

# JS\_ST\_Assignment\_07

December 1, 2015

## 0.1 Part 1

This notebook assumes that the database created in Assignment 5 exists and is accessible to the person running this notebook. If the database has not been created, please run that notebook first so that you would be able to run this notebook.

Based on the analyses completed in Assignment 5, we knew that it would be important for an analyst to be able to easily access ratings based on user dimension data and movie genre data. However, the structure of the transactional database required complex joins, especially to access movie genre data. To simplify queries, we decided to convert the data to a dimensional model.

The main business process under observation is that users provide ratings for movies, which fall within various genres. Therefore, we created a fact table called Ratings whose grain is a movie rating by user and genre. User does not refer to a single person, because no unique user id is provided. Instead, it refers to a category to which a user belongs. We are told the user's age range, gender, occupation, and zip code. The intersection of these categories is captured by the user\_dimension table, in which each unique combination of values receives a user\_dim\_id. This was the structure in the transactional database created in Assignment 5. The dimensional model utilizes the same structure, but with the addition of a surrogate key called user\_key.

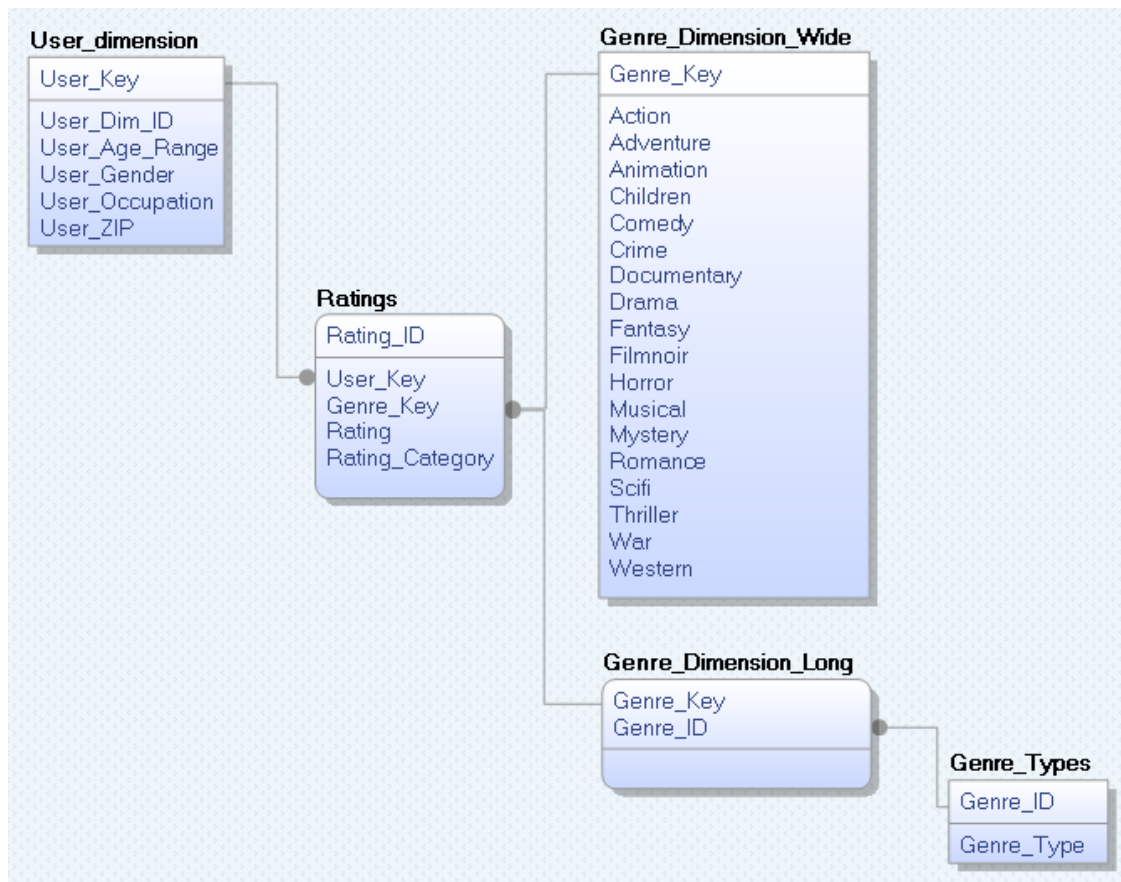
In the relational database, the genre data was normalized so that multiple genre row entries appeared for each rating id. For example, if rating id 1 was associated with a movie in 3 genres (Action, Adventure, and Children) then rating id 1 will have 3 rows in the genre table, one for each genre. Certain types of analyses benefit from this structure. A drawback is that joining to that type of table causes duplication of ratings table rows. A count or an average using such a structure would not be accurate. In order to meet such analytical requirements, we created another type of genre dimension table as well. In this structure, we defined all genre types and then listed the different combinations in a bridge table (genre\_dimension\_long). This bridge table contains the same genre keys for the groups that we had before in the genre dimension table (genre\_dimension\_wide).

The ETL process required extracting data from the database to csv files and reading the csv files into the data warehouse. The following path indicates the location in which the csv files will be placed. Please make sure that MySQL has permission to write files into this folder, otherwise this process would fail.

```
In [129]: path = '/home/vagrant/DWH Notebooks/'
```

```
In [130]: fn = '%sJS_ST_Assignment07_ERD.PNG'% path
          from IPython.display import Image
          Image(filename=fn)
```

```
Out[130]:
```



```
In [131]: %load_ext sql
```

The sql extension is already loaded. To reload it, use:  

```
%reload_ext sql
```

```
In [132]: !echo "DROP DATABASE IF EXISTS movielensdwh" | mysql --user=mysqluser --password=mysqlpass
```

```
In [133]: !echo "CREATE DATABASE movielensdwh" | mysql --user=mysqluser --password=mysqlpass
```

```
In [134]: %sql mysql://mysqluser:mysqlpass@localhost/movielensdwh
```

```
Out[134]: u'Connected: mysqluser@movielensdwh'
```

In order to move the data from our transactional database to our data warehouse, we defined two cursors. cursor\_db connects to the database movielensdb created in Assignment 5. cursor\_dwh connects to the data warehouse.

```
In [135]: import MySQLdb as myDB
conn_db = myDB.connect('localhost', 'mysqluser', 'mysqlpass', 'movielensdb')
cursor_db = conn_db.cursor()
```

```
In [136]: conn_dwh = myDB.connect('localhost', 'mysqluser', 'mysqlpass', 'movielensdwh')
cursor_dwh = conn_dwh.cursor()
sql = 'SET AUTOCOMMIT = 1'
cursor_dwh.execute(sql)
```

Out[136]: OL

Creating the tables as per the star schema for the dimensional model.

```
In [137]: %%sql
CREATE TABLE genre_dimension_wide (
    genre_key int NOT NULL AUTO_INCREMENT,
    Action VARCHAR(10),
    Adventure VARCHAR(10),
    Animation VARCHAR(10),
    Children VARCHAR(10),
    Comedy VARCHAR(10),
    Crime VARCHAR(10),
    Documentary VARCHAR(10),
    Drama VARCHAR(10),
    Fantasy VARCHAR(10),
    Filmnoir VARCHAR(10),
    Horror VARCHAR(10),
    Musical VARCHAR(10),
    Mystery VARCHAR(10),
    Romance VARCHAR(10),
    Scifi VARCHAR(10),
    Thriller VARCHAR(10),
    War VARCHAR(10),
    Western VARCHAR(10),
    PRIMARY KEY(genre_key)
)
```

0 rows affected.

Out[137]: []

```
In [138]: %%sql
CREATE TABLE user_dimension (
    user_key int NOT NULL AUTO_INCREMENT,
    user_dim_id int,
    user_age_range VARCHAR(10),
    user_gender VARCHAR(6),
    user_occupation VARCHAR(30),
    user_ZIP VARCHAR(20),
    PRIMARY KEY(user_key)
)
```

0 rows affected.

Out[138]: []

```
In [139]: %%sql
CREATE TABLE ratings (
    rating_id int,
    user_key int,
    genre_key int,
    rating int,
    PRIMARY KEY(rating_id),
    FOREIGN KEY (user_key) REFERENCES user_dimension(user_key),
    FOREIGN KEY (genre_key) REFERENCES genre_dimension_wide(genre_key)
)
```

0 rows affected.

Out[139]: []

## 0.2 Part 2

We exported the data from each of the three tables of the transactional database into csv files. We then loaded the data from the csv files into staging tables. These tables were then used for the transformation process (if needed) before the data was loaded into the final tables.

```
In [140]: sql = '''SELECT * FROM user_dimension INTO OUTFILE '%suser_data.csv'
            FIELDS TERMINATED BY ','
            OPTIONALLY ENCLOSED BY '"'
            ESCAPED BY '\\\\'
            LINES TERMINATED BY '\\n' ;''' % path
            cursor_db.execute(sql)
```

Out[140]: 5796L

```
In [141]: %%sql
CREATE TABLE user_staging (
    user_dim_id int,
    user_age_range VARCHAR(10),
    user_gender VARCHAR(6),
    user_occupation VARCHAR(30),
    user_ZIP VARCHAR(20)
)
```

0 rows affected.

Out[141]: []

```
In [142]: sql = '''LOAD DATA INFILE '%suser_data.csv' INTO TABLE user_staging
            FIELDS TERMINATED BY ','
            OPTIONALLY ENCLOSED BY '"'
            ESCAPED BY '\\\\'
            LINES TERMINATED BY '\\n' ;''' % path
            cursor_dwh.execute(sql)
```

Out[142]: 5796L

No transformations were necessary for the user\_dimension table. The data was inserted into the final user\_dimension table and user\_key was generated.

```
In [143]: sql = '''INSERT INTO user_dimension (user_dim_id, user_age_range, user_gender, user_occupation,
            SELECT user_dim_id, user_age_range, user_gender, user_occupation, user_ZIP
            FROM user_staging'''
            cursor_dwh.execute(sql)
```

Out[143]: 5796L

```
In [144]: sql = 'SELECT * FROM user_dimension LIMIT 10'
            cursor_dwh.execute(sql)
            cursor_dwh.fetchall()
```

```
Out[144]: ((1L, 712L, '18-24', 'FEMALE', 'academic/educator', '01562'),
(2L, 735L, '18-24', 'FEMALE', 'academic/educator', '02109'),
(3L, 704L, '18-24', 'FEMALE', 'academic/educator', '02144'),
(4L, 700L, '18-24', 'FEMALE', 'academic/educator', '04240'),
(5L, 734L, '18-24', 'FEMALE', 'academic/educator', '07666'),
(6L, 706L, '18-24', 'FEMALE', 'academic/educator', '11375'),
(7L, 733L, '18-24', 'FEMALE', 'academic/educator', '20003'),
(8L, 719L, '18-24', 'FEMALE', 'academic/educator', '33143'),
(9L, 724L, '18-24', 'FEMALE', 'academic/educator', '45458'),
(10L, 729L, '18-24', 'FEMALE', 'academic/educator', '48067'))
```

```
In [145]: sql = '''SELECT rating_id, user_dim_id, rating FROM ratings INTO OUTFILE '%ratings_data.csv'
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
ESCAPED BY '\\\\'
LINES TERMINATED BY '\\n' ;''' % path
cursor_db.execute(sql)
```

```
Out[145]: 1000209L
```

```
In [146]: sql = '''
CREATE TABLE ratings_staging (
    rating_id int,
    user_dim_id int,
    rating int
)
'''
cursor_dwh.execute(sql)
```

```
Out[146]: 0L
```

We created an index on the user\_dim\_id. This is because the final ratings table includes a user\_key which would require a join with the user\_dimension table. An index would help in loading the data faster.

```
In [147]: sql = '''
CREATE INDEX idx_user_dim on ratings_staging(user_dim_id)
'''
cursor_dwh.execute(sql)
```

```
Out[147]: 0L
```

```
In [148]: sql = '''
EXPLAIN SELECT * FROM ratings_staging WHERE user_dim_id = 900
'''
cursor_dwh.execute(sql)
cursor_dwh.fetchall()
```

```
Out[148]: ((1L,
'SIMPLE',
'ratings_staging',
'ref',
'idx_user_dim',
'idx_user_dim',
'5',
'const',
1L,
'Using where'),)
```

```
In [149]: sql = '''LOAD DATA INFILE '%ratings_data.csv' INTO TABLE ratings_staging
            FIELDS TERMINATED BY ','
            OPTIONALLY ENCLOSED BY '"'
            ESCAPED BY '\\\\'
            LINES TERMINATED BY '\\n' ;''' % path
            cursor_dwh.execute(sql)
```

```
Out[149]: 1000209L
```

```
In [150]: sql = '''
            INSERT INTO ratings (rating_id, user_key, rating)
            SELECT rt.rating_id, u.user_key, rt.rating
            FROM ratings_staging rt
            JOIN user_dimension u on rt.user_dim_id=u.user_dim_id
            '''
            cursor_dwh.execute(sql)
```

```
Out[150]: 1000209L
```

The staging table created below is a temporary table from which we will select the distinct combinations of genres to be loaded into genre.dimension.wide.

```
In [151]: %%sql
CREATE TABLE staging (
    rating_id int,
    Action VARCHAR(10) DEFAULT 'FALSE',
    Adventure VARCHAR(10) DEFAULT 'FALSE',
    Animation VARCHAR(10) DEFAULT 'FALSE',
    Children VARCHAR(10) DEFAULT 'FALSE',
    Comedy VARCHAR(10) DEFAULT 'FALSE',
    Crime VARCHAR(10) DEFAULT 'FALSE',
    Documentary VARCHAR(10) DEFAULT 'FALSE',
    Drama VARCHAR(10) DEFAULT 'FALSE',
    Fantasy VARCHAR(10) DEFAULT 'FALSE',
    Filmnoir VARCHAR(10) DEFAULT 'FALSE',
    Horror VARCHAR(10) DEFAULT 'FALSE',
    Musical VARCHAR(10) DEFAULT 'FALSE',
    Mystery VARCHAR(10) DEFAULT 'FALSE',
    Romance VARCHAR(10) DEFAULT 'FALSE',
    Scifi VARCHAR(10) DEFAULT 'FALSE',
    Thriller VARCHAR(10) DEFAULT 'FALSE',
    War VARCHAR(10) DEFAULT 'FALSE',
    Western VARCHAR(10) DEFAULT 'FALSE',
    PRIMARY KEY(rating_id)
)
```

```
0 rows affected.
```

```
Out[151]: []
```

```
In [152]: sql = '''INSERT INTO staging (rating_id)
            SELECT distinct rating_id
            FROM ratings
            '''
            cursor_dwh.execute(sql)
```

Out[152]: 1000209L

```
In [153]: sql = '''
SELECT * FROM genre INTO OUTFILE '%sgenre_data.csv'
        FIELDS TERMINATED BY ','
        OPTIONALLY ENCLOSED BY '"'
        ESCAPED BY '\\\\'
        LINES TERMINATED BY '\\n' ;
''' % path
cursor_db.execute(sql)
```

Out[153]: 2101815L

```
In [154]: sql = '''
CREATE TABLE genre_staging (
rating_id int,
genre varchar(20))'''
cursor_dwh.execute(sql)
```

Out[154]: 0L

```
In [155]: sql = '''
LOAD DATA INFILE '%sgenre_data.csv'
INTO TABLE genre_staging
        FIELDS TERMINATED BY ','
        OPTIONALLY ENCLOSED BY '"'
        ESCAPED BY '\\\\'
        LINES TERMINATED BY '\\n' ;''' % path
cursor_dwh.execute(sql)
```

Out[155]: 2101815L

The staging table has rows inserted into it for each row in the ratings table. The default values for all the genre columns was set to FALSE. We need to update the genre columns to reflect which genres are TRUE for each rating id. We do this by updating the table column by column.

```
In [156]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Action = "TRUE"
WHERE gt.genre = "Action"'''
cursor_dwh.execute(sql)
```

Out[156]: 257457L

```
In [157]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Adventure = "TRUE"
WHERE gt.genre = "Adventure"'''
cursor_dwh.execute(sql)
```

Out[157]: 133953L

```
In [158]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Animation = "TRUE"
WHERE gt.genre = "Animation"'''
cursor_dwh.execute(sql)
```

Out[158]: 43293L

```
In [159]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Children = "TRUE"
WHERE gt.genre = "Children"'''
cursor_dwh.execute(sql)
```

Out[159]: 72186L

```
In [160]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Comedy = "TRUE"
WHERE gt.genre = "Comedy"'''
cursor_dwh.execute(sql)
```

Out[160]: 356580L

```
In [161]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Crime = "TRUE"
WHERE gt.genre = "Crime"'''
cursor_dwh.execute(sql)
```

Out[161]: 79541L

```
In [162]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Documentary = "TRUE"
WHERE gt.genre = "Documentary"'''
cursor_dwh.execute(sql)
```

Out[162]: 7910L

```
In [163]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Drama = "TRUE"
WHERE gt.genre = "Drama"'''
cursor_dwh.execute(sql)
```

Out[163]: 354529L

```
In [164]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Fantasy = "TRUE"
WHERE gt.genre = "Fantasy"'''
cursor_dwh.execute(sql)
```

Out[164]: 36301L



```

In [165]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Filmnoir = "TRUE"
WHERE gt.genre = "Film-Noir"'''
cursor_dwh.execute(sql)

Out[165]: 18261L

In [166]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Horror = "TRUE"
WHERE gt.genre = "Horror"'''
cursor_dwh.execute(sql)

Out[166]: 76386L

In [167]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Musical = "TRUE"
WHERE gt.genre = "Musical"'''
cursor_dwh.execute(sql)

Out[167]: 41533L

In [168]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Mystery = "TRUE"
WHERE gt.genre = "Mystery"'''
cursor_dwh.execute(sql)

Out[168]: 40178L

In [169]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Romance = "TRUE"
WHERE gt.genre = "Romance"'''
cursor_dwh.execute(sql)

Out[169]: 147523L

In [170]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Scifi = "TRUE"
WHERE gt.genre = "Sci-Fi"'''
cursor_dwh.execute(sql)

Out[170]: 157294L

In [171]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Thriller = "TRUE"
WHERE gt.genre = "Thriller"'''
cursor_dwh.execute(sql)

```

Out[171]: 189680L

```
In [172]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET War = "TRUE"
WHERE gt.genre = "War"'''
cursor_dwh.execute(sql)
```

Out[172]: 68527L

```
In [173]: sql = '''
UPDATE staging AS s
INNER JOIN genre_staging gt on s.rating_id=gt.rating_id
SET Western = "TRUE"
WHERE gt.genre = "Western"'''
cursor_dwh.execute(sql)
```

Out[173]: 20683L

We select the distinct combinations of genres from the staging table and add them to genre\_dimension\_wide. This generates a unique genre\_key for each combination.

```
In [174]: sql = '''
INSERT INTO genre_dimension_wide (Action, Adventure, Animation, Children, Comedy,
    Crime, Documentary, Drama, Fantasy, Filmnoir, Horror, Musical, Mystery,
    Romance, Scifi, Thriller, War, Western)
SELECT DISTINCT Action, Adventure, Animation, Children, Comedy,
    Crime, Documentary, Drama, Fantasy, Filmnoir, Horror, Musical, Mystery,
    Romance, Scifi, Thriller, War, Western
FROM staging'''
cursor_dwh.execute(sql)
```

Out[174]: 301L

Once the genre\_dimension\_wide table has been created, we update the ratings table to include the genre\_key.

```
In [175]: sql = '''
UPDATE ratings as r
JOIN
    (SELECT genre_key, rating_id
    FROM staging s
    JOIN genre_dimension_wide g on s.Action=g.Action and s.Adventure=g.Adventure and s.Animation=g.Animation
    and s.Comedy=g.Comedy and s.Crime=g.Crime and s.Documentary=g.Documentary and s.Drama=g.Drama
    and s.Filmnoir=g.Filmnoir and s.Horror=g.Horror and s.Musical=g.Musical and s.Mystery=g.Mystery
    and s.Scifi=g.Scifi and s.Thriller=g.Thriller and s.War=g.War and s.Western=g.Western) a
on r.rating_id=a.rating_id
SET r.genre_key = a.genre_key
'''
cursor_dwh.execute(sql)
```

Out[175]: 1000209L

As explained before, certain types of analyses are better suited to a long genre dimension table which would have the genre types as values rather than column names. Genre represents a multi-valued dimension

attribute. In such cases, it is recommended to create a bridge and an outrigger table. We created a genre\_types table which contains the 18 genre types and a unique id. The genre\_dimension\_long table functions as a bridge. It contains each unique combination of genres as a list, and each combination has a unique genre\_key. This key is the same as the key generated earlier for the genre\_dimension\_wide table.

```
In [176]: sql = '''
CREATE TABLE genre_types (
genre_id int NOT NULL AUTO_INCREMENT,
genre_type varchar(25),
PRIMARY KEY(genre_id));
'''
cursor_dwh.execute(sql)
```

Out[176]: 0L

```
In [177]: sql = '''
INSERT INTO genre_types (genre_type)
SELECT * FROM
(SELECT distinct genre
FROM genre_staging
ORDER BY genre) A;'''
cursor_dwh.execute(sql)
```

Out[177]: 18L

```
In [178]: cursor_dwh.execute('SELECT * FROM genre_types')
cursor_dwh.fetchall()
```

```
Out[178]: ((1L, 'Action'),
(2L, 'Adventure'),
(3L, 'Animation'),
(4L, 'Children'),
(5L, 'Comedy'),
(6L, 'Crime'),
(7L, 'Documentary'),
(8L, 'Drama'),
(9L, 'Fantasy'),
(10L, 'Film-Noir'),
(11L, 'Horror'),
(12L, 'Musical'),
(13L, 'Mystery'),
(14L, 'Romance'),
(15L, 'Sci-Fi'),
(16L, 'Thriller'),
(17L, 'War'),
(18L, 'Western'))
```

```
In [179]: sql = '''
CREATE TABLE genre_dimension_long (
genre_key int,
genre_id int,
PRIMARY KEY (genre_key, genre_id))'''
cursor_dwh.execute(sql)
```

Out[179]: 0L

```
In [180]: sql = '''
INSERT INTO genre_dimension_long
SELECT distinct r.genre_key, gt.genre_id
FROM genre_staging gs
JOIN ratings r on gs.rating_id = r.rating_id
JOIN genre_types gt on gs.genre = gt.genre_type
'''

cursor_dwh.execute(sql)
```

```
Out[180]: 882L
```

Once all the loading was complete, we ran a few queries to ensure the accuracy of the loaded data.

```
In [181]: %%sql
SELECT * FROM ratings LIMIT 10
```

10 rows affected.

```
Out[181]: [(1L, 5774L, 1L, 3L),
(2L, 5774L, 2L, 4L),
(3L, 5774L, 3L, 1L),
(4L, 5774L, 3L, 3L),
(5L, 5774L, 4L, 3L),
(6L, 5774L, 5L, 4L),
(7L, 5774L, 6L, 4L),
(8L, 5774L, 7L, 5L),
(9L, 5774L, 3L, 5L),
(10L, 5774L, 8L, 4L)]
```

Check that the number of distinct user\_keys in the ratings table matches the number of distinct user\_keys in the user\_dimension table.

```
In [182]: %%sql
SELECT count(distinct user_key) FROM ratings;
```

1 rows affected.

```
Out[182]: [(5796L,)]
```

```
In [183]: %%sql
SELECT count(*) FROM user_dimension;
```

1 rows affected.

```
Out[183]: [(5796L,)]
```

We ran some tests on ratings table row with rating\_id 10 and checked if its associated data was properly inserted in all the tables.

```
In [184]: %%sql
SELECT * FROM ratings WHERE rating_id=10;
```

1 rows affected.

```
Out[184]: [(10L, 5774L, 8L, 4L)]
```

Ensure that the user\_key was accurately loaded.



5 rows affected.

```
Out[191]: [(3L,), (4L,), (1L,), (5L,), (2L,)]
```

```
In [192]: %%sql
          ALTER TABLE ratings
          ADD rating_category VARCHAR(10);
```

1000209 rows affected.

```
Out[192]: []
```

```
In [193]: %%sql
          SELECT * FROM ratings LIMIT 10;
```

10 rows affected.

```
Out[193]: [(1L, 5774L, 1L, 3L, None),
           (2L, 5774L, 2L, 4L, None),
           (3L, 5774L, 3L, 1L, None),
           (4L, 5774L, 3L, 3L, None),
           (5L, 5774L, 4L, 3L, None),
           (6L, 5774L, 5L, 4L, None),
           (7L, 5774L, 6L, 4L, None),
           (8L, 5774L, 7L, 5L, None),
           (9L, 5774L, 3L, 5L, None),
           (10L, 5774L, 8L, 4L, None)]
```

```
In [194]: %%sql
          UPDATE ratings
          SET rating_category = CASE WHEN rating in (1,2) THEN 'LOW'
          WHEN rating in (3) THEN 'MEDIUM'
          ELSE 'HIGH' END;
```

1000209 rows affected.

```
Out[194]: []
```

```
In [195]: %%sql
          SELECT distinct rating, rating_category FROM ratings
          order by rating;
```

5 rows affected.

```
Out[195]: [(1L, 'LOW'), (2L, 'LOW'), (3L, 'MEDIUM'), (4L, 'HIGH'), (5L, 'HIGH')]
```

### 0.3 Part 3

We did some analyses of the new data warehouse.

```
In [196]: %matplotlib inline
```

To start, we wanted a general summary of the average rating and the number of ratings associated with each genre type. We observed that Film-Noir has the highest average rating. However, it does not have very many ratings.

```
In [197]: %%sql
SELECT gt.genre_type, avg(r.rating) as avg_rating, count(*) as num_ratings
FROM genre_dimension_long gl
JOIN genre_types gt on gl.genre_id = gt.genre_id
JOIN ratings r on r.genre_key = gl.genre_key
GROUP BY gt.genre_type
ORDER BY avg_rating DESC
```

18 rows affected.

```
Out[197]: [('Film-Noir', Decimal('4.0752'), 18261L),
('Documentary', Decimal('3.9331'), 7910L),
('War', Decimal('3.8933'), 68527L),
('Drama', Decimal('3.7663'), 354529L),
('Crime', Decimal('3.7087'), 79541L),
('Animation', Decimal('3.6849'), 43293L),
('Mystery', Decimal('3.6681'), 40178L),
('Musical', Decimal('3.6655'), 41533L),
('Western', Decimal('3.6378'), 20683L),
('Romance', Decimal('3.6075'), 147523L),
('Thriller', Decimal('3.5705'), 189680L),
('Comedy', Decimal('3.5221'), 356580L),
('Action', Decimal('3.4912'), 257457L),
('Adventure', Decimal('3.4773'), 133953L),
('Sci-Fi', Decimal('3.4665'), 157294L),
('Fantasy', Decimal('3.4474'), 36301L),
('Children', Decimal('3.4220'), 72186L),
('Horror', Decimal('3.2150'), 76386L)]
```

Next we compared the total number of ratings for each genre. The genre with the highest number of ratings is Comedy.

```
In [198]: %%sql
SELECT gt.genre_type, count(*) as num_ratings
FROM genre_dimension_long gl
JOIN genre_types gt on gl.genre_id = gt.genre_id
JOIN ratings r on r.genre_key = gl.genre_key
GROUP BY gt.genre_type
ORDER BY num_ratings desc
```

18 rows affected.

```
Out[198]: [('Comedy', 356580L),
('Drama', 354529L),
('Action', 257457L),
('Thriller', 189680L),
('Sci-Fi', 157294L),
('Romance', 147523L),
('Adventure', 133953L),
('Crime', 79541L),
('Horror', 76386L),
('Children', 72186L),
('War', 68527L),
('Animation', 43293L),
('Musical', 41533L),
```

```

('Mystery', 40178L),
('Fantasy', 36301L),
('Western', 20683L),
('Film-Noir', 18261L),
('Documentary', 7910L)]

```

```

In [199]: result = _
          result.pie()

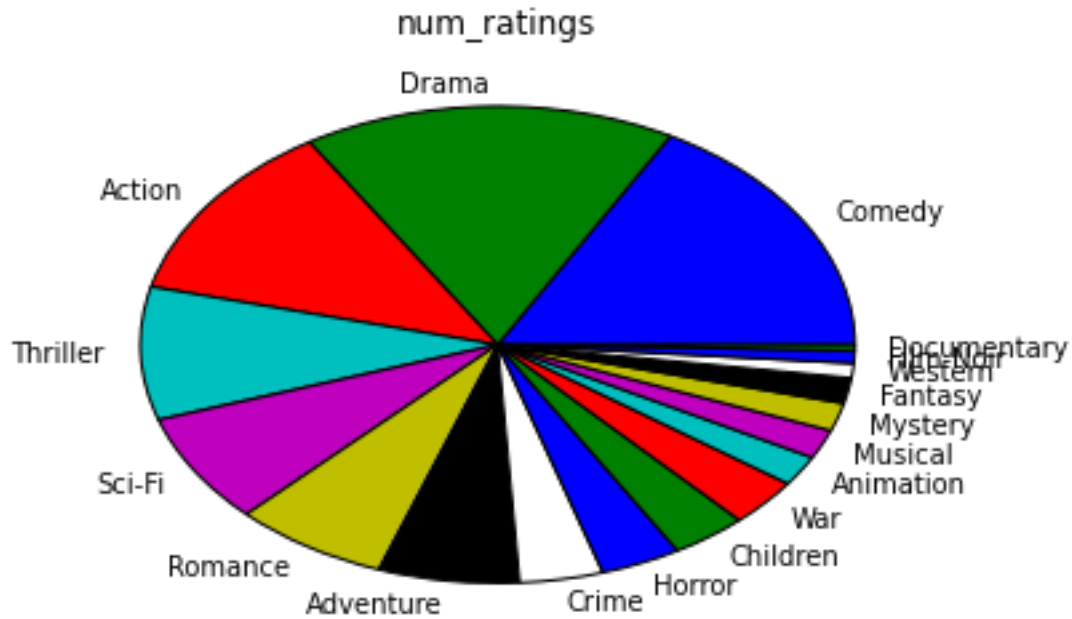
```

```

Out[199]: ([<matplotlib.patches.Wedge at 0x7ff3fc349f50>,
             <matplotlib.patches.Wedge at 0x7ff3fc2fd950>,
             <matplotlib.patches.Wedge at 0x7ff3fc2e7310>,
             <matplotlib.patches.Wedge at 0x7ff3fc2e7c90>,
             <matplotlib.patches.Wedge at 0x7ff3fc2fc650>,
             <matplotlib.patches.Wedge at 0x7ff3fc2fcfd0>,
             <matplotlib.patches.Wedge at 0x7ff3fc311990>,
             <matplotlib.patches.Wedge at 0x7ff3fc2a8350>,
             <matplotlib.patches.Wedge at 0x7ff3fc2a8cd0>,
             <matplotlib.patches.Wedge at 0x7ff3fc2b5690>,
             <matplotlib.patches.Wedge at 0x7ff3fc2c1050>,
             <matplotlib.patches.Wedge at 0x7ff3fc2c19d0>,
             <matplotlib.patches.Wedge at 0x7ff3fc2cf390>,
             <matplotlib.patches.Wedge at 0x7ff3fc2cfd10>,
             <matplotlib.patches.Wedge at 0x7ff3fc2db6d0>,
             <matplotlib.patches.Wedge at 0x7ff3fc268090>,
             <matplotlib.patches.Wedge at 0x7ff3fc268a10>,
             <matplotlib.patches.Wedge at 0x7ff3fc2743d0>],
            [<matplotlib.text.Text at 0x7ff3fc2fd550>,
             <matplotlib.text.Text at 0x7ff3fc2fd10>,
             <matplotlib.text.Text at 0x7ff3fc2e78d0>,
             <matplotlib.text.Text at 0x7ff3fc2fc290>,
             <matplotlib.text.Text at 0x7ff3fc2fcc10>,
             <matplotlib.text.Text at 0x7ff3fc3115d0>,
             <matplotlib.text.Text at 0x7ff3fc311f50>,
             <matplotlib.text.Text at 0x7ff3fc2a8910>,
             <matplotlib.text.Text at 0x7ff3fc2b52d0>,
             <matplotlib.text.Text at 0x7ff3fc2b5c50>,
             <matplotlib.text.Text at 0x7ff3fc2c1610>,
             <matplotlib.text.Text at 0x7ff3fc2c1f90>,
             <matplotlib.text.Text at 0x7ff3fc2cf950>,
             <matplotlib.text.Text at 0x7ff3fc2db310>,
             <matplotlib.text.Text at 0x7ff3fc2dbc90>,
             <matplotlib.text.Text at 0x7ff3fc268650>,
             <matplotlib.text.Text at 0x7ff3fc268fd0>,
             <matplotlib.text.Text at 0x7ff3fc274990>])

```





We wanted to know the general patterns in the ratings for each genre, i.e., how often did users rate movies High, Medium or Low. We found that in general, most people tend to rate movies as High regardless of the genre. This might indicate that people are more likely to rate movies that they like than to rate movies that they don't like.

```
In [200]: %%sql
SELECT genre_type,
       total_ratings,
       low_ratings/total_ratings as percent_low,
       medium_ratings/total_ratings as percent_medium,
       high_ratings/total_ratings as percent_high
FROM(
SELECT genre_type, COUNT(distinct r.rating_id) as total_ratings
      ,SUM(CASE WHEN rating_category = 'Low' then 1 else 0 end) as low_ratings
      ,SUM(CASE WHEN rating_category = 'Medium' then 1 else 0 end) as medium_ratings
      ,SUM(CASE WHEN rating_category = 'High' then 1 else 0 end) as high_ratings
FROM genre_dimension_long gl
JOIN genre_types gt on gl.genre_id = gt.genre_id
JOIN ratings r on r.genre_key = gl.genre_key
GROUP BY genre_type) A
```

18 rows affected.

```
Out[200]: [('Action', 257457L, Decimal('0.1863'), Decimal('0.2747'), Decimal('0.5390')),
           ('Adventure', 133953L, Decimal('0.1876'), Decimal('0.2827'), Decimal('0.5297')),
           ('Animation', 43293L, Decimal('0.1313'), Decimal('0.2535'), Decimal('0.6152')),
           ('Children', 72186L, Decimal('0.2010'), Decimal('0.2825'), Decimal('0.5164')),
           ('Comedy', 356580L, Decimal('0.1758'), Decimal('0.2719'), Decimal('0.5523')),
           ('Crime', 79541L, Decimal('0.1339'), Decimal('0.2457'), Decimal('0.6204')),
           ('Documentary', 7910L, Decimal('0.0946'), Decimal('0.1828'), Decimal('0.7226')),
           ('Drama', 354529L, Decimal('0.1164'), Decimal('0.2392'), Decimal('0.6443')),
```

```
( 'Fantasy', 36301L, Decimal('0.1955'), Decimal('0.2890'), Decimal('0.5155')),
( 'Film-Noir', 18261L, Decimal('0.0600'), Decimal('0.1766'), Decimal('0.7635')),
( 'Horror', 76386L, Decimal('0.2736'), Decimal('0.2802'), Decimal('0.4463')),
( 'Musical', 41533L, Decimal('0.1414'), Decimal('0.2540'), Decimal('0.6046')),
( 'Mystery', 40178L, Decimal('0.1434'), Decimal('0.2518'), Decimal('0.6048')),
( 'Romance', 147523L, Decimal('0.1464'), Decimal('0.2740'), Decimal('0.5796')),
( 'Sci-Fi', 157294L, Decimal('0.1984'), Decimal('0.2727'), Decimal('0.5289')),
( 'Thriller', 189680L, Decimal('0.1652'), Decimal('0.2643'), Decimal('0.5705')),
( 'War', 68527L, Decimal('0.1065'), Decimal('0.1997'), Decimal('0.6938')),
( 'Western', 20683L, Decimal('0.1458'), Decimal('0.2590'), Decimal('0.5952'))]
```

We wanted to see if any particular genre had especially high ratings. We also looked at the differences between MALE and FEMALE viewers. Both men and women tend to rate Film-Noir and Documentary movies as High. Horror movies are unlikely to be rated High by both genders. Women are more likely than men to rate Musicals high.

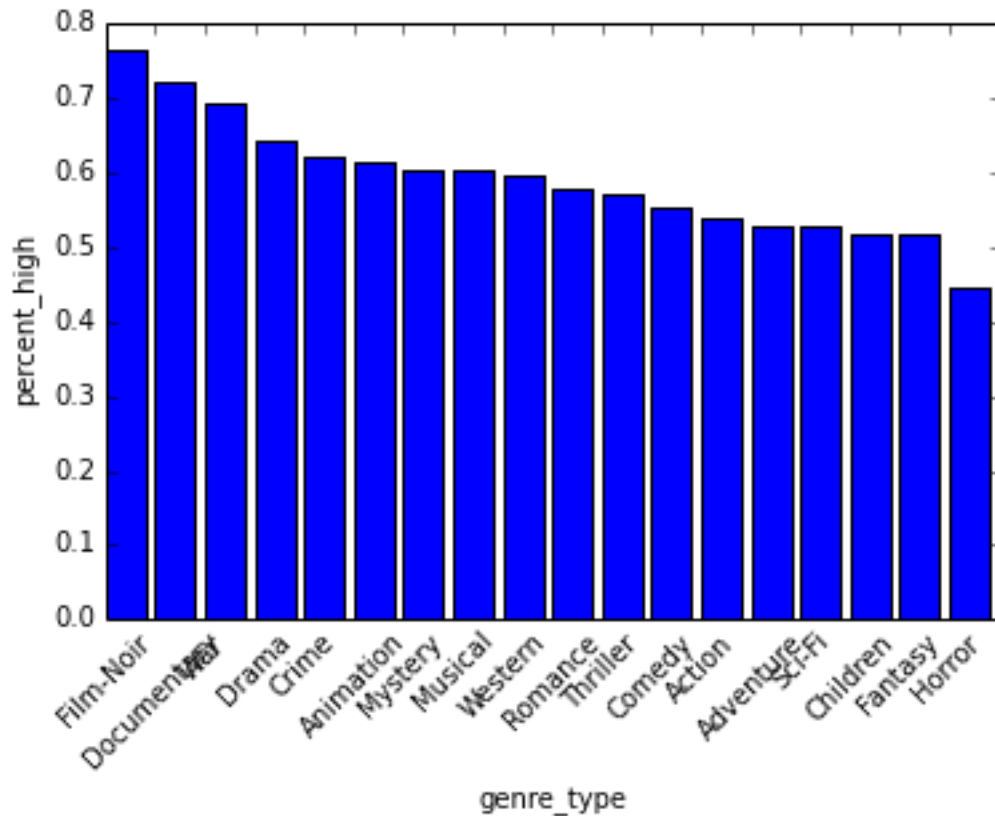
```
In [201]: %%sql
SELECT genre_type,
        high_ratings/total_ratings as percent_high
FROM(
SELECT genre_type, COUNT(distinct r.rating_id) as total_ratings
      ,SUM(CASE WHEN rating_category = 'High' then 1 else 0 end) as high_ratings
FROM genre_dimension_long gl
JOIN genre_types gt on gl.genre_id = gt.genre_id
JOIN ratings r on r.genre_key = gl.genre_key
GROUP BY genre_type) A
ORDER BY percent_high desc
```

18 rows affected.

```
Out[201]: [('Film-Noir', Decimal('0.7635')),
('Documentary', Decimal('0.7226')),
('War', Decimal('0.6938')),
('Drama', Decimal('0.6443')),
('Crime', Decimal('0.6204')),
('Animation', Decimal('0.6152')),
('Mystery', Decimal('0.6048')),
('Musical', Decimal('0.6046')),
('Western', Decimal('0.5952')),
('Romance', Decimal('0.5796')),
('Thriller', Decimal('0.5705')),
('Comedy', Decimal('0.5523')),
('Action', Decimal('0.5390')),
('Adventure', Decimal('0.5297')),
('Sci-Fi', Decimal('0.5289')),
('Children', Decimal('0.5164')),
('Fantasy', Decimal('0.5155')),
('Horror', Decimal('0.4463'))]
```

```
In [202]: result = _
result.bar()
```

```
Out[202]: <Container object of 18 artists>
```



```
In [203]: %%sql
SELECT genre_type,
        high_ratings/total_ratings as percent_high
FROM(
SELECT genre_type, COUNT(distinct r.rating_id) as total_ratings
      ,SUM(CASE WHEN rating_category = 'High' then 1 else 0 end) as high_ratings
FROM genre_dimension_long gl
JOIN genre_types gt on gl.genre_id = gt.genre_id
JOIN ratings r on r.genre_key = gl.genre_key
JOIN user_dimension u on u.user_key = r.user_key
where u.user_gender = 'MALE'
GROUP BY genre_type) A
ORDER BY percent_high desc
```

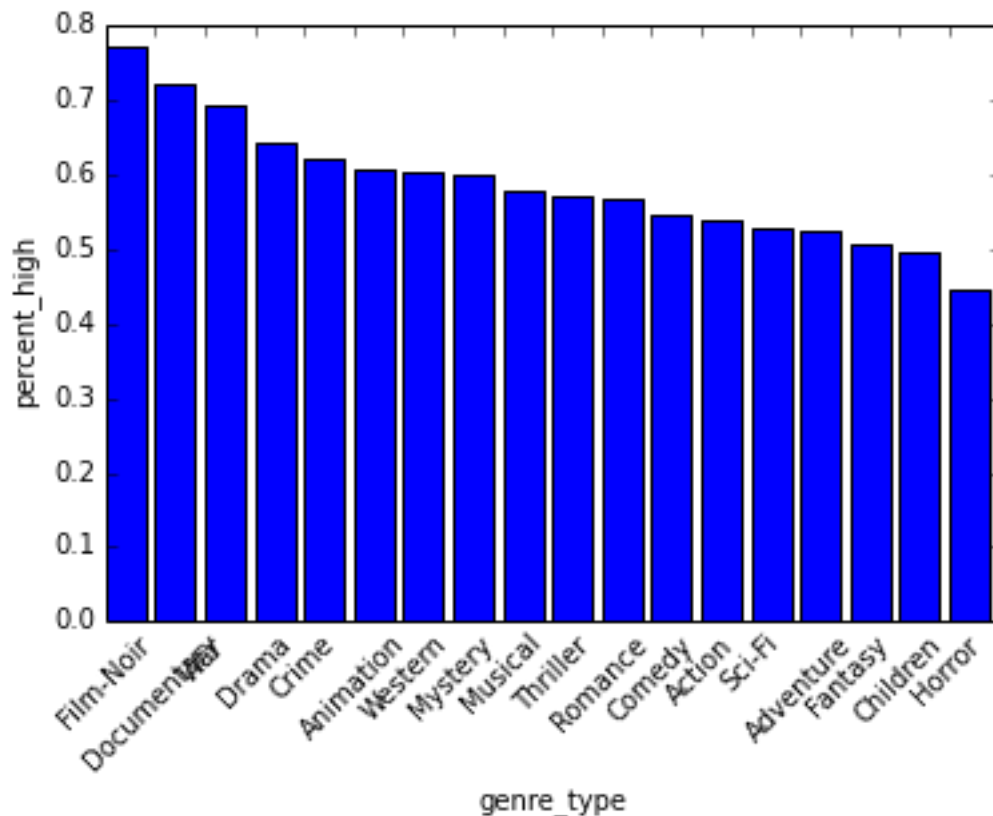
18 rows affected.

```
Out[203]: [('Film-Noir', Decimal('0.7717')),
            ('Documentary', Decimal('0.7218')),
            ('War', Decimal('0.6934')),
            ('Drama', Decimal('0.6438')),
            ('Crime', Decimal('0.6218')),
            ('Animation', Decimal('0.6071')),
            ('Western', Decimal('0.6032')),
            ('Mystery', Decimal('0.6007')),
            ('Musical', Decimal('0.5793')),
```

```
('Thriller', Decimal('0.5694')),
('Romance', Decimal('0.5662')),
('Comedy', Decimal('0.5456')),
('Action', Decimal('0.5385')),
('Sci-Fi', Decimal('0.5293')),
('Adventure', Decimal('0.5260')),
('Fantasy', Decimal('0.5076')),
('Children', Decimal('0.4948')),
('Horror', Decimal('0.4462'))]
```

```
In [204]: result = _
          result.bar()
```

```
Out[204]: <Container object of 18 artists>
```



```
In [205]: %%sql
          SELECT genre_type,
                 high_ratings/total_ratings as percent_high
          FROM(
          SELECT genre_type, COUNT(distinct r.rating_id) as total_ratings
                 ,SUM(CASE WHEN rating_category = 'High' then 1 else 0 end) as high_ratings
          FROM genre_dimension_long gl
          JOIN genre_types gt on gl.genre_id = gt.genre_id
          JOIN ratings r on r.genre_key = gl.genre_key
          JOIN user_dimension u on u.user_key = r.user_key
```

```

        where u.user_gender = 'FEMALE'
        GROUP BY genre_type) A
ORDER BY percent_high desc

```

18 rows affected.

```

Out[205]: [('Film-Noir', Decimal('0.7361')),
           ('Documentary', Decimal('0.7253')),
           ('War', Decimal('0.6951')),
           ('Musical', Decimal('0.6571')),
           ('Drama', Decimal('0.6457')),
           ('Animation', Decimal('0.6360')),
           ('Mystery', Decimal('0.6173')),
           ('Crime', Decimal('0.6151')),
           ('Romance', Decimal('0.6054')),
           ('Thriller', Decimal('0.5748')),
           ('Comedy', Decimal('0.5706')),
           ('Children', Decimal('0.5679')),
           ('Western', Decimal('0.5557')),
           ('Adventure', Decimal('0.5439')),
           ('Action', Decimal('0.5412')),
           ('Fantasy', Decimal('0.5407')),
           ('Sci-Fi', Decimal('0.5271')),
           ('Horror', Decimal('0.4464'))]

```

```

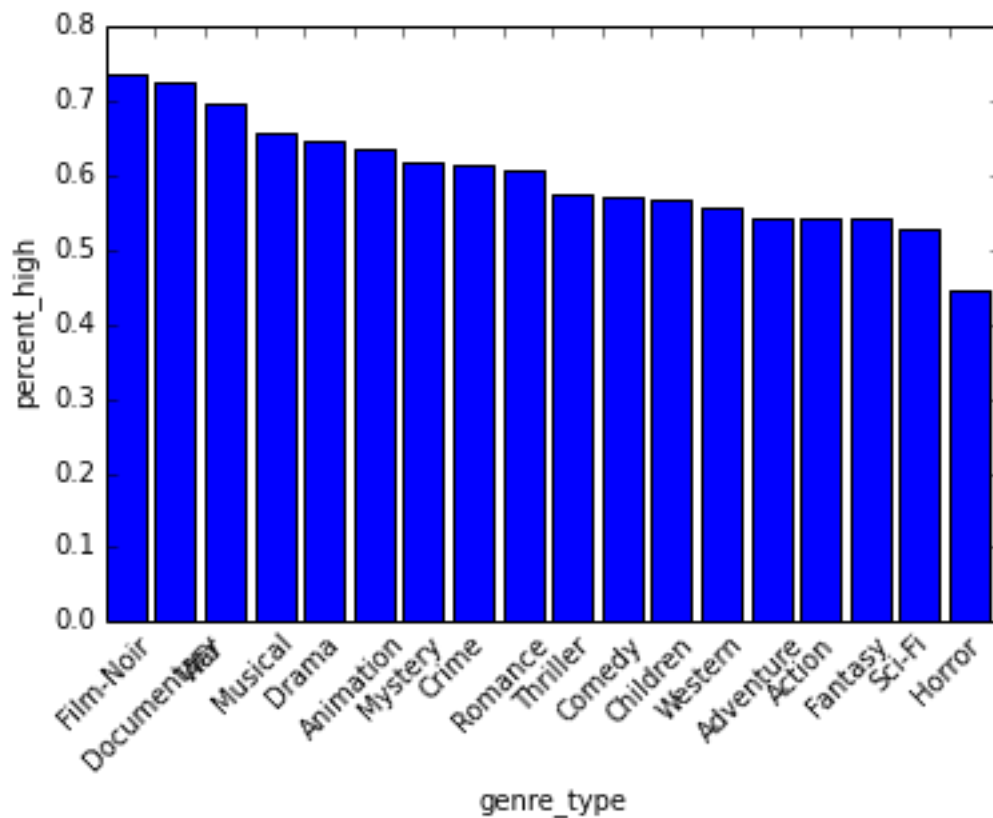
In [206]: result = _
          result.bar()

```

```

Out[206]: <Container object of 18 artists>

```



Next we wanted to know what combinations of genre tended to receive High ratings. This query was a better fit for the genre\_dimension\_wide table because we did not want ratings table rows to be counted more than once. First we found the average rating for movies that were both Action and Adventure and compared this against movies that were both Action and Drama. Movies that were Action and Drama were rated higher.

```
In [207]: %%sql
SELECT avg(rating) as avg_rating, count(*) as num_ratings
FROM ratings r
JOIN genre_dimension_wide gd on r.genre_key = gd.genre_key
WHERE gd.Action = 'TRUE' and gd.Adventure = 'TRUE'
```

1 rows affected.

```
Out[207]: [(Decimal('3.4974'), 90323L)]
```

```
In [208]: %%sql
SELECT avg(rating) as avg_rating, count(*) as num_ratings
FROM ratings r
JOIN genre_dimension_wide gd on r.genre_key = gd.genre_key
WHERE gd.Action = 'TRUE' and gd.Drama = 'TRUE'
```

1 rows affected.

```
Out[208]: [(Decimal('3.7612'), 48916L)]
```

We expanded this query to include user age range so that we could search for differences between the age groups.

```
In [209]: %%sql
SELECT ud.user_age_range, avg(rating) as avg_rating, count(*) as num_ratings
FROM ratings r
JOIN genre_dimension_wide gd on r.genre_key = gd.genre_key
JOIN user_dimension ud on r.user_key = ud.user_key
WHERE gd.Action = 'TRUE' and gd.Adventure = 'TRUE'
GROUP BY ud.user_age_range
```

7 rows affected.

```
Out[209]: [('18-24', Decimal('3.4539'), 18064L),
           ('25-34', Decimal('3.4671'), 36569L),
           ('35-44', Decimal('3.5249'), 17701L),
           ('45-49', Decimal('3.5391'), 6876L),
           ('50-55', Decimal('3.6253'), 5776L),
           ('56+', Decimal('3.6029'), 2795L),
           ('Under 18', Decimal('3.5311'), 2542L)]
```

```
In [210]: %%sql
SELECT ud.user_age_range, avg(rating) as avg_rating, count(*) as num_ratings
FROM ratings r
JOIN genre_dimension_wide gd on r.genre_key = gd.genre_key
JOIN user_dimension ud on r.user_key = ud.user_key
WHERE gd.Action = 'TRUE' and gd.Drama = 'TRUE'
GROUP BY ud.user_age_range
```

7 rows affected.

```
Out[210]: [('18-24', Decimal('3.7787'), 8666L),
            ('25-34', Decimal('3.7118'), 19693L),
            ('35-44', Decimal('3.7809'), 9787L),
            ('45-49', Decimal('3.7494'), 3986L),
            ('50-55', Decimal('3.8498'), 3734L),
            ('56+', Decimal('3.8972'), 1995L),
            ('Under 18', Decimal('3.8284'), 1055L)]
```

### 0.3.1 Benefits

The analyses performed above demonstrate the benefits of having both types of genre dimensions. When we want to compare stats for each genre, a group by clause is required. For this, the `genre_dimension_long` is well suited. On the other hand, when we want to look at genre combinations, it is better to use the `genre_dimension_wide` table as it gives us the flexibility of taking one or more genres under consideration.

The process of loading the data into the data warehouse is very fast. We used csv to export data from the transactional database and load it into the data warehouse. We used staging tables to hold the data temporarily so that we would be able to transform the data into the new structure. Whenever we had to insert or update data, we used joins instead of loops or where clauses which significantly sped up the processing time.

### 0.3.2 Drawbacks

The data does not contain unique user IDs. Knowing the unique user IDs would have enabled us to perform more predictive types of analysis. For example, do users who like Comedy also like other types of movies. This would have enabled us to make recommendations to users based on their preferences. Since this information is critical for the recommendation process, it is likely that the company has this information but does not share it publically.

The data also does not contain movie titles or unique movie IDs. Because of this, we can only look at ratings for different combinations of genres but not for individual movies. Again, this stops us from making recommendations. We also don't know if a particular movie is significantly influencing the overall rating for an entire genre.

### 0.3.3 Improvements

To make exploratory analyses simpler, we added a new attribute of `rating_category` which binned the rating values into High, Medium and Low. Instead of only looking at averages of ratings, analysts can look at the distribution of ratings across these three categories to understand which types of movies are preferred.

If this were a professional project, other improvements might include the addition of the missing data mentioned previously. Also we could add more detail about geographic location. The user data currently includes zip code. The addition of fields like city and state could make for more interesting analyses.

Furthermore, a professional project would require some additional steps as the data is moved between the staging and the final tables. Currently, we assumed that the tables in the data warehouse were empty. In the future, we would have to ensure that only new data is added so that there is no duplication.

### 0.3.4 Contributions of Each Group Member

We worked together on all parts of this assignment. We discussed the structure, implemented the design, suggested and implemented improvements and completed the writeup together.