SQL – язык структурированных запросов

Типы команд SQL

- DDL – язык определения данных,
- DML – язык манипулирования данными,
- DQL – язык запросов,
- DCL – язык управления данными,
- команды администрирования данных,
- команды управления транзакциями.

Data Definition Language (DDL)

Основные команды языка DDL:

- **CREATE TABLE**
- **DROP TABLE**
- **ALTER TABLE**
- **CREATE INDEX**
- **DROP INDEX**
- **ALTER INDEX**

Data Definition Language (**DDL**) определение данных

Создание таблиц

CREATE TABLE имя таблицы
[CHECK(условие)](имя_поля тип данных [NULL|NOT NULL][,…]), где
[CHECK(условие)] – ограничение на значение столбца.

Например:

CREATE TABLE stud (ФИО varchar(20) NOT NULL,

Дисциплина varchar(20) NOT NULL,

Оценка smallint NOT NULL);

CREATE TABLE

Кроме того должны включаться средства поддержки целостности данных

PRIMARY KEY

FOREIGN KEY ()

ON UPDATE CASCADE

ON DELETE CASCADE

CREATE TABLE

Пример1:

```
CREATE TABLE Клиент
(КодКлиента INT IDENTITY (1,1) PRIMARY
   KEY,
Фирма VARCHAR(50) NOT NULL,
ФИО VARCHAR(50) NOT NULL,
Город VARCHAR(50) NOT NULL,
Телефон CHAR(10) NOT NULL
   CHECK(Телефон LIKE '[1-9][0-9][0-9]-
   [0-9][0-9]- [0-9][0-9]'))
```

CREATE TABLE

Пример2:

```
CREATE TABLE Заказ
(КодЗаказа INT IDENTITY (1,1) PRIMARY KEY,
КодКлиента INT NOT NULL,
КодТовара INT NOT NULL,
Количество INT NOT NULL DEFAULT 0,
Дата DATETIME NOT NULL DEFAULT GETDATE(),
CONSTRAINT fk_Товар
FOREIGN KEY(КодТовара) REFERENCES Товар,
CONSTRAINT fk_Клиент
FOREIGN KEY(КодКлиента) REFERENCES Клиент)
```

Изменение структуры таблицы

- **Для добавления столбцов в таблицу**

ALTER TABLE *<имя таблицы>* **ADD** (*<имя столбца> <тип данных> <размер>*)

- **Возможно изменение описания столбцов**

ALTER TABLE *<имя таблицы>* **MODIFY** *<имя столбца> <тип данных> <размер/точность>*

Возможно изменение описания столбцов, если

- столбец пуст, то можно изменить тип данных и размер/точность;
- столбец заполнен, то можно размер/точность можно увеличить, но нельзя понизить;
- ни одно значение столбца не содержит NULL, можно установить NOT NULL.

Разрешается изменять значения по умолчанию.

Data Manipulation Language (**DML**)

Используется для манипулирования информацией внутри объектов РБД.

Содержит всего три команды:

- **INSERT**
- **UPDATE**
- **DELETE**

Data Query Language (**DQL**)

Имеет всего один оператор, но зато какой?!

SELECT

Оператор **SELECT**

Предназначен для выборки информации из таблиц БД и представления нужным образом результата.

Для выборки необходимо указать как минимум две вещи:

- что выбрать,
- откуда выбрать.

Data Control Language (**DCL**)

- Позволяет управлять доступом к информации, хранящейся внутри БД,
- Используется для создания объектов, связанных с доступом к данным,
- Служит для контроля над привилегиями пользователей

GRANT, REVOKE

Команды управления транзакциями

- **COMMIT** – для сохранения изменений,
- **ROLLBACK** - для отмены изменений,
- **SAVEPOINT** – для установки точек возврата,
- **SET TRANSACTION** – для установки режима транзакции

Преимущества языка SQL

- Стандартность (стандартизовано международными организациями),
- Независимость от конкретной СУБД,
- Возможность использования как для локальных так и для многопользовательских СУБД,
- Реляционная основа языка, а потому простота изучения,
- Возможность создания интерактивных запросов,

Преимущества языка SQL (продолжение)

- Возможность программного доступа к БД – язык легко использовать в приложениях,
- Возможности, предоставляемые представлениями,
- Возможность манипулировать структурой БД,
- Поддержка архитектуры клиент-сервер.

Типы данных языка SQL

Символьные:

- char (длина) 1 символ - 1 байт
- varchar (длина) символьное поле переменной длины
- nchar (длина) символьное поле UNICODE
- nvarchar символьное поле переменной длины UNICODE
- UNICODE
- 1символ = 2 байта

Числовые:

- Int - целое 4 байта
- Smallint - целое 2 байта
- Datetime - дата/время (~до микросекунд)
- Smalldatetime- дата/время (~1 минута, 1.01.1900-6.06.2079).

Отсутствующие данные

- Для обозначения неизвестного в данный момент времени значения атрибута используется ключевое слово **NULL**.
- Использования в качестве аргумента функций count и AVG атрибутов, имеющих значение NULL, дает правильный результат.

Встроенные функции

ISNUMERIC(выражение)	Определяет, имеет ли выражение числовой тип данных
SIGN(число)	Определяет знак числа
RAND(целое число)	Вычисляет случайное число с плавающей запятой в интервале от 0 до 1
ROUND(число, точность)	Выполняет округление числа с указанной точностью
POWER(число, степень)	Возводит число в степень
SQRT(число)	Извлекает квадратный корень из числа
LEN(строка)	Вычисляет длину строки в символах
LTRIM(строка)	Удаляет пробелы в начале строки
RTRIM(строка)	Удаляет пробелы в конце строки
LEFT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого левого символа
RIGHT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого правого символа
GETDATE()	Возвращает текущее системное время
ISDATE(строка)	Проверяет строку на соответствие одному из форматов даты и времени
DATEADD(тип, число, дата)	Прибавляет к дате указанное число единиц заданного типа (год, месяц, день, час и т.п.)
CAST(выражение AS	Преобразование типов

Функция **IDENTITY**

Для типов данных int или decimal можно создать автоинкрементный столбец, чьи значения будут гарантировано уникальными, но он должен быть определен как NOT NULL. Такой столбец удобен в качестве первичного ключа.

IDENTITY(n,m), где n-начальное значение, m-приращение.

Получение информации о типах данных

- Список всех используемых типов данных можно узнать из системной таблицы systypes

SELECT * FROM systypes

Приступим к изучению оператора 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 <условие для
группы>]
[ORDER BY имя столбца [,…n]]

ЧТО, СТРАШНО?

Нет, все достаточно просто, если разобраться.

Начнем с простого случая, когда данные извлекаются из одной таблицы.

Итак, что главное.

SELECT *что вывести*
(какие столбцы)
FROM *откуда вывести (из каких таблиц)*

Примеры

1. Выбрать номера зачеток и фамилии всех студентов

```
SELECT N_зачетки, ФИО
FROM Студент;
```

2. Выбрать все данные о преподавателях из таблицы преподаватели

```
SELECT *
FROM Преподаватели
```

Вычисляемые поля

SELECT [список полей,]
выражение [**AS** имя поля результата]
FROM …
Следует помнить, что полученный результат не сохраняется в таблице.

Сортировка выбранных данных – предложение **ORDER BY**

Предложение **ORDER BY** должно быть последним в операторе SELECT.

Сортировать можно по нескольким столбцам.

По умолчанию сортировка по возрастанию (**ASC**).

Для сортировки по убыванию необходимо в конце указать ключевое слово **DESCENDING** сокращенно **DESC**

ПРИМЕРЫ:

1. Вывести по алфавиту фамилии всех студентов и номера их зачеток

```
SELECT ФИО, N_зачетки
FROM Студент
ORDER BY ФИО;
```

ПРИМЕРЫ:

2. Вывести список студентов по убыванию значений поля «год рождения»

```
SELECT ФИО, N_зачетки, год_рождения
FROM Студент
ORDER BY ФИО, год_рождения DESC ;
```

Предложение **WHERE** для фильтрации записей в соответствии с условием

Существует 5 основных типов условий поиска:

- Сравнение
- Попадание в заданный диапазон
- Принадлежность множеству,
- Соответствие строкового значения заданному шаблону
- Проверка на значение NULL

СРАВНЕНИЕ

WHERE логическое выражение

Например:

SELECT *

FROM Ведомость

WHERE Оценка>3

Логическое выражение может быть сложным

Порядок действий при вычислении выражений

- Выражение вычисляется слева направо.
- Сначала вычисляется то, что в скобках.
- Логические операции выполняются в следующей последовательности сначала NOT затем AND потом OR.

Диапазон в WHERE

- Оператор **BETWEEN** используется для поиска значения внутри диапазона
- **BETWEEN** минимальное значение диапазона **AND** максимальное значение диапазона (граничные значения включаются)
- **NOT BETWEEN** – для поиска значений вне диапазона

Диапазон в WHERE

ПРИМЕР:

SELECT Наименование, Цена

FROM Товар

WHERE Цена **BETWEEN** 500 **AND** 1500

Принадлежность множеству

Оператор **IN(список заданных значений)** проверяет соответствие списку заданных значений,

NOT IN – наоборот несоответствие.

Например:

```
SELECT *
FROM Ведомость
WHERE оценка IN(4,5);
```

Соответствие строкового значения заданному шаблону

Используется оператор **LIKE** ”шаблон”.

В шаблоне используются следующие символы:

% -заменяет любое количество любых символов;

_-заменяет один символ;

[]-предлагает набор символов;

[^]-предлагает все символы кроме указанных.

Например:

Найти студентов, у которых в номере телефона:

- Третья цифра 8

…**WHERE** Студент.Телефон **LIKE ”_ _8%”**

- Вторая цифра 5 или 9

WHERE Студент.Телефон **LIKE ”_[59]%”**

- Вторая цифра 5,6 или 7

WHERE Студент.Телефон **LIKE ”_[5-7]%”**

- Встречается 23

WHERE Студент.Телефон **LIKE ”%23% ”**

продолжение

WHERE Дисциплина **LIKE** ”математика**%** ”

WHERE Дисциплина **LIKE** ”**%** математика**%** ”

WHERE Дисциплина **LIKE** ”м**%**а ”

Во всех этих случаях поиск идет медленно особенно, если метасимвол используется вначале шаблона.

Проверка на значение NULL

- Найти студентов без телефона:

SELECT ФИО, Телефон
FROM Студент
WHERE Телефон **IS NULL;**

- Найти студентов с телефоном:

…
WHERE Телефон **IS NOT NULL;**

Вычисления в SELECT

SELECT список полей, выражение **AS** имя поля-результата.

Например, вывести список фамилий студентов с указанием только года рождения:

SELECT ФИО, **YEAR**(дата_рождения) **AS** Год_Рождения

FROM СТУДЕНТЫ;

Подведение итогов в запросах.

SELECT имя поля, функция **AS** имя результата

FROM имя таблицы;

Пример 1: *вычислить общее количество студентов*

SELECT COUNT(*) **AS** всего_студентов

FROM СТУДЕНТЫ;

Подведение итогов в запросах (продолжение).

Пример *2:вычислить общее количество студентов в группе 434*

```
SELECT COUNT(ФИО) AS
   всего_студентов
FROM СТУДЕНТЫ
WHERE группа='434';
```

Подведение итогов в запросах (продолжение).

А как быть, если надо подсчитать количество студентов в каждой группе по факультету?

Подведение итогов в запросах (продолжение).

Правильно, нужно использовать группировку.

В чем состоит суть операции группировки?

- При группировке все множество записей таблицы разбивается на группы, в которых собираются записи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.
- Агрегатные функции вычисляют одиночное значение для каждой группы таблицы.

Список агрегатных функций:

- COUNT - Количество строк или непустых значений полей, которые выбрал запрос
- SUM- Сумма всех выбранных значений данного поля
- AVG - Среднеарифметическое значение всех выбранных значений данного поля
- MIN - Наименьшее из всех выбранных значений данного поля
- MAX - Наибольшее из всех выбранных значений данного поля

- Агрегатные функции используются подобно именам полей в операторе SELECT, но с одним исключением: они берут имя поля как аргумент. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

Предложение **GROUP BY**

GROUP BY имя поля, по которому нужно группировать.

В нашем примере будет

SELECT Группа, **COUNT**(ФИО) **AS** всего_студентов

FROM СТУДЕНТЫ

GROUP BY Группа;

Выражение **COUNT**(ФИО) вычисляется по одному разу для каждой группы

А как применить условие к сгруппированным данным?

Например, необходимо найти группы с количеством студентов > 20.

Предложение **HAVING**

HAVING логическое выражение

HAVING для групп тоже, что **WHERE** для записей.

WHERE фильтрует строки, а **HAVING –** группы.

WHERE фильтрует до группировки, а **HAVING** – после.

Пример 1

Определить группы с количеством студентов более 20

```
SELECT Группа, COUNT(ФИО) AS всего_студентов
FROM СТУДЕНТЫ
GROUP BY Группа
HAVING COUNT(ФИО)>20;
```

Пример 2

Получить список клиентов, сделавших хотя бы 2 заказа.

```
SELECT Код_клиента, COUNT(*) AS всего_заказов
FROM Заказы
GROUP BY Код_клиента
HAVING COUNT(*)>=2;
```

А можно ли использовать WHERE, если есть HAVING?

Конечно, если нужно.

Например, *нужно найти группы только 4-го курса, где количество студентов >20*

SELECT Группа, **COUNT**(ФИО) **AS** всего_студентов

FROM СТУДЕНТЫ

WHERE Группа LIKE ”_4*”

GROUP BY Группа

HAVING COUNT(ФИО)>20;

ОБЪЕДИНЕНИЯ ТАБЛИЦ

Объединение – это самая мощная операция SQL.

Чтобы извлечь данные, хранящиеся в нескольких таблицах с помощью одного оператора SELECT, необходимо их объединить.

Чтобы объединить, нужно указать все необходимые таблицы и «объяснить», как они между собой связаны.

Объединение таблиц с помощью предложения **WHERE**

Пусть нужно выбрать сведения об оценках, полученных студентами на экзаменах

SELECT ФИО, N_зачетки, Оценка
FROM Студент, Ведомость;

Результатом такого запроса будет декартово произведение. Например, если в таблице Студент 30 записей, а таблице Ведомость 120 записей, то в результате такого запроса получится 3600 записей.

Как вам такой результат?!

Объединение таблиц с помощью предложения WHERE – правильное решение

```
SELECT ФИО, N_зачетки,
  Оценка FROM Студент,
  Ведомость
WHERE Студент.N_зачетки=
  Ведомость. N_зачетки;
```

WHERE для объединения

Если требуется объединить несколько таблиц, то это записывается в одном предложении WHERE с использованием операции AND.

Например, пусть нужно выбрать сведения об оценках, полученных студентами на экзаменах, с указанием дисциплины

РЕШЕНИЕ:

```
SELECT ФИО, N_зачетки,
  Дисциплина, Оценка
FROM Студент, Ведомость,
  Дисциплины
WHERE Студент.N_зачетки=
  Ведомость. N_зачетки AND
  Ведомость.
  Код_дисциплины=Дисциплины.
  Код_дисциплины;
```

Внутреннее соединение с помощью конструкции **INNER JOIN ON**

Объединение задается в предложении FROM, вместо WHERE используется предложение ON

Пример:

```
SELECT ФИО, N_зачетки,
  Оценка
FROM Студент INNER JOIN
  Ведомость
ON Студент.N_зачетки=
  Ведомость. N_зачетки;
```

А если таблиц больше двух?

Ничего страшного!

…

```
FROM tabl1 INNER JOIN (tabl2 INNER
   JOIN tabl3 ON tabl2.id2=tabl3.id3)
   ON tadl1.id1=tabl3.id3
```

В нашем примере будет:

В нашем примере будет:

```
SELECT ФИО, N_зачетки, дисциплина,
  Оценка
FROM Студент INNER JOIN (Ведомость
  INNER JOIN Дисциплины ON
  Ведомость.
  Код_дисциплины=Дисциплины.
  Код_дисциплины)
ON Студент.N_зачетки= Ведомость.
  N_зачетки;
```

Ну, ОЧЕНЬ громоздко! Легко ошибиться в именах таблиц. Есть выход.

В части FROM оператора SELECT допустимо применять синонимы (псевдонимы) к именам таблицы, если при формировании запроса нам требуется более чем один экземпляр некоторого отношения. Синонимы задаются с использованием ключевого слова AS, которое может быть вообще опущено. Поэтому часть FROM может выглядеть следующим образом:

FROM RI AS A, RI AS B.

Псевдонимы

Например:

```
SELECT ФИО, N_зачетки,
   дисциплина, Оценка
FROM Студент AS С INNER JOIN
   (Ведомость AS В INNER JOIN
   Дисциплины AS Д ON В.
   Код_дисциплины=Д.
   Код_дисциплины)
ON С.N_зачетки= В. N_зачетки;
```

Подзапросы

Применяют, когда в предложении WHERE значение, с которым надо сравнивать должно быть вычислено в момент выполнения оператора SELECT.

Текст подзапроса заключается в скобки.

Существует два типа подзапросов:

Скалярный – возвращает единственное значение,

Табличный – возвращает множество значений.

Подзапросы

С помощью SQL можно вкладывать запросы внутрь друг друга. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса (в предложении WHERE или HAVING), определяющего, верно оно или нет. Совместно с подзапросом можно использовать предикат EXISTS, который возвращает истину, если вывод подзапроса не пуст.

- В сочетании с другими возможностями оператора выбора, такими как группировка, подзапрос представляет собой мощное средство для достижения нужного результата. Предикат EXISTS (SubQuery) истинен, когда подзапрос SubQuery не пуст, то есть содержит хотя бы один кортеж, в противном случае предикат EXISTS ложен.
- Предикат NOT EXISTS обратно — истинен только тогда, когда подзапрос SubQuery пуст.

Пример 1

Определить даты, когда продажи превышали средний уровень.

```
SELECT Дата, Количество
FROM Продажа
WHERE Количество > (SELECT AVR(Количество) FROM Продажа);
```

Пример 2

Определить даты, когда среднее количество проданного за день товара оказалось >25 единиц.

```
SELECT Продажа.Дата,
  AVG(Продажа.Количество) AS
  Среднедневное
FROM Продажа
GROUP BY Продажа.Дата
HAVING AVG (Продажа.Количество)
  >25;
```

Правильно, это простой запрос без подзапроса!

А если требуется найти тоже, но больше среднего показателя по всем сделкам вообще. Тогда:

...

```
GROUP BY Продажа.Дата
HAVING AVG (Продажа.Количество) >
(SELECT AVG(Продажа.Количество)
  FROM Продажа)
```

Подзапросы, возвращающие множество значений

- Можно добавить данные из одной таблицы в другую

```
INSERT INTO табл1
SELECT *
FROM табл2
WHERE  поле IN(SELECT …)
```

Подзапросы, возвращающие множество значений

- Можно удалять данные из одной таблицы, по результатам, полученным в другой

```
DELETE
FROM студенты
WHERE номер_зачетки
 IN (SELECT номер_зачетки
FROM ведомость
WHERE оценка=2)
```

Условный оператор и оператор цикла в SQL

Условие: Условный оператор IF условие
оператор или Begin опреатор1,…END
else оператор или блок

Select №группы From группы Where
№группы=@группа

if Exists (Select №группы From группы
Where № группы =@группа) ELSE Print
“Нет такой группы”

Цикл

WHILE (Select avg(цена) FROM товары)<100 Begin UPDATE товары SETцена=цена*1 .02

IF (Select max(цена) FROM товары)>1000 Break ELSE Continue

Комбинированные запросы. Оператор UNION

Например, создать общий список студентов и преподавателей, фамилии которых начинаются на букву К

```
SELECT ФИО, Адрес
FROM Студенты
WHERE Студенты.ФИО LIKE”К%”
UNION
SELECT ФИО, Адрес
FROM Преподаватели
WHERE Преподаватели.ФИО LIKE”К%”;
```

Язык манипулирования данными DML

Ввод данных:

INSERT INTO имя_таблицы [(список полей)]

VALUES (список значений)

Например:

INSERT INTO STUD (ФИО,ГРУППА)

VALUES (‘ИвановИИ’, ‘425’)

Язык манипулирования данными DML (продолж.)

Правила:

Если задаются значения всех полей, то список полей не нужен.

Если столбец при описании таблицы имеет признак NOT NULL, то ввод данных в это поле в каждой записи обязателен.

Если имеется хотя бы один необязательный столбец, в который не вводится значение, задание списка имен столбцов обязательно.

Можно по запросу извлечь значения из одной таблицы и разместить их в другой:

INSERT INTO табл.куда

FROM табл. Откуда

WHERE условие

Язык манипулирования данными DML (продолж.)

Добавление выбранных данных

INSERT INTO *табл.куда (список полей)*

SELECT …

Копирование данных из одной таблицы в другую

SELECT *

INTO *новая табл*

FROM *старая табл*

INSERT INTO

- Можно по запросу извлечь значения из одной таблицы и разместить их в другой

INSERT INTO табл.-куда

FROM табл.-откуда

WHERE л.в.

Язык манипулирования данными DML

Обновление данных:

UPDATE имя таблицы

SET имя_поля1=нов.знач.1 [,имя поля2=нов.знач.2] и т.д.

[WHERE условие отбора**]**

Например:

UPDATE Ведомость

SET Ведомость.Оценка=5

WHERE Ведомость.Оценка=2 **AND** Ведомость.Дисциплина=’23’

Язык манипулирования данными DML

DELETE
FROM имя-таблицы
[WHERE условие отбора**]**

Если не указать условие, то удаляются все строки и таблица будет пуста!

Следите за нарушением целостности!

Представление

- это именованная виртуальная таблица, содержимое которой является результатом запроса, заданного при описании представления.

CREATE VIEW имя_представления **AS**

- **SELECT** список_столбцов
- **FROM …**
- **WHERE …**

- Представление является хранимой инструкцией SELECT.
- Представления запрашиваются так же, как таблицы, и не принимают параметры.

Представление позволяет:

- Ограничивать число столбцов;
- Ограничивать число строк;
- Выводить дополнительные столбцы, преобразованные из других столбцов;
- Выводить группы строк.

Для чего нужны представления:

- Обеспечивают независимость пользовательских программ от изменений логической структуры БД.
- Для каждого пользователя может быть создано «свое окно в БД».
- От определенных пользователей м.б. скрыты некоторые данные.
- Для повторного использования операторов SQL.
- Для упрощения выполнения сложных операций, например, объединения таблиц.
- Для изменения форматирования и отображения данных.