Язык SQL

SQL (Structured Query Language) - структурированный язык запросов) основывается на некоторой смеси алгебраических и логических конструкций.

Язык SQL в настоящее время является промышленным стандартом, который в большей или меньшей степени поддерживает любая СУБД, претендующая на звание "реляционной".

1.Типы данных SQL.

- Символьные типы данных - содержат буквы, цифры и специальные символы.
- **CHAR** или **CHAR(n)** -символьные строки фиксированной длины. Длина строки определяется параметром **n**. **CHAR** без параметра соответсвует **CHAR(1)**. Для хранения таких данных всегда отводится **n** байт вне зависимости от реальной длины строки.
- **VARCHAR(n)** - символьная строка переменной длины. Для хранения данных этого типа отводится число байт, соответствующее реальной длине строки.

- Целые типы данных - поддерживают только целые числа (дробные части и десятичные точки не допускаются). Над этими типами разрешается выполнять арифметические операции и применять к ним агрегирующие функции (определение максимального, минимального, среднего и суммарного значения столбца реляционной таблицы).
  - **INTEGER** или **INT**- целое, для хранения которого отводится, как правило, 4 байта. *(Замечание: число байт, отводимое для хранения того или иного числового типа данных зависит от используемой СУБД и аппаратной платформы, здесь приводятся наиболее "типичные" значения)* Интервал значений от - 2147483647 до + 2147483648
  - **SMALLINT** - короткое целое (2 байта), интервал значений от - 32767 до +32768
- Вещественные типы данных - описывают числа с дробной частью.
  - **FLOAT** и **SMALLFLOAT** - числа с плавающей точкой

- **DECIMAL(p)** - тип данных аналогичный **FLOAT** с числом значащих цифр **p**.
- **DECIMAL(p,n)** - аналогично предыдущему, **p** - общее количество десятичных цифр, **n** - количество цифр после десятичной запятой.
- Денежные типы данных - описывают, естественно, денежные величины. Если в ваша система такого типа данных не поддерживает, то используйте DECIMAL(p,n).
- **MONEY(p,n)** - все аналогично типу **DECIMAL(p,n)**. Вводится только потому, что некоторые СУБД предусматривают для него специальные методы форматирования.
- Дата и время - используются для хранения даты, времени и их комбинаций.
- **DATE** - тип данных для хранения даты.
- **TIME** - тип данных для хранения времени.
- **INTERVAL** - тип данных для хранения временного интервала.)

- **DATETIME** - тип данных для хранения моментов времени (год + месяц + день + часы + минуты + сек. + доли сек.).
- Двоичные типы данных - позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.). Определения этих типов наиболее сильно различаются от системы к системе, часто используются ключевые слова:
- **BINARY**
- **BYTE**
- **BLOB**
- Последовательные типы данных - используются для представления возрастающих числовых последовательностей.
- Для всех типов данных имеется общее значение **NULL** - "не определено". При создании таблицы можно явно указать СУБД могут ли элементы того или иного столбца иметь значения **NULL** (это не допустимо, для столбца, являющего первичным ключом).

2. Операторы создания схемы базы данных.

При описании команд предполагается, что:

- текст, набранный строчными буквами (например, **CREATE TABLE**) является обязательным
- текст, набранный прописными буквами и заключенный в угловые скобки (например, **<имя_базы_данных>**) обозначает переменную, вводимую пользователем
- в квадратные скобки (например, **[NOT NULL]**) заключается необязательная часть команды
- взаимоисключающие элементы команды разделяются вертикальной чертой (например, **[UNIQUE | PRIMARY KEY]**).

Операторы базы данных

Команда	Описание
CREATE DATABASE <имя_базы_данных>	Создание базы данных
DROP DATABASE <имя_базы_данных>	Удаление базы данных

Создание таблицы:

```
CREATE TABLE
<имя_таблицы>(<имя_столбца><тип_столбца>
[NOT NULL]
[UNIQUE | PRIMARY KEY]
[REFERENCES <имя_мастер_таблицы> [<имя_столбца>]] ,
...)
```

Пользователь обязан указать имя таблицы и список столбцов. Для каждого столбца обязательно указываются его имя и тип (см. таблицу в предыдущем разделе), а также опционально могут быть указаны параметры

- **NOT NULL** - в этом случае элементы столбца всегда должны иметь определенное значение (не NULL)
- один из взаимоисключающих параметров **UNIQUE** - значение каждого элемента столбца должно быть уникальным или **PRIMARY KEY** - столбец является первичным ключом.
- **REFERNECES <имя_мастер_таблицы> [<имя_столбца>]** - эта конструкция определяет, что данный столбец является внешним ключом и указывает на ключ какой мастер таблицы он ссылается.

Контроль за выполнением указанных условий осуществляет СУБД.

Пример: создание базы данных ***publications: // (ПУБЛИКАЦИИ)***

```
CREATE DATABASE publications;
CREATE TABLE authors (au_id INT PRIMARY KEY,
            author VARCHAR(25) NOT NULL);
CREATE TABLE publishers (pub_id INT PRIMARY KEY,
        publisher VARCHAR(255) NOT NULL,
url VARCHAR(255));
CREATE TABLE titles (title_id INT PRIMARY KEY,
   title VARCHAR(255) NOT NULL,
   yearpub INT,
   pub_id INT REFERENCES publishers(pub_id));
CREATE TABLE titleautors (
au_id INT REFERENCES authors(au_id),
title_id INT REFERENCES titles(title_id));
CREATE TABLE wwwsites (site_id INT PRIMARY KEY,
                       site VARCHAR(255) NOT NULL,
                       url VARCHAR(255));
```

```
 CREATE TABLE wwwsiteauthors (au_id INT
REFERENCES authors(au_id),
                                site_id INT
REFERENCES wwwsites(site_id));
```

Удаление таблицы:

DROP TABLE <имя_таблицы>

Модификация таблицы:

Добавить столбцы	ALTER TABLE <имя_таблицы> ADD (<имя_столбца> <тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY] [REFERENCES <имя_мастер_таблицы> (<имя_столбца>)]]

Модификация таблицы:

Удалить столбцы	ALTER TABLE <имя_таблицы> DROP (<имя_столбца>,...)
Модификация типа столбцов	ALTER TABLE <имя_таблицы> MODIFY <имя_столбца> <тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY] [REFERENCES <имя_мастер_таблицы> (<имя_столбца>)]

3. Операторы управления правами доступа.

По соображениям безопасности не каждому пользователю прикладной системы может быть разрешено получать информацию из какой-либо таблицы, а тем более изменять в ней данные. Для определения прав пользователей относительно объектов базы данных (таблицы, представления, индексы) в **SQL** определена пара команд GRANT и REVOKE. Синтаксис операции передачи прав на таблицу:

```
GRANT <тип_права_на_таблицу>
  ON  <имя_таблицы> [<список_столбцов>]
  TO  <имя_пользователя>
```

Права пользователя на уровне таблицы определяются следующими ключевыми словами (как мы увидим чуть позже эти ключевые слова совпадают с командами выборки и изменения данных):

- SELECT - получение информации из таблицы
- UPDATE - изменение информации в таблице
- INSERT - добавление записей в таблицу

- DELETE - удаление записей из таблицы
- INDEX - индексирование таблицы
- ALTER - изменение схемы определения таблицы
- ALL - все права

В поле <тип_права_на_таблицу> может быть указано либо ключевое слово ALL или любая комбинация других ключевых слов. Например, предоставим все права на таблицу **publishers** пользователю **andy**:

```
GRANT ALL ON publishers TO andy;
```

Пользователю **peter** предоставим права на извлечение и дбавление записей на эту же таблицу:

```
GRANT SELECT INSERT ON publishers TO peter;
```

В том случае, когда одинаковые права надо предоставить сразу всем пользователям, вместо выполнения команды GRANT для каждого из них, можно вместо имени пользователя указать ключевое слово PUBLIC:

```
GRANT SELECT ON publishers TO PUBLIC;
```

Отмена прав осуществляется командой REVOKE:

```
REVOKE <тип_права_на_таблицу>
   ON  <имя_таблицы> [<список_столбцов>]
   FROM  <имя_пользователя>
```

Все ключевые слова данной команды эквивалентны оператору GRANT.

Большинство систем поддерживают также команду GRANT для назначения привилегий на базу данных в целом. В этом случае формат команды:

```
GRANT <тип_права_на_базу_данных>
   ON  <имя_базы данных>
   TO  <имя_пользователя>
```

Способы задания прав на базу данных различны для разных СУБД, и точную их формулировку нужно уточнять в документации

4. Команды модификации данных.

К этой группе относятся операторы добавления, изменения и удаления записей.

Добавить новую запись в таблицу:

```
INSERT INTO <имя_таблицы> [
(<имя_столбца>,<имя_столбца>,...) ]
                                    VALUES
(<значение>,<значение>,..)
```

Список столбцов в данной команде не является обязательным параметром. В этом случае должны быть указаны значения для всех полей таблицы в том порядке, как эти столбцы были перечислены в команде `CREATE TABLE`, например:

```
INSERT INTO publishers VALUES (16,"Microsoft Press","http://www.microsoft.com");
```

Пример с указанием списка столбцов:

```
INSERT INTO publishers (publisher,pub_id)
VALUES ("Super Computer Publishing",17);
```

Модификация записей:

UPDATE <имя_таблицы>
SET имя_столбца>=<значение>,...[WHERE <условие>]

Если задано ключевое слово WHERE и условие, то команда UPDATE применяется только к тем записям, для которых оно выполняется. Если условие не задано, UPDATE применяется ко всем записям.

Пример:

```
UPDATE publishers SET url="http://www.superpub.com"
WHERE pub_id=17;
```

В качестве условия используются логические выражения над константами и полями. В условиях допускаются:

- операции сравнения: > , < , >= , <= , = , <> , != . В SQL эти операции могут применяться не только к числовым значениям, но и к строкам ("<" означает раньше, а ">" позже в алфавитном порядке) и датам ("<" раньше и ">" позже в хронологическом порядке).
- операции проверки поля на значение NULL: IS NULL, IS NOT NULL

- операции проверки на вхождение в диапазон: `BETWEEN` и `NOT BETWEEN`.
- операции проверки на вхождение в список: `IN` и `NOT IN`
- операции проверки на вхождение подстроки: `LIKE` и `NOT LIKE`
- отдельные операции соединяются связями `AND, OR, NOT` и группируются с помощью скобок.

Пример:

```
UPDATE publishers SET url="url not defined" WHERE url IS NULL;
```

Эта команда находит в таблице **`publishers`** все неопределенные значения столбца **`url`** и заменяет их строкой "url not defined".

Удаление записей

`DELETE FROM <имя_таблицы> [ WHERE <условие> ]`

Удаляются все записи, удовлетворяющие указанному условию. Если ключевое слово `WHERE` и условие отстутствуют, из таблицы удаляются все записи.

Пример:

DELETE FROM publishers WHERE publisher = "Super Computer Publishing";

Эта команда удаляет запись об издательстве Super Computer Publishing

5. Выборка данных.

Для извлечения записей из таблиц в SQL определен оператор **SELECT**. С помощью этой команды осуществляется не только операция реляционной алгебры "выборка" (горизонтальное подмножество), но и предварительное соединение (join) двух и более таблиц. Это наиболее сложное и мощное средство SQL, полный синтаксис оператора **SELECT** имеет вид:

```
SELECT [ALL | DISTINCT] <список_выбора>
FROM <имя_таблицы>, ...
[ WHERE <условие> ]
[ GROUP BY <имя_столбца>,... ]
[ HAVING <условие> ]
[ORDER BY <имя_столбца> [ASC | DESC],... ]
```

Оператор всегда начинается с ключевого слова SELECT. В конструкции <список_выбора> определяется столбец или столбцы, включаемые в результат. Он может состоять из имен одного или нескольких столбцов, или из одного символа * (звездочка), определяющего все столбцы.

Пример: получить список всех авторов

SELECT author FROM authors;

получить список всех полей таблицы **authors:**

SELECT * FROM authors;

В том случае, когда нас интересуют не все записи, а только те, котрые удовлетворяют некому условию, это условие можно указать после ключевого слова WHERE. Например, найдем все книги, опубликованные после 1996 года:

SELECT title FROM titles WHERE yearpub > 1996;

Допустим теперь, что нам надо найти все публикации за интервал 1995 - 1997 гг. Это условие можно записать в виде:

SELECT title FROM titles WHERE yearpub>=1995 AND yearpub<=1997;

Другой вариант этой команды можно получить с использованием логической операции проверки на вхождение в интервал:

```
SELECT title FROM titles WHERE yearpub BETWEEN 1995 AND 1997;
```

При использовании конструкции `NOT BETWEEN` находятся все строки, не входящие в указанный диапазон.

Еще один вариант этой команды можно построить с помощью логической операции проверки на вхождение в список:

```
SELECT title FROM titles WHERE yearpub IN (1995,1996,1997);
```

Здесь мы задали в явном виде список интересующих нас значений. Конструкция `NOT IN` позволяет найти строки, не удовлетворяющие условиям, перечисленным в списке.

Наиболее полно преимущества ключевого слова `IN` проявляются во вложенных запросах, также называемых подзапросами.

Предположим, нам нужно найти все издания, выпущенные компанией "Oracle Press". Наименования издательских компаний содержатся в таблице **`publishers`**, названия книг в таблице **`titles`**. Ключевое слово `NOT IN` позволяет объединить обе таблицы и извлечь при этом нужную информацию:

```
SELECT title FROM titles WHERE pub_id IN
    (SELECT pub_id FROM publishers WHERE publisher='Oracle Press');
```

При выполнении этой команды СУБД вначале обрабатывает вложенный запрос по таблице **`publishers`**, а затем его результат передает на вход основного запроса по таблице **`titles`**.

Некоторые задачи нельзя решить с использованием только операторов сравнения. Например, мы хоти найти web-site издательства "Wiley", но не знаем его точного наименования. Для решения этой задачи предназначено ключевое слово `LIKE`, его синтаксис имеет вид:

WHERE <имя_столбца> LIKE <образец> [ESCAPE <ключевой_символ>]

Образец заключается в кавычки и должен содержать шаблон подстроки для поиска. Обычно в шаблонах используются два символа:

- % (знак процента) - заменяет любое количество символов
- _ (подчеркивание) - заменяет одиночный символ.

Попробуем найти искомый web-site:

SELECT publiser, url FROM publishers WHERE publisher LIKE '%Wiley%';

В соотвествии с шаблоном СУБД найдет все строки включающие в себя подстроку "Wiley". Другой пример: найти все книги, название которых начинается со слова "SQL":

SELECT title FROM titles WHERE title LIKE 'SQL%';

В том случае, когда надо найти значение, которое само содержит один из символов шаблона, используют ключевое слово ESCAPE и <ключевой_символ>. Литерал, следующий в шаблоне после ключевого символа, рассматривается как обычный символ, все последующие символы имеют обычное значение. Например, нам надо найти ссылку на web-страницу, о которой известно, что в ее url содержится подстрока "my_works":

SELECT site, url FROM wwwsites WHERE url LIKE'%my@_works%' ESCAPE '@';

При выполнении оператора `SELECT` результирующее отношение (**но не таблица!**) может иметь несколько записей с одинаковыми значениями всех полей. Чтобы исключить повторяющиеся записи из выборки используется **`DISTINCT`**. **`ALL`** указывает, что в результат необходимо включать все строки.

6. Выборка из нескольких таблиц.

Очень часто возникает ситуация, когда выборку данных надо производить из отношения, которое является результатом слияния (join) двух других отношений. Например, нам нужно получить из базы данных **publications** информацию о всех печатных изданиях в виде следующей таблицы:

```
-----------------------------------------
|название_книги | год_выпуска | издательство   |
-----------------------------------------
|                |             |                |
```

Для этого СУБД предварительно должна выполнить слияние таблиц **`titles`** и **`publishers`**.

Для выполнения операции такого рода в операторе SELECT после ключевого слова FROM указывается список таблиц, по которым производится поиск данных. После ключевого слова WHERE указывается условие, по которому производится слияние. Для выполнить данный запрос, нужно дать команду:

```
SELECT titles.title,titles.yearpub,publishers.publisher
FROM titles,publishers
WHERE titles.pub_id=publishers.pub_id;
```

Пример, где одновременно задаются условия и слияния, и выборки (результат предыдущего запроса ограничивается изданиями после 1996 года):

```
SELECT titles.title,titles.yearpub,publishers.publisher
FROM titles,publishers
WHERE titles.pub_id=publishers.pub_id AND
    titles.yearpub>1996;
```

Когда в разных таблицах присутствуют одноименные поля, то для устранения неоднозначности перед именем поля указывается имя таблицы и знак "." (точка).

Имеется возможность производить слияние и более чем двух таблиц. Например, чтобы дополнить описанную выше выборку именами авторов книг необходимо составить оператор следующего вида:

```
SELECT authors.author,titles.title,titles.yearpub,publishers.publisher
  FROM titles,publishers,titleauthors
  WHERE titleauthors.au_id=authors.au_id AND
        titleauthors.title_id=titles.title_id AND
        titles.pub_id=publishers.pub_id AND
        titles.yearpub > 1996;
```

7. Вычисления внутри SELECT.

SQL позволяет выполнять различные арифметические операции над столбцами результирующего отношения. В конструкции <список_выбора> можно использовать константы, функции и их комбинации с арифметическими операциями и скобками.
чтобы узнать сколько лет прошло с 1992 года до публикации той или иной книги можно выполнить команду:

SELECT title, yearpub-1992 FROM titles WHERE yearpub > 1992;

В арифметических вражения допускаются операции сложения (+), вычитания (-), деления (/), умножения (*), а также различные функции (COS, SIN, ABS - абсолютное значение и т.д.). Также в запрос можно добавить строковую константу:

SELECT 'the title of the book is', title, yearpub-1992
FROM titles WHERE yearpub > 1992;

В SQL также определены так называемые агрегатные функции, которые совершают действия над совокупностью одинаковых полей в группе записей. Среди них:

- **AVG(<имя поля>)** - среднее по всем значениям данного поля
- **COUNT(<имя поля>)** или **COUNT (*)** - число записей
- **MAX(<имя поля>)** - максимальное из всех значений данного поля
- **MIN(<имя поля>)** - минимальное из всех значений данного поля
- **SUM(<имя поля>)** - сумма всех значений данного поля

Каждая агрегирующая функция возвращает единственное значение.

Примеры: определить дату публикации самой "древней" книги в нашей базе данных : SELECT MIN(yearpub) FROM titles;

подсчитать количество книг в нашей базе данных:

```
SELECT COUNT(*) FROM titles;
```

Область действия данных функции можно ограничить с помощью логического условия. Например, количество книг, в названии которых есть слово "SQL":

```
SELECT COUNT(*) FROM titles WHERE title LIKE '%SQL%';
```

8. Групировка данных.

Группировка данных в операторе `SELECT` осуществляется с помощью ключевого слова `GROUP BY` и ключевого слова `HAVING`, с помощью которого задаются условия разбиения записей на группы. `GROUP BY` неразрывно связано с агрегирующими функциями, без них оно практически не используется. `GROUP BY` разделяет таблицу на группы, а агрегирующая функция вычисляет для каждой из них итоговое значение.

Определим для примера количество книг каждого издательства в нашей базе данных:

```
SELECT publishers.publisher, count(titles.title)
    FROM titles,publishers
    WHERE titles.pub_id=publishers.pub_id
    GROUP BY publisher;
```

Ключевое слово `HAVING` работает следующим образом: сначала `GROUP BY` разбивает строки на группы, затем на полученные наборы накладываются условия `HAVING`. Например, устраним из предыдущего запроса те издательства, которые имеют только одну книгу:

```
SELECT publishers.publisher, count(titles.title)
   FROM titles,publishers
   WHERE titles.pub_id=publishers.pub_id
   GROUP BY publisher
    HAVING COUNT(*)>1;
```

Другой вариант использования HAVING - включить в результат только те издательства, название которых оканчивается на подстроку "Press":

```
SELECT publishers.publisher, count(titles.title)
    FROM titles,publishers
    WHERE titles.pub_id=publishers.pub_id
    GROUP BY publisher
      HAVING publisher LIKE '%Press';
```

Во втором варианте условие отбора записей мы могли поместить в раздел ключевого слова WHERE, в первом же варианте этого сделать не удастся, поскольку WHERE не допускает использования агрегирующих функций.

9. Сортировка данных.

Для сортировки данных, получаемых при помощи оператора SELECT служит ключевое слово ORDER BY. Можно сортировать результаты по любому столбцу или выражению, указанному в <списке_выбора>. Данные могут быть упорядочены как по возрастанию, так и по убыванию.

Пример: сортировать список авторов по алфавиту:

```
SELECT author FROM authors ORDER BY author;
```

Более сложный пример: получить список авторов, отсортированный по алфавиту, и список их публикаций, причем для каждого автора список книг сортируется по времени издания в обратном порядке (т. е. сначала более "свежие" книги, затем все более "древние"):

```
SELECT authors.author,titles.title,titles.yearpub,publishers.publisher
    FROM authors,titles,publishers,titleauthors
    WHERE titleauthors.au_id=authors.au_id AND
        titleauthors.title_id=titles.title_id AND
        titles.pub_id=publishers.pub_id
    ORDER BY authors.author ASC, titles.yearpub DESC;
```

Ключевое слово `DESC` задает здесь обратный порядок сортировки по полю **`yearpub`**, ключевое слов `ASC` (его можно опускать) - прямой порядок сортировки по полю **`author`**.

11.Использование представлений.

До сих пор мы говорили о таблицах, которые *реально* хранятся в базе данных. Это, так называемые, базовые таблицы (base tables). Существует другой вид таблиц, получивший название "представления" (иногда их называют"представляемые таблицы").

Определение:
Представление (view) - это таблица, содержимое которой берется из других таблиц посредством запроса. При этом новые копии данных не создаются

Когда содержимое базовых таблиц меняется, СУБД автоматически перевыполняет запросы, создающие view, что приводит к соответствующи изменениям в представлениях.

Представление определяется с помощью команды

```
CREATE VIEW <имя_представления>
[<имя_столбца>,...] AS <запрос>
```

При этом должны соблюдаться следующие ограничения:

1) представление должно базироваться на единственном запросе (UNION не допустимо)
2) выходные данные запроса, формирующего представление, должны быть не упорядочены (ORDER BY не допустимо)
Создадим представление, хранящее информацию об авторах, их книгах и издателях этих книг:

```
 CREATE VIEW books AS
 SELECT  authors.author, titles.title,
titles.yearpub, publishers.publisher
   FROM authors,titles,publishers,titleauthors
   WHERE titleauthors.au_id=authors.au_id
AND titleauthors.title_id=titles.title_id
AND titles.pub_id=publishers.pub_id
```

Теперь любой пользователь, чьих прав на доступ к данному представлению достаточно, может осуществлять выборку данных из **books**. Например:

```
SELECT titles FROM books WHERE author LIKE
'%Date'
```

```
SELECT author,count(title) FROM books GROUP BY
author
```

(Права пользователей на доступ в представлениям назначаются также с помощью команд `GRANT / REVOKE.`)

Смысл использования представлений:

если запросы типа "выбрать все книги данного автора с указанием издательств" выполняются достаточно часто, то создание представляемой таблицы **books** значительно сократит накладные расходы на выполнение соединеия четырех базовых таблиц **authors**, **titles**, **publishers** и **titleauthors**. Кроме того, в представлении может быть представлена информация, явно не хранимая ни в одной из базовых таблиц. Например, один из столбцов представления может быть вычисляемым:

```
CREATE VIEW amount (publisher, books_count) AS
SELECT publishers.publisher, count(titles.title)
FROM titles,publishers
WHERE titles.pub_id=publishers.pub_id
GROUP BY publisher;
```

Здесь использована еще одна, ранее не описанная, возможность SQL - присвоение новых имен столбцам представления. В приведенном примере число изданий, осуществленных каждым издатетлем, будет хранится в столбце с именем **books_count**. Если мы хотим присвоить новые имена столбцам представления, нужно указывать имена для **всех** столбцов. Тип данных столбца представления и его нулевой статус всегда зависят от того, как он был определен в базовой таблице (таблицах).

Запрос на выборку данных к представлению выглядит абсолютно аналогично запросу к любой другой таблице. Но на изменение данных в представлении накладываются ограничения:
- если представление основано на одной таблице, изменения данных в нем допускаются (изменяются данные в связанной с ним таблице).
- если представление основано более чем на одной таблице, то изменения данных в нем не допускаются (СУБД не может правильно восстановить схему таблиц из схемы представления).

Удаление представления производится с помощью оператора:

```
DROP VIEW <имя_представления>
```

12. Дополнительные возможности SQL.

Следующие возможности представлены в той или иной мере практически во всех современных СУБД.

- **Хранимые процедуры**.

Практический опыт создания приложений обработки данных показывает, что ряд операций над данными, реализующих общую для всех пользователей логику и не связанных с пользовательским интерфейсом, целесообразно вынести на сервер.

Для написания процедур, реализующих эти операции стандартных возможностей SQL не достаточно - здесь необходимы операторы обработки ветвлений, циклов и т.д. Поэтому во многих СУБД существуют **процедурные** расширения SQL (PL/SQL Oracle и т.д.). Эти расширения содержат логические операторы (IF ... THEN ... ELSE), операторы перехода по условию (SWITCH ... CASE ...), операторы циклов (FOR, WHILE, UNTIL) и операторы передачи управления в процедуры (CALL, RETURN). С помощью этих средств создаются функциональные модули, хранящиеся на сервере вместе с базой данных.

Обычно такие модули называют ***хранимыми процедурами***. Они могут быть вызваны с передачей параметров любым пользователем, имеющим на то соответствующие права. В некоторых системах хранимые процедуры могут быть реализованы и в виде внешних по отношению к СУБД модулей на языках общего назначения, таких как *C* или *Java*. Пример:

```
 CREATE FUNCTION <имя_функции>
([<тип_параметра1>,...<тип_параметра2>])
              RETURNS <возвращаемые_типы>
              AS [ <SQL_оператор> |
<имя_объектного_модуля> ]
                LANGUAGE 'SQL' | 'C' | 'internal'
```

Вызов созданной функции осуществляется из оператора SELECT (также, как вызываются функции агрегирования).

Синтаксис процедур разнится в зависимости от выбранной СУБД и не может быть непосредственно перенесен в исходном коде.

- **Триггеры**.

  Для каждой таблицы может быть назначена хранимая процедура без параметров, которая вызывается при выполнении оператора модификации этой таблицы (INSERT, UPDATE, DELETE). Такие хранимые процедуры получили название ***триггеров***. Триггеры выполняются автоматически, независимо от того, что именно является причиной модификации данных - действия человека, оператора или прикладной программы. "Усредненный" синтаксис оператора создания триггера:

```
CREATE TRIGGER <имя_триггера>
              ON <имя_таблицы>
          FOR { INSERT | UPDATE | DELETE }
              [, INSERT | UPDATE | DELETE ] ...
               AS <SQL_оператор>
```

Ключевое слово **ON** задает имя таблицы, для которой определяется триггер, ключевое слово **FOR** указывает какая команда (команды) модификации данных активирует триггер.

Операторы SQL после ключевого слова **AS** описывают действия, которые выполняет триггер и условия выполнения этих действий. Здесь может быть перечислено любое число операторов SQL, вызовов хранимых процедур и т.д. Использование триггеров очень удобно для выполнения операций контроля ограничений целостности.

- **Мониторы событий**.

Ряд СУБД допускает создание таких хранимых процедур, которые непрерывно сканируют одну или несколько таблиц на предмет обнаружения тех или иных событий (например, среднее значение какого-либо столбца достигает заданного предела). В случае наступления события может быть инициирован запуск триггера, хранимой процедуры, приложения и т.п. Пример: пусть база данных является частью автоматизированной системы управления технологическим процессом. В поле одной из таблиц заносятся показания датчика температуры, установленного на резце токарного станка. Если это значение превышает заданный предел, запускается внешняя программа, изменяющая параметры работы станка

Транзакции, блокировки и многопользовательский доступ к данным.

Любая база данных годна к использованию только тогда, когда ее состояние соответствует состоянию предметной области. Такие состояния называют **целостными**. Очевидно, что при изменении данных БД должна переходить от одного целостного состояния к другому. Однако, в процессе обновления данных возможны ситуации, когда состояние целостности нарушается. Например:

- В банковской системе производится перевод денежных средств с одного счета на другой. На языке SQL эта операция описывается последовательностью двух команд `UPDATE`:

```
UPDATE accounts SET summa=summa-1000 WHERE
account="PC_1"
UPDATE accounts SET summa=summa+1000 WHERE
account="PC_2”
```

после выполнения первой команды и до завершения второй команды база данных не находится в целостном состоянии (искомая сумма списана с первого счета, но не зачислена на второй).

Если в этот момент в системе произойдет сбой (например, выключение электропитания), то целостное состояние БД будет безвозвратно утеряно.

- Целостность БД может нарушаться и во время обработки одной команды SQL. Пусть выполняется операция увеличения зарплаты всех сотрудников фирмы на 20%:

```
UPDATE employers SET salary=salary*1.2
```

При этом СУБД последовательно обрабатывает все записи, подлежащие обновлению, т.е. существует временной интервал, когда часть записей содержит новые значения, а часть - старые. Во избежание таких ситуаций в СУБД вводится понятие **транзакции** - атомарного действия над БД, переводящего ее из одного целостного состояния в другое целостное состояние. Другими словами, транзакция - это последовательность операций, которые должны быть или все выполнены или все не выполнены (**все или ничего**). Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) произведенные изменения в БД во внешней памяти, либо ни одно из этих изменений не отражается на БД.

Каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения. Этот принцип делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД.
Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти - аварийное завершение работы СУБД по причине ошибки в программе или в результате некоторого аппаратного сбоя или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной.

Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении второй требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью. в которую поступают записи обо всех изменениях основной части БД.
Обычно придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL).
Грубо говоря, эта стратегия заключается в том, что запись об

изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты,

модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти.
Этот процесс содержит много тонкостей, в основном относящихся к ведению администратора базы данных.
Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Для нормального восстановления БД после

жесткого сбоя необходимо, чтобы журнал не пропал.
восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя.
В СУБД различных поставщиков начало транзакции может задаваться явно (например, командой **BEGIN TRANSACTION**), либо предполагаться неявным (так определено в стандарте SQL), т.е. очередная транзакция открывается автоматически сразу же после удачного или неудачного завершения предыдущей. Для завершения транзакции обычно используют команды SQL:

- **COMMIT** - успешно завершить транзакцию
- **ROLLBACK** - откатить транзакцию, т.е. вернуть БД в состояние, в котором она находилась на момент начала транзакции.

Стандарт SQL определяет, что транзакция начинается с первого SQL-оператора, инициируемого пользователем или содержащегося в прикладной программе. Все последующие SQL-операторы составляют тело транзакции - вплоть до **END TRANSACTION**.

Транзакция завершается одним из возможных способов:
1.оператор **COMMIT** означает успешное завершение транзакции, все изменения, внесенные в базу данных делаются постоянными
2.оператор **ROLLBACK** прерывает транзакцию и отменяет все внесенные ею изменения
3.успешное завершение программы, инициировавшей транзакцию, означает успешное завершение транзакции (как использование **COMMIT**)
4. ошибочное завершение программы прерывает транзакцию (как **ROLLBACK**)
Пример явно заданной транзакции:

```
        BEGIN TRANSACTION;
/* Начать транзакцию */
        DELETE ...;
/* Изменения */
        UPDATE ...;
/* данных    */
        if (обнаружена_ошибка) ROLLBACK;
   else COMMIT;              /* Завершить транзакцию */
```

Пример неявно заданной транзакции:

```
        COMMIT;                          /* Окончание
предыдущей транзакции */
        DELETE ...;                        /*
Изменения */
        UPDATE ...;                        /*
данных    */
        if (обнаружена_ошибка) ROLLBACK;
        else COMMIT;                       /*
Завершить транзакцию */
```

Описанный механизм транзакций гарантирует обеспечение целостного состояния базы данных только в том случае, когда все транзакции выполняются последовательно, т.е. в каждую единицу времени активна только одна транзакция.