Введение

- **Излагаются принципы управления данными с помощью языка структурированных запросов SQL на примере различных реляционных СУБД**

Что будет изучено

- Правила Кодда
- Основы SQL
- Нормальные формы.
- Синтаксис основных команд SQL
- Основные возможности и особенности СУБД Microsoft Office Access.

Словарь (СУБД)

- **База данных** (БД, database) - поименованная совокупность структурированных данных, относящихся к определенной предметной области.
- **Предметная область** - некоторая часть реально существующей системы, функционирующая как самостоятельная единица. Полная предметная область может представлять собой экономику страны или группы союзных государств, однако на практике для информационных систем наибольшее значение имеет предметная область масштаба отдельного предприятия или корпорации.
- **Система управления базами данных** (СУБД) - комплекс программных и языковых средств, необходимых для создания и модификации базы данных, добавления, модификации, удаления, поиска и отбора информации, представления информации на экране и в печатном виде, разграничения прав доступа к информации, выполнения других операций с базой.

Словарь (таблица)

- **Реляционная БД** - основной тип современных баз данных. Состоит из таблиц, между которыми могут существовать связи по ключевым значениям.
- **Таблица** базы данных (table) - регулярная структура, которая состоит из однотипных строк (записей, records), разбитых на столбцы (поля, fields).
- В теории реляционных баз данных синоним таблицы - **отношение** (relation), в котором строка называется **кортежем**, а столбец называется **атрибутом**.
- В концептуальной модели реляционной БД аналогом таблицы является сущность (entity), с определенным набором свойств - **атрибутов**, способных принимать определенные значения (набор допустимых значений - **домен**).

1-6 правило Кодда

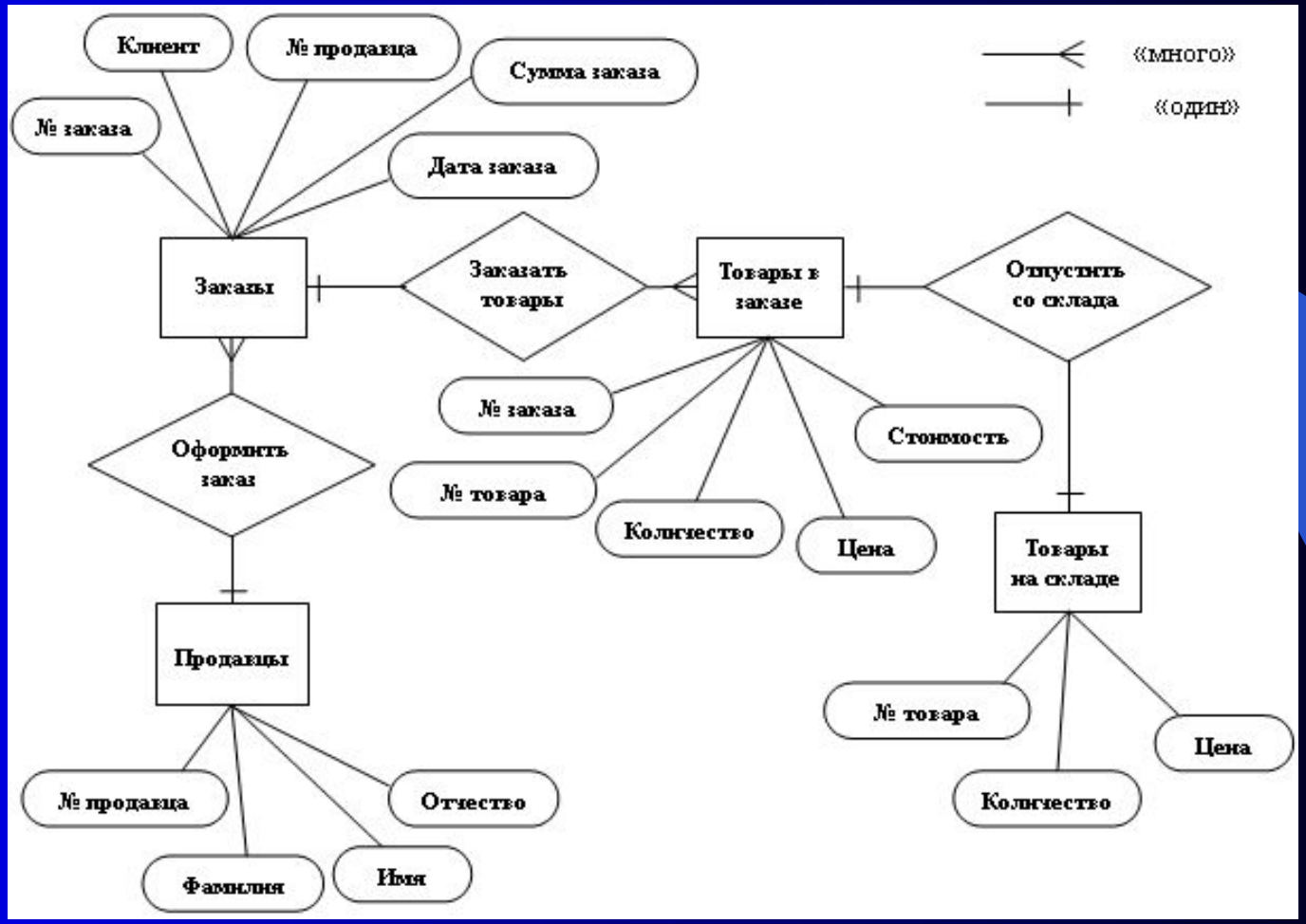
Реляционная СУБД должна быть способна полностью управлять базой данных через ее реляционные возможности.

- **Информационное правило** - вся информация в реляционной БД (включая имена таблиц и столбцов) должна определяться строго как значения в таблицах.
- **Гарантированный доступ** - любое значение в реляционной БД должно быть гарантированно доступно для использования через комбинацию имени таблицы, значения первичного ключа и имени столбца
- **Поддержка пустых значений** (null value) - СУБД должна уметь работать с пустыми значениями (неизвестными или неиспользованными значениями), в отличие от значений по умолчанию и независимо для любых *доменов*.
- **Онлайновый реляционный каталог** - описание БД и ее содержания должны быть представлены на логическом уровне как таблицы, к которым можно применять запросы, используя язык базы данных.
- **Исчерпывающий язык управления данными** - по крайней мере, один из поддерживаемых языков должен иметь четко определенный синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила целостности, авторизацию и транзакции.
- **Правило обновления представлений** (views) - все представления, теоретически обновляемые, могут быть обновлены через систему.

7-12 правило Кодда

- **Вставка, обновление и удаление** - СУБД поддерживает не только запрос на отбор данных, но и вставку, обновление и удаление
- **Физическая независимость данных** - на программы-приложения и специальные программы логически не влияют изменения физических методов доступа к данным и структур хранилищ данных.
- **Логическая независимость данных** - на программы-приложения и специальные программы логически не влияют, в пределах разумного, изменения структур таблиц.
- **Независимость целостности** - язык БД должен быть способен определять правила целостности. Они должны сохраняться в онлайновом справочнике, и не должно существовать способа их обойти.
- **Независимость распределения** - на программы-приложения и специальные программы логически не влияет, первый раз используются данные или повторно.
- **Неподрывность** - невозможность обойти правила целостности, определенные через язык базы данных, использованием языков низкого уровня

Диаграмма «сущность-связи»



Нормализация

- ***Нормализация*** - это формальный метод анализа *отношений* на основе их первичного ключа и существующих связей. Ее задача - это замена одной схемы (или совокупности *отношений*) БД другой схемой, в которой *отношения* имеют более простую и регулярную структуру.

При работе с реляционной моделью для создания *отношений* приемлемого качества достаточно выполнения требований первой нормальной формы.

1НФ

Первая нормальная форма (1НФ) связана с понятиями простого и сложного *атрибутов*.

Простой ***атрибут*** - это *атрибут*, значения которого атомарны (т.е. неделимы).

Сложный *атрибут* может иметь значение, представляющее собой объединение нескольких значений одного или разных доменов.

В первой нормальной форме устраняются повторяющиеся *атрибуты* или группы *атрибутов*, т.е. производится выявление неявных *сущностей*, "замаскированных" под *атрибуты*.

ФИО	Оклад
Иванов Иван Иванович	1 000,00
Петров Петр Петрович	2 000,00
Васильев Василий Васильевич	1 500,00

Поле1
25.10.2008 суббота
26.10.2008 воскресенье

Коллекционер	Хобби
Иванов	значки, монеты, боны
Петров	модели, награды, вымпелы, этикетки
Васильев	значки, тарелки

1НФ

Отношение приведено к 1НФ, если все его атрибуты - простые, т.е. значение *атрибута* не должно быть множеством или повторяющейся группой.

Для приведения таблиц к 1НФ необходимо разбить сложные *атрибуты* на простые, а многозначные *атрибуты* вынести в отдельные *отношения*.

Фамилия	Имя	Отчество	Оклад
Иванов	Иван	Иванович	1 000,00р.
Петров	Петр	Петрович	2 000,00р.
Васильев	Василий	Васильевич	1 500,00р.

Код	Дата	День недели
1	25.10.2008	суббота
2	26.10.2008	воскресенье

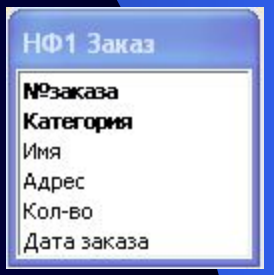
Коллекционер	Хобби
Васильев	тарелки
Васильев	значки
Иванов	боны
Иванов	монеты
Иванов	значки
Петров	этикетки
Петров	вымпелы
Петров	награды
Петров	модели

2 НФ

Вторая нормальная форма (2НФ) применяется к *отношениям* с составными ключами (состоящими из двух и более *атрибутов*) и связана с понятиями функциональной зависимости.

Если в любой момент времени каждому значению *атрибута* A соответствует единственное значение *атрибута* B, то B функционально зависит от A (A B). Атрибут (группа ***атрибутов***) A называется детерминантом.

№заказа	Категория	Имя	Адрес	Кол-во	Дата заказа
123	17	Евгения	Б. Великой Победы, дом №38	30	05.05.2002
123	23	Евгения	Б. Великой Победы, дом №38	50	05.05.2002
129	8	Екатерина	Б. Великой Победы, дом №34	10	30.03.2002
129	20	Екатерина	Б. Великой Победы, дом №34	25	30.03.2002



- Адрес клиента может быть, если есть его заказ;
- Если удалить заказанный продукт, то удаляются данные о клиенте и заказе;
- Если измениться адрес клиента, то нужно редактировать несколько записей.

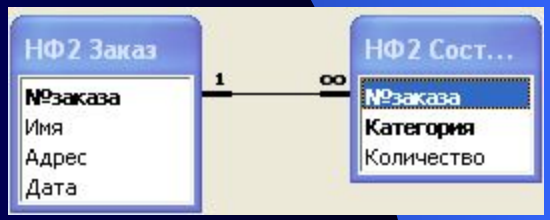
2 НФ

Во второй нормальной форме устраняются *атрибуты*, зависящие только от части уникального ключа. Эта часть уникального ключа определяет отдельную *сущность*.

Отношение находится во 2НФ, если оно приведено к 1НФ и каждый неключевой атрибут функционально полно зависит от составного первичного ключа.

№заказа	Имя	Адрес	Дата
123	Евгения	Б. Великой Победы, дом №38	05.05.2002
129	Екатерина	Б. Великой Победы, дом №34	30.03.2002

№заказа	Категория	Количество
123	17	30
123	23	50
129	8	10
129	20	25



3 НФ

Третья нормальная форма (3НФ) связана с понятием транзитивной зависимости. Пусть A, B, C - *атрибуты* некоторого *отношения*. При этом A B и B C, но обратное соответствие отсутствует, т.е. C не зависит от B или B не зависит от A. Тогда говорят, что C транзитивно зависит от A (A C).

В третьей нормальной форме устраняются *атрибуты*, которые зависят от *атрибутов*, не входящих в уникальный ключ. Эти *атрибуты* являются основой отдельной *сущности*.

3 НФ

Отношение находится в 3НФ, если оно находится во 2НФ и не имеет атрибутов, не входящих в первичный ключ и находящихся в транзитивной зависимости от первичного ключа.

Имя	Адрес
Евгения	Б. Великой Победы, дом №38
Екатерина	Б. Великой Победы, дом №34

№ заказа	Имя	Дата заказа
123	Евгения	05.05.2002
129	Екатерина	30.03.2002

№заказа	Категория	Количество
123	17	30
123	23	50
129	8	10
129	20	25

3 НФ

Отношение находится в 3НФ в том и только в том случае, если все неключевые атрибуты отношения взаимно независимы и полностью зависят от первичного ключа.

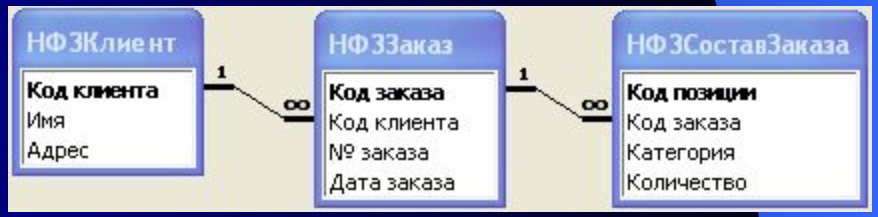
Код клиента	Имя	Адрес
1	Евгения	Б. Великой Победы, дом №38
2	Екатерина	Б. Великой Победы, дом №34

Код позиции	Код заказа	Категория	Количество
1	1	17	30
2	1	23	50
3	2	8	10
4	2	20	25

Код заказа	Код клиента	№ заказа	Дата заказа
1	1	123	05.05.2002
2	2	129	30.03.2002

Имя	Адрес	№ заказа	Дата заказа	Категория	Количество
Евгения	Б. Великой Победы, дом №38	123	05.05.2002	17	30
Евгения	Б. Великой Победы, дом №38	123	05.05.2002	23	50
Екатерина	Б. Великой Победы, дом №34	129	30.03.2002	8	10
Екатерина	Б. Великой Победы, дом №34	129	30.03.2002	20	25

```
SELECT НФ3Клиент.Имя, НФ3Клиент.Адрес, НФ3Заказ.[№ заказа],
НФ3Заказ.[Дата заказа], НФ3СоставЗаказа.Категория, НФ3СоставЗаказа.
Количество
FROM (НФ3Клиент INNER JOIN НФ3Заказ ON НФ3Клиент.[Код
клиента] = НФ3Заказ.[Код клиента]) INNER JOIN НФ3СоставЗаказа ON
НФ3Заказ.[Код заказа] = НФ3СоставЗаказа.[Код заказа];
```



4,5 НФ

Существуют также нормальная форма Бойса-Кодда (НФБК), 4НФ и 5НФ. Однако наибольшее значение имеет 1НФ, т.к. последующие НФ связаны с понятиями о составных ключах и сложных зависимостях от ключей, а на практике встречаются обычно более простые случаи.

Проблемы нормализации

Моделирование структуры базы данных при помощи алгоритма *нормализации* имеет серьезные недостатки:

Методика *нормализации* предполагает первоначальное размещение всех *атрибутов* проектируемой предметной области в одном *отношении*, что является очень неестественной операцией. Интуитивно разработчик сразу проектирует несколько *отношений* в соответствии с обнаруженными *сущностями*. Даже если совершить насилие над собой и создать одно или несколько *отношений*, включив в них все предполагаемые *атрибуты*, то совершенно неясен смысл полученного *отношения*.

Невозможно сразу определить полный список *атрибутов*. Пользователи имеют привычку называть разными именами одни и те же вещи или наоборот, называть одними именами разные вещи.

Для проведения процедуры *нормализации* необходимо выделить зависимости *атрибутов*, что тоже очень нелегко.

Ссылочная целостность

Соблюдение условий ссылочной целостности в реляционной базе данных

- **Правило соответствия внешних ключей первичным** - основное правило соблюдения условий ссылочной целостности. Для каждого значения внешнего ключа должно существовать соответствующее значение первичного ключа в родительской таблице
- Ссылочная целостность может нарушиться в результате операций вставки (добавления), обновления и удаления записей в таблицах. В определении ссылочной целостности участвуют две таблицы - родительская и дочерняя, для каждой из них возможны эти операции, поэтому существует шесть различных вариантов, которые могут привести либо не привести к нарушению ссылочной целостности.
- Для родительской таблицы:
- **Вставка.** Возникает новое значение первичного ключа. Существование записей в родительской таблице, на которые нет ссылок из дочерней таблицы, допустимо, операция **не нарушает ссылочной целостности**.
- **Обновление**. Изменение значения первичного ключа в записи **может привести к нарушению ссылочной целостности**.
- **Удаление**. При удалении записи удаляется значение первичного ключа. Если есть записи в дочерней таблице, ссылающиеся на ключ удаляемой записи, то значения внешних ключей станут некорректными. Операция **может привести к нарушению ссылочной целостности.**
- Для дочерней таблицы:
- **Вставка.** Нельзя вставить запись в дочернюю таблицу, если для новой записи значение внешнего ключа некорректно. Операция **может привести к нарушению ссылочной целостности**.
- **Обновление.** При обновлении записи в дочерней таблице можно попытаться некорректно изменить значение внешнего ключа. Операция **может привести к нарушению ссылочной целостности.**
- **Удаление.** При удалении записи в дочерней таблице **ссылочная целостность не нарушается**.
- Таким образом, ссылочная целостность в принципе может быть нарушена при выполнении одной из четырех операций:
- Обновление записей в родительской таблице.
- Удаление записей в родительской таблице.
- Вставка записей в дочерней таблице.
- Обновление записей в дочерней таблице.

СЦ родительской таблицы

Для родительской таблицы:

- **Вставка.** Возникает новое значение первичного ключа. Существование записей в родительской таблице, на которые нет ссылок из дочерней таблицы, допустимо, операция **не нарушает ссылочной целостности**.
- **Обновление**. Изменение значения первичного ключа в записи **может привести к нарушению ссылочной целостности**.
- **Удаление**. При удалении записи удаляется значение первичного ключа. Если есть записи в дочерней таблице, ссылающиеся на ключ удаляемой записи, то значения внешних ключей станут некорректными. Операция **может привести к нарушению ссылочной целостности.**

СЦ дочерней таблицы

Для дочерней таблицы:

- **Вставка.** Нельзя вставить запись в дочернюю таблицу, если для новой записи значение внешнего ключа некорректно. Операция **может привести к нарушению ссылочной целостности**.
- **Обновление.** При обновлении записи в дочерней таблице можно попытаться некорректно изменить значение внешнего ключа. Операция **может привести к нарушению ссылочной целостности.**
- **Удаление.** При удалении записи в дочерней таблице **ссылочная целостность не нарушается**.

Стратегия поддержания СЦ

Основные:

- **RESTRICT** (ОГРАНИЧИТЬ) - не разрешать выполнение операции, приводящей к нарушению ссылочной целостности.
- **CASCADE** (КАСКАДНОЕ ИЗМЕНЕНИЕ) - разрешить выполнение требуемой операции, но внести при этом необходимые изменения в связанных таблицах так, чтобы не допустить нарушения ссылочной целостности и сохранить все имеющиеся связи. Изменение начинается в родительской таблице и каскадно выполняется в дочерних таблицах.

Дополнительные:

- **IGNORE** (ИГНОРИРОВАТЬ) - разрешить выполнять операцию без проверки ссылочной целостности. В этом случае в дочерней таблице могут появляться некорректные значения внешних ключей, вся ответственность за целостность базы данных ложится на программиста или пользователя.
- **SET NULL** (ЗАДАТЬ ЗНАЧЕНИЕ NULL) - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на null-значения. Эта стратегия имеет два недостатка. Во-первых, для нее требуется разрешение на использование null-значений. Во-вторых, записи дочерней таблицы теряют связь с записями родительской таблицы.
- **SET DEFAULT** (ЗАДАТЬ ЗНАЧЕНИЕ ПО УМОЛЧАНИЮ) - разрешить выполнение требуемой операции, но все возникающие некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию. Достоинство этой стратегии по сравнению с предыдущей в том, что она позволяет не пользоваться null-значениями.

Обозначения

- ::= Равно по определению
- | Необходимость выбора одного из нескольких приведенных значений
- <…> Описанная с помощью *метаязыка* структура языка
- {…} Обязательный выбор некоторой конструкции из списка
- […] Необязательный выбор некоторой конструкции из списка
- [,…n] Необязательная возможность повторения конструкции от нуля до нескольких раз

Типы данных

- *Символьный* **CHAR | VARCHAR**
- *Битовый* **BIT | BIT VARYING**
- *Точные числа* **NUMERIC | DECIMAL | INTEGER | SMALLINT**
- *Округленные числа* **FLOAT | REAL | DOUBLE PRECISION**
- *Дата/время* **DATE | TIME | TIMESTAMP**
- Интервал**INTERVAL**

Получить список всех *типов данных*, включая *пользовательские*, можно из системной таблицы systypes:

SELECT * FROM systypes

Преобразование типов возможно с помощью операторов

CAST(выражение AS тип_данных)

CONVERT(тип_данных[(длина)], выражение [,стиль])

Создание БД

CREATE DATABASE имя_базы_данных [**ON**
[**PRIMARY**] [<определение_файла> [,...n]]
[,<определение_группы> [,...n]]] [**LOG ON**
{<определение_файла>[,...n] }] [**FOR LOAD**
| **FOR ATTACH**]

<определение_файла>::= ([
NAME=логическое_имя_файла,]
FILENAME='физическое_имя_файла'
[,**SIZE**=размер_файла]
[,**MAXSIZE**={max_размер_файла
|**UNLIMITED** }] [,
FILEGROWTH=величина_прироста])[,...n]

Пример создания БД

```
CREATE DATABASE Archive ON PRIMARY ( NAME=Arch1,
    FILENAME='c:\user\data\archdat1.mdf', SIZE=100MB,
    MAXSIZE=200, FILEGROWTH=20), (NAME=Arch2,
    FILENAME='c:\user\data\archdat2.mdf', SIZE=100MB,
    MAXSIZE=200, FILEGROWTH=20), (NAME=Arch3,
    FILENAME='c:\user\data\archdat3.mdf', SIZE=100MB,
    MAXSIZE=200, FILEGROWTH=20) LOG ON
    (NAME=Archlog1, FILENAME='c:\user\data\archlog1.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20),
    (NAME=Archlog2, FILENAME='c:\user\data\archlog2.ldf',
    SIZE=100MB, MAXSIZE=200, FILEGROWTH=20)
```

Изменение БД

- <изменение_базы_данных> ::= **ALTER DATABASE** имя_базы_данных { **ADD FILE** <определение_файла>[,...n] [**TO FILEGROUP** имя_группы_файлов] | **ADD LOG FILE** <определение_файла>[,...n] | **REMOVE FILE** логическое_имя_файла | **ADD FILEGROUP** имя_группы_файлов | **REMOVE FILEGROUP** имя_группы_файлов | **MODIFY FILE** <определение_файла> | **MODIFY FILEGROUP** имя_группы_файлов <свойства_группы_файлов>}

Основные объекты структуры базы данных SQL-сервера

- **Tables** Таблицы базы *данных*, в которых хранятся собственно данные
- **Views** Просмотры (виртуальные таблицы) для отображения *данных* из таблиц
- **Stored Procedures** Хранимые процедуры
- **Triggers** Триггеры – специальные хранимые процедуры, вызываемые при изменении *данных* в таблице
- **User Defined function** Создаваемые пользователем функции
- **Indexes** Индексы – дополнительные структуры, призванные повысить производительность работы с *данными*
- **User Defined Data Types** Определяемые пользователем *типы данных*
- **Keys** Ключи – один из видов ограничений целостности данных
- **Constraints** Ограничение целостности – *объекты* для обеспечения логической целостности данных
- **Users** Пользователи, обладающие доступом к базе данных
- **Roles** Роли, позволяющие объединять пользователей в группы
- **Rules** Правила базы *данных*, позволяющие контролировать логическую целостность данных
- **Defaults** Умолчания или стандартные установки базы данных

Create table

CREATE [{ **GLOBAL** | **LOCAL** }] **TEMPORARY] TABLE** имя_таблицы ({ column | [table_constraint] } . , .. [**ON COMMIT** { **DELETE** | **PRESERVE**} **ROWS**]);

column определяется как

имя_поля {domain | datatype [size]} [column_constraint:] [**DEFAULT** default_value] [**COLLATE** collate_value]

Ограничения столбца

- ***NOT NULL*** - в любой добавляемой или изменяемой строке столбец всегда должен иметь значение, отличное от NULL.
- ***UNIQUE*** - все значения столбца должны быть уникальны.
- ***PRIMARY KEY*** - устанавливает один столбец как первичный ключ и одновременно подразумевает, что все значения столбца будут уникальны.
- ***CHECK*** (condition) - указываемое в скобках условие использует для сравнения значение столбца и возвращает TRUE, FALSE или UNKNOWN. Если при попытке выполнения SQL-оператора возвращаемое значение равно FALSE, то оператор выполнен не будет.
- ***REFERENCES*** table (fields_list) - ограничение требует совпадения значений столбцов данной таблицы с указанными столбцами родительской таблицы.

Ограничения таблицы

- *CHECK* (condition) - указываемое в скобках условие использует для сравнения значение столбца и возвращает TRUE, FALSE или UNKNOWN. Если при попытке выполнения SQL-оператора возвращаемое значение равно FALSE, то оператор выполнен не будет.
- ***FOREIGN KEY*** (fields_list) - это ограничение по внешнему ключу аналогично ограничению *REFERENCES* для столбцов и гарантирует, что все значения, указанные во внешнем ключе, будут соответствовать значениям родительского ключа, обеспечивая ссылочную целостность. Следует отметить, что типы данных столбцов, используемых в этом ограничении, должны совпадать, а типы таблиц (постоянная базовая таблица, глобальная временная таблица, локальная временная таблица) родительского и внешнего ключа - соответствовать друг другу.

Пример создания

- **CREATE TABLE**

  Товар

  (Название **VARCHAR**(50) **NOT NULL**,
  Цена **MONEY NOT NULL**,
  Тип **VARCHAR(50) NOT NULL**,
  Сорт **VARCHAR(50)**,
  ГородТовара **VARCHAR(50))**

Изменение таблицы

- **ALTER TABLE** имя_таблицы
  {[**ALTER COLUMN** имя_столбца
  {новый_тип_данных [(точность[,
  масштаб])] [**NULL** | **NOT NULL**]}] |
  ADD { [имя_столбца тип_данных] |
  имя_столбца AS выражение } [,...n] |
  DROP {**COLUMN** имя_столбца}[,...n] }

Select

SELECT [**ALL** | **DISTINCT**]
{*|[имя_столбца [**AS** новое_имя]]} [,...n]
FROM имя_таблицы [[**AS**] псевдоним]
[,...n] [**WHERE** <условие_поиска>]
[**GROUP BY** имя_столбца [,...n]]
[**HAVING** <критерии выбора групп>]
[**ORDER BY** имя_столбца [,...n]]

Очередность выполнения

- FROM – определяются имена используемых таблиц;
- WHERE – выполняется *фильтрация строк* объекта в соответствии с заданными условиями;
- GROUP BY – образуются *группы строк* , имеющих одно и то же значение в указанном столбце;
- HAVING – фильтруются группы строк объекта в соответствии с указанным условием;
- SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
- ORDER BY – определяется упорядоченность результатов выполнения операторов.

Select … Where

- *Сравнение*: сравниваются результаты вычисления одного выражения с результатами вычисления другого.
- *Диапазон*: проверяется, попадает ли результат вычисления выражения в заданный *диапазон* значений.
- *Принадлежность множеству*: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.
- *Соответствие шаблону*: проверяется, отвечает ли некоторое строковое значение заданному шаблону.
- *Значение* NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Select … ORDER BY

SELECT Клиент.Фамилия, Клиент.Фирма
FROM Клиент **ORDER BY** Клиент.
Фамилия

SELECT Клиент.Фирма, Клиент.Фамилия
FROM Клиент **ORDER BY** Клиент.
Фирма, Клиент.Фамилия **DESC**

Выборка

Операция ***выборки*** - построение горизонтального подмножества, т.е. подмножества кортежей, обладающих заданными свойствами.

Операция *выборки* работает с одним отношением R и определяет результирующее отношение, которое содержит только те кортежи (строки) отношения R, которые удовлетворяют заданному условию F (предикату).

σF(R) или σпредикат(R).

Операция выборки в SQL.

Выборка σ(a2=1)(R)={(a, 1), (b, 1)} записывается следующим образом:

SELECT a1, a2 **FROM** R **WHERE** a2=1

Проекция

Операция ***проекции*** - построение вертикального подмножества отношения, т.е. подмножества кортежей, получаемого выбором одних и исключением других атрибутов.

Операция *проекции* работает с одним отношением R и определяет новое отношение, которое содержит вертикальное подмножество отношения R, создаваемое посредством извлечения значений указанных атрибутов и исключения из результата строк-дубликатов.

Πa1, a2,... an(R)

Операция проекции в SQL.

Проекция Πb2(S)={(h), (g)) записывается следующим образом:

SELECT b2 **FROM S**

Декартово произведение

Декартово произведение RxS двух отношений (двух таблиц) определяет новое отношение - результат конкатенации (т.е. сцепления) каждого кортежа (каждой записи) из отношения R с каждым кортежем (каждой записью) из отношения S.

RxS={(a, 1, 1, h), (a, 2, 1, h), (b, 1, 1, h), ... }

SELECT R.a1, R.a2, S.b1, S.b2 **FROM** R, S

Соединение

Соединение - это процесс, когда две или более таблицы объединяются в одну. Способность объединять информацию из нескольких таблиц или запросов в виде одного логического набора данных обусловливает широкие возможности SQL.

В языке SQL для задания типа *соединения* таблиц в логический набор записей, из которого будет выбираться необходимая информация, используется операция JOIN в предложении FROM.

Формат операции:

FROM имя_таблицы_1 {**INNER | LEFT | RIGHT**} **JOIN** имя_таблицы_2 **ON** условие_соединения

Варианты соединения

- тета-соединение
- *соединение по эквивалентности*
- естественное соединение
- *внешнее соединение*
- *полусоединение*

Операция тета-соединения

Операция тета-соединения определяет отношение, которое содержит кортежи из *декартова произведения* отношений R и S, удовлетворяющие предикату F. Предикат F имеет вид R.ai Θ S.bj, где вместо Θ может быть указан один из операторов сравнения (>, >=, <, <=, =, <>).

Если предикат F содержит только оператор равенства (=), то *соединение* называется *соединением по эквивалентности*.

Операция тета-соединения в языке SQL называется INNER JOIN (внутреннее *соединение*) и используется, когда нужно включить все строки из обеих таблиц, удовлетворяющие условию объединения. Внутреннее *соединение* имеет место и тогда, когда в предложении WHERE сравниваются значения полей из разных таблиц. В этом случае строится *декартово произведение* строк первой и второй таблиц, а из полученного набора данных отбираются записи, удовлетворяющие условиям объединения.

SELECT R.a1, R.a2, S.b1, S.b2 **FROM** R, S **WHERE** R.a2=S.b1

или

SELECT R.a1, R.a2, S.b1, S.b2 **FROM** R **INNER JOIN** S ON R.a2=S.b1

Естественное соединение

- Естественным *соединением* называется *соединение по эквивалентности* двух отношений R и S, выполненное по всем общим атрибутам, из результатов которого исключается по одному экземпляру каждого общего атрибута.

SELECT R.a1, R.a2, S.b2 **FROM** R, S **WHERE** R.a2=S.b1

или

SELECT R.a1, S.b1, S.b2 **FROM** R **INNER JOIN** S **ON** R.a2=S.b1

SELECT Товар.Название, Сделка.Количество, Сделка. Дата, Клиент.Фирма
FROM
Клиент **INNER JOIN**
(Товар **INNER JOIN** Сделка **ON** Товар. КодТовара=Сделка.КодТовара)
ON Клиент.КодКлиента=Сделка.КодКлиента

Внешние и внутренние соединения

Внешнее соединение похоже на внутреннее, но в результирующий набор данных включаются также записи ведущей таблицы *соединения*, которые объединяются с пустым множеством записей другой таблицы.

Какая из таблиц будет ведущей, определяет вид *соединения*.

LEFT - левое *внешнее соединение*, ведущей является таблица, расположенная слева от вида *соединения*;

RIGHT - правое *внешнее соединение*, ведущая таблица расположена справа от вида *соединения*.

Левое и правое внешнее соединение

Левым *внешним соединением* называется *соединение*, при котором кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения S, также включаются в результирующее отношение.

SELECT R.a1, R.a2, S.b1, S.b2 **FROM** R **LEFT JOIN** S **ON** R.a2=S.b1

Существует и правое *внешнее соединение* R S, называемое так потому, что в результирующем отношении содержатся все кортежи правого отношения. Кроме того, имеется и полное *внешнее соединение*, в его результирующее отношение помещаются все кортежи из обоих отношений, а для обозначения несовпадающих значений кортежей в нем используются определители NULL.

SELECT R.a1, R.a2, S.b1, S.b2 **FROM** R **RIGHT JOIN** S **ON** R.a2=S.b1

Полусоединение

Операция ***полусоединения*** определяет отношение, содержащее те кортежи отношения R, которые входят в *соединение* отношений R и S.

SELECT R.a1, R.a2 **FROM** R, S **WHERE** R.a2=S.b1

или

SELECT R.a1, R.a2 **FROM** R **INNER JOIN** S ON R.a2=S.b1

Объединение

Объединение (UNION) R S отношений R и S можно получить в результате их конкатенации с образованием одного отношения с исключением кортежей-дубликатов. При этом отношения R и S должны быть совместимы, т.е. иметь одинаковое количество полей с совпадающими типами данных. Иначе говоря, отношения должны быть совместимы по *объединению*.

Объединением двух таблиц R и S является таблица, содержащая все строки, которые имеются в первой таблице R, во второй таблице S или в обеих таблицах сразу.

SELECT R.a1, R.a2 **FROM** R **UNION SELECT** S.b2, S.b1 **FROM** S

Пересечение

Операция ***пересечения*** (INTERSECT) R S=R-(R-S) определяет отношение, которое содержит кортежи, присутствующие как в отношении R, так и в отношении S. Отношения R и S должны быть совместимы по *объединению*.

Пересечением двух таблиц R и S является таблица, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.

SELECT R.a1, R.a2 **FROM** R,S **WHERE** R.a1=S.b1 **AND** R.a2=S.b2

или

SELECT R.a1, R.a2 **FROM** R **WHERE** R.a1 **IN** (**SELECT** S.b1 **FROM** S **WHERE** S.b1=R.a1) **AND** R.a2 **IN** (**SELECT** S.b2 **FROM** S **WHERE** S.b2=R.a2)

Разность

Разность (EXCEPT) R-S двух отношений R и S состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. Причем отношения R и S должны быть совместимы по *объединению*.

Разностью двух таблиц R и S является таблица, содержащая все строки, которые присутствуют в таблице R, но отсутствуют в таблице S.

SELECT R.a1, R.a2 **FROM** R **WHERE NOT EXISTS (SELECT** S.b1,S.b2 **FROM** S **WHERE** S.b1=R.a2 **AND** S.b2=R.a1)

Вычисляемые поля

В общем случае для создания *вычисляемого (производного) поля* в списке SELECT следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. Можно указать имя любого столбца (поля) таблицы или запроса, но использовать имя столбца только той таблицы или запроса, которые указаны в списке предложения FROM соответствующей инструкции. При построении сложных выражений могут понадобиться скобки.

Стандарты SQL позволяют явным образом задавать имена столбцов результирующей таблицы, для чего применяется фраза AS.

Рассчитать общую стоимость для каждой сделки. Этот запрос использует расчет результирующих столбцов на основе арифметических выражений.

SELECT Товар.Название, Товар.Цена, Сделка.Количество, Товар.Цена*Сделка.Количество **AS** Стоимость **FROM** Товар **INNER JOIN** Сделка **ON** Товар.КодТовара=Сделка.КодТовара

Итоговые функции

С помощью *итоговых (агрегатных) функций* в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные *итоговые функции*:

Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;

Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;

Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.

Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Select … Group By

Часто в запросах требуется формировать промежуточные итоги, что обычно отображается появлением в запросе фразы "для каждого...". Для этой цели в операторе SELECT используется предложение GROUP BY. Запрос, в котором присутствует GROUP BY, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная суммарная строка. Стандарт SQL требует, чтобы предложение SELECT и фраза GROUP BY были тесно связаны между собой. При наличии в операторе SELECT фразы GROUP BY каждый элемент списка в предложении SELECT должен иметь единственное значение для всей группы. Более того, предложение SELECT может включать только следующие типы элементов: имена полей, *итоговые функции*, константы и выражения, включающие комбинации перечисленных выше элементов.

Все имена полей, приведенные в списке предложения SELECT, должны присутствовать и во фразе GROUP BY - за исключением случаев, когда имя столбца используется в *итоговой функции*. Обратное правило не является справедливым - во фразе GROUP BY могут быть имена столбцов, отсутствующие в списке предложения SELECT.

Если совместно с GROUP BY используется предложение WHERE, то оно обрабатывается первым, а *группированию* подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом SQL определено, что при проведении *группирования* все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значение NULL и идентичные значения во всех остальных непустых группируемых столбцах, они помещаются в одну и ту же группу.

Вычислить средний объем покупок, совершенных каждым покупателем.

SELECT Клиент.Фамилия, Avg(Сделка.Количество) **AS** Среднее_количество **FROM** Клиент **INNER JOIN** Сделка **ON** Клиент.КодКлиента=Сделка.КодКлиента **GROUP BY** Клиент.Фамилия

Select … having

При помощи HAVING отражаются все предварительно сгруппированные посредством GROUP BY блоки данных, удовлетворяющие заданным в HAVING условиям. Это дополнительная возможность "профильтровать" выходной набор.

Условия в HAVING отличаются от условий в WHERE:

HAVING исключает из результирующего набора данных группы с результатами агрегированных значений;

WHERE исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;

в условии поиска WHERE нельзя задавать агрегатные функции.

Определить фирмы, у которых общее количество сделок превысило три.

SELECT Клиент.Фирма, Count(Сделка.Количество) **AS** Количество_сделок **FROM** Клиент **INNER JOIN** Сделка **ON** Клиент.КодКлиента=Сделка.КодКлиента **GROUP BY** Клиент.Фирма **HAVING** Count(Сделка.Количество)>3

Подзапросы

Подзапрос **– это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст *подзапроса* должен быть заключен в скобки. К *подзапросам* применяются следующие правила и ограничения:**

- **фраза ORDER BY не используется, хотя и может присутствовать во внешнем *подзапросе*;**
- **список в предложении SELECT состоит из имен отдельных столбцов или составленных из них выражений – за исключением случая, когда в *подзапросе* присутствует ключевое слово *EXISTS*;**
- **по умолчанию имена столбцов в *подзапросе* относятся к таблице, имя которой указано в предложении FROM. Однако допускается ссылка и на столбцы таблицы, указанной во фразе FROM внешнего запроса, для чего применяются квалифицированные имена столбцов (т.е. с указанием таблицы);**
- **если *подзапрос* является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операции.**

Существует два типа *подзапросов*:

- **Скалярный *подзапрос* возвращает единственное значение. В принципе, он может использоваться везде, где требуется указать единственное значение.**
- **Табличный *подзапрос* возвращает множество значений, т.е. значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Он возможен везде, где допускается наличие таблицы.**

Подзапросы, возвращающие единичное значение

Определить дату продажи максимальной партии товара.

SELECT Дата, Количество FROM Сделка WHERE Количество=(SELECT Max(Количество) FROM Сделка)

Определить даты сделок, превысивших по количеству товара среднее значение и указать для этих сделок превышение над средним уровнем.

SELECT Дата, Количество, Количество-(SELECT Avg(Количество) FROM Сделка) AS Превышение FROM Сделка WHERE Количество> (SELECT Avg(Количество) FROM Сделка)

Подзапросы, возвращающие множество значений

Во многих случаях значение, подлежащее сравнению в предложениях WHERE или HAVING, представляет собой не одно, а несколько значений. Вложенные *подзапросы* генерируют непоименованное промежуточное отношение, временную таблицу. Оно может использоваться только в том месте, где появляется в *подзапросе*. К такому отношению невозможно обратиться по имени из какого-либо другого места запроса. Применяемые к *подзапросу* операции основаны на тех операциях, которые, в свою очередь, применяются к множеству, а именно:

- { WHERE | HAVING } выражение [NOT] *IN* (*подзапрос*);
- { WHERE | HAVING } выражение оператор_сравнения { ALL | SOME | ANY }(*подзапрос*);
- {WHERE | HAVING } [NOT] *EXISTS* (*подзапрос*);

Использование операций IN и NOT IN

Оператор *IN* используется для сравнения некоторого значения со списком значений, при этом проверяется, входит ли значение в предоставленный список или сравниваемое значение не является элементом представленного списка.

Пример 7.7. Определить список товаров, которые имеются на складе.

- SELECT Название FROM Товар WHERE КодТовара In (SELECT КодТовара FROM Склад)

Exist

Ключевые слова *EXISTS* и NOT EXISTS предназначены для использования только совместно с *подзапросами*. Результат их обработки представляет собой логическое значение TRUE или FALSE. Для ключевого слова *EXISTS* результат равен TRUE в том и только в том случае, если в возвращаемой *подзапросом* результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица *подзапроса* пуста, результатом обработки операции *EXISTS* будет значение FALSE. Для ключевого слова NOT EXISTS используются правила обработки, обратные по отношению к ключевому слову *EXISTS*. Поскольку по ключевым словам *EXISTS* и NOT EXISTS проверяется лишь наличие строк в результирующей таблице *подзапроса*, то эта таблица может содержать произвольное количество столбцов.

Определить список имеющихся на складе товаров

SELECT Название **FROM** Товар **WHERE** КодТовара **In** (**SELECT** КодТовара **FROM** Склад)

или

SELECT Название **FROM** Товар **WHERE EXISTS** (**SELECT** КодТовара **FROM** Склад **WHERE** Товар.КодТовара=Склад.КодТовара)

Определить список отсутствующих на складе товаров.

SELECT Название **FROM** Товар **WHERE** КодТовара **Not In** (**SELECT** КодТовара **FROM** Склад)

или

SELECT Название **FROM** Товар **WHERE NOT EXISTS** (**SELECT** КодТовара **FROM** Склад **WHERE** Товар.КодТовара=Склад.КодТовара)

Модификация данных

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью *запросов действия* пользователь может добавить, удалить или обновить блоки данных. Существует три вида *запросов действия*:

- **INSERT INTO** – *запрос добавления*;
- **DELETE** – *запрос удаления*;
- **UPDATE** – *запрос обновления*.

Запрос добавления

Оператор INSERT применяется для *добавления записей* в таблицу. Формат оператора:

<оператор_вставки>::=INSERT INTO <имя_таблицы> [(имя_столбца [,...n])] {VALUES (значение[,...n])| <SELECT_оператор>}

Список значений (VALUES) должен следующим образом соответствовать списку столбцов:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
- типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример добавления

```
INSERT INTO Товар
    (Название, Тип, Цена)
    VALUES(" Славянский ", " шоколад ", 12)
Или
INSERT INTO Товар
    VALUES (" Славянский ", " шоколад ", 12)
Или
INSERT INTO Итог
    (Название, Месяц, Стоимость )
SELECT Товар.Название, Month(Сделка.Дата) AS Месяц,
    Sum(Товар.Цена*Сделка.Количество) AS Стоимость
FROM Товар INNER JOIN
    Сделка ON Товар.КодТовара= Сделка.КодТовара
GROUP BY Товар.Название, Month(Сделка.Дата)
```

Запрос удаления

Оператор **DELETE** предназначен для *удаления группы записей* из таблицы.

Формат оператора:

<оператор_удаления> ::=**DELETE FROM** <имя_таблицы>[**WHERE** <условие_отбора>]

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

DELETE FROM Сделка **WHERE Year**(Сделка. Дата)=**Year(GETDATE**())-1

Запрос обновления

Оператор UPDATE применяется для *изменения* значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

<оператор_изменения> ::= **UPDATE** имя_таблицы **SET** имя_столбца= <выражение>[,...n] [**WHERE** <условие_отбора>]

Примеры:

UPDATE Товар
SET Товар.Цена=140, Товар.Остаток=20
WHERE Товар.Сорт=" Первый "

или

UPDATE Сделка
SET Сделка.Количество= Сделка.Количество*1.1
WHERE Сделка.Количество=
(**SELECT** Max(Сделка.Количество) **FROM** Сделка)

Определение представления

Представления, или *просмотры* (VIEW), представляют собой временные, производные (иначе - виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним. Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. *Представление* не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц. Применение *представлений* позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности. Содержимое *представлений* выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в *представлении* автоматически меняются. *Представление* - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде. Результат выполнения этого запроса в каждый момент времени становится содержанием *представления*. У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

<определение_просмотра> ::= { CREATE| ALTER} VIEW имя_просмотра [(имя_столбца [,...n])] [WITH ENCRYPTION] AS SELECT_оператор [WITH CHECK OPTION]

Примеры представлений

```
CREATE VIEW view1 AS
    SELECT КодКлиента, Фамилия,
    ГородКлиента
    FROM Клиент
    WHERE ГородКлиента='Москва'
```

Пример обращения:

```
SELECT * FROM view1
```

Понятие функции пользователя

Функции пользователя представляют собой самостоятельные объекты базы данных, такие, например, как *хранимые процедуры* или триггеры. *Функция пользователя* располагается в определенной базе данных и доступна только в ее контексте.

В SQL Server имеются следующие классы *функций пользователя*:

- ***Scalar*** – *функции* возвращают обычное скалярное значение, каждая может включать множество команд, объединяемых в один блок с помощью конструкции BEGIN...END;
- **Inline** – *функции* содержат всего одну команду SELECT и возвращают пользователю набор данных в виде значения *типа данных TABLE*;
- ***Multi-statement*** – *функции* также возвращают пользователю значение *типа данных TABLE*, содержащее набор данных, однако в теле *функции* находится множество команд SQL (INSERT, UPDATE и т.д.). Именно с их помощью и формируется набор данных, который должен быть возвращен после выполнения *функции*.

Пользовательские функции сходны с *хранимыми процедурами*, но, в отличие от них, могут применяться в запросах так же, как и системные *встроенные функции*. *Пользовательские функции*, возвращающие таблицы, могут стать альтернативой просмотрам. Просмотры ограничены одним выражением SELECT, а *пользовательские функции* способны включать дополнительные выражения, что позволяет создавать более сложные и мощные конструкции.

Источники информации

- http://www.intuit.ru/department/database/sql/
- http://www.sql.ru