Язык **SQL** или **Structured Query Language** (язык структурированных запросов) предназначен для управления данными в системе реляционных баз данных (RDBMS). С помощью него пишутся специальные запросы (так называемые SQL инструкции) к базе данных с целью получения данных из базы данных или для манипулирования этими данными.

СУБД – это система управления базами данных, сокращенно СУБД. Основные СУБД:

- **Microsoft SQL Server** – это СУБД от Microsoft. Она очень популярна в корпоративном секторе. Это целый комплекс приложений, позволяющий хранить и модифицировать данные, анализировать их, осуществлять безопасность этих данных и многое другое;
- **Oracle Database** – это СУБД от компании Oracle. Это также очень популярная СУБД среди крупных компаний - конкурент Microsoft SQL Server, стоимость их полнофункциональных версий очень высока;
- **MySQL** – СУБД от Oracle, распространяется бесплатно. MySQL получила очень широкую популярность в интернет сегменте, т.е. большинство сайтов в интернете используют эту СУБД как средство хранения данных;
- **PostgreSQL** – эта система управления базами данных также является бесплатной, и она очень популярна и функциональна.

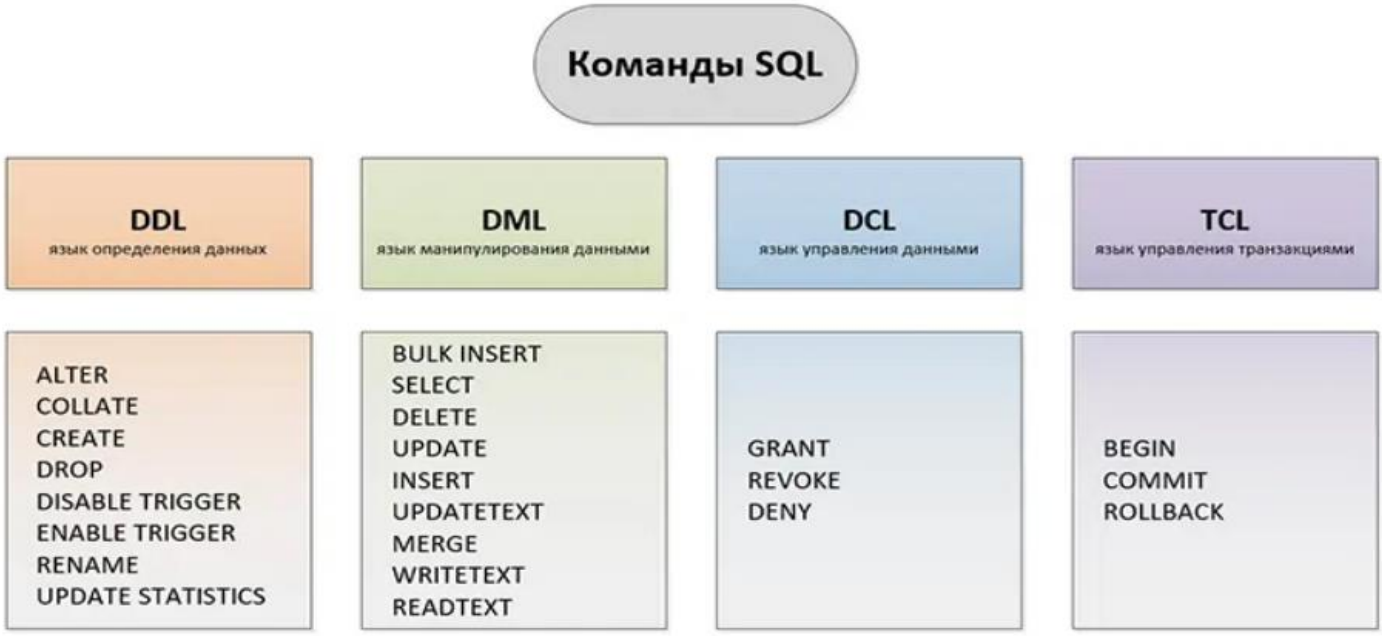
Виды SQL запросов

DDL (Data Definition Language) - язык определения данных. Задачей DDL запросов является создание БД и описание ее структуры. Запросами такого вида устанавливаются правила того, в каком виде различные данные будут размещаться в БД.

DML (Data Manipulation Language) — язык манипулирования данными. В число запросов этого типа входят различные команды, используя которые непосредственно производятся некоторые манипуляции с данными. DML-запросы нужны для добавления изменений в уже внесенные данные, для получения данных из БД, для их сохранения, для обновления различных записей и для их удаления из БД. В число элементов DML-обращений входит основная часть SQL операторов.

DCL (Data Control Language) — язык управления данными. Включает в себя запросы и команды, касающиеся разрешений, прав и других настроек СУБД.

TCL (Transaction Control Language) — язык управления транзакциями. Конструкции такого типа применяют чтобы управлять изменениями, которые производятся с использованием DML запросов. Конструкции TCL позволяют нам производить объединение DML запросов в наборы транзакций.



Настройка базы данных

Перед началом создайте БД с тестовыми данными. Для работы вам понадобится скачать два файла: DLL.sql и InsertStatements.sql. После установите MySQL, откройте терминал и войдите в консоль MySQL с помощью команды:

```
mysql -u root -p
```

Затем введите пароль и выполните следующую команду. Назовём базу данных «university»:

```
CREATE DATABASE university;

USE university;

SOURCE <path_of_DLL.sql_file>;

SOURCE <path_of_InsertStatements.sql_file>;
```

SHOW DATABASES

SQL-команда, которая отвечает за просмотр доступных баз данных.

CREATE DATABASE

Команда для создания новой базы данных.

USE

С помощью этой SQL-команды `USE <database_name>` выбирается база данных, необходимая для дальнейшей работы с ней.

SOURCE

А `SOURCE <file.sql>` позволит выполнить сразу несколько SQL-команд, содержащихся в файле с расширением .sql.

DROP DATABASE

Стандартная SQL-команда для удаления целой базы данных.

SHOW TABLES

Увидеть все таблицы, которые доступны в базе данных.

CREATE TABLE

SQL-команда для создания новой таблицы:

```
CREATE TABLE <table_name1> (
  <col_name1><col_type1>,
  <col_name2><col_type2>,
  <col_name3><col_type3>
  PRIMARY KEY(<col_name1>),
  FOREIGN KEY(<col_name2>) REFERENCES <table_name2>(<col_name2>)
);
```

Ограничения целостности при использовании `CREATE TABLE`

Может понадобиться создать ограничения для определённых столбцов в таблице. При создании таблицы можно задать следующие ограничения:

- ячейка таблицы не может иметь значение `NULL`;
- первичный ключ — `PRIMARY KEY(col_name1, col_name2, …)`;
- внешний ключ — `FOREIGN KEY(col_namex1, …, col_namexn) REFERENCES table_name(col_namex1, …, col_namexn)`.

Можно задать больше одного первичного ключа. В этом случае получится составной первичный ключ.

Пример: Создайте таблицу «instructor»:

```
CREATE TABLE instructor (
  ID CHAR(5),
  name VARCHAR(20) NOT NULL,
  dept_name VARCHAR(20),
  salary NUMERIC(8,2),
  PRIMARY KEY (ID),
  FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);
```

DESCRIBE

С помощью `DESCRIBE <table_name>` можно просмотреть различные сведения (тип значений, является ключом или нет) о столбцах таблицы.

INSERT

Команда `INSERT INTO <table_name>` в SQL отвечает за добавление данных в таблицу:

```
INSERT INTO <table_name> (<col_name1>, <col_name2>, <col_name3>, …)
  VALUES (<value1>, <value2>, <value3>, …);
```

При добавлении данных в каждый столбец таблицы не требуется указывать названия столбцов.

```
INSERT INTO <table_name>
  VALUES (<value1>, <value2>, <value3>, …);
```

UPDATE

SQL-команда для обновления данных таблицы:

```
UPDATE <table_name>
  SET <col_name1> = <value1>, <col_name2> = <value2>, ...
  WHERE <condition>;
```

DELETE

SQL-команда `DELETE FROM <table_name>` используется для удаления данных из таблицы.

DROP TABLE

А так можно удалить всю таблицу целиком.

SELECT

Далее мы рассмотрим основные команды SQL, которые позволяют работать непосредственно с данными. К одной из таких SQL-команд относится `SELECT` для получения данных из выбранной таблицы:

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>;
```

Следующей командой можно вывести все данные из таблицы:

```
SELECT * FROM <table_name>;
```

SELECT DISTINCT

В столбцах таблицы могут содержаться повторяющиеся данные. Используйте `SELECT DISTINCT` для получения только неповторяющихся данных.

```
SELECT DISTINCT <col_name1>, <col_name2>, …
  FROM <table_name>;
```

WHERE

Можно использовать ключевое слово `WHERE` в `SELECT` для указания условий в запросе:

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <condition>;
```

В запросе можно задавать следующие условия:

- сравнение текста;
- сравнение численных значений;
- логические операции `AND` (и), `OR` (или) и `NOT` (отрицание).

Пример

Попробуйте выполнить следующие команды. Обратите внимание на условия, заданные в `WHERE`:

```
SELECT * FROM course WHERE dept_name=’Comp. Sci.’;
SELECT * FROM course WHERE credits>3;
SELECT * FROM course WHERE dept_name='Comp. Sci.' AND credits>3;
```

GROUP BY

Оператор `GROUP BY` часто используется с агрегатными функциями, такими как `COUNT`, `MAX`, `MIN`, `SUM` и `AVG`, для группировки выходных значений.

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  GROUP BY <col_namex>;
```

Пример

Выведем количество курсов для каждого факультета:

```
SELECT COUNT(course_id), dept_name
```

```
  FROM course
  GROUP BY dept_name;
```

HAVING

Ключевое слово `HAVING` было добавлено в SQL по той причине, что `WHERE` не может использоваться для работы с агрегатными функциями.

```
SELECT <col_name1>, <col_name2>, ...
  FROM <table_name>
  GROUP BY <column_namex>
  HAVING <condition>
```

Пример

Выведем список факультетов, у которых более одного курса:

```
SELECT COUNT(course_id), dept_name
  FROM course
  GROUP BY dept_name
  HAVING COUNT(course_id)>1;
```

ORDER BY

`ORDER BY` используется для сортировки результатов запроса по убыванию или возрастанию. `ORDER BY` отсортирует по возрастанию, если не будет указан способ сортировки `ASC` или `DESC`.

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  ORDER BY <col_name1>, <col_name2>, … ASC|DESC;
```

Пример

Выведем список курсов по возрастанию и убыванию количества кредитов:

```
SELECT * FROM course ORDER BY credits;
SELECT * FROM course ORDER BY credits DESC;
```

BETWEEN

`BETWEEN` используется для выбора значений данных из определённого промежутка. Могут быть использованы числовые и текстовые значения, а также даты.

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <col_namex> BETWEEN <value1> AND <value2>;
```

Пример

Выведем список инструкторов, чья зарплата больше 50 000, но меньше 100 000:

```
SELECT * FROM instructor
  WHERE salary BETWEEN 50000 AND 100000;
```

LIKE

Оператор `LIKE` используется в `WHERE`, чтобы задать шаблон поиска похожего значения.

Есть два свободных оператора, которые используются в `LIKE`:

- `%` (ни одного, один или несколько символов);
- `_` (один символ).

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <col_namex> LIKE <pattern>;
```

Пример

Выведем список курсов, в имени которых содержится «to», и список курсов, название которых начинается с «CS-»:

```
SELECT * FROM course WHERE title LIKE ‘%to%’;
SELECT * FROM course WHERE course_id LIKE 'CS-___';
```

IN

С помощью `IN` можно указать несколько значений для оператора `WHERE`:

```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <col_namen> IN (<value1>, <value2>, …);
```

Пример

Выведем список студентов с направлений Comp. Sci., Physics и Elec. Eng.:

```
SELECT * FROM student
  WHERE dept_name IN (‘Comp. Sci.’, ‘Physics’, ‘Elec. Eng.’);
```

JOIN

`JOIN` используется для связи двух или более таблиц с помощью общих атрибутов внутри них. На изображении ниже показаны различные способы объединения в SQL. Обратите внимание на разницу между левым внешним объединением и правым внешним объединением:


```
SELECT <col_name1>, <col_name2>, …
  FROM <table_name1>
  JOIN <table_name2>
  ON <table_name1.col_namex> = <table2.col_namex>;
```

Пример

Выведем список всех обязательных курсов и детали о них:

```
SELECT prereq.course_id, title, dept_name, credits, prereq_id
  FROM prereq
  LEFT OUTER JOIN course
  ON prereq.course_id=course.course_id;
```

VIEW

`VIEW` — это виртуальная таблица SQL, созданная в результате выполнения выражения. Она содержит строки и столбцы и очень похожа на обычную SQL-таблицу. `VIEW` всегда показывает самую свежую информацию из базы данных.

Создание

```
CREATE VIEW <view_name> AS
  SELECT <col_name1>, <col_name2>, …
  FROM <table_name>
  WHERE <condition>;
```

Удаление

```
DROP VIEW <view_name>;
```

Агрегатные функции

Это не совсем основные команды SQL, однако знать их тоже желательно. Агрегатные функции используются для получения совокупного результата, относящегося к рассматриваемым данным:

- `COUNT(col_name)` — возвращает количество строк;
- `SUM(col_name)` — возвращает сумму значений в данном столбце;
- `AVG(col_name)` — возвращает среднее значение данного столбца;
- `MIN(col_name)` — возвращает наименьшее значение данного столбца;
- `MAX(col_name)` — возвращает наибольшее значение данного столбца.

Вложенные подзапросы

Вложенные подзапросы — это SQL-запросы, которые включают выражения `SELECT`, `FROM` и `WHERE`, вложенные в другой запрос.

Пример

Найдём курсы, которые преподавались осенью 2009 и весной 2010 годов:

```
SELECT DISTINCT course_id
  FROM section
  WHERE semester = 'Fall' AND year= 2009 AND course_id IN (
    SELECT course_id
    FROM section
    WHERE semester = 'Spring' AND year= 2010
  );
```

Шпаргалка по SQL

create database *имя_базы_данных*;
use *имя_базы_данных*;
create table *имя_таблицы (имя_первого_столбца тип, имя_второго_столбца тип, …, имя_последнего_столбца тип)*;
show *databases;* — показать все имеющиеся БД.
show *tables;* — показать список таблиц текущей БД (предварительно ее надо выбрать с помощью оператора use).
describe *имя_таблицы;* — показать описание столбцов указанной таблицы
drop database *имя_базы данных;* — удалить БД.
drop table *имя_таблицы;* -удалить таблицу.
AUTO_INCREMENT — высчитывает максимальное значение этого столбца, полученное значение увеличивает на 1 и заносит его в столбец.
PRIMARY KEY ()
FOREIGN KEY *(имя_столбца_которое_является_внешним_ключом)* **REFERENCES** *имя_таблицы_родителя (имя_столбца_родителя);*
INSERT INTO *имя_таблицы* **VALUES** *('значение_первого_столбца', 'значение_второго_столбца', …, 'значение_последнего_столбца')*;
INSERT INTO *имя_таблицы ('имя_столбца',*
'имя_столбца') **VALUES** *('значение_первого_столбца','значение_второго_столбца');*
SELECT *что_выбрать* **FROM** *откуда_выбрать;*
SELECT *** **FROM** *откуда_выбрать;* — выбрать все столбцы таблицы
SELECT *имя_столбца* **FROM** *имя_таблицы* **ORDER BY** *имя_столбца_сортировки;* — сортировка
SELECT *имя_столбца* **FROM** *имя_таблицы* **ORDER BY** *имя_столбца_сортировки;* — сортировка **DESC**;
SELECT *имя_столбца* **FROM** *имя_таблицы* **WHERE** *условие;*

Необычные операторы:

BETWEEN *меньшее_число* **AND** *большее_число* **—** отбираются значения, находящиеся между указанными.
IS NOT NULL (IS NULL) — отбираются строки, (не) имеющие значения в указанном поле.
IN (NOT IN) — отбираются значения, (кроме) соответствующие указанным
LIKE (NOT LIKE) — отбираются значения, (не) соответствующие образцу. Самый распространенный метасимвол — %. Он означает любые символы. Например, если нам надо найти слова, начинающиеся с букв «вел», то мы напишем LIKE 'вел%', а если мы хотим найти слова, которые содержат символы «клуб», то мы напишем LIKE '%клуб%'.
SELECT *имя_столбца* **FROM** *имя_таблицы* **WHERE** *часть*
условия **IN** *(***SELECT** *имя_столбца* **FROM** *имя_таблицы* **WHERE** *часть*
условия **IN** *(***SELECT** *имя_столбца* **FROM** *имя_таблицы* **WHERE** *условие))* **; —** подзапросы
SELECT *имена_столбцов_таблицы_1, имена_столбцов_таблицы_2* **FROM** *имя_таблицы_1,*
имя_таблицы_2; — объединение
SELECT *имя_таблицы_1.имя_столбца, имя_таблицы_2.имя_столбца* **FROM** *имя_таблицы_1* **ТИП ОБЪЕДИНЕНИЯ** *имя_таблицы_2* **ON** *условие_объединения;*— где **ТИП ОБЪЕДИНЕНИЯ** — либо **LEFT OUTER JOIN**, либо **RIGHT OUTER JOIN.** Чтобы взять все строки с таблицы, а не только полные.
COUNT() – подсчет количества строк в таблице
SELECT COUNT*(**имя_столбца****)* FROM** *имя_таблицы;*
SELECT *имя_столбца* **COUNT***(имя_столбца)* **FROM** *имя_таблицы* **GROUP BY** *имя_столбца;*
SELECT *имя_столбца* **COUNT***(имя_столбца)* **FROM** *имя_таблицы* **GROUP BY** *имя_столбца* **HAVING COUNT** *условие;* — **HAVING** исполняет функции
ALTER TABLE *имя_таблицы* **ADD COLUMN** *имя_столбца тип;* — добавление столбца.
FIRST — новый столбец будет первым, и **AFTER** — указывает после какого столбца поместить новый.
UPDATE *имя_таблицы* **SET** *имя_столбца=значение_столбца* **WHERE** *условие* — для обновления уже существующих данных.
ALTER TABLE *имя_таблицы* **CHANGE** *старое_имя_столбца новое_имя_столбца тип;* — измение названия столбца.
ALTER TABLE *имя_таблицы* **MODIFY** *имя_столбца новый_тип;* — изменение типа данных столбца.
DELETE FROM *имя_таблицы* **WHERE** *условие;* — удаление строк из столбца.
AVG() — Функция возвращает среднее значение столбца.
COUNT() — Функция возвращает число строк в столбце.
MAX() — Функция возвращает самое большое значение в столбце.

MIN() — Функция возвращает самое маленькое значение в столбце.
SUM() — Функция возвращает сумму значений столбца.
SELECT *имя_таблицы_1.имя_столбца* имя_таблицы_2.имя_столбца* **AS** *имя_вычисляемого_столбца* **FROM** *имя_таблицы_1, имя_таблицы_2* – создание дополнительного столбца для вывода данных. Ее нельзя использовать, так как она не находиться в какой-либо таблице. Для таких случаев существуют Представления.
CREATE VIEW *имя_представления* **AS** *запрос*; — создание представления.
CONCAT(*str1,str2…***)** Возвращает строку, созданную путем объединения аргументов (аргументы указываются в скобках — str1,str2…). Добавляем пробел » «, как аргумент, для читабельности.
SELECT CONCAT_WS(*' ', имя_столбца1, имя_столбца2***) FROM** *имя_таблицы*; — если аргументов много, используем этот синтаксис для рациональности. Первым аргументом ставим разделитель.
INSERT(*str, pos, len, new_str***)** Возвращает строку str, в которой подстрока, начинающаяся с позиции pos и имеющая длину len символов, заменена подстрокой new_str.
LPAD(*str, len, dop_str***)** Возвращает строку str, дополненную слева строкой dop_str до длины len.
RPAD(*str, len, dop_str***)** Возвращает строку str, дополненную справа строкой dop_str до длины len.
LTRIM(*str***)** Возвращает строку str, в которой удалены все начальные пробелы. Эта строковая функция удобна для корректного отображения информации в случаях, когда при вводе данных допускаются случайные пробелы.
RTRIM(*str***)** Возвращает строку str, в которой удалены все конечные пробелы.
TRIM(*str***)** Возвращает строку str, в которой удалены все начальные и конечные пробелы.
LOWER(*str***)** Возвращает строку str, в которой все символы переведены в нижний регистр. С русскими буквами работает некорректно, поэтому лучше не применять.
UPPER(*str***)** Возвращает строку str, в которой все символы переведены в верхний регистр. С русскими буквами так же лучше не применять.
LENGTH(*str***)** Возвращает длину строки str.
LEFT(*str, len***)** Возвращает len левых символов строки str.
RIGHT(*str, len***)** Возвращает len правых символов строки str.
REPEAT(*str, n***)** Возвращает строку str n-количество раз.
REPLACE(*str, pod_str1, pod_str2***)** Возвращает строку str, в которой все подстроки pod_str1 заменены подстроками pod_str2.
REVERSE(*str***)** Возвращает строку str, записанную в обратном порядке.
LOAD_FILE(*file_name***)** Эта функция читает файл file_name и возвращает его содержимое в виде строки.
CURDATE(), **CURTIME()** и **NOW()** — Первая функция возвращает текущую дату, вторая — текущее время, а третья — текущую дату и время.
ADDDATE(*date*, **INTERVAL** *value***)** — Функция возвращает дату *date*, к которой прибавлено значение *value*. Значение *value* может быть отрицательным, тогда итоговая дата уменьшится. В качестве значения *value* могут выступать не только дни, но и недели (**WEEK**), месяцы (**MONTH**), кварталы (**QUARTER**) и годы (**YEAR**).
SUBDATE(*date*, **INTERVAL** *value***)** — функция идентична предыдущей, но производит операцию вычитания, а не сложения.
PERIOD_ADD(*period, n***)** — функция добавляет n месяцев к значению даты period. Нюанс: значение даты должно быть представлено в формате YYYYMM.
TIMESTAMPADD(*interval, n, date***)** — функция добавляет к дате *date* временной интервал *n*, значения которого задаются параметром *interval*. Возможные значения параметра interval: **FRAC_SECOND** — микросекунды **SECOND** — секунды **MINUTE** — минуты **HOUR** — часы **DAY** — дни **WEEK** — недели **MONTH** — месяцы **QUARTER** — кварталы **YEAR** – годы.
TIMEDIFF(*date1, date2***)** — вычисляет разницу в часах, минутах и секундах между двумя датами.
DATEDIFF(*date1, date2***)** — вычисляет разницу в днях между двумя датами.
PERIOD_DIFF(*period1, period2***)** — функция вычисляет разницу в месяцах между двумя датами, представленными в формате YYYYMM.
TIMESTAMPDIFF(*interval, date1, date2***)** — функция вычисляет разницу между датами *date2* и *date1* в единицах, указанных в параметре *interval*.
SUBTIME(*date, time***)** функция вычитает из времени date время time.
DATE(*datetime***)** — возвращает дату, отсекая время.
TIME(*datetime***)** — возвращает время, отсекая дату.
TIMESTAMP(*date***)** — функция принимает дату *date* и возвращает полный вариант со временем.
DAY(*date***)** и **DAYOFMONTH(***date***)** — функции-синонимы, возвращают из даты порядковый номер дня месяца.

DAYNAME(date**)**, **DAYOFWEEK(***date***)** и **WEEKDAY(***date***) —** функции возвращают день недели, первая — его название, вторая — номер дня недели (отсчет от 1 — воскресенье до 7 — суббота), третья — номер дня недели (отсчет от 0 — понедельник, до 6 – воскресенье.
WEEK(*date***)**, **WEEKOFYEAR(***datetime***) —** обе функции возвращают номер недели в году, первая для типа *date*, а вторая — для типа *datetime*, у первой неделя начинается с воскресенья, у второй — с понедельника.
MONTH(*date***) и MONTHNAME(***date***)** — обе функции возвращают значения месяца. Первая — его числовое значение (от 1 до 12), вторая — название месяца.
QUARTER(*date***)** — функция возвращает значение квартала года (от 1 до 4).
YEAR(*date***) —** функция возвращает значение года (от 1000 до 9999).
DAYOFYEAR(*date*) возвращает порядковый номер дня в году (от 1 до 366).
HOUR(*datetime***)** возвращает значение часа для времени (от 0 до 23).
MINUTE(*datetime***)** возвращает значение минут для времени (от 0 до 59).
SECOND(*datetime***)** возвращает значение секунд для времени (от 0 до 59).
EXTRACT(*type* **FROM** *date***)** возвращает часть date определяемую параметром *type*:
SELECT EXTRACT(*YEAR* **FROM** *'2011-04-17 23:15:18'***) AS** *year***,**
EXTRACT(*MONTH* **FROM** *'2011-04-17 23:15:18'***) AS** *mon***,**
EXTRACT(*DAY* **FROM** *'2011-04-17 23:15:18'***) AS** *day***,**
EXTRACT(*HOUR* **FROM** *'2011-04-17 23:15:18'***) AS** *hour***,**
EXTRACT(*MINUTE* **FROM** *'2011-04-17 23:15:18'***) AS** *min***,**
EXTRACT(*SECOND* **FROM** *'2011-04-17 23:15:18'***) AS** *sec***;**
TO_DAYS(*date***)** и **FROM_DAYS(***n***)** взаимообратные функции. Первая преобразует дату в количество дней, прошедших с нулевого года. Вторая, наоборот, принимает число дней, прошедших с нулевого года и преобразует их в дату.
UNIX_TIMESTAMP(*date***)** и **FROM_UNIXTIME(***n***)** взаимообратные функции. Первая преобразует дату в количество секунд, прошедших с 1 января 1970 года. Вторая, наоборот, принимает число секунд, с 1 января 1970 года и преобразует их в дату.
TIME_TO_SEC(*time***)** и **SEC_TO_TIME(***n***)** взаимообратные функции. Первая преобразует время в количество секунд, прошедших от начала суток. Вторая, наоборот, принимает число секунд с начала суток и преобразует их во время.
MAKEDATE(*year, n***)** функция принимает год и номер дня в году и преобразует их в дату.