

Базовый синтаксис

В сегодняшнем занятии мы с вами разобрали основные понятия в SQL и базовый синтаксис формирования запроса. Помимо этого мы изучили понятия DDL и DML и основные команды создания таблиц и представлений.

Различие между SQL и MySQL

Существует два различных понятия, SQL и СУБД.

SQL является языком программирования, который используется во множестве СУБД. SQL описывает концепции и синтаксис основных операций. Почти каждая СУБД включает в себя дополнительный функционал, специфичные для себя особенности и расширения.

Резюмируя, можно сказать, что SQL это набор правил для языка, а СУБД (Mysql, oracle, MS SQL) это реализация со своими специфичными особенностями.

SQL vs NoSQL

Все большую популярность набирает такое понятие, как NoSQL и возникают два вопроса:

- В чем отличие SQL от NoSQL
- Что лучше и перспективнее

Давайте разберемся по порядку на примере базы данных, которая описывает взаимодействие сущностей **Customer** (покупатель) и **Order** (заказ)

В чем отличие SQL от NoSQL

SQL позволяет нам работать с привычной для нас таблицей, у которой есть столбцы и строки. Если рассматривать наш пример, мы создадим две отдельные таблицы **Customer** и **Order** и при необходимости объединим данные из этих таблиц

В NoSql ситуация совершенно иная, данные представляются в виде графа и имеют каскадную структуру, скажем в наборе данных о **Customer** у нас будет свойство **Order**, которое содержит в себе все заказы покупателя.

Что лучше и перспективнее

На данный момент NoSQL приобретает популярность и является "модным" направлением, он действительно хорош для решения узко профилированных задач, однако SQL существует значительно дольше и накопленный опыт и экспертизы позволяют нам реализовывать достаточно сложную архитектуру, делая ее масштабируемой и удобной в использовании.

Резюмируя можно сказать, что SQL еще множество лет будет впереди и в любом случае, даже при желании использовать NoSQL необходимо знать SQL на достаточно хорошем уровне.

Основные понятия у таблицы

Для более удобной работы нам необходимо ввести ряд понятий:

- 1) Поле, атрибут, столбец. Часть таблицы, определяющая одно из свойств. (располагается вертикально)
- 2) Запись, строка. Часть таблицы, которая хранит информацию об одной описанной единице. (располагается горизонтально)
- 3) Поле. Ячейка в таблице, которую можно однозначно определить по строке и столбцу.
- 3) Таблица. Сущность в базе данных, которая хранит в себе данные. Имеет строгую структуру и типизированные поля. Хранится на жестком диске.
- 4) Выборка. Набор данных, полученных в результате запроса SELECT. Хранится в оперативной памяти.
- 5) Запрос. Команда на SQL, предназначенная для получения или изменение данных, или структуры сущностей в базе данных.

В результате запроса из таблицы можно получить выборку с определенными записями и полями.

Формирование выборки

Для формирования выборки необходимо использовать специальный запрос SELECT имеющий следующую структуру:



```
SELECT
*
FROM users
WHERE name = 'Анатолий'
and lastname = 'Ушанов'
```

В разделе SELECT можно указать множество полей через запятую или * (все поля)

В разделе FROM указывается таблицы источник.

В разделе WHERE указываются условия.

Для формирования условий используются следующие операторы.

 синтаксис	 функционал
<u>=</u> , <u><</u> , <u>></u> , <u><></u> , <u><=</u> , <u>>=</u>	равен, меньше, больше, не равен, меньше или равно, больше или равно
<u>between</u> , <u>not between</u>	значение в диапазоне
<u>in</u> , <u>not in</u>	значение в списке
<u>like</u> , <u>not like</u>	значение подходит по маске (% - множество символов, _ - один символ)
<u>and</u> , <u>or</u>	логические и, или
(.)	указание приоритета

```
-- запрос будет работать при условии существования таблицы
-- users с используемыми полями
```

```
SELECT
name,
lastname
FROM users
WHERE (name = 'Анатолий'
and lastname = 'Ушанов'
and age between 12 and 34
and city not in ('Moscow', 'Erevan')
and email like '%@gmail.com'
) or gender = 'М'
```

Разбор задания из класса

Задание:

Из таблицы **HR.EMPLOYEES**

```
-- 1 задание (найти всех сотрудников, с job_id = IT_PROG)
SELECT
    *
from HR.EMPLOYEES
where job_id = 'IT_PROG'

-- 2 задание (найти сотрудников, с зп больше 10 000)
SELECT
    *
from HR.EMPLOYEES
where salary > 10000

-- 3 задание (найти сотрудников, с зп от 10 000 до 20 000 (включая концы))
SELECT
    *
from HR.EMPLOYEES
where salary between 10000 and 20000

-- 4 задание (найти сотрудников не из 60, 30 и 100 департамента)
SELECT
    *
from HR.EMPLOYEES
where department_id not in (60, 30, 100)

-- 5 задание (найти сотрудников у которых есть две буквы ll подряд в середине имени)
SELECT
    *
from HR.EMPLOYEES
where FIRST_NAME like '%_ll_%'

-- 6 задание (найти сотрудников, у которых фамилия кончается на а)
SELECT
    *
from HR.EMPLOYEES
where LAST_NAME like '%a'
```

Особенности значения Null

В SQL существует значение Null, которое обозначает отсутствие значения в поле. Не стоит путать значение Null со значением 0 и пустой строкой (''). И их нельзя сравнивать ранее изученными операторами.

Для лучшего представления смысла значения Null я хочу вас познакомить с Петей и Колей.

П: "Коля, какая завтра погода?"

К: "Не знаю, я не планировал завтра никуда выходить."



П: "Хорошо, может ты знаешь, какая после завтра погода?"

К: "Нет, вышел новый сезон Рика и Морти и я планирую всю неделю засматривать его до дыр"

Из этого диалога мы можем сделать вывод, что у Коли отличный вкус на сериалы и что он не знает, какая будет погода завтра или послезавтра.

- Можем ли мы сказать, что погода будет 0 градусов? **Нет**
- Можем ли мы сказать, что завтра и после завтра будет одинаковая погода? **Нет.** Может в итоге и окажется, что она и будет одинаковой, однако сейчас это утверждение ложно.
- Можем ли мы сказать, что завтра и после завтра будет разная погода? **Нет.** Все по той же причине.

Мы можем лишь определить, является данное значение Null или нет, для этого существуют специальные операторы

 синтаксис	 функционал
<u>is null</u>	является пустым значением
<u>is not null</u>	не является пустым значением

Оператор distinct

Оператор distinct позволяет убрать дубли у запроса, это достаточно мощный инструмент, однако использовать его нужно только в уместных случаях.

Зачастую возникновение дублей сигнализирует нам о том, что мы сделали в запросе что-то неверно. Использование distinct в таком случае не только неправильно с архитектурной точки зрения (в простонародий костыль), но и сильно негативно влияет на время выполнения запроса.

Все дело в том, что для нахождения дублей distinct прежде сортирует записи, а сортировка очень ресурса затратный процесс.

Используйте distinct только в тех случаях, где вы понимаете причину возникновения дублей, и что она заключается не в неправильном запросе.

```
-- запрос вернет уникальные пары имени и фамилии

SELECT distinct
  name,
  lastname
FROM users
```

Разбор задания из класса

Задание:

Из таблицы **HR.EMPLOYEES**

```
-- 1 задание (найти все имеющиеся job_id)
SELECT distinct
  job_id
from HR.EMPLOYEES

-- 2 задание (найти job_id у которых нет COMMISSION_PCT или зп меньше 3000)
SELECT distinct
  *
from HR.EMPLOYEES
where COMMISSION_PCT is null
or salary < 3000
```

DDL/DML/TCL

Команды SQL делятся на несколько групп, две из которых будут активно нами использоваться:

DDL (Data Definition Language).

Команды, направленные на создание, изменение или удаление сущностей базы данных

DML (Data Manipulation Language).

Команды, направленные на добавление, изменение или удаление данных внутри базы данных.

Создание таблиц

Команда создания таблицы позволяет указать перечень полей, их типы и ограничения (Constraints).



```
-- синтаксис формирования таблицы

create table <table_name> (
  <column1_name> <data_type>,
  <column2_name> <data_type>,
  <column3_name> <data_type>,
  ...
)

-- пример создания таблицы



create table toys (
  toy_name varchar(100),
  weight  number
);
```

Типы данных

 синтаксис	 функционал
<u>varchar2</u>	текст
<u>varchar</u>	текст
<u>char</u>	символы
<u>integer</u>	целое число
<u>number</u>	число
<u>date</u>	дата
<u>timestamp</u>	момент времени

Ограничения

Ограничения позволяют указать определенные правила для данных, которые будут вставляться в таблицу. В случае, если мы попытаемся вставить данные, которые этим правилам не соответствуют, произойдет ошибка.

 синтаксис	 функционал
<u>NOT NULL</u>	Значение в поле не может быть null
<u>UNIQUE</u>	Значение в поле должно быть уникальным
<u>PRIMARY KEY</u>	Значение в поле первичный ключ не null уникальное
<u>CHECK</u>	Соответствует условию

Ограничение может указываться не только на поле, но и на несколько полей.

```
-- пример указания ограничений

CREATE TABLE ARTIST
(
    NAME VARCHAR2(255) NOT NULL,
    LASTNAME VARCHAR2(255) NOT NULL,
    gender char(1) check (gender in ('f', 'm')),
    primary key (NAME, LASTNAME)
);
```

Ограничение может указываться не только на поле, но и на несколько полей.

Разбор задания из класса

Создать таблицу goods с полями

- id (первичный ключ)
- title (строка максимум в 30 символов)
- quantity (число)
- in_stock (символ (Y/N))

```
CREATE TABLE goods(
    id integer primary key,
    title varchar(30),
```



```
quantity number,  
in_stock char(1) check (in_stock in ('Y', 'N'))  
);
```

Добавление данных

Для того чтобы заполнить таблицу данными нам необходимо воспользоваться командой **INSERT**.

```
-- шаблон  
insert into <table_name>(  
    <column1_name>,  
    <column2_name>,  
    <column3_name>  
)values(  
    <value1>,  
    <value2>,  
    <value3>  
)  
  
-- пример  
insert into goods(  
    title,  
    quantity,  
    in_stock  
)values(  
    "Велосипед",  
    10,  
    "Y"  
)
```

При вызове команды **insert** указывать список полей, в которые данные должны попасть, не обязательно, однако такой подход позволяет избежать ошибок, так как данные подставляются в поля позиционно (первый в первый, второй во второй и т.д.)

Представления (view)

Представления позволяют нам сохранить не сами данные, а запрос. При каждом обращении к представлению выполняется запрос и формируется набор данных из таблицы источника.

```
-- шаблон
create view v_table as
select
    <column1_name>,
    <column2_name>,
    <column3_name>
where <condition1>
    and <condition2>;
```

Разбор задания

1. Добавьте в таблицу goods 4 товара
2. Увеличьте цену велосипеда на 10000
3. Создать представление, которое выводит только те товары, которые стоят меньше 10000

```
-- создание таблицы
create table goods(
    id integer primary key,
    title varchar(128),
    price integer check(price > 0),
    quantity integer check(price between 0 and 10)
);

-- Добавьте в таблицу goods 4 товара

insert into goods (id, title, price, quantity) values(1, 'велосипед', 12000, 4);
insert into goods (id, title, price, quantity) values(2, 'лыжи', 10000, 5);
insert into goods (id, title, price, quantity) values(3, 'коньки', 6000, 7);
insert into goods (id, title, price, quantity) values(4, 'скейт', 10000, 2);

-- Увеличьте цену велосипеда на 10000

update goods
set price = 10000
where title = 'велосипед';

-- Создать представление, которое выводит только те товары, которые стоят меньше 10000

create view v_goods_cheap as
select * from goods
where price < 10000;
```

Изменение данных

Для изменения данных таблицы можно использовать команду UPDATE, которая позволяет по условию изменить данные в какой-либо таблице

```
--шаблон

update <table_name>
set <column1_name> = <value1>,
    <column2_name> = <value2>
where <condition1>
    and <condition2>
```

Разбор задания

1. Добавьте в таблицу goods 4 товара
2. Увеличьте кол-во велосипедов на 10000
3. Создать представление, которое выводит только те товары, которые стоят меньше 10000

```
-- создание таблицы
create table goods(
    id integer primary key,
    title varchar(128),
    price integer check(price > 0),
    quantity integer check(price between 0 and 10)
);

-- Добавьте в таблицу goods 4 товара

insert into goods (id, title, price, quantity) values(1, 'велосипед', 12000, 4);
insert into goods (id, title, price, quantity) values(2, 'лыжи', 10000, 5);
insert into goods (id, title, price, quantity) values(3, 'коньки', 6000, 7);
insert into goods (id, title, price, quantity) values(4, 'скейт', 10000, 2);

-- Увеличьте цену велосипеда на 10000

update goods
set price = 10000
```

```
where title = 'велосипед';

-- Создать представление, которое выводит только те товары, которые стоят меньше 10000

create view v_goods_cheap as
  select * from goods
  where price < 10000;
```

Удаление данных

Для удаления данных из таблицы можно использовать две команды

- delete (DML)
- truncate (DDL)

У этих двух способов есть ряд особенностей.

Использование commit

В силу того, что delete это DML, для того, чтобы применить удаление на всю базу, а не только в рамках вашей сессии, необходимо использовать команду commit. В truncate напротив, при использовании commit запускается автоматически, тем самым накатывая не только очистку данных таблицы, но и все ранее совершенные DML операции.

Скручивание автоинкремента

При создании поля с суррогатным ключом, который должен автоматически увеличиваться, часто используют автоинкремент, который позволяет автоматически при добавлении строки увеличивать ключ на единицу. При использовании delete новая запись будет продолжать ряд ключей, которых уже нет, не обнулив автоинкремент.

Truncate напротив, скидывает счетчик и все новые записи будут считаться с 1.

Так же стоит подчеркнуть, что в truncate нельзя использовать условие, он очищает всю таблицу полностью и за счет этого работает быстрее, delete напротив, может работать с условием.