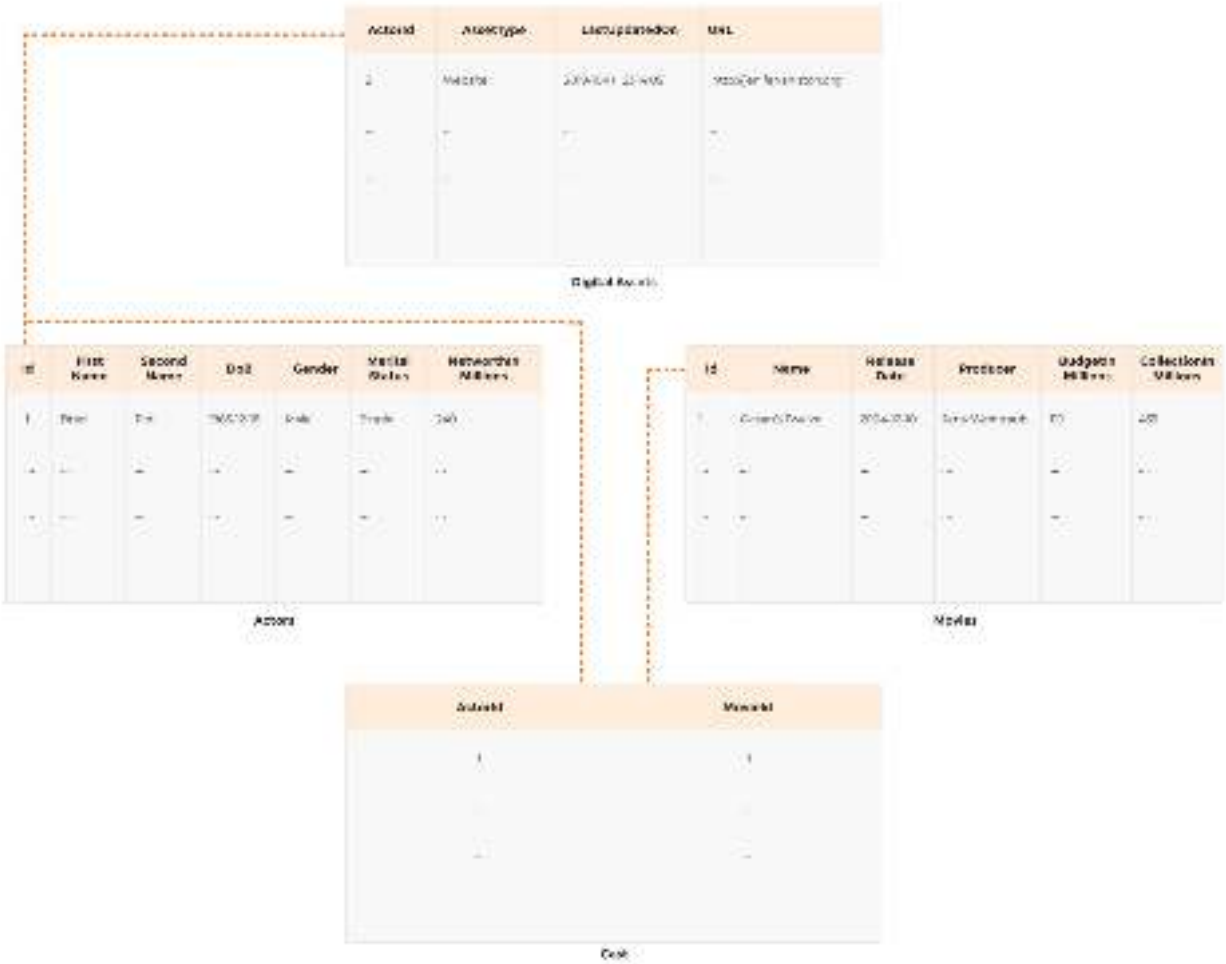
Solution Practice Set 3

Solution Practice Set 3

The database relationship model is reprinted below for reference.



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/quiz.sh** and wait for the MySQL prompt to

start-up.

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

-- Question # 1, Query 1
SELECT AVG(BudgetInMillions)
FROM Movies;

-- Question # 1, Query 2
SELECT Name
FROM Movies
WHERE BudgetInMillions > (SELECT AVG(BudgetInMillions)
                          FROM Movies);

-- Question # 2, Query 1
SELECT * FROM DigitalAssets
RIGHT JOIN Actors
ON Id = ActorId;

-- Question # 2, Query 2
SELECT CONCAT(FirstName, " ", SecondName)
AS Actors_With_No_Online_Presence
FROM DigitalAssets
RIGHT JOIN Actors
ON Id = ActorId
WHERE URL IS NULL;

-- Question # 3, Query 1
SELECT CONCAT(FirstName, " ", SecondName)
FROM Actors
WHERE NOT EXISTS (SELECT ActorId
                  FROM DigitalAssets
                  WHERE ActorId = Id);

-- Question # 4, Query 1
SELECT Name, CollectionInMillions
FROM Movies
ORDER BY CollectionInMillions DESC;

-- Question # 4, Query 2
SELECT Name,
CollectionInMillions AS Collection_In_Millions
FROM Movies
ORDER BY CollectionInMillions DESC
LIMIT 1 OFFSET 4;

-- Question # 4, Query 3
SELECT Name,
CollectionInMillions AS Collection_In_Millions
FROM Movies
ORDER BY CollectionInMillions DESC
LIMIT 4, 1;

-- Question # 5, Query 1
SELECT LastUpdatedOn, Id
FROM Actors
INNER JOIN DigitalAssets
ON ActorId = Id;

-- Question # 5, Query 2
```

```
SELECT *
FROM Cast
INNER JOIN (SELECT LastUpdatedOn, Id

            FROM Actors
            INNER JOIN DigitalAssets
            ON ActorId = Id) AS tbl
ON tbl.Id = ActorId;

-- Question # 5, Query 3
SELECT *
FROM Movies AS m
INNER JOIN (SELECT *
            FROM Cast
            INNER JOIN (SELECT LastUpdatedOn, Id
                        FROM Actors
                        INNER JOIN DigitalAssets
                        ON ActorId = Id) AS tbl1
            ON tbl1.Id = ActorId) AS tbl2
ON tbl2.MovieId = m.Id;

-- Question # 5, Query 4
SELECT DISTINCT Name
AS Actors_Posting_Online_Within_Five_Days_Of_Movie_Release
FROM Movies AS m
INNER JOIN (SELECT *
            FROM Cast
            INNER JOIN (SELECT LastUpdatedOn, Id
                        FROM Actors
                        INNER JOIN DigitalAssets
                        ON ActorId = Id) AS tbl1
            ON tbl1.Id = ActorId) AS tbl2
ON tbl2.MovieId = m.Id
WHERE ADDDATE(ReleaseDate, INTERVAL -5 Day) <= LastUpdatedOn
AND ADDDATE(ReleaseDate, INTERVAL +5 Day) >= LastUpdatedOn;
```

Terminal

Question # 1

Write a query to display all those movie titles whose budget is greater than the average budget of all the movies.

This question also requires flexing MySQL's aggregation capabilities. First we'll write a query to calculate the average budget for all the films as follows:

```
SELECT AVG(BudgetInMillions)
FROM Movies;
```

```
mysql> SELECT AVG(BudgetInMillions)
    -> FROM Movies;
+-------------------------+
| AVG(BudgetInMillions) |
+-------------------------+
|                 97.8000 |
+-------------------------+
1 row in set (0.00 sec)
```

Now, we can plug the above query as a sub-query and list all the movies whose budget was greater than the average budget across all movies.

```
SELECT Name
FROM Movies
WHERE BudgetInMillions > (SELECT AVG(BudgetInMillions)
                          FROM Movies);
```

```
mysql> SELECT Name
    -> FROM Movies
    -> WHERE BudgetInMillions > (SELECT AVG(BudgetInMillions)
    ->                           FROM Movies);
+--------------------------------+
| Name                           |
+--------------------------------+
| Ocean's Twelve                 |
| Mr & Mrs. Smith                |
| Mission: Impossible - Fallout  |
| Keeping Up with the Kardashians |
| Avengers: Endgame              |
+--------------------------------+
5 rows in set (0.00 sec)
```

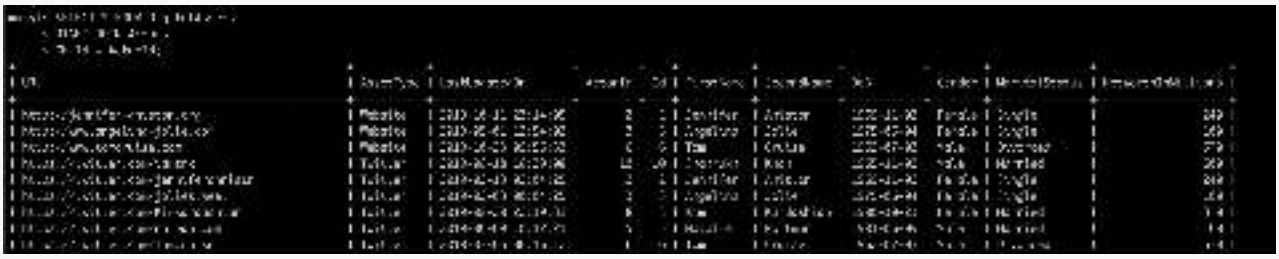
Question # 2

Find all those actors who don't have any digital media presence using a right join statement.

The **Actors** table has the ID column which is the same as the ActorID column of the **DigitalAssets** table. In a right join, the table on the right side of the join has all the rows included which don't satisfy the join criteria. In this case, we want to include all the actors that don't have their ID present in the **DigitalAssets** table, so we need to place the **Actors** table on the right of the **RIGHT JOIN** clause.

```
SELECT * FROM DigitalAssets
RIGHT JOIN Actors
ON Id = ActorId;
```

The output in the below screenshot is truncated for lack of space.



The result set of the above query sets NULL for columns from **DigitalAssets** table for those rows from the **Actors** table that don't have a corresponding entry in the **DigitalAssets** table. We can predicate on the column URL being NULL to identify those actors who don't have a social media presence.

```
SELECT CONCAT(FirstName, " ", SecondName)
AS Actors_With_No_Online_Presence
FROM DigitalAssets
RIGHT JOIN Actors
ON Id = ActorId
WHERE URL IS NULL;
```

```
mysql> SELECT CONCAT(FirstName, " ", SecondName)
    -> AS Actors_With_No_Online_Presence
    -> FROM DigitalAssets
    -> RIGHT JOIN Actors
    -> ON Id = ActorId
    -> WHERE URL IS NULL;
+--------------------------------+
| Actors_With_No_Online_Presence |
+--------------------------------+
| Kylie Jenner                   |
| priyanka Chopra                |
| Khloe Kardashian               |
| Kourtney Kardashian            |
| Abhishek Bachchan              |
| Frank Sinatra                  |
| Fahim Haq                      |
+--------------------------------+
7 rows in set (0.00 sec)
```

Question # 3

Can you rewrite the previous query without a join and using EXISTS operator?

We can grab all the actor IDs that exist in the **DigitalAssets** table and then select those actors from the **Actors** table whose ID doesn't appear in the result set of the subquery.

```
SELECT CONCAT(FirstName, " ", SecondName)
FROM Actors
WHERE NOT EXISTS (SELECT ActorId
                  FROM DigitalAssets
```

```
                WHERE ActorId = Id);
```

```
mysql> SELECT CONCAT(FirstName, " ", SecondName)
    -> FROM Actors
    -> WHERE NOT EXISTS (SELECT ActorId
    ->                   FROM DigitalAssets
    ->                   WHERE ActorId = Id);
+-------------------------------------+
| CONCAT(FirstName, " ", SecondName) |
+-------------------------------------+
| Kylie Jenner                        |
| priyanka Chopra                     |
| Khloe Kardashian                    |
| Kourtney Kardashian                 |
| Abhishek Bachchan                   |
| Frank Sinatra                       |
| Fahim Haq                           |
+-------------------------------------+
7 rows in set (0.00 sec)
```

Question # 4

Write a query to print the name of the fifth highest grossing movie at the box office.

It's trivial to print the list of Movies sorted by how much they made at the box office in descending fashion as follows:

```
SELECT Name, CollectionInMillions
FROM Movies
ORDER BY CollectionInMillions DESC;
```

```
mysql> SELECT Name, CollectionInMillions
    -> FROM Movies
    -> ORDER BY CollectionInMillions DESC;
+----------------------------------+----------------------+
| Name                             | CollectionInMillions |
+----------------------------------+----------------------+
| Avengers: Endgame                |           2798.00000 |
| Mission: Impossible - Fallout    |            791.10000 |
| Mr & Mrs. Smith                  |            478.20000 |
| Ocean's 11                       |            450.70000 |
| Ocean's Twelve                   |            435.00000 |
| Keeping Up with the Kardashians  |            300.00000 |
| Regarding Henry                  |             43.00000 |
| Mohabbatein                      |             12.55000 |
| Ocean's 11                       |              5.00000 |
| London Fields                    |              0.48700 |
+----------------------------------+----------------------+
10 rows in set (0.00 sec)
```

To print the 5th highest grossing movie, we need to leverage the **OFFSET** and **LIMIT** clauses. **OFFSET** allows us to print results starting from a specific row in the sorted result set. We'll want to print the row immediately after the first 4 rows, which will be the fifth row in the sorted result and the 5th highest grossing movie. Finally, we'll use the **LIMIT** clause set to 1 to print only one row. The final query is shown below:

```
SELECT Name,
CollectionInMillions AS Collection_In_Millions
FROM Movies
ORDER BY CollectionInMillions DESC
LIMIT 1 OFFSET 4;
```

```
mysql> SELECT Name,
    -> CollectionInMillions AS Collection_In_Millions
    -> FROM Movies
    -> ORDER BY CollectionInMillions DESC
    -> LIMIT 1 OFFSET 4;
+----------------+-------------------------+
| Name           | Collection_In_Millions |
+----------------+-------------------------+
| Ocean's Twelve |               435.00000 |
+----------------+-------------------------+
1 row in set (0.00 sec)
```

Alternative syntax would be:

```
SELECT Name,
CollectionInMillions AS Collection_In_Millions
FROM Movies
ORDER BY CollectionInMillions DESC
LIMIT 4, 1;
```

```
mysql> SELECT Name,
    -> CollectionInMillions AS Collection_In_Millions
    -> FROM Movies
    -> ORDER BY CollectionInMillions DESC
    -> LIMIT 4, 1;
+----------------+-------------------------+
| Name           | Collection_In_Millions |
+----------------+-------------------------+
| Ocean's Twelve |               435.00000 |
+----------------+-------------------------+
1 row in set (0.00 sec)
```

Question # 5

Find those movies, whose cast latest activity on social media occurred between the span of 5 days before and 5 days after the

release date.

This question is an example of a complex query. We are asked to find the names of those movies whose cast's latest activity on their digital assets was around the same time when the movie was released. We store the last time an actor updated any of his digital accounts in the **DigitalAssets** table whereas the release date of the movie is in the **Movies** table. We'll need to connect all information from the **DigitalAssets** table to the **Movies** using the **Cast** table step by step. Let's see how we can do that:

1. First, we'll retrieve the latest times for all the actors when they made updates to their online accounts along with their IDs.

```
SELECT LastUpdatedOn, Id
FROM Actors
INNER JOIN DigitalAssets
ON ActorId = Id;
```

```
mysql> SELECT LastUpdatedOn, Id
    -> FROM Actors
    -> INNER JOIN DigitalAssets
    -> ON ActorId = Id;
+---------------------+----+
| LastUpdatedOn       | Id |
+---------------------+----+
| 2019-10-11 23:14:05 |  2 |
| 2019-05-01 12:54:02 |  3 |
| 2019-10-23 09:56:33 |  6 |
| 2019-08-18 18:39:08 | 10 |
| 2019-02-13 03:04:25 |  2 |
| 2019-02-03 00:04:25 |  3 |
| 2019-08-28 22:19:33 |  8 |
| 2019-06-09 10:12:21 |  5 |
| 2018-07-05 06:16:12 |  6 |
| 2020-01-01 23:12:13 |  1 |
| 2000-10-30 01:00:54 | 10 |
| 2019-11-11 15:00:00 |  2 |
| 2018-07-11 17:17:18 |  4 |
| 2019-09-04 18:07:38 |  8 |
| 2019-06-09 09:14:20 |  5 |
| 2019-10-28 19:39:40 |  6 |
| 2000-10-24 18:39:08 |  9 |
| 2019-05-15 16:25:02 |  1 |
| 2018-03-15 13:25:00 |  8 |
| 2018-11-24 15:06:59 |  5 |
| 2019-06-04 03:44:36 |  3 |
| 2019-06-09 09:14:20 |  5 |
+---------------------+----+
22 rows in set (0.00 sec)
```

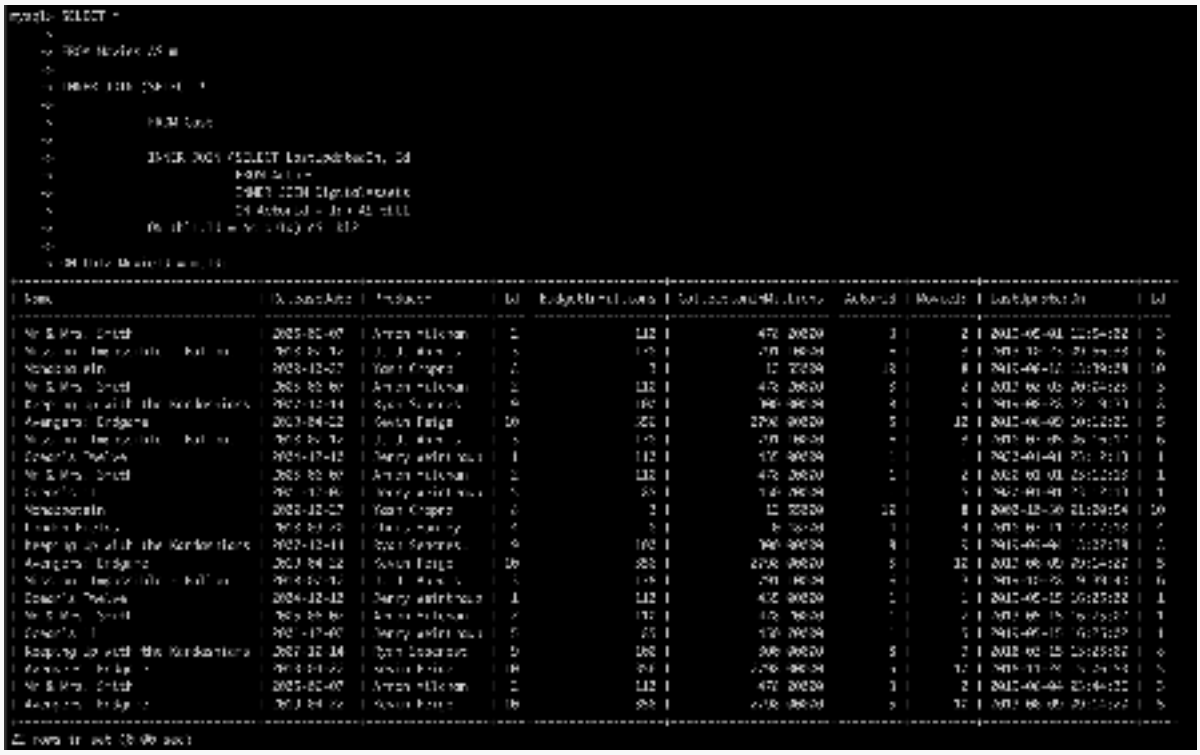
2. Next, we'll join the result of the previous step with the **Cast** table based on the actor ID. The result of this step will be another derived table whose each row will consist of a movie ID, an actor that was part of the movie, and time of their latest online activity.

```
SELECT *

FROM Cast

INNER JOIN (SELECT LastUpdatedOn, Id
            FROM Actors
            INNER JOIN DigitalAssets
            ON ActorId = Id) AS tbl

ON tbl.Id = ActorId;
```

```
mysql> SELECT *
    ->
    -> FROM Cast
    ->
    -> INNER JOIN (SELECT LastUpdatedOn, Id
    ->              FROM Actors
    ->              INNER JOIN DigitalAssets
    ->              ON ActorId = Id) AS tbl
    ->
    -> ON tbl.Id = ActorId;
+---------+---------+---------------------+----+
| ActorId | MovieId | LastUpdatedOn       | Id |
+---------+---------+---------------------+----+
|       3 |       2 | 2019-05-01 12:54:02 |  3 |
|       6 |       3 | 2019-10-23 09:56:33 |  6 |
|      10 |       8 | 2019-08-18 18:39:08 | 10 |
|       3 |       2 | 2019-02-03 00:04:25 |  3 |
|       8 |       9 | 2019-08-28 22:19:33 |  8 |
|       5 |      10 | 2019-06-09 10:12:21 |  5 |
|       6 |       3 | 2018-07-05 06:16:12 |  6 |
|       1 |       1 | 2020-01-01 23:12:13 |  1 |
|       1 |       2 | 2020-01-01 23:12:13 |  1 |
|       1 |       5 | 2020-01-01 23:12:13 |  1 |
|      10 |       8 | 2000-10-30 01:00:54 | 10 |
|       4 |       4 | 2018-07-11 17:17:18 |  4 |
|       8 |       9 | 2019-09-04 18:07:38 |  8 |
|       5 |      10 | 2019-06-09 09:14:20 |  5 |
|       6 |       3 | 2019-10-28 19:39:40 |  6 |
|       1 |       1 | 2019-05-15 16:25:02 |  1 |
|       1 |       2 | 2019-05-15 16:25:02 |  1 |
|       1 |       5 | 2019-05-15 16:25:02 |  1 |
|       8 |       9 | 2018-03-15 13:25:00 |  8 |
|       5 |      10 | 2018-11-24 15:06:59 |  5 |
|       3 |       2 | 2019-06-04 03:44:36 |  3 |
|       5 |      10 | 2019-06-09 09:14:20 |  5 |
+---------+---------+---------------------+----+
22 rows in set (0.00 sec)
```

3. In the third step we can join the derived table of the second step with the **Movies** table based on the movie ID. The derived table resulting from this joining will have the movie name, movie release date, an actor participating in that movie, and the latest time of that actor's online activity. By now we have all the necessary columns we need to compare.

```
SELECT *

FROM Movies AS m

INNER JOIN (SELECT *

            FROM Cast

            INNER JOIN (SELECT LastUpdatedOn, Id
                        FROM Actors
                        INNER JOIN DigitalAssets
                        ON ActorId = Id) AS tbl1
            ON tbl1.Id = ActorId) AS tbl2

ON tbl2.MovieId = m.Id;
```



4. In the fourth step we'll add a **WHERE** clause and set the LastUpdatedColumn to be between the plus/minus 5 days from the

date of the movie release. Also, remember that two actors cast in the same movie could have posted about their upcoming movie, but we want to print the name of the movie only once, therefore, we also add the **DISTINCT** clause.

```
SELECT DISTINCT Name

AS Actors_Posting_Online_Within_Five_Days_Of_Movie_Release

FROM Movies AS m

INNER JOIN (SELECT *

            FROM Cast

            INNER JOIN (SELECT LastUpdatedOn, Id
                        FROM Actors
                        INNER JOIN DigitalAssets
                        ON ActorId = Id) AS tbl1
            ON tbl1.Id = ActorId) AS tbl2

ON tbl2.MovieId = m.Id

WHERE ADDDATE(ReleaseDate, INTERVAL -5 Day) <= LastUpdatedOn

AND ADDDATE(ReleaseDate, INTERVAL +5 Day) >= LastUpdatedOn;
```

```
mysql> SELECT DISTINCT Name
    ->
    -> AS Actors_Posting_Online_Within_Five_Days_Of_Movie_Release
    ->
    -> FROM Movies AS m
    ->
    -> INNER JOIN (SELECT *
    ->
    ->              FROM Cast
    ->
    ->              INNER JOIN (SELECT LastUpdatedOn, Id
    ->                          FROM Actors
    ->                          INNER JOIN DigitalAssets
    ->                          ON ActorId = Id) AS tbl1
    ->              ON tbl1.Id = ActorId) AS tbl2
    ->
    -> ON tbl2.MovieId = m.Id
    ->
    -> WHERE ADDDATE(ReleaseDate, INTERVAL -5 Day) <= LastUpdatedOn
    ->
    -> AND ADDDATE(ReleaseDate, INTERVAL +5 Day) >= LastUpdatedOn;
+-----------------------------------------------------------+
| Actors_Posting_Online_Within_Five_Days_Of_Movie_Release |
+-----------------------------------------------------------+
| Mohabbatein                                               |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```