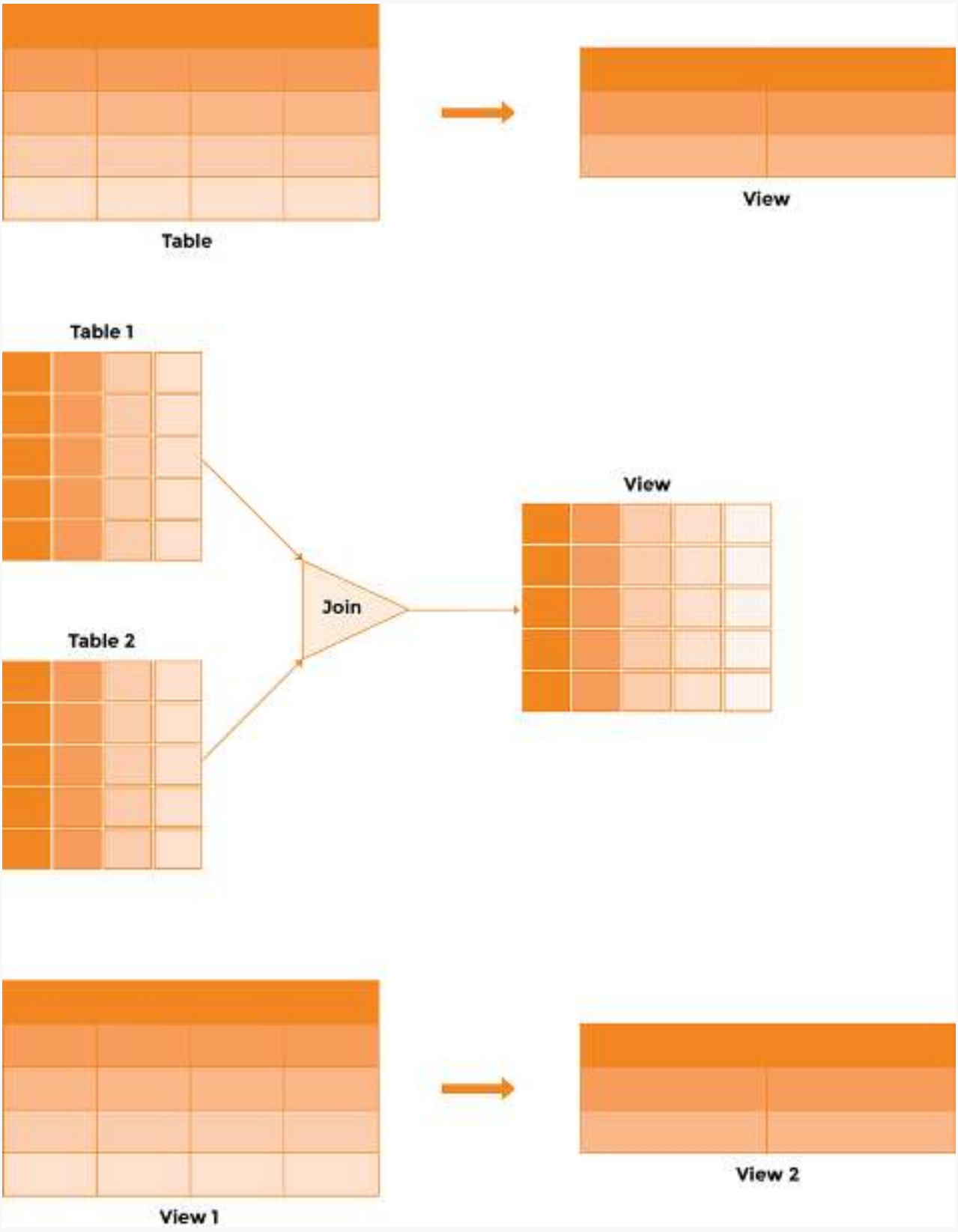
Creating a View

In this lesson we will learn about three different ways of creating views.

Creating a View

Views are virtual tables that are created as a result of a SELECT query. They offer a number of advantages such as showing only a subset of data that is meaningful to users or restricting the number of rows and columns shown for security reasons. A view containing columns from multiple tables can simplify queries by changing a multi-table query to a single-table query against a view. Views are stored in the database along with tables.

A view can be created from a single table, by joining two tables, or from another view.



Syntax

> CREATE [OR REPLACE] VIEW **view_name** AS
>
> SELECT **col1**, **col2**, ...**coln**
>
> FROM **table**

> WHERE < **condition**>

> 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/41lesson.sh** and wait for the mysql prompt to start-up.

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

-- Query 1
CREATE VIEW DigitalAssetCount AS
SELECT ActorId, COUNT(AssetType) AS NumberOfAssets
FROM DigitalAssets
GROUP BY ActorId;

-- Query 2
SELECT * FROM DigitalAssetCount;

-- Query 3
CREATE VIEW ActorsTwitterAccounts AS
SELECT FirstName, SecondName, URL
FROM Actors
INNER JOIN DigitalAssets
ON Actors.Id = DigitalAssets.ActorID
WHERE AssetType = 'Twitter';

-- Query 4
CREATE OR REPLACE VIEW ActorsTwitterAccounts AS
SELECT CONCAT(FirstName, ' ', SecondName) AS ActorName, URL
FROM Actors
INNER JOIN DigitalAssets
ON Actors.Id = DigitalAssets.ActorID
WHERE AssetType = 'Twitter';

-- Query 5
CREATE VIEW RichActors AS
SELECT FirstName, SecondName, Gender, NetWorthInMillions
FROM Actors
WHERE NetWorthInMillions > (
SELECT AVG(NetWorthInMillions)
FROM Actors)
ORDER BY NetWorthInMillions DESC;

-- Query 6
CREATE VIEW RichFemaleActors AS
SELECT * FROM RichActors
WHERE Gender = 'Female';

-- Query 7
CREATE VIEW ActorDetails (ActorName, Age, MaritalStatus, NetWorthInMillions) AS
SELECT CONCAT(FirstName,' ',SecondName) AS ActorName,
       TIMESTAMPDIFF(YEAR, DoB, CURDATE()) AS Age,
       MaritalStatus, NetWorthInMillions
FROM Actors;
```

```
FROM Actors;

-- Query 8
SELECT ActorName, Age, NetWorthInMillions
    FROM ActorDetails
    ORDER BY Age DESC;
```

Terminal

1. A view can be created from a single table. The SELECT query specifies the columns in the view. Use the following query to create a **DigitalAssetCount** view from the DigitalAssets table.

```
CREATE VIEW DigitalAssetCount AS
SELECT ActorId, COUNT(AssetType) AS NumberOfAssets
FROM DigitalAssets
GROUP BY ActorId;
```

```
mysql> CREATE VIEW DigitalAssetCount AS
    -> SELECT ActorId, COUNT(AssetType) AS NumberOfAssets
    -> FROM DigitalAssets
    -> GROUP BY ActorId;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

This view shows the number of digital assets owned by an actor. A view can be queried in the same manner as a table. Run the following query on the view we just created:

```
SELECT * FROM DigitalAssetCount;
```

```
mysql> SELECT * FROM DigitalAssetCount;
+---------+----------------+
| ActorId | NumberOfAssets |
+---------+----------------+
|       1 |              2 |
|       2 |              3 |
|       3 |              3 |
|       4 |              1 |
|       5 |              4 |
|       6 |              3 |
|       8 |              3 |
|      10 |              2 |
+---------+----------------+
8 rows in set (0.00 sec)
```

Views are stored as virtual tables and also appear in the list of tables when SHOW TABLES is executed.

```
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_MovieIndustry |
+---------------------------+
| Actors                    |
| DigitalAssetCount         |
| DigitalAssets             |
+---------------------------+
3 rows in set (0.00 sec)
```

MovieIndustry database has two tables and the DigitalAssetCount view is shown along with them. To find out which entities in the above image are tables and which are views, the SHOW FULL TABLES command is used. The Table_Type column in the result specifies whether the object is a view or a table as shown below:

```
mysql> SHOW FULL TABLES;
+---------------------------+------------+
| Tables_in_MovieIndustry | Table_type |
+---------------------------+------------+
| Actors                    | BASE TABLE |
| DigitalAssetCount         | VIEW       |
| DigitalAssets             | BASE TABLE |
+---------------------------+------------+
3 rows in set (0.00 sec)
```

2. A view can be created from multiple tables using JOIN. Let's suppose we want to create a view of Actors who have Twitter accounts. This can be done by joining the Actors and DigitalAssets tables as follows:

```
CREATE VIEW ActorsTwitterAccounts AS
SELECT FirstName, SecondName, URL
FROM Actors
INNER JOIN DigitalAssets
ON Actors.Id = DigitalAssets.ActorID
WHERE AssetType = 'Twitter';
```

```
mysql>
mysql> mysql> CREATE VIEW ActorsTwitterAccounts AS
    -> SELECT FirstName, SecondName, URL
    -> FROM Actors
    -> INNER JOIN DigitalAssets
    -> ON Actors.Id = DigitalAssets.ActorID
    -> WHERE AssetType = 'Twitter';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM ActorsTwitterAccounts;
+-----------+------------+-----------------------------------+
| FirstName | SecondName | URL                               |
+-----------+------------+-----------------------------------+
| Shahrukh  | Khan       | https://twitter.com/iamsrk        |
| Jennifer  | Aniston    | https://twitter.com/jenniferannistn |
| Angelina  | Jolie      | https://twitter.com/joliestweet   |
| Kim       | Kardashian | https://twitter.com/KimKardashian |
| Natalie   | Portman    | https://twitter.com/natpdotcom    |
| Tom       | Cruise     | https://twitter.com/TomCruise     |
+-----------+------------+-----------------------------------+
6 rows in set (0.00 sec)
```

We can use the **[OR REPLACE]** clause to make changes to a view that

we just created. If the view does not exist, the OR REPLACE clause has no effect.

```
CREATE OR REPLACE VIEW ActorsTwitterAccounts AS
SELECT CONCAT(FirstName, ' ', SecondName) AS ActorName, URL
FROM Actors
INNER JOIN DigitalAssets
ON Actors.Id = DigitalAssets.ActorID
WHERE AssetType = 'Twitter';
```

```
mysql>
mysql> CREATE OR REPLACE VIEW ActorsTwitterAccounts AS
    -> SELECT CONCAT(FirstName, ' ', SecondName) AS ActorName, URL
    -> FROM Actors
    -> INNER JOIN DigitalAssets
    -> ON Actors.Id = DigitalAssets.ActorID
    -> WHERE AssetType = 'Twitter';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM ActorsTwitterAccounts;
+-----------------+-------------------------------------+
| ActorName       | URL                                 |
+-----------------+-------------------------------------+
| Shahrukh Khan   | https://twitter.com/iamsrk          |
| Jennifer Aniston | https://twitter.com/jenniferannistn |
| Angelina Jolie  | https://twitter.com/joliestweet     |
| Kim Kardashian  | https://twitter.com/KimKardashian   |
| Natalie Portman | https://twitter.com/natpdotcom      |
| Tom Cruise      | https://twitter.com/TomCruise       |
+-----------------+-------------------------------------+
6 rows in set (0.00 sec)
```

Here we have modified the view created above to show the first and last names in one column instead of two separate columns.

3. The SELECT statement that creates a view can also have a nested subquery. If we want to create a table of those actors whose net worth is more than the average net worth, we can do so using a nested subquery as follows:

```
CREATE VIEW RichActors AS
SELECT FirstName, SecondName, Gender, NetWorthInMillions
FROM Actors
WHERE NetWorthInMillions > (
SELECT AVG(NetWorthInMillions)
FROM Actors)
ORDER BY NetWorthInMillions DESC;
```

```
mysql> CREATE VIEW RichActors AS
    -> SELECT FirstName, SecondName, Gender, NetWorthInMillions
    -> FROM Actors
    -> WHERE NetWorthInMillions > (
    -> SELECT AVG(NetWorthInMillions)
    -> FROM Actors)
    -> ORDER BY NetWorthInMillions DESC;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM RichActors;
+-----------+------------+--------+--------------------+
| FirstName | SecondName | Gender | NetWorthInMillions |
+-----------+------------+--------+--------------------+
| Kylie     | Jenner     | Female |               1000 |
| Shahrukh  | Khan       | Male   |                600 |
| Tom       | Cruise     | Male   |                570 |
| Amitabh   | Bachchan   | Male   |                400 |
| Kim       | Kardashian | Female |                370 |
+-----------+------------+--------+--------------------+
5 rows in set (0.01 sec)
```

The view shows five rich actors whose net worth is more than the average of all actors in our database.

4. A view can also be created from another view. Let's suppose we want to create a view, **RichFemaleActors**, based on the RichActors view we created in the last step.

```
CREATE VIEW RichFemaleActors AS
SELECT * FROM RichActors
WHERE Gender = 'Female';
```

```
mysql> CREATE VIEW RichFemaleActors AS
    -> SELECT * FROM RichActors
    -> WHERE Gender = 'Female';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM RichFemaleActors;
+-----------+------------+--------+--------------------+
| FirstName | SecondName | Gender | NetWorthInMillions |
+-----------+------------+--------+--------------------+
| Kylie     | Jenner     | Female |               1000 |
| Kim       | Kardashian | Female |                370 |
+-----------+------------+--------+--------------------+
2 rows in set (0.00 sec)
```

5. In the previous examples, the SELECT statement specifies the columns of the view. We can explicitly define columns in a view by

listing them in parentheses after the view name. Run the following

query to create the **ActorDetails** view where we define a column, Age, that is based on the DoB column in the Actors table:

```
CREATE VIEW ActorDetails (ActorName, Age, MaritalStatus, NetWorth
InMillions) AS
SELECT CONCAT(FirstName,' ',SecondName) AS ActorName,
       TIMESTAMPDIFF(YEAR, DoB, CURDATE()) AS Age,
       MaritalStatus, NetWorthInMillions
FROM Actors;
```

The following query lists the actors from the view created above according to age:

```
SELECT ActorName, Age, NetWorthInMillions
    FROM ActorDetails
    ORDER BY Age DESC;
```

```
    -> FROM Actors;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT ActorName, Age, NetWorthInMillions
    ->     FROM ActorDetails
    ->     ORDER BY Age DESC;
+------------------+------+--------------------+
| ActorName        | Age  | NetWorthInMillions |
+------------------+------+--------------------+
| Amitabh Bachchan |   77 |                400 |
| Tom Cruise       |   57 |                570 |
| Brad Pitt        |   56 |                240 |
| Johnny Depp      |   56 |                200 |
| Shahrukh Khan    |   54 |                600 |
| Jennifer Aniston |   50 |                240 |
| Angelina Jolie   |   44 |                100 |
| Kim Kardashian   |   39 |                370 |
| Natalie Portman  |   38 |                 60 |
| priyanka Chopra  |   37 |                 28 |
| Kylie Jenner     |   22 |               1000 |
+------------------+------+--------------------+
11 rows in set (0.00 sec)
```