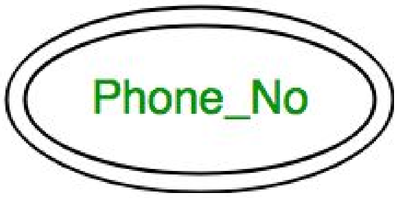
Birla Institute of Technology & Science Pilani, K.K Birla Goa Campus
CS-F212 Database Systems
Sem2 2020-2021
Lab 3 Reading Material - ER to Relational , Foreign Key and DQL Commands
- Harshit, Vivek,Tanishq,Aanand,Aditya

- **Multivalued Attribute**

  An attribute consisting of more than one value for a given entity. For example, a student can have multiple phone numbers, so **Phone_No** attribute is a multi-valued attribute. In the ER diagram, the multi-valued attribute is represented by a double oval.

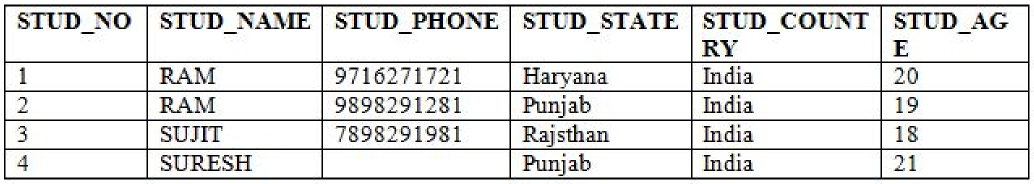


  Note: The multi-valued attribute can't be represented in a single table; we need another table to represent it. We can represent it in another table using **Foreign Key.**

- **Foreign Key**

Foreign Key is a field (or collection of fields) in a table(child table) that refers to the primary key in another table (parent table). The purpose of the foreign key is to link the two tables together. The table containing the foreign key is called the child table & table that it refers to (primary key) is called the reference/parent table.

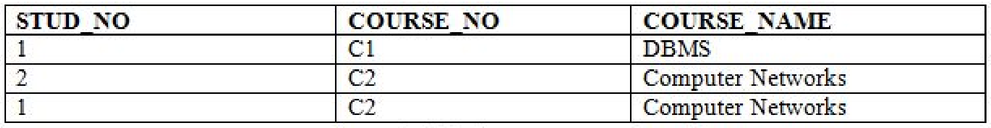
STUDENT



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE



STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

In the figure above the STUD_NO attribute in STUDENT_COURSE table is the foreign key referring to the STUD_NO attribute in the STUDENT (Parent) table.

<u>SQL Commands related to Foreign Key:</u>

<u>Using Create Table</u>

You can declare foreign key in the create table itself while creating the child table.

```
CREATE TABLE STUDENT_COURSE (
         STUD_NO int NOT NULL,
    COURSE_NO varchar(10) NOT NULL,
    COURSE_NAME varchar(30) not NULL,
    FOREIGN KEY (STUD_NO) REFERENCES STUDENT(STUD_NO)
);
```

This is the Foreign key (in the child table)

This is the of parent table

This is the primary key in the parent table that foreign Key refers to.

Note: It is better if you name the foreign key constraint so that you can drop (if required) or make any changes to this constraint later on.
If the FOREIGN KEY clause included a contraint name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the constraint name value is internally generated by InnoDB when the foreign key is created.
This name can be found using the `SHOW CREATE TABLE <tablename>;`

```
 CREATE TABLE STUDENT_COURSE (
      STUD_NO int NOT NULL,
      COURSE_NO varchar(10) NOT NULL,
    COURSE_NAME varchar(30) not NULL,
    CONSTRAINT Fk_Stud_course FOREIGN KEY (STUD_NO) REFERENCES
STUDENT(STUD_NO)
);
```

Using Alter Table

If you have created the child table without declaring the foreign key, then you can add it later using alter command.

```
ALTER TABLE STUDENT_COURSE
ADD CONSTRAINT FK_Stud_course
FOREIGN KEY (STUD_NO) REFERENCES STUDENT(STUD_NO);
```

Drop Foreign Key

```
 ALTER TABLE Orders DROP CONSTRAINT FK_Stud_course;
```

Integrity Constraints:

- **Domain integrity constraints:**
  These constraints set a range, and any violations that take place will prevent the user from performing the manipulations that caused the breach.
  Two types are:
  1. `NOT NULL:` By default, the table can contain null values. This constraint ensures that the table has a value.
  2. `CHECK`: This can be defined to allow only a particular range of values.

- **Entity integrity constraints:**
  There are two types of entity integrity constraints.

1. `UNIQUE`: The unique constraint designates a column or a group of columns as a unique key. This constraint allows only unique values to be stored in a column and thus it rejects duplication of records.
2. `PRIMARY KEY`: This constraint is the same as a unique constraint,, but in addition to preventing duplication it also does not allow null values. This constraint cannot be put on the column having ‘long’ data type.

- **Referential integrity constraints:**
  It enforces the relationship between tables. It designates a column or combination of columns as a foreign key. The foreign key establishes a relationship with a specified primary or unique key in another table called the referenced key. In this relationship, the table containing the foreign key is called the child table, and the table containing the referenced key is called the parent table.

Storage Engine

A storage engine is a software module a database management system uses to CREATE, READ and UPDATE from a database. There are two types of Storage Engines in MySQL:

1. Transactional: Transactional means that the write operations on the databases can be rolled back if they do not complete. These operations are known as transactions.
2. Non-Transactional: The impact of no Rollback/Commit is felt. To perform rollback operation, the user will need to do it manually with codes.

For MySQL 5.5 and later InnoDB is the default MySQL storage engine. InnoDB is an ACID-compliant storage engine. It supports row-level locking, crash recovery and multi-version concurrency control.
It is the only engine which provides foreign key referential integrity constraint.

To see the status information about the storage engines and checking whether a storage engine is supported, or to see what the default engine type:

```
SHOW ENGINES;
```

Conditions regarding Foreign Key Definitions:

- Foreign key relationships involve a parent table that holds the central data values and a child table with identical values pointing back to its parent. The FOREIGN KEY clause is specified in the child table. The parent and child tables must both be InnoDB tables. They must not be TEMPORARY tables.

- Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB to be compared without a type conversion. The size and sign of integer types must be the same. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.

- InnoDB permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are listed as the first columns in the same order.

- InnoDB rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table if there is no matching candidate key value in the parent table.

When an UPDATE or DELETE operation affects a key value in the parent table that has matching rows in the child table, the result depends on the referential action specified using ON UPDATE and ON DELETE subclauses of the FOREIGN KEY clause.

InnoDB supports majorly three options regarding the action to be taken:

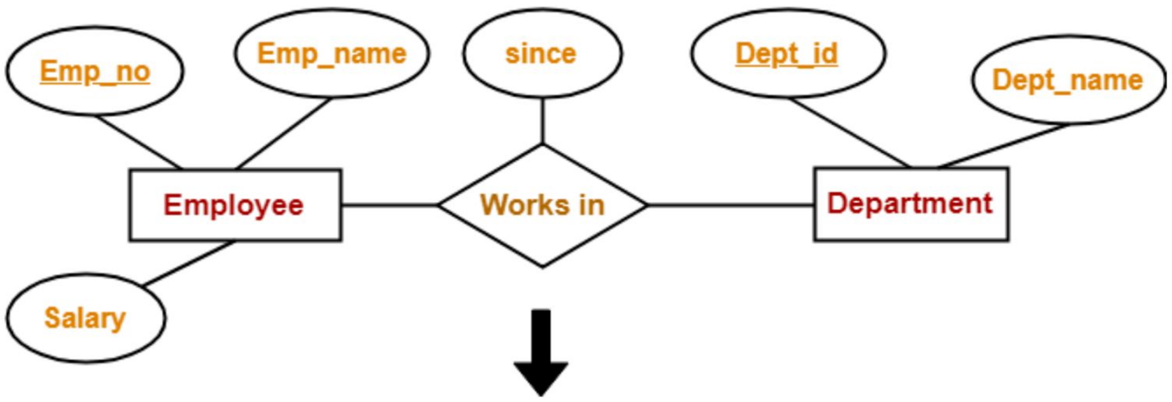
1. `CASCADE`: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table. Both ON DELETE CASCADE and ON UPDATE CASCADE are supported.

2. `SET NULL`: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to NULL. Both ON DELETE SET NULL and ON UPDATE SET NULL clauses are supported. If you specify a SET NULL action, make sure that you have not declared the columns in the child table as NOT NULL.

3. `RESTRICT`: Rejects the delete or update operation for the parent table. Specifying RESTRICT (or NO ACTION(deferred check) InnoDB doesn't do deferred checks hence these two are equivalent) is the same as omitting the ON DELETE or ON UPDATE clause. This is the default action if ON DELETE and ON UPDATE are not specified.

Example Question

Given the ER Diagram, make the corresponding tables:



```
Create table Employee(emp_no int primary key, emp_name varchar(30), salary int);
Create table Department(dept_id varchar(10) primary key, dept_name varchar(15));
Create table WorksIn(emp_no int, dept_id varchar(10), since date, primary
key(emp_no,dept_id), foreign key(emp_no) references Employee(emp_no), foreign key(dept_id)
references Department(dept_id) ON UPDATE CASCADE ON DELETE RESTRICT);
```

DQL Commands

Retrieving information from a table

The SELECT statement is used to pull information from a table.
Syntax:

SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
what_to_select can be a list of columns, or * to indicate all columns.

Selecting all data
mysql> SELECT * FROM <tablename>;
Example://to display all information of all students
mysql >SELECT * FROM students;

Selecting distinct rows
mysql>SELECT DISTINCT <column name> FROM<tablename>;
Example://to display names of all students ignoring duplicate names
mysql > SELECT DISTINCT name from student;

Selecting particular rows i.e; select with where clause

```
mysql>select column name1, column name2 ….. from tablename where condition (and or
conditions);
```

Examples:

```
/*to display name of a particular student*/
mysql >SELECT name FROM students where id = ‘PS99305018’;

/*to display id and name of students in hostel no 10 having percentage more than 90.6*/
mysql >SELECT id,name FROM students where percentage>90.6 and hostel=10;

/*to display id and name of students having percentage more than 90.6 but do not belong to
hostel no 10*/
mysql >SELECT id,name FROM students where percentage>90.6 and hostel != 10;
                                        OR
mysql >SELECT id,name FROM students where percentage>90.6 and not hostel = 10;

/*to display id and name of students having percentage between 60 and 90.6*/
mysql >SELECT id,name FROM students where percentage>=60.0 and percentage<=90.6;


/*to display id and name of students having percentage 60 and 90.6*/
mysql >SELECT id,name FROM students where percentage=60.0 or percentage=90.6;
```

Note: AND and OR may be intermixed. If you do that, it's a good idea to use parentheses to indicate how conditions should be grouped.

```
/*to view current_date*/
mysql> select curdate( );

/*to view current date and time*/
mysql> select now( );
     OR
   select sysdate( );
```

TRY THIS:->

```
 /*to do arithmetic in select clause*/
mysql>select percentage * 10.0 from students where hostel = 10;
```

It will display percentage*10 of those students that reside in Hostel=10

```
/* to name the result */
mysql>select percentage * 10.0 product from students where hostel = 10;
```

This is just to view the result of arithmetic operation and not to make any changes in the table. To do so we need another DML command named update.

Using inbuilt functions

```
/* display the names of those students whose birthday is in month of june. */
mysql> select name from students where month(bdate)=6;
```

similarly there are inbuilt functions for arithmetic, string, numeric and date operations which you can google.

Selecting particular columns

select column name ….from table name
Example:

```
mysql >SELECT id FROM students;
mysql >SELECT id, name FROM students;
```

SORTING ROWS

You may have noticed in the preceding examples that the result rows are displayed in no particular order. However, it's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an ORDER BY clause.

Example://to display names of all students sorted by percentage in ascending order

```
mysql> SELECT name, percentage FROM students ORDER BY percentage;
```

To sort in reverse order, add the DESC (descending) keyword to the name of the column you are sorting by:

```
mysql> SELECT name, percentage FROM students ORDER BY percentage DESC;
```

You can sort on multiple columns. For example, to sort by hostel no of students, then by name, use the following query:

```
mysql> SELECT name, hostelno, percentage FROM students ORDER BY hostelno, name
DESC;
```

The above query will first sort the rows based on hostelno (in ascending order), and then for same hostelno, sorting is done based on name (in descending or reverse order)

SQL LIMIT Clause

The LIMIT clause is used to specify the number of records to return from the top. The LIMIT clause can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Syntax:

```
SELECT column_name(s)
FROM table_name
LIMIT number;
```

Example:

- To list first 3 id nos:

```
mysql> SELECT name FROM students limit 3;
```

SQL pattern matching

SQL pattern matching allows you to use ‘_’ (underscore) to match any single character, and ‘%’ (percentage) to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case insensitive by default. Some examples are shown below. Note that you do not use = or != when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead
.
Examples:

- To find names beginning with `b':

```
mysql> SELECT * FROM students WHERE name LIKE "b%";
```

- To find names ending with `fy':

```
mysql> SELECT * FROM students WHERE name LIKE "%fy";
```

- ‘To find names containing a `w':

```
mysql> SELECT * FROM students WHERE name LIKE "%w%";
```

- To find names containing exactly five characters, use the `_' pattern character:

```
mysql> SELECT * FROM students WHERE name LIKE "_____";
```

SQL AGGREGATE FUNCTIONS:

SQL provides specialized functions to perform operations using the data manipulation commands. A function takes one or more arguments and returns a value.

1. Average function **avg( )**: This function returns the average of values of the column specified in the argument.

Example://to find average percentage of students

- mysql> SELECT AVG(percentage) FROM students

2. Min function **min( ):** This function gives the least of all values of the column present in the argument.

Example://to find minimum percentage of students

- mysql> SELECT MIN(percentage) FROM students;

3. Max function **max( )**: This function gives the maximum of a set of values of the column present in the argument.

Example://to find maximum percentage of students

- mysql> SELECT MAX(percentage) FROM students;

4. Sum function **sum( )**: This function is used to obtain the sum of a range of values of a record set.

Example://to find sum of percentage of all students

- mysql> SELECT SUM(percentage) FROM students;

5. Count function **count( )**: This function is used to count the number of non-NULL results.

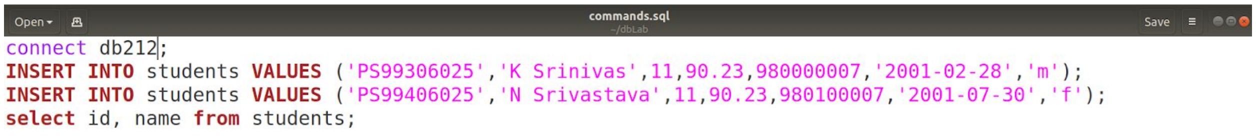
Example://to count total number of students

- mysql> SELECT COUNT(*) FROM students;

Source Command:

Used to run sql script from within the MySql shell

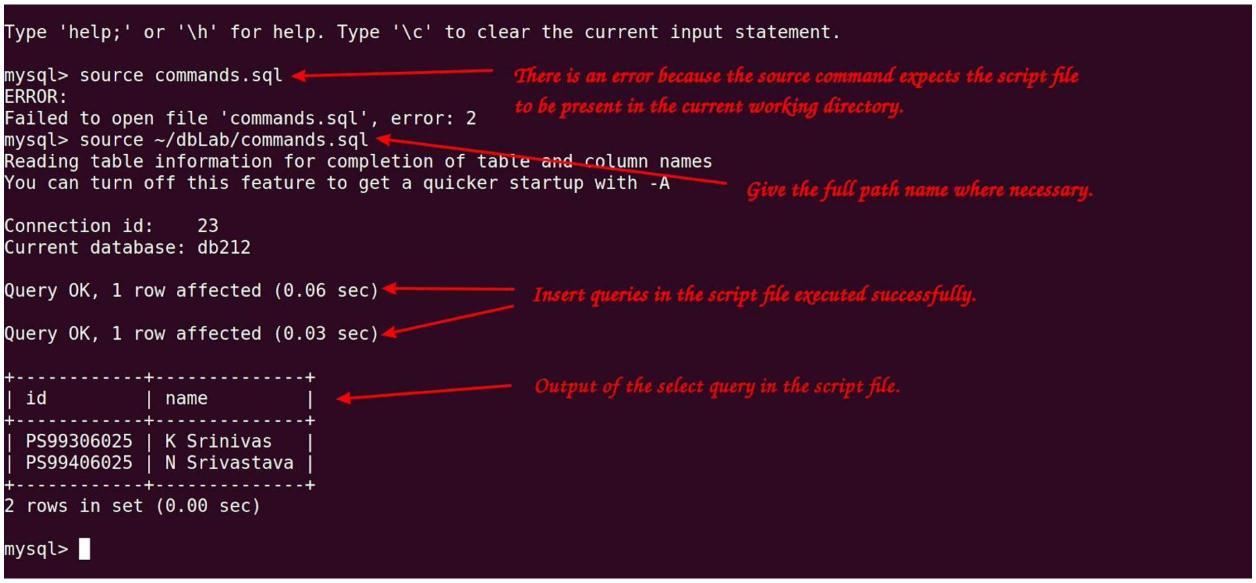
Suppose you have an SQL script as shown below. You can edit the script using any text editor.



```
connect db212;
INSERT INTO students VALUES ('PS99306025','K Srinivas',11,90.23,980000007,'2001-02-28','m');
INSERT INTO students VALUES ('PS99406025','N Srivastava',11,90.23,980100007,'2001-07-30','f');
select id, name from students;
```

The SQL script shown above contains three different SQL commands. The script can be run from within the MySQL shell as shown below.

The screenshot below shows how to run the script from within the MySQL shell.


There is an error because the source command expects the script file to be present in the current working directory.
Give the full path name where necessary.
Insert queries in the script file executed successfully.
Output of the select query in the script file.

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> source commands.sql
ERROR:
Failed to open file 'commands.sql', error: 2
mysql> source ~/dbLab/commands.sql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Connection id:    23
Current database: db212

Query OK, 1 row affected (0.06 sec)

Query OK, 1 row affected (0.03 sec)

+------------+--------------+
| id         | name         |
+------------+--------------+
| PS99306025 | K Srinivas   |
| PS99406025 | N Srivastava |
+------------+--------------+
2 rows in set (0.00 sec)

mysql> 
```

In the next labs, you won't be manually inserting values into the table . We will be providing you with data dump (in sql file) , you can load this info by source command in just a single go without manually copy pasting the single commands in the MySql Shell

For example in lab1 , you were asked to insert the following values into the respective tables

Q5) Insert the following entries into the corresponding tables:

Movies:

1) 125, Good Will Hunting, 1997,126, English, 1998-06-03, UK
2) 126, Back To The Future, 1985, 116, English, 1985-12-04, UK
3) 127, Seven Samurai, 1954, 207, Japanese, 1954-04-26, JP
4) 128, Jurassic Park, 1993, 128, English, 1993-06--09, US
5) 129, Uri: The Surgical Strike, 2019, 138, Hindi, 2019-1-11, IND

Critics:

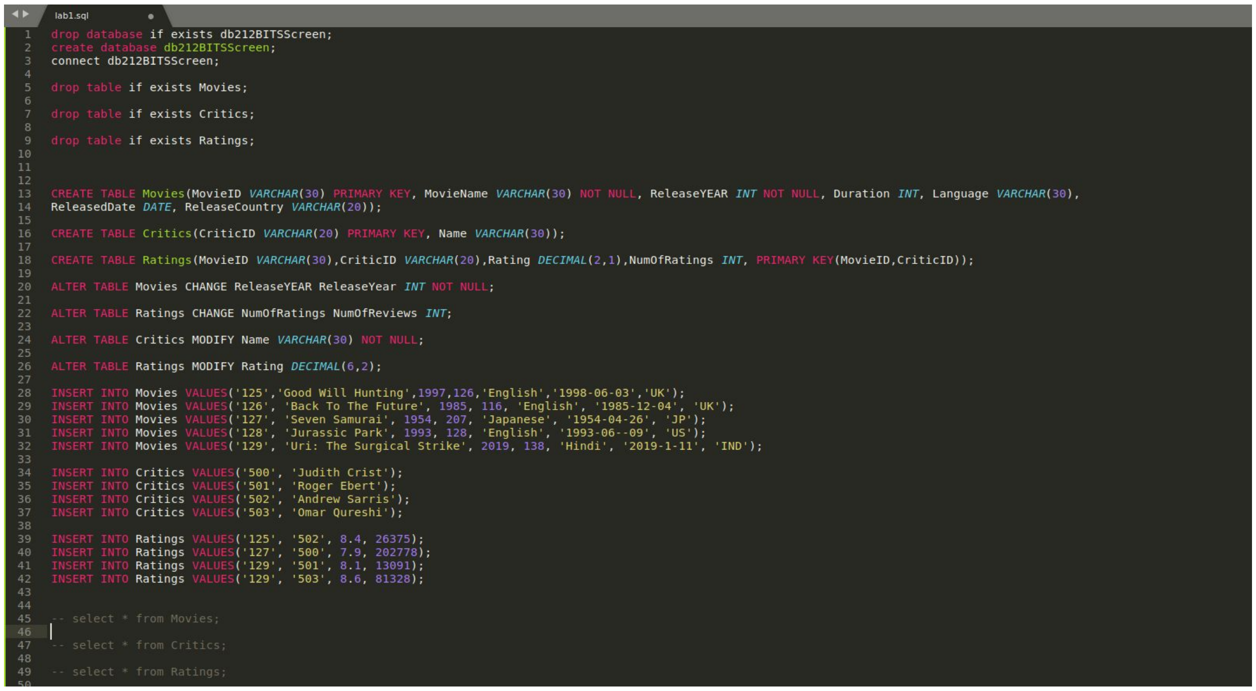
1) 500, Judith Crist
2) 501, Roger Ebert
3) 502, Andrew Sarris
4) 503, Omar Qureshi

Ratings:

1) 125, 502, 8.4, 26375
2) 127, 500, 7.9, 202778
3) 129, 501, 8.1, 13091
4) 129, 503, 8.6, 81328

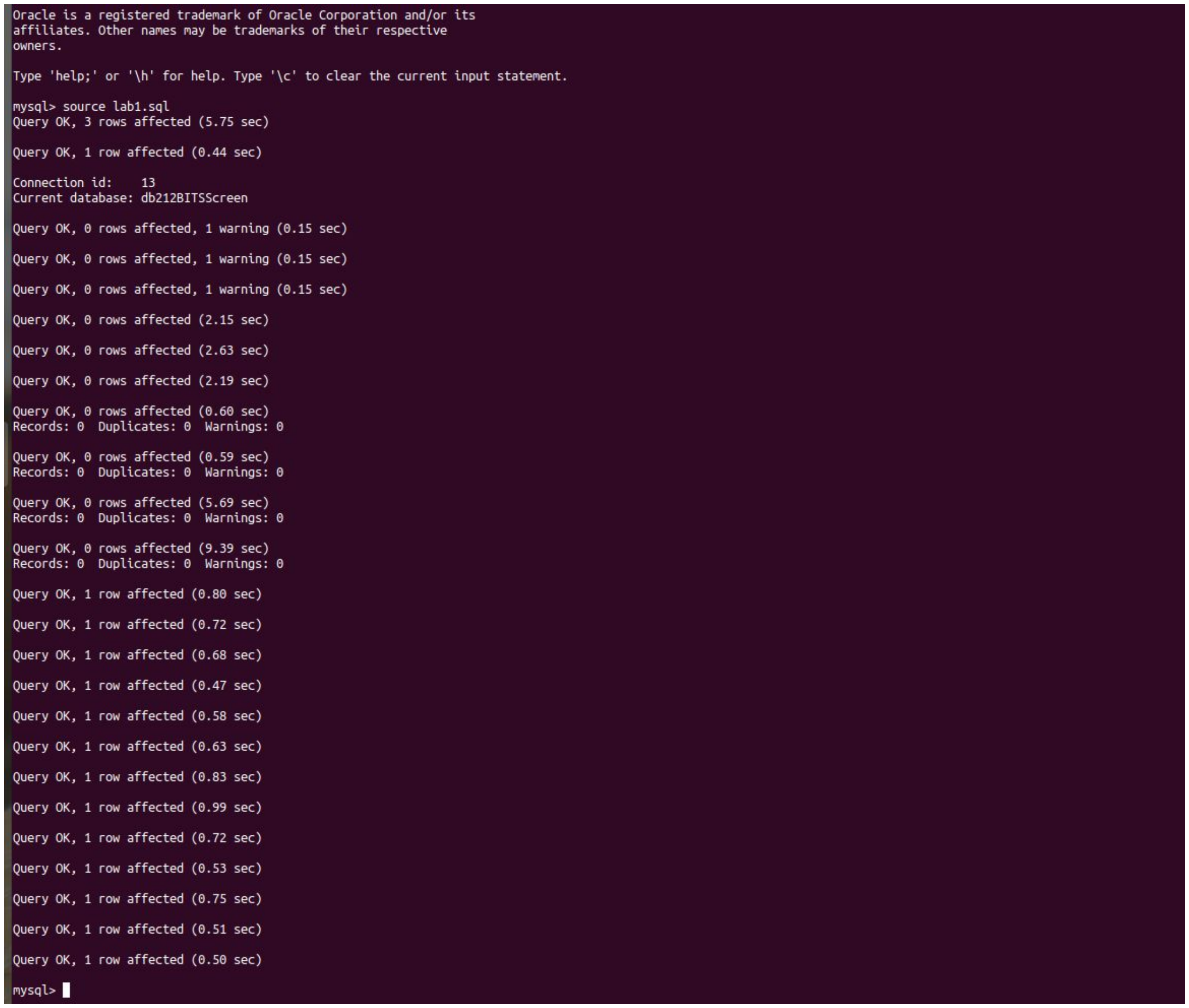
In next labs, we'll provide with you a dump like this, which you can run in mysql shell using source command

Dump file Link



```sql
drop database if exists db212BITSScreen;
create database db212BITSScreen;
connect db212BITSScreen;

drop table if exists Movies;

drop table if exists Critics;

drop table if exists Ratings;



CREATE TABLE Movies(MovieID VARCHAR(30) PRIMARY KEY, MovieName VARCHAR(30) NOT NULL, ReleaseYEAR INT NOT NULL, Duration INT, Language VARCHAR(30),
ReleasedDate DATE, ReleaseCountry VARCHAR(20));

CREATE TABLE Critics(CriticID VARCHAR(20) PRIMARY KEY, Name VARCHAR(30));

CREATE TABLE Ratings(MovieID VARCHAR(30),CriticID VARCHAR(20),Rating DECIMAL(2,1),NumOfRatings INT, PRIMARY KEY(MovieID,CriticID));

ALTER TABLE Movies CHANGE ReleaseYEAR ReleaseYear INT NOT NULL;

ALTER TABLE Ratings CHANGE NumOfRatings NumOfReviews INT;

ALTER TABLE Critics MODIFY Name VARCHAR(30) NOT NULL;

ALTER TABLE Ratings MODIFY Rating DECIMAL(6,2);

INSERT INTO Movies VALUES('125','Good Will Hunting',1997,126,'English','1998-06-03','UK');
INSERT INTO Movies VALUES('126', 'Back To The Future', 1985, 116, 'English', '1985-12-04', 'UK');
INSERT INTO Movies VALUES('127', 'Seven Samurai', 1954, 207, 'Japanese', '1954-04-26', 'JP');
INSERT INTO Movies VALUES('128', 'Jurassic Park', 1993, 128, 'English', '1993-06--09', 'US');
INSERT INTO Movies VALUES('129', 'Uri: The Surgical Strike', 2019, 138, 'Hindi', '2019-1-11', 'IND');

INSERT INTO Critics VALUES('500', 'Judith Crist');
INSERT INTO Critics VALUES('501', 'Roger Ebert');
INSERT INTO Critics VALUES('502', 'Andrew Sarris');
INSERT INTO Critics VALUES('503', 'Omar Qureshi');

INSERT INTO Ratings VALUES('125', '502', 8.4, 26375);
INSERT INTO Ratings VALUES('127', '500', 7.9, 202778);
INSERT INTO Ratings VALUES('129', '501', 8.1, 13091);
INSERT INTO Ratings VALUES('129', '503', 8.6, 81328);


-- select * from Movies;

-- select * from Critics;

-- select * from Ratings;
```

Source command



```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> source lab1.sql
Query OK, 3 rows affected (5.75 sec)

Query OK, 1 row affected (0.44 sec)

Connection id:    13
Current database: db212BITSScreen

Query OK, 0 rows affected, 1 warning (0.15 sec)

Query OK, 0 rows affected, 1 warning (0.15 sec)

Query OK, 0 rows affected, 1 warning (0.15 sec)

Query OK, 0 rows affected (2.15 sec)

Query OK, 0 rows affected (2.63 sec)

Query OK, 0 rows affected (2.19 sec)

Query OK, 0 rows affected (0.60 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.59 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (5.69 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (9.39 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 1 row affected (0.80 sec)

Query OK, 1 row affected (0.72 sec)

Query OK, 1 row affected (0.68 sec)

Query OK, 1 row affected (0.47 sec)

Query OK, 1 row affected (0.58 sec)

Query OK, 1 row affected (0.63 sec)

Query OK, 1 row affected (0.83 sec)

Query OK, 1 row affected (0.99 sec)

Query OK, 1 row affected (0.72 sec)

Query OK, 1 row affected (0.53 sec)

Query OK, 1 row affected (0.75 sec)

Query OK, 1 row affected (0.51 sec)

Query OK, 1 row affected (0.50 sec)

mysql> 
```