEW

Типы таблиц, с которыми вы имели дело до сих пор, назывались - базовыми таблицами. Это - таблицы, которые содержат данные. Однако имеется другой вид таблиц: - представления.

Представления - это таблицы чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных.

Представления - подобны окнам, через которые вы просматриваете информацию (как она есть, или в другой форме, как вы потом увидите), которая фактически хранится в базовой таблице. Представление - это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

Вы создаете представление командой CREATE VIEW. Она состоит из слов CREATE VIEW (СОЗДАТЬ ПРЕДСТАВЛЕНИЕ), имени представления которое нужно создать, слова AS (КАК), и далее запроса:

CREATE VIEW <имя представления> [(<имя столбца> [,<имя столбца>...])] AS<подзапрос>[WITH CHECK OPTION];

<подзапрос>::=<оператор SELECT>;

Пример:

```
CREATE VIEW Londonstaff
AS SELECT *
FROM Salespeople
WHERE city = 'London';
```

Теперь Вы имеете представление, называемое Londonstaff. Вы можете использовать это представление точно так же как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из, и соединена с, другими таблицами и представлениями.

Команда создания индексов CREATE INDEX позволяет определять простые и составные индексы, устанавливать ключи.

Следует заметить, что синтаксис команд определения данных в различных СУБД может иметь отличия от стандарта.

Индекс - это упорядоченный (буквенный или числовой) список столбцов или групп столбцов в таблице. Таблицы могут иметь большое количество строк, а, так как строки не находятся в каком-нибудь определенном порядке, их поиск по указанному значению может потребовать времени. Индексный адрес - это и забота, и в то же врем обеспечение способа объединения всех значений в группы из одной или больше строк, которые отличаются одна от другой.

Индексы - это средство SQL, которое родил сам рынок, а не ANSI. Поэтому, сам по себе стандарт ANSI в настоящее время не поддерживает индексы, хотя они очень полезны и широко применяемы.

В то врем как индекс значительно улучшает эффективность запросов, использование индекса несколько замедляет операции модификации DML(такие как INSERT и DELETE), а сам индекс занимает объем памяти. Следовательно, каждый раз когда вы создаете таблицу Вы должны принять решение, индексировать ее или нет. Индексы могут состоять из

многочисленных полей. Если больше чем одно поле указывается для одного индекса, второе упорядочивается внутри первого, третье внутри второго, и так далее. Если вы имели первое и последнее имя в двух различных полях таблицы, вы могли бы создать индекс, который бы упорядочил предыдущее поле внутри последующего. Это может быть выполнено независимо от способа упорядочивания столбцов в таблице.

Синтаксис для создания индекс - обычно следующий (помните, что это не ANSI стандарт):

CREATE [UNIQUE] INDEX <имя индекса> ON<имя таблицы>(< имя столбца>,[<упоряд.>][,<имя столбца> [<упоряд.>]...]);

<упоряд.>::=ASC (возрастание)| DESC (убывание);
{по умолчанию ASC }

Таблица, конечно, должна уже быть создана и должна содержать им столбца. Им индекса не может быть использовано для чего-то другого в базе данных (любым пользователем). Однажды созданный, индекс будет невидим пользователю. SQL сам решает когда он необходим чтобы ссылаться на него и делает это автоматически. Если, например, таблица Заказчиков будет наиболее часто упоминаемой в запросах продавцов к их собственной клиентуре, было бы правильно создать такой индекс в поле snum таблицы Заказчиков.

CREATE INDEX Clientgroup ON Customers (snum);