**What is DataBase Testing?**

DataBase – storage area (no matter whether it’s mobile or desktop or web application software, each of them store data in database; if we want to communicate with database we should know SQL; this comes to backend testing.

Here mainly we’re gonna test operations related to database; it can be conducted along with UI side of application/software. DataBase testing consists of both black box and white box testing. We’ll focus on database not on UI.

Изображение выглядит как текст, диаграмма, снимок экрана, дизайн

Автоматически созданное описание

On the picture above there are various types of testing presented which are linked with DataBase testing – our main flow and focus during this course.

Pre-requisite for conducting database testing is knowing SQL, so this is gonna be our first Module here.

-----------------------------------  
  
  
  
  
**Module 1: SQL**

**Day-1**

At first there was not any database System, data was stored in different text files, then it became overwhelming to store data like that. Excel files came to picture, and it was much simpler but still not good enough to encompass all the operations which were necessary to retrieve the data.   
That’s how Data Base Management Systems appeared. There was much convenience to retrieve the data and execute diverse queries in general.

And then RDBMS (Relational Data Base Management Systems) conquered the market and it turned out possible to work with data linked through different tables of data base.

RDBMS: products:

Oracle, MS-SQL Server, DB2, MySQL, MS-Access; some of them are opensource some of them expect license subscription. MySQL – opensource.

We surely are gonna stick to RDBMS.

**SQL (Structural Query Language)** – special language to interact with data stored in the form of tables within a data base. Almost every client software for communicating with data bases provides the same set of requisites, syntax, and stuff. For creating, updating, deleting, retrieving data (for manipulating with data) we use SQL.

Every data base has two components: **Client and Server**. Server is responsible for storing the data, I perform any operations/actions from the client. Client is a tool from where we need to execute all desirable commands, server and clients are both software. Client is a very light tool doesn’t occupy much space. But all commands are executed on the sever side.   
Server usually is on the remote machine (picturing a case when you work for a company remotely). We have client on our machine (MS SQL Server), and we connect with server by providing certain set of credentials.

Type clients:  
 🡪 **Graphical mode(GUI)**  
 🡪 **CL I (Command Line Interface)**  
  
For example:   
MySQL -------------🡪 **MySQLWorkbench (GUI)**  
 **MySQL CommandLinetool (CL I)**

So we can connect to any data base, produced by any company through any client.

Then I installed MySQL WorkBench and MySQL Shell for later on purposes.

---------------------------------------

**Day-2**

SQL Language characteristics:

1. **DDL** (Data Definition Language)

2. **DML** (Data Manipulation Language)

3. **DRL** (Data Retrieval Language)

4. **TCL** (Transaction Control Language)

5. **DCL** (Data Control Language)

All these commands are selected by circumstances and conditions. I mean Administrators, Developers, Testers, Designers – everyone will use one or few languages to complete their tasks.

**DDL** commands: CREATE, ALTER, DROP, TRUNCATE, RENAME (if we want to create or change or delete the tables we refer to these commands)

**DML** commands: INSERT, UPDATE, DELETE (here we work with already existing tables, databases to manipulate with data)

**DRL** commands: SELECT (most popular and important)

**TCL** commands: COMMIT, ROLLBACK, SAVE POINT (after completing the action it will not automatically commit changes here we need to use commit, or by mistake I made an action and I have to get back in time, I can use rollback)

**DCL** commands: GRANT, REVOKE (used by administrators)

-------------------

Basic entities in database testing: DataBase and Tables;

Firstly, let us consider prominent commands for creating and deleting new database:

**CREATE DATABASE <name\_of\_db>;** - creating new database

**DROP DATABASE <name\_of\_db>;** - removing database

**CREATE SCHEMA <name\_of\_db>;** - creating new database (used in MySQL as well)

**DROP SCHEMA <name\_of\_db>;** - removing database (used in MySQL as well)  
  
But to avoid a possible error it’s better to use such command:

**CREATE DATABASE IF NOT EXISTS <name\_of\_db>;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Next essential thing is creating of tables, follow this syntax:  
**create table <table\_name> (col1 datatype, col2 datatype, col3 datatype,….)**

Example:

USE mydb;

CREATE TABLE STUDENT(SNO INT(5),SNAME VARCHAR(15),MARKS INT(3));

and there’s one command called DESCRIBE <name\_of\_table> which will generate the output with information about metadata of the table.

Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание

----------------------  
  
  
  
Inserting data into table – INSERT command:

INSERT INTO <TABLE\_NAME> VALUES(VAL1, VAL2, VAL3,…);

Example:

USE mydb;

**INSERT INTO STUDENT VALUES(101, ‘KIRAN’, 80);**  
  
after execution this insertion I was given such an output:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Which means that this manipulation was carried out successfully.

There’s also another way to insert data into table, observe this:

**INSERT INTO STUDENT (SNAME, SNO, MARKS) VALUES (‘RAM’, 102, 60);**

Pay attention that in the previous approach we were inserting data in order as fields were created, but here we intentionally define the order by providing fields’ names after name of the table!!!

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Sometimes we might face the situation when we simply don’t know which value to transmit to the table as an inserting value, in such cases we may pass NULL value:

**INSERT INTO STUDENT VALUES (103, ‘KRISHNA’, NULL);**

Here NULL is not an empty, but it’s not an actual value. Just unknown field.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

-------------------------  
  
  
  
Selecting rows from a table – SELECT command:

For the start I simply execute some sample commands to get adjusted to the tool MySQL and to the SQL language itself:

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Here I specified just 3 columns from table employee (this database by the way I imported from opensource platform)

**SELECT <col1, col2, …> FROM <Name\_of\_Table>** OR **SELECT \* FROM <Name\_of\_Table>**

Suppose I want to assign table’s columns in my way. I can achieve that by giving aliases to selected columns:

**SELECT <col1 al1, col2 al2,…> FROM <Name\_of\_Table>** where al1, al2 – aliases of corresponding columns.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

And also I may add some number to numeric column (for ex.: in my case I want to append 5000 to my salary column and give it an alias name as well)

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

SQL Data types:

* Numeric
* Text
* Data/time

But those were just basic prominent categories/big clusters of data in SQL which might be as well split into more detailed subsets.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

------------------------  
  
  
  
  
**Day-3**

**WHERE Keyword** – filtering records based on condition)

Here I’m gonna execute various number of queries containing this keyword implying the filtering data from the table:

Template: **SELECT <col1,col2…> FROM <Name\_of\_table> WHERE condition(s)**   
  
**SELECT \* FROM employees WHERE salary > 60000:**

Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание

**SELECT \* FROM employees WHERE salary <= 70000:**

Изображение выглядит как текст, программное обеспечение, число, снимок экрана

Автоматически созданное описание

**SELECT \* FROM employees WHERE person\_id = 9:**

**Изображение выглядит как текст, программное обеспечение, снимок экрана, число

Автоматически созданное описание**

**SELECT \* FROM employees WHERE leave\_reason is NULL:**

**Изображение выглядит как текст, число, линия, Шрифт

Автоматически созданное описание**

**SELECT \* FROM employees WHERE hire\_date = ‘2023-03-03’:**

**Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание**

There’s another keyword we often employ, this word is **DISINCT** here I’m gonna throw a couple examples how we may specify this word in our queries:  
  
In the screenshot below you might observe that I impart DISTINCT keyword to one particular column ‘currencies’ and that means that as an outcome I’ll receive only this column with unrepeatable values:

At first I’ll execute query without distinct keyword:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

And now with it:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

We’re moving on and now it’s time to talk about Logical Operators

(AND, OR, NOT)

Here we’d like to check more than one condition in case of ‘AND’ and ‘OR’.

First operator for observing is AND:

Sample: **SELECT <col1,col2,…> FROM <name\_of\_table> WHERE <cond1> AND <cond2>**

**SELECT \* FROM employees WHERE salary > 70000 AND id < 109:**

**Изображение выглядит как текст, число, программное обеспечение, линия

Автоматически созданное описание**

Now let’s display query using OR operator:

Sample: **SELECT <col1,col2,…> FROM <name\_of\_table> WHERE <cond1> OR <cond2>**

**SELECT \* FROM employees WHERE salary > 70000 OR id < 109:**

**Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание**

Now it’s turn of NOT operator:

Sample: **SELECT <col1,col2,…> FROM <name\_of\_table> WHERE NOT <cond>**

**SELECT \* FROM employees WHERE NOT currency = 22:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

Next thing for discussion is Between and In operators

Between is used to display the rows which is following in the range of values (the opposite operator is not between):

Sample: **SELECT <col1,col2> FROM <table\_name> WHERE <any\_col> BETWEEN <val1> AND <val2>:**

**SELECT \* FROM employees WHERE salary BETWEEN 60000 AND 90000:**

**Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание**

**NOT BETWEEN** operator**:**

Sample: **SELECT <col1,col2> FROM <table\_name> WHERE <anycol> NOT BETWEEN <val1> AND <val2>:**

**SELECT \* FROM employees WHERE salary NOT BETWEEN 60000 AND 90000:**

**Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание**

Here In keyword comes:

In operator return the rows which are matching with certain values from the given list (There’s also NOT IN operator):

Sample: **SELECT <col1,col2..> FROM <tablename> WHERE <anycol> IN (val1,val2,…).**

**SELECT \* FROM employees WHERE salary IN (25600.350, 95600.350, 105600.350):**

Изображение выглядит как текст, программное обеспечение, число, снимок экрана

Автоматически созданное описание

Let’s now consider backward operator NOT IN:

Sample: **SELECT <col1,col2..> FROM <tablename> WHERE <anycol> NOT IN (val1,val2,…).**

**SELECT \* FROM employees WHERE salary NOT IN (25600.350, 95600.350, 105600.350):**

**Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание**

The next topic we’re about to look at is Pattern Matching. Such method works with characters. When I want to retrieve the data from the table there are couple ways to do that employing patterns:

% 🡪 many characters

\_ 🡪 single character.

Sample:

**SELECT <col1,col2..> FROM <tablename> WHERE <anycol> LIKE <pattern>;**

For example:

**SELECT \* FROM persons WHERE first\_name LIKE ‘%a%’;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

Such command implies retrieving the data with the first names where there’s a character ‘a’.

Because I % means any number of characters even 0.

**SELECT \* FROM persons WHERE first\_name LIKE ‘\_ani%’;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание**

And as It was at the previous instances we can use NOT with LIKE Keyword as well (negation):

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

----------------------------------------

**Day-4**

**DDL Commands (Data Definition Language):**

1) CREATE

2) ALTER

3) DROP

4) TRUNCATE

5) RENAME

**Create & Alter:**

**CREATE** is used to create database objects(Tables, views, synonymes, etc…)

**ALTER**:

* Adding a new column
* Dropping an existing column
* Modifying an existing column (Increase/Decrease size of any column & change a data type of a column)
* Renaming a column

More detailed tangible analysis:

At first I want to create new table within mydb database. For that I need to execute the following command:

**CREATE TABLE STUDENT (SID INT (4), SNAME VARCHAR (15)):**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Then I just put three records into the table to work with them later on.

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Now we’ll observe the usage of ALTER command:

For the firs step I’ll examine appending new column, my requirement will be to add student grade column. Let’s see.

Sample: **ALTER TABLE <name\_of\_table> ADD(<name\_of\_column> <type\_of\_column(length)>);**  
**ALTER TABLE STUDENT ADD(GRADE VARCHAR(2));**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Suppose, I want to drop a column from the table. It means I want to cut it off, eliminate from any column. I need to employ Alter option as well:

Sample: **ALTER TABLE <name\_of\_table> DROP COLUMN <name\_of\_column>;**  
**ALTER TABLE STUDENT DROP COLUMN GRADE;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

Now we’ll see how to modify the existing column (for instance increasing size of the column):

Sample: **ALTER TABLE <tablename> MODIFY COLUMN <column\_name> <preferrable\_type/size>;  
ALTER TABLE STUDENT MODIFY COLUMN SNAME VARCHAR(20);**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание**

And there’s another operation with alter keyword considering renaming process (I’d rather change the name of some column).

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

We’re moving further and the next commands in our vision is DROP commands:

If we want to completely remove any database object we normally use DROP command.

Compared to DROP command, by using TRUNCATE command we only delete all the data from database, table or others database objects, structure of these objects stays put.

There’s one similar command named DELETE, where we also may delete all the data but we can roll it back lately.

Before executing these commands we need to set up one property: **SET autocommit = 0**; because currently this parameter is configured as 1. To implement ROLLBACK command after DELETE command we must not commit our action, that’s why it’s important to switch ‘autocommit’ to 0.  
  
Let’s investigate DELETE command first (assumably I want to delete data from any table):

Sample: **DELETE FROM <name\_of\_table>**

**DELETE FROM STUDENT:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Therefore I got idle table with no data in it. So DELETE command executed successfully. But what if I want my data back. In this case I need to apply **ROLLBACK** command:

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Alternative command for deleting the data is TRUNCATE, the only twist here is that I can not roll my data back here:  
  
sample: **TRUNCATE TABLE <name\_of\_table>**

**TRUNCATE TABLE STUDENT;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Finally we’re rendering DROP command, where I intent to remove not only the data but table itself.

Sample: **DROP TABLE <name\_of\_table>**

**DROP TABLE STUDENT:**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

There’s one more command I want to unveil: RENAME (when I want to rename the table name):

Sample: **RENAME TABLE <table\_name> TO <new\_table\_name>**

**RENAME TABLE memos TO memos;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

-----------------------------------------------

**Day-5**

**Build-in functions in MySQL** are presenting keywords which we can describe while executing some query to obtain data from database.

4 types of functions available:

1) String functions – operate on string data types

2) Numeric functions – operate on numeric data types

3) Date functions – operate on date data types

4) Aggregate functions – operate on all of the data types and produce summarized result sets.

**String functions:**

**1) Upper():** converts into upper case letters.

Sample**: SELECT UPPER(<name\_of\_column(string)>) FROM <name\_of\_table>;**

**SELECT UPPER(first\_name) FROM persons;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

**2)** **Lower():** converts into upper case letters.

Sample: **SELECT LOWER(<name\_of\_column(string)>) FROM <name\_of\_table>;**

**SELECT LOWER (first\_name) FROM persons;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

**3) Length():** return the length of a string.

Samples: **SELECT LENGTH(<name\_of\_column>) FROM <name\_of\_table>;**

**SELECT \* FROM<name\_of\_table> WHERE LENGTH(<name\_of\_column>) = 4;**

**SELECT \* FROM persons WHERE LENGTH(first\_name) = 4;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**4) TRIM()**: removing specific characters from both sides of the string

Sample: **SELECT TRIM(“ <any\_characters> ”)**

**SELECT TRIM(‘f’ from ‘f<any\_characters>f’)**

**SELECT TRIM(‘zz’ from ‘zzORACLEzz’);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**5) INSTR():** Returns the position of the character within a string:

Sample**: SELECT INSTR(‘any\_characters’, ‘spec\_char’);**

**SELECT INSTR('Welcome', 'm');**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**6) SUBSTR()/SUBSTRING():** returns the substring of a string

Sample: **SELECT SUBSTR(‘string’, <digit\_of\_start\_pos>, <number\_of\_char\_to\_count>)**

**SELECT SUBSTR(‘ORACLE’, 2, 3);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**SELECT SUBSTRING(‘crocodile’, 5, 3);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**7) CONCAT()** – aggregate two strings into one.

Sample: **SELECT CONCAT(‘<one\_column\_name>’, ‘another\_column\_name’) FROM <name\_of\_table>**

**SELECT CONCAT(first\_name, last\_name) FROM persons;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**Numeric functions:**

**1) ABS()** – returns absolute value

Sample: **SELECT ABS(<num\_value>);**

**SELECT ABS(-40); SELECT ABS(40);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**2) SQRT()** – return squared root of the number;

Sample: **SELECT SQRT(<num\_value>)**

**SELECT SQRT(36);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**3) MOD()** – returns the remainder from the division

Sample: **SELECT MOD(<first number>, <divider>)**

**SELECT MOD(10,3);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**4) POWER()** – returns the number raised into n-degree

Sample: **SELECT POWER(<number1>, <number2>)**

**SELECT POWER(2, 5);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**5) TRUNCATE()** – truncates a number to the specified number of decimal places

Sample: **SELECT TRUNCATE(<number>, <spec\_number\_to\_truncate>)**

**SELECT TRUNCATE(40.1234, 3);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**SELECT TRUNCATE(40.1234, 2);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**SELECT TRUNCATE(4012421, -2);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**6) GREATEST() & LEAST()** returns greatest, least values in the provided values.

Sample: **SELECT GREATEST(<num1>, <num2>…);**

**SELECT GREATEST(100,200,300,400,500);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**SELECT LEAST(100,200,340);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**Date Functions  
 1) CURDATE()/CURRENT\_DATE()** –returns current date

Sample **SELECT CURDATE(); SELECT CURRENT\_DATE();**

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

**2) CURTIME()/CURRENT\_TIME()** – returns current time

Sample: **SELECT CURTIME(); SELECT CURRENT\_TIME();**

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

**3) NOW()** – returns both current date and time

Sample: **SELECT NOW();**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

**4) SYSDATE()** – returns current date and time

Sample: **SELECT SYSDATE();**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**5) YEAR(), MONTH(), DAY()** – returns year, month, day correspondingly

Sample: **SELECT MONTH(<”given\_date”>), YEAR(<”given\_date”>), DAY(<”given\_date”>);**

**SELECT MONTH("2023-07-19") as Mon, YEAR("2023-07-19") as Yea, DAY("2023-07-19") as Day;**

Изображение выглядит как текст, программное обеспечение, число, Значок на компьютере

Автоматически созданное описание

Let’s execute one trial query:

**SELECT \* FROM employees WHERE YEAR(leave\_date) = "2024";**

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание

**Aggregate functions**They are all performing calculations on multiple rows of a single column of a table and returning a single value.

**1)MAX()** – returns max value from the column

Sample: **SELECT MAX(<col\_name>) FROM <name\_of\_table>**

**SELECT MAX(salary) FROM employees;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**2) AVG()** – returns average value from the column

Sample: **SELECT AVG(<col\_name>) FROM <name\_of\_table>**

**SELECT AVG(salary) FROM employees;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**3) MIN()** – returns minimun value from the column

Sample: **SELECT MIN (<col\_name>) FROM <name\_of\_table>**

**SELECT MIN (salary) FROM employees;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**4) SUM()** – returns sum of the values from the column

Sample: **SELECT SUM (<col\_name>) FROM <name\_of\_table>**

**SELECT SUM (salary) FROM employees;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

**5) COUNT()** – returns number of…

Sample: **SELECT COUNT (<col\_name>) FROM <name\_of\_table>**

**SELECT COUNT (\*) FROM employees;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

--------------------------------------

**Day – 6**

**Group By, Having, Order By.**

**1) Group by operator**:   
Several statements about this operator:

* The GROUP BY clause groups records into summary rows
* GROUP BY returns one record for each group
* GROUP BY typically involves aggregates: COUNT, MAX, MIN, AVG…
* GROUP BY can group by one or more columns.
* All the columns in the select list should include in group by clause.

Let’s incur a close look to the execution of queries with usage of GROUP BY clause:

Sample: **SELECT <name\_of\_col>, agg\_fun FROM <name\_of\_table> GROUP BY <name\_of\_col>**

Current database I’m gonna work with:

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание

**SELECT department\_id, SUM(salary) FROM employees GROUP BY department\_id;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

**SELECT division\_id, COUNT(\*) FROM employees GROUP BY division\_id;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

In the next example I exhibit how it’s literally possible to group rows by several columns:

**SELECT division\_id, department\_id, COUNT(\*) FROM employees GROUP BY department\_id, division\_id;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

To sum up we can retrieve the results by implementing group by clause, and by that we group our data.

But supposedly I want to apply some conditions, impose some restraints on my grouping, that’s when having keyword appears.

**2) Having keyword**.

This special keyword we only employ on top of group by clause if we want to filter some data:

Sample: **SELECT <name\_of\_col>, agg\_fun FROM <name\_of\_table> GROUP BY <name\_of\_col> HAVING <condition>**

**SELECT division\_id, SUM(salary) FROM employees GROUP BY division\_id HAVING SUM(salary) > 200000;**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Here by giving specified condition under HAVING keyword we only got 2 divisions out of 4!

We can also execute query having look like this:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Where I used Where keyword as well to impose condition in advance. I can act like this. So I write where keyword before the grouping.

**3)** Finally, I’ve moved to **ORDER BY** clause. I exploit this particular clause when I want to arrange the rows in some order (ascending or descending). – Sorting the records.

Simple Sample: **SELECT <name\_of\_col> FROM <name\_of\_table> ORDER BY <name\_of\_any\_col>;**

**SELECT \* FROM employees ORDER BY department\_id;**

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание

And we may type our complex queries using all the discussed commands, clauses all together, ORDER BY, GROUP BY, WHEN, HAVING

There you go:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

--------------------------------------

**Day-7**   
Set operators: **UNION, UNION ALL**, INTERSECT, MINUS

But MySQL doesn’t support INTERSECT and MINUS so we’ll focus on UNION and UNION ALL.

Basically these operators will be used to combine several tables, so we intend to retrieve data from multiple tables.

There are 3 methods to capture the data from a few tables, first method – our today’s object of attention – UNION. The others two we’ll discuss later on.

When I use just UNION It won’t provide me duplications, but UNION ALL will also give the duplicated values(data).

There are several statements(restraints) we have to comply while working with UNION operator:

* The UNION operator is used to combine the result-set of two or more SELECT statements.
* Each SELECT statement within UNION must have the same number of columns.
* The columns must also have similar data types.
* The columns in each SELECT statement must also be in the same order.

Firstly, I composed a couple of tables to work with:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

**SELECT NUM FROM A UNION SELECT NUM FROM B;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание**

But If I want to consider duplicated values as well implying this particular example I should employ UNION ALL operator:

**SELECT NUM FROM A UNION ALL SELECT NUM FROM B;**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание**

**----------------------------------------**

**Day – 8**

**JOIN** operators**.**

With help of them we may retrieve data from two or more tables.

The tables are mutually related through primary and foreign keys.

Types of JOIN:

* Inner JOIN/Simple JOIN
* Left JOIN
* Right JOIN
* Full JOIN
* Self JOIN

Изображение выглядит как текст, снимок экрана, диаграмма, круг

Автоматически созданное описание

For the commence created a couple of tables for later considerations:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Now I’ visually demonstrate how those JOINs are gonna work:

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

That was a brief example of Inner JOIN. Now it’s turn of Left JOIN:

**Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание**

At the example above we managed to see how some records got matched and a couple of values from left table (TAB1) were unmatched.

By using Right JOIN we’ll get the opposite case but for right table (TAB2):

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Full JOIN is not supported in MySQL, but it’s not prohibited to exploit others SQL servers.

Now let’s delve into another instance:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

It’s even conceivable to employ aliases for tables in my queries:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

The next topic in our roster is subqueries. It’s a query inside another query. It’s helpful in order to reduce queries. Make them look smaller.

Let’s consider a simple example:

**SELECT salary FROM employees WHERE salary > (SELECT salary FROM employees**

**WHERE person\_id=10);**

Here I want to extract salaries which are bigger than person\_id=10’s salary:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

We may take a look at another example (now my requirement is to find out what’s the second biggest salary in the company. Here I come):

**SELECT MAX(salary) FROM employees**

**WHERE salary < (SELECT MAX(salary) FROM employees);**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

And now the third salary from the top:

**SELECT MAX(salary) FROM employees WHERE salary < (SELECT MAX(salary) FROM employees**

**WHERE salary < (SELECT MAX(salary) FROM employees));**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Now my desire is to display all employees whose salaries equal to the greatest salary in the company:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Each instance above concerning subqueries was comprised of single row subquery. But now I suggest that we’ll immense deeper into multi-rows subqueries. Why don’t we?

**SELECT \* FROM employees WHERE salary IN**

**(SELECT salary FROM employees WHERE department\_id = 5);**

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание

OR:

**SELECT \* FROM employees WHERE salary > ANY**

**(SELECT salary FROM employees WHERE department\_id = 5);**

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

------------------------------------

**Day – 9**

**Integrity Constraints** – special rules/conditions imposed on columns of tables. Some sets of restrictions you might say. There are a few groups of these constraints. Let’s observe them.

Constraints can be described when table is being created (CREATE TABLE …) or when we want to alter any pending table (ALTER TABLE …):

* NOT NULL – ensures that a column does not have any null values
* UNIQUE - ensures that each value of this column is unique.
* PRIMARY KEY - a combination of NOT NULL and UNIQUE options. Exclusively identifies each row.
* FOREIGN KEY - uniquely specifies each row/record in another table.
* CHECK – ensures that every value in this column satisfies a certain condition.
* DEFAULT – sets a default value for a column when no value is specified.

Now we’re gonna consider one by one more closely. First one is **NOT NULL**:

First of all I need to create new table (student1) where I put some NOT NULL constraint on ‘sno’ column:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

By next step I attempted to append improper record to the table, what I got a conforming error message for:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

For a later exploration there’s a mandatory to drop the table which is why I have removed the corresponding (student1) table via a well-known DROP command.

Here’s my review on **UNIQUE** command. Suppose I’m not in a delight from seeing duplicate values in my column or probably there’s a requirement from a customer or logic issue resolving. Anyway, I’d like to identify each value, that’s why I refer to UNIQUE command.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

we may also use this constraint on a table level with asserting definite columns which we’d like to subdue under this condition. Here’s a picture:

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

I can literally list all the columns within parentheses, kind of like (sno, sname…) which means those columns would be imposed by this constraint concurrently. This way of assignment is much easier when we need to cover several columns at the same time.

Here I was trying to add duplicate value to the table which wound up to be unsuccessful:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

But a little twist appeared on my way of research. It turns out NULL values are acceptable and moreover in unlimited amount. It happens because NULL values are just unknown, idle ones that don’t have a peculiar weight and can be recurred unlimitedly.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

The next prominent constraint is **‘Primary key’**. If supposedly I want to establish a relationship between two tables, one of the tables surely must contain primary key. It’s a blend of NOT NULL and UNIQUE constraints. Here we also might prescribe this stipulation both on a column and table levels. There must be only one primary key constraint for a table.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

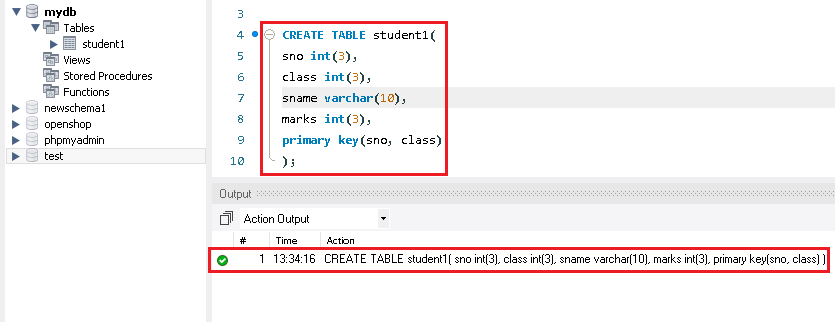
Автоматически созданное описание

Let me execute a few queries some of which are improper due to primary key constraint:

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Here’s a tiny swirl. Creation of composite primary key is conceivable. So I can consolidate two columns into one primary key. Vividly on a table level:



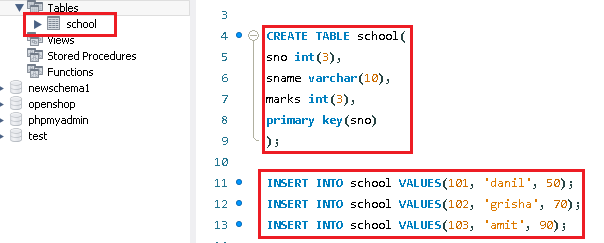
Time comes to a detailed talk about **FOREIGN KEY**:

A FOREIGN KEY is a key used to link two tables together.

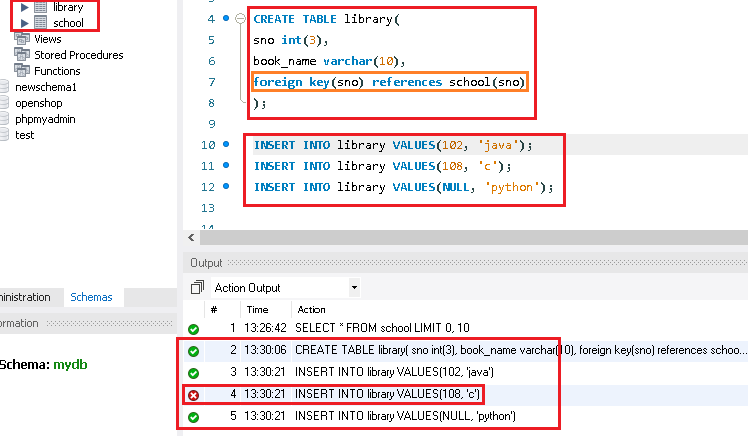
A FOREIGN KEY is a key (or collection of fields) in one table that refers to a PRIMARY KEY in another table.

For accomplishing further tasks I need to create two tables in one database with employing PRIMARY and FOREIGN keys.

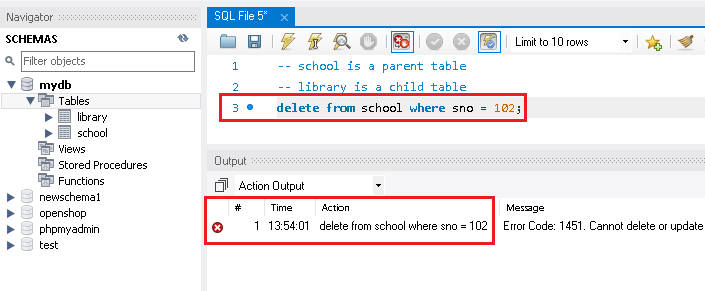
Firstly I have created a parent table (school) and put three records in it:



Then I have created a child table with foreign key imposed on column ‘sno’ with reference on corresponding column ‘sno’ of tenant table ‘school’. And I also tried to incorporate a few records where one of them wound up to be an error because there wasn’t matching value from a parent table(‘school’). Note that NULL value passed and turned out to be acceptable.



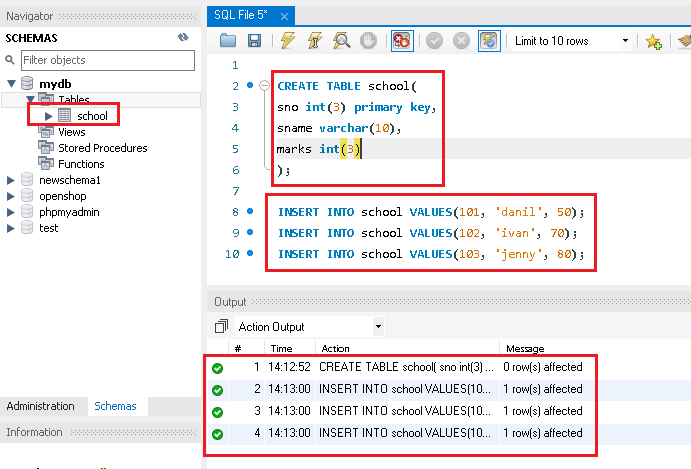
But here’s a twirling moment I’d like to pay attention on: according to all stipulations which I composed my tables on, I cannot eliminate record with ‘sno’ = 102 from the parent table because this record is referred by conforming record from a child table! Let’s throw a gaze on this attempt:

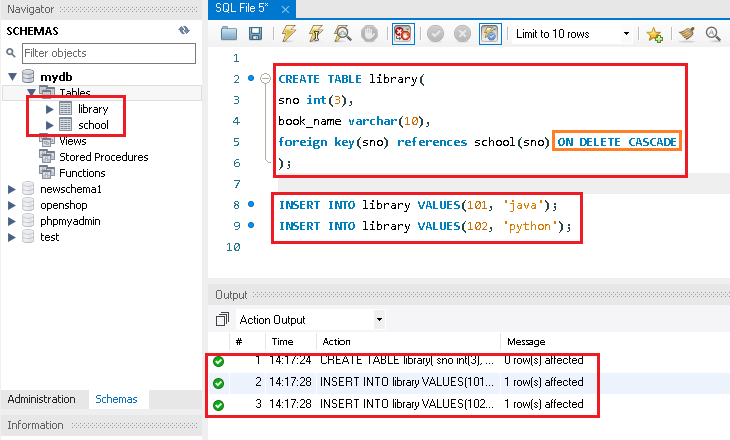


We’re supposed to delete all the rows from the child table containing all 102 records and then removing from the parent table of matching record will be allowed.

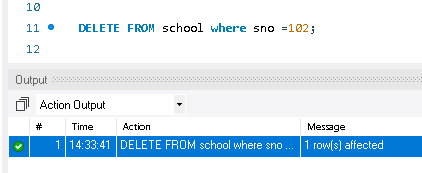
\*But there’s a circumventing option which means that we’ll be able to delete a record from a parent table even though there are some dependencies in a child table regarding to that particular row in the parent table. We should use ‘ON DELETE CASCADE’ stipulation (although it’s an exceptional case):

First of all I’ve dropped existing tables and recreated them again but with a slight alteration. Let’s see.





Here I’ve composed two tables like in the previous example, but additionally I ascribed **‘ON DELETE CASCADE’** option to child table where the reference comes from. This entitles me to delete parent table’s records independently from a child table. All records from child table referring to the matching parent table’s values will be removed as well.



Now we can see it’s conceivable.

Moving further and the next integrity constraint in our roster is **‘Check Constraint’**. This sort of stipulation is used to allow user insert only specific values to the column. Let’s observe an example.

I have created a table where I’ve tried to insert some records. Couple rows have been rejected due to ‘check’ constraint letting insert values from 50 to 100 into ‘marks’ column.

Изображение выглядит как текст, Шрифт, программное обеспечение, линия

Автоматически созданное описание

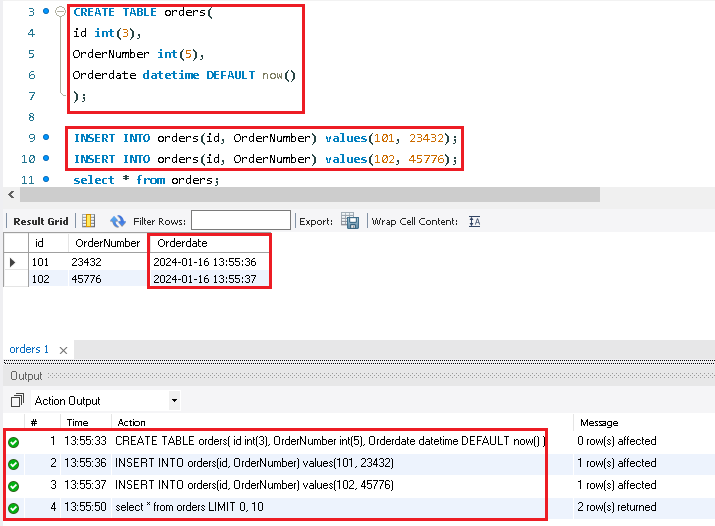
Going to next instance, where I’ve created new table and imposed peculiar constraint on ‘city’ column which I specified being chosen from list of 3 cities:

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

The second insert wasn’t pulled off since there was an attempt to insert a record with a city not from predefined list of cities.

Finally the last but not least constraint for today is **‘Default’** constraint. It’s employed when we need a column to have a default value in case we haven’t provided our own one. But if we put our value the row will be added(updated) with our value independently from default value assigned by creation or alteration of table.



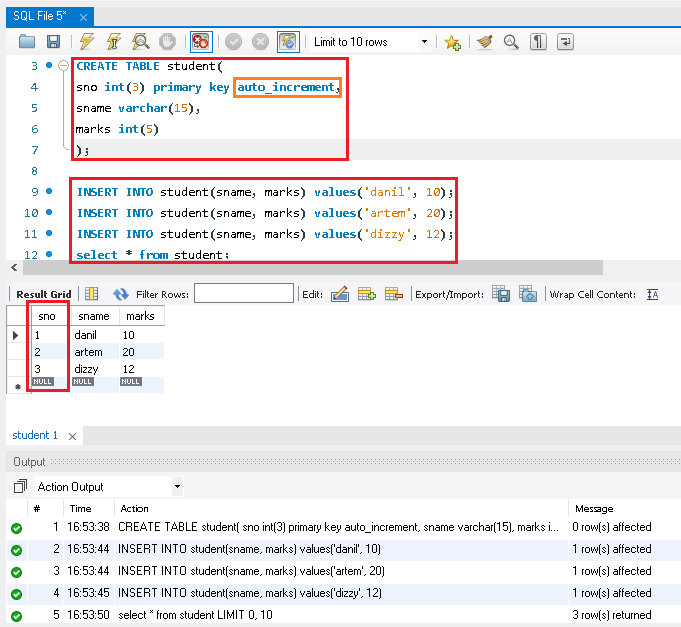
-----------------------------------

**Day – 10**

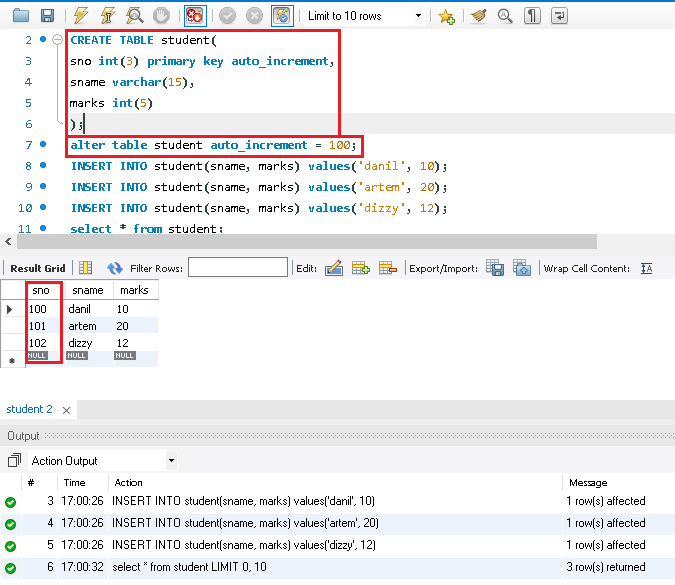
**AUTO INCREMENT & LIMIT**

Auto increment is a function applied to numeric data type which generates sequential numbers as records get added to the table. It must be set on a field having primary key instruction.

For perceptible instance I have created new table where I tuned primary key along with **AUTO INCREMENT** function to ‘sno’ column.

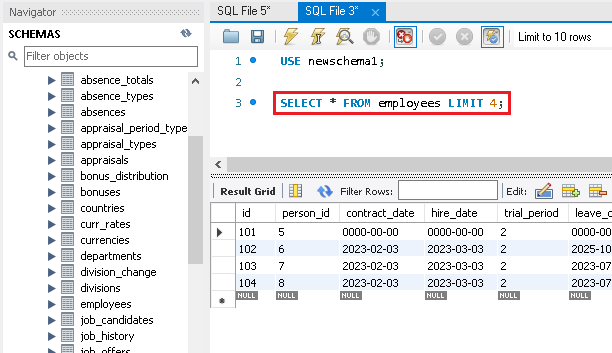


I even may set up the start digit of incrementing: like at the picture below I used an alter command to serve this. Now observe:



The next thing I want to discuss is **LIMIT**; suppose I want to retrieve a certain number of rows, I can configure that option through UI of MySQL client or directly through a query:

Like here I displayed only 4 first records of the table:



-------------------------------

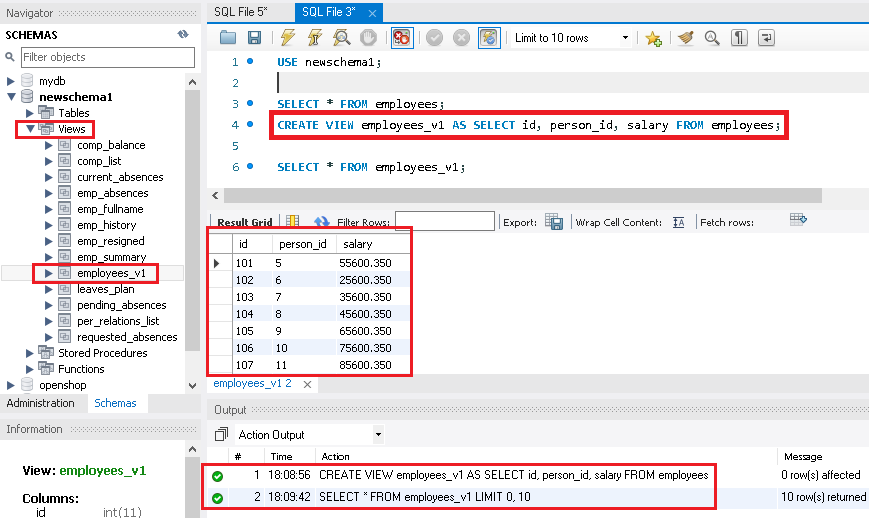
**Day – 11**

**Views & Indexes**

Views is a special database object (a virtual object) or we even may say abstract object shaped on base of result-set of SQL statement.

View represents rows and columns just like a single table. Consists of data from one or several tables.

And by taking a view separately we can conduct the same queries like with a real table (JOIN, WHERE statements…).



I can delete this view by implementing well-known command (drop view <desired view>)

One more important thing: if I change some common record for table(s) and view in a real table it will affect view by changing it too, but id doesn’t work backwards).

**INDEX:**

They’re used to speed up the query execution (performance-related database objects)

Not visible for users. They’re usually applied to columns comprised of complexed data.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание  
-----------------------------

**Day-12**

**TCL (Transactional control commands)**. Until now we’ve been mingling about DML commands. Now I want to sharpen my unprejudiced attention on two important TCL commands which are **‘autocommit’** and **‘rollback’**. Their names speak for themselves, but observation of several examples here will not be redundant. By default in MySQL autocommit command is enabled (equals to 1), but sometimes it’s disabled (equals to 0). In such cases after execution DML commands we’re compelled to apply ‘commit;’ command to confirm our data manipulation.

But if we preset autocommit as 0 (disabled), we may use rollback command for getting our data back!

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

What I’ve done here is created a table, inserted three records, then configured autocommit option as 0 (turned it off), deleted one row and then got it back by implementing rollback option. There’s a prominent moment I have to say about: if I switch autocommit to 1, I will not be able to employ rollback commant, because all DML commands will be committed automatically.

----------------------------