SELECT and INSERT

This lesson discusses inserting data into a table using SELECT and INSERT in a single query.

SELECT and INSERT

MySQL provides us the facility to insert several rows from another table into an existing table using a combination of select and insert statements. In fact, we can also create a table on the fly and fill it up with rows from another table.

Syntax to Insert in an Existing Table

```
INSERT INTO table1 (col1, col2)
SELECT col3, col4
FROM table2;
```

Syntax to Insert in a New Table

```
CREATE TABLE newTable (col1 <datatype>, <col2>)
SELECT col3, col4
FROM table2;
```

Connect to the terminal below by clicking in the widget. Once connected, the command line prompt will show up. Enter or copy and paste the command **./DataJek/Lessons/35lesson.sh** and wait for the MySQL prompt to start up.

prompt to start-up.

```
-- The lesson queries are reproduced below for convenient copy/paste into the terminal.

-- Query 1
CREATE TABLE Names (name VARCHAR(20),
                    PRIMARY KEY(name));

-- Query 2
INSERT INTO Names(name)
SELECT SecondName FROM Actors;

-- Query 3
INSERT IGNORE INTO Names(name)
SELECT SecondName
FROM Actors WHERE Id = 1;

-- Query 4
CREATE TABLE MyTempTable SELECT * FROM Actors;

-- Query 5
CREATE TABLE NamesWithDoBs (
Id INT AUTO_INCREMENT,
Name VARCHAR(20) NOT NULL DEFAULT "unknown",
DoB DATE,
PRIMARY KEY(Id), KEY(Name), KEY(DoB))  SELECT FirstName, DoB FROM Actors;

-- Query 6
CREATE TABLE CopyOfActors LIKE Actors;
```

Terminal

1. We can populate data into a table from another table using **INSERT** and **SELECT** in a single query. Say, we want to create a table of all the second names of actors. We'll create a table as follows:

   ```
   CREATE TABLE Names (name VARCHAR(20),
                       PRIMARY KEY(name));
   ```

   Now we can insert using **INSERT** and **SELECT** in a single statement as follows:

   ```
   INSERT INTO Names(name)
   SELECT SecondName FROM Actors;
   ```

```
mysql> CREATE TABLE Names (name VARCHAR(20),
    ->                     PRIMARY KEY(name));
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> INSERT INTO Names(name)
    -> SELECT SecondName FROM Actors;
Query OK, 11 rows affected (0.00 sec)
Records: 11  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Names;
+------------+
| name       |
+------------+
| Aniston    |
| Bachchan   |
| Chopra     |
| Cruise     |
| Depp       |
| Jenner     |
| Jolie      |
| Kardashian |
| Khan       |
| Pitt       |
| Portman    |
+------------+
11 rows in set (0.00 sec)
```

2. Note that in creating the table **Names** we have set the only column as the primary key, however, we aren't guaranteed that the values being selected from the **Actors** table will not contain duplicates. Let's try to insert an existing row into the table:

```
mysql> INSERT INTO Names(name) SELECT SecondName FROM Actors WHERE Id=1;
ERROR 1062 (23000): Duplicate entry 'Pitt' for key 'PRIMARY'
```

As you can see, trying to add a duplicate rows result in an error.

MySQL provides a way to bypass this error and continue execution using the **IGNORE** clause. It doesn't mean a duplicate row is added to the table, rather it means that MySQL issues a warning instead of issuing an error and aborting.

```
INSERT IGNORE INTO Names(name)
SELECT SecondName
FROM Actors WHERE Id = 1;
```

```
mysql> INSERT IGNORE INTO Names(name)
    -> SELECT SecondName
    -> FROM Actors WHERE Id = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)
Records: 1  Duplicates: 1  Warnings: 1

mysql> SELECT * FROM Names;
+-------------+
| name        |
+-------------+
| Aniston     |
| Bachchan    |
| Chopra      |
| Cruise      |
| Depp        |
| Jenner      |
| Jolie       |
| Kardashian  |
| Khan        |
| Pitt        |
| Portman     |
+-------------+
11 rows in set (0.00 sec)
```

Note that the query finishes successfully but informs the user about the duplicate row. Both the duplicate and warning counts read one.

3. In one of the previous examples we created the table first and then inserted data into the table. We can do both tasks in one shot. Consider the following query:

```
CREATE TABLE MyTempTable SELECT * FROM Actors;
```

```
mysql> CREATE TABLE MyTempTable SELECT * FROM Actors;
Query OK, 11 rows affected (0.02 sec)
Records: 11  Duplicates: 0  Warnings: 0

mysql> DESC MyTempTable;
+-------------------+-----------------------------------+------+-----+---------+-------+
| Field             | Type                              | Null | Key | Default | Extra |
+-------------------+-----------------------------------+------+-----+---------+-------+
| Id                | int(11)                           | NO   |     | 0       |       |
| FirstName         | varchar(20)                       | YES  |     | NULL    |       |
| SecondName        | varchar(20)                       | YES  |     | NULL    |       |
| DoB               | date                              | YES  |     | NULL    |       |
| Gender            | enum('Male','Female','Transgender') | YES  |     | NULL    |       |
| MaritalStatus     | enum('Married','Divorced','Single') | YES  |     | NULL    |       |
| NetWorthInMillions | decimal(10,0)                    | YES  |     | NULL    |       |
+-------------------+-----------------------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM MyTempTable;
+----+-----------+------------+------------+--------+---------------+--------------------+
| Id | FirstName | SecondName | DoB        | Gender | MaritalStatus | NetWorthInMillions |
+----+-----------+------------+------------+--------+---------------+--------------------+
|  1 | Brad      | Pitt       | 1963-12-18 | Male   | Single        |                240 |
|  2 | Jennifer  | Aniston    | 1969-11-02 | Female | Single        |                240 |
|  3 | Angelina  | Jolie      | 1975-06-04 | Female | Single        |                100 |
|  4 | Johnny    | Depp       | 1963-06-09 | Male   | Single        |                200 |
|  5 | Natalie   | Portman    | 1981-06-09 | Male   | Married       |                 60 |
|  6 | Tom       | Cruise     | 1962-07-03 | Male   | Divorced      |                570 |
|  7 | Kylie     | Jenner     | 1997-08-10 | Female | Married       |               1000 |
|  8 | Kim       | Kardashian | 1980-10-21 | Female | Married       |                370 |
|  9 | Amitabh   | Bachchan   | 1942-10-11 | Male   | Married       |                400 |
| 10 | Shahrukh  | Khan       | 1965-11-02 | Male   | Married       |                600 |
| 11 | priyanka  | Chopra     | 1982-07-18 | Female | Married       |                 28 |
+----+-----------+------------+------------+--------+---------------+--------------------+
11 rows in set (0.00 sec)
```

As you can see we get a copy of the data with the above query but if you pay attention to the describe statement, you'll notice that the table we create doesn't inherit the primary key constraints. In fact, creating and copying data as above will not create foreign or primary key constraints on the copy table.

4. All the modifiers that can be used in a stand-alone create table statement can also be used in a combined create and populate table statement.

```
CREATE TABLE NamesWithDoBs (
Id INT AUTO_INCREMENT,
Name VARCHAR(20) NOT NULL DEFAULT "unknown",
DoB DATE,
PRIMARY KEY(Id), KEY(Name), KEY(DoB))  SELECT FirstName, DoB FROM
 Actors;
```

```
mysql> CREATE TABLE NamesWithDoBs (
    -> Id INT AUTO_INCREMENT,
    -> Name VARCHAR(20) NOT NULL DEFAULT "unknown",
    -> DoB DATE,
    -> PRIMARY KEY(Id), KEY(Name), KEY(DoB))  SELECT FirstName, DoB FROM Actors;
Query OK, 11 rows affected (0.02 sec)
Records: 11  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM NamesWithDoBs;
+----+---------+-----------+------------+
| Id | Name    | FirstName | DoB        |
+----+---------+-----------+------------+
|  1 | unknown | Brad      | 1963-12-18 |
|  2 | unknown | Jennifer  | 1969-11-02 |
|  3 | unknown | Angelina  | 1975-06-04 |
|  4 | unknown | Johnny    | 1963-06-09 |
|  5 | unknown | Natalie   | 1981-06-09 |
|  6 | unknown | Tom       | 1962-07-03 |
|  7 | unknown | Kylie     | 1997-08-10 |
|  8 | unknown | Kim       | 1980-10-21 |
|  9 | unknown | Amitabh   | 1942-10-11 |
| 10 | unknown | Shahrukh  | 1965-11-02 |
| 11 | unknown | priyanka  | 1982-07-18 |
+----+---------+-----------+------------+
11 rows in set (0.00 sec)
```

5. We can also create a copy of an existing table without the data using the **LIKE** operator. For instance:

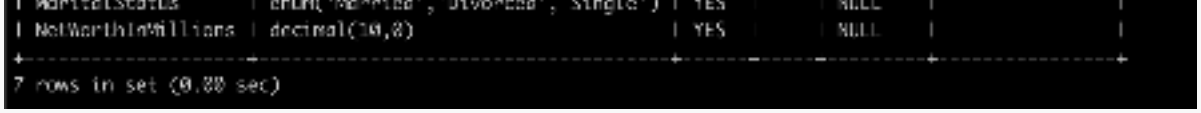
```
CREATE TABLE CopyOfActors LIKE Actors;
```

```
mysql> CREATE TABLE CopyOfActors LIKE Actors;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM CopyOfActors;
Empty set (0.00 sec)
```

As you can see, the copy table doesn't contain any data, but its structure is exactly the same as the source table. The primary keys and any indexes defined on the source table are also defined on the copy table.

```
mysql> DESC CopyOfActors;
+-------------------+-----------------------------------+------+-----+---------+----------------+
| Field             | Type                              | Null | Key | Default | Extra          |
+-------------------+-----------------------------------+------+-----+---------+----------------+
| Id                | int(11)                           | NO   | PRI | NULL    | auto_increment |
| FirstName         | varchar(20)                       | YES  |     | NULL    |                |
| SecondName        | varchar(20)                       | YES  |     | NULL    |                |
| DoB               | date                              | YES  |     | NULL    |                |
| Gender            | enum('Male','Female','Transgender') | YES  |     | NULL    |                |
```


| MaritalStatus | enum('Married','Divorced','Single') | YES | | NULL | |
| NetWorthInMillions | decimal(10,0) | YES | | NULL | |
7 rows in set (0.00 sec)