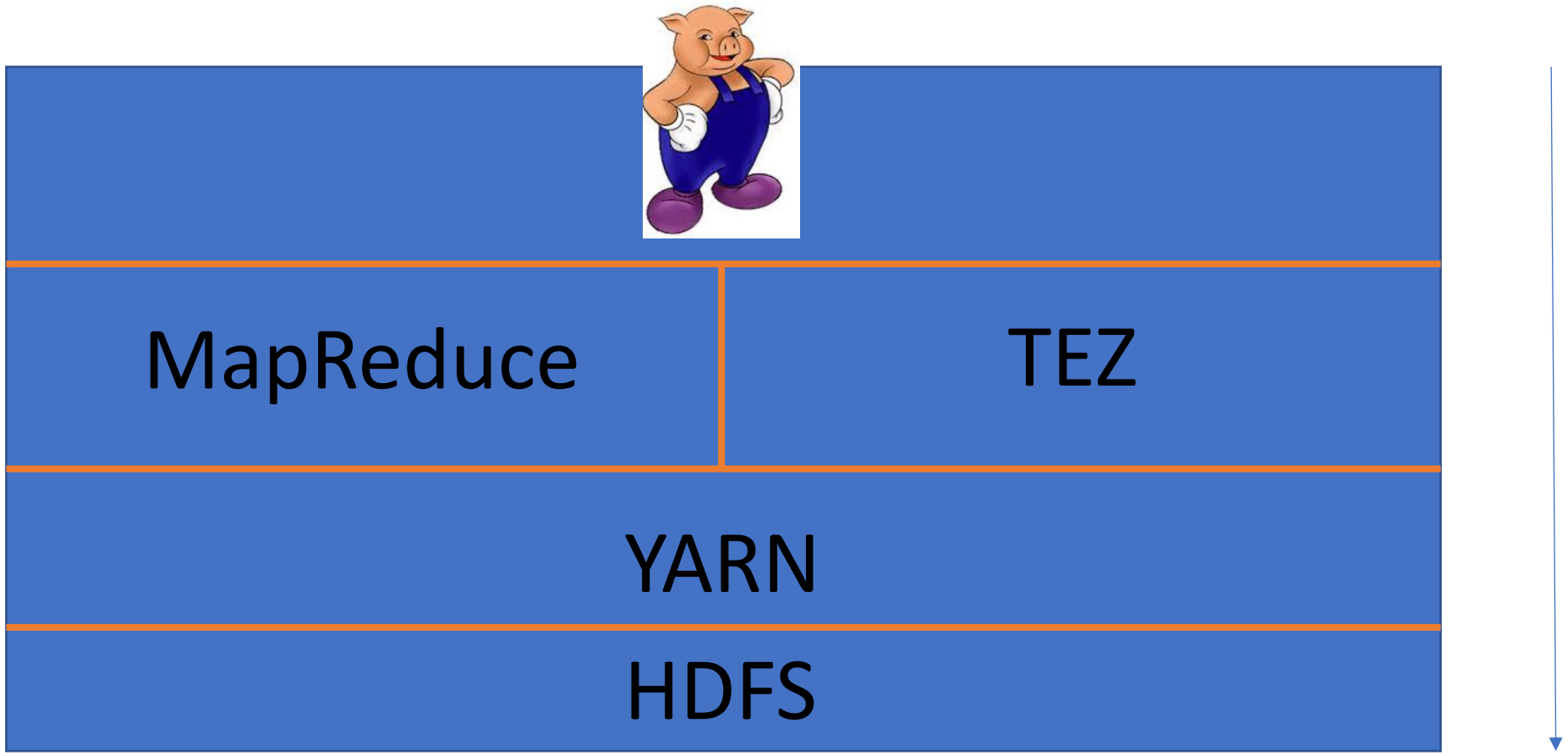
Programming Hadoop with Pig

Bernard Lee Kok Bang

Why PIG?

- a high-level platform for ***processing and analyzing large datasets*** in Apache Hadoop
- part of the Hadoop ecosystem and provides a scripting language known as ***Pig Latin***
- data flow language that allows users to express their data processing tasks in a simple and concise syntax
- abstracts the ***complexity of writing MapReduce*** programs
- Pig Latin is designed to be easy to learn and use, especially for those ***familiar with SQL*** or scripting languages

Pig Architecture



Data Types: Simple Types

Data Type	Examples
Int	10, 5, 12
long	10L, 11L
float, double	Numbers with decimal place; 10.56
Chararray	String in UTF-8 format: Hello World
Boolean	True [T] / false [F]

https://bit.ly/3rYsiPj

Data Types: Complex Types

Data Type	Description	Examples
tuple	An ordered set of fields	(19,2)
bag	A collection of tuples **[unordered]**	{(19, 2), (18,1)}
map	A set of key value pairs	[name#John,phone#5551212]

tuple: - similar to a row in a relational database table
- often used to represent individual records or rows of data

bag: an unordered collection of tuples
: used to represent a collection of records or
: as the result of group operations

https://bit.ly/3rYsiPj

Load/Store Functions

Function	Syntax	Description
BinStorage	A = LOAD 'data' USING BinStorage();	Loads and stores data in machine-readable format.
JsonLoader, JsonStorage	A = load 'a.json' using JsonLoader();	Load or store JSON data.
PigDump	STORE X INTO 'output' USING PigDump();	Stores data in UTF-8 format.
PigStorage	A = LOAD 'student' USING PigStorage('\t') AS (name: chararray, age:int, gpa: float);	Loads and stores data as structured text files.
TextLoader	A = LOAD 'data' USING TextLoader();	Loads unstructured data in UTF-8 format.

https://bit.ly/3rYsiPj

Relational Operator

Operator	Description	Example
JOIN	Performs an inner join of two or more relations based on common field values.	X = JOIN A BY a1, B BY b1; DUMP X (1,2,1,3) (1,2,1,2) (4,5,4,7) A = (1,2) (4,5) B = (1,3) (1,2) (4,7)
LOAD	Loads data from the file system.	A = LOAD 'myfile.txt'; LOAD 'myfile.txt' AS (f1:int, f2:int, f3:int);
UNION	Computes the union of two or more relations. (Does not preserve the order of tuples)	X = UNION A, B; DUMP X; (1,2,3) (4,2,1) (2,4) (8,9) (1,3) A = (1,2,3) (4,2,1) B = (2,4) (8,9) (1,3)
ORDERBY	Sorts a relation based on one or more fields.	A = LOAD 'mydata' AS (x: int, y: map[]); B = ORDER A BY x;

https://bit.ly/3rYsiPj

Relational Operator

Operator	Description	Example	
DISTINCT	Removes duplicate tuples in a relation.	X = DISTINCT A; DUMP X; (1,2,3) (4,3,3) (8,3,4)	A = (8,3,4) (1,2,3) (4,3,3) (4,3,3) (1,2,3)
FILTER	Generates transformation of data for each row as specified	X = FILTER A BY f3 == 3; DUMP X; (1,2,3) (4,3,3) (8,4,3)	A = (1,2,3) (4,5,6) (7,8,9) (4,3,3) (8,4,3)
FOREACH	Selects tuples from a relation based on some condition.	X = FOREACH A GENERATE a1, a2; DUMP X; (1,2) (4,2) (8,3)	A = (1,2,3) (4,2,5) (8,3,6)

https://bit.ly/3rYsiPj

Running Pig Script

- Grunt **- Command line interpreter**
- Script **- write Pig script in a file and run from the command line**
- Ambari / Hue (cloudera)

An example

- Find the oldest (timestamp) **5-star (avgRating > 4)** movies
- Using Movielens datasets [ML-100k]:
- **Sorted by their release date [in UNIX epoch seconds]**
- **Pig scripting is case sensitive; every punctuation matters !!!!**

Create a relation named "ratings" with a given schema

[path in HDFS]

> ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

- **Schema -> assigns name to the field (column) and declares data type of the field**

- **"AS" clause -> gives us the schema of the data**

- **output is a tuple of information**

Output is a tuple

(196,242,3,881250949)
(186,302,3,891717742)
(22,377,1,878887116)

userID	movieID	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596
298	474	4	884182806
115	265	2	881171488
253	465	5	891628467
305	451	3	886324817
6	86	3	883603013

Use PigStorage if we need a different delimiter [Pig expects tab delimited "\t" by default]

```
> metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING
        PigStorage('|') AS (movieID:int, movieTitle:chararray,
        releaseDate:chararray, videoRelease:chararray,
        imdbLink:chararray);
```

- DUMP metadata; -> DUMP will output the result
- **movieID, movieTitle, releaseDate**, link to the movie [in u.data]

movieid **movietitle** **releasedate**

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-ex
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-ex
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-e
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-e
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exac
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan
7|Twelve Monkeys (1995)|01-Jan-1995||http://us.imdb.com/M/tit
8|Babe (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?B
9|Dead Man Walking (1995)|01-Jan-1995||http://us.imdb.com/M/t
10|Richard III (1995)|22-Jan-1996||http://us.imdb.com/M/title
```

Create a relation from another relation; FOREACH /GENERATE

> nameLookup = FOREACH metadata GENERATE movieID, movieTitle, ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) as releaseTime;

[convert into Unix time stamp -> seconds since 1st Jan 1970] **releasetime**

Relation: metadata

movieid movietitle releasedate

movieID, movieTitle, releaseDate

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-ex
2|GoldenEye (1995)|01-Jan-1995||http://us.imdb.com/M/title-ex
3|Four Rooms (1995)|01-Jan-1995||http://us.imdb.com/M/title-e
4|Get Shorty (1995)|01-Jan-1995||http://us.imdb.com/M/title-e
5|Copycat (1995)|01-Jan-1995||http://us.imdb.com/M/title-exac
6|Shanghai Triad (Yao a yao yao dao waipo qiao) (1995)|01-Jan
7|Twelve Monkeys (1995)|01-Jan-1995||http://us.imdb.com/M/tit
8|Babe (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?B
9|Dead Man Walking (1995)|01-Jan-1995||http://us.imdb.com/M/t
10|Richard III (1995)|22-Jan-1996||http://us.imdb.com/M/title
```

convert → **(1,Toy Story (1995),788918400)**

Group By [Creating a bag of tuples]

> ratingsByMovie = GROUP ratings by movieID;

➢DUMP ratingsByMovie;

userID,movieID,rating,timeStamp

```
(1,{(807,1,4,892528231),(554,1,3,876231938),(49,1,2,888068651),(79,1,4,891271870),(8
(2,{(429,2,3,882387599),(551,2,2,892784780),(774,2,1,888557383),(521,2,3,886063310),
(3,{(459,3,2,879563288),(145,3,3,875271562),(886,3,3,876032330),(181,3,2,878963441),
(4,{(497,4,3,879310825),(207,4,4,876198457),(194,4,4,879521397),(450,4,3,882373865),
(5,{(776,5,4,892920320),(388,5,4,886441083),(633,5,3,877212085),(504,5,4,887912462),
(6,{(9,6,5,886960055),(181,6,1,878962866),(79,6,4,891271901),(537,6,2,886029806),(18
(7,{(89,7,5,879441422),(693,7,4,875483947),(192,7,4,881367791),(201,7,3,884112201),(
(8,{(488,8,3,891295067),(301,8,4,882076494),(379,8,5,880525194),(215,8,2,891436177)
```

Results

```
ratingsByMovie: {group: int,ratings: {(userID: int,movieID: int,rating: int,ratingTime: int)}}
```

AVG **Compute the average ratings for each movie**

```
> avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID,
        AVG(ratings.rating) AS avgRating;
> DUMP avgRatings;
```

```
(1,3.8783185840707963)
(2,3.2061068702290076)
(3,3.0333333333333333)
(4,3.550239234449761)
(5,3.302325581395349)
(6,3.576923076923077)
(7,3.798469387755102)
(8,3.9954337899543377)
(9,3.8963210702341136)
(10,3.831460674157303)
(11,3.847457627118644)
(12,4.385767790262173)
```

movieID

avgRating

Diagnostics

```
> DESCRIBE ratings;
> DESCRIBE ratingsByMovie;
> DESCRIBE avgRatings;
```

DESCRIBE: is used to display the schema of a relation or data set

FILTER **[Good movies only, with average ratings more than 4]**

```
> fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;
```

JOIN

```
fiveStarMoviesWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

DESCRIBE fiveStarMoviesWithData;

DUMP fiveStarMoviesWithData;
```

ORDER BY

```
> oldestFiveStarMovie = ORDER fiveStarMoviesWithData BY
        nameLookup::releaseTime;
> DUMP oldestFiveStarMovie;
```

Oldest Good Movies ☺

Putting together

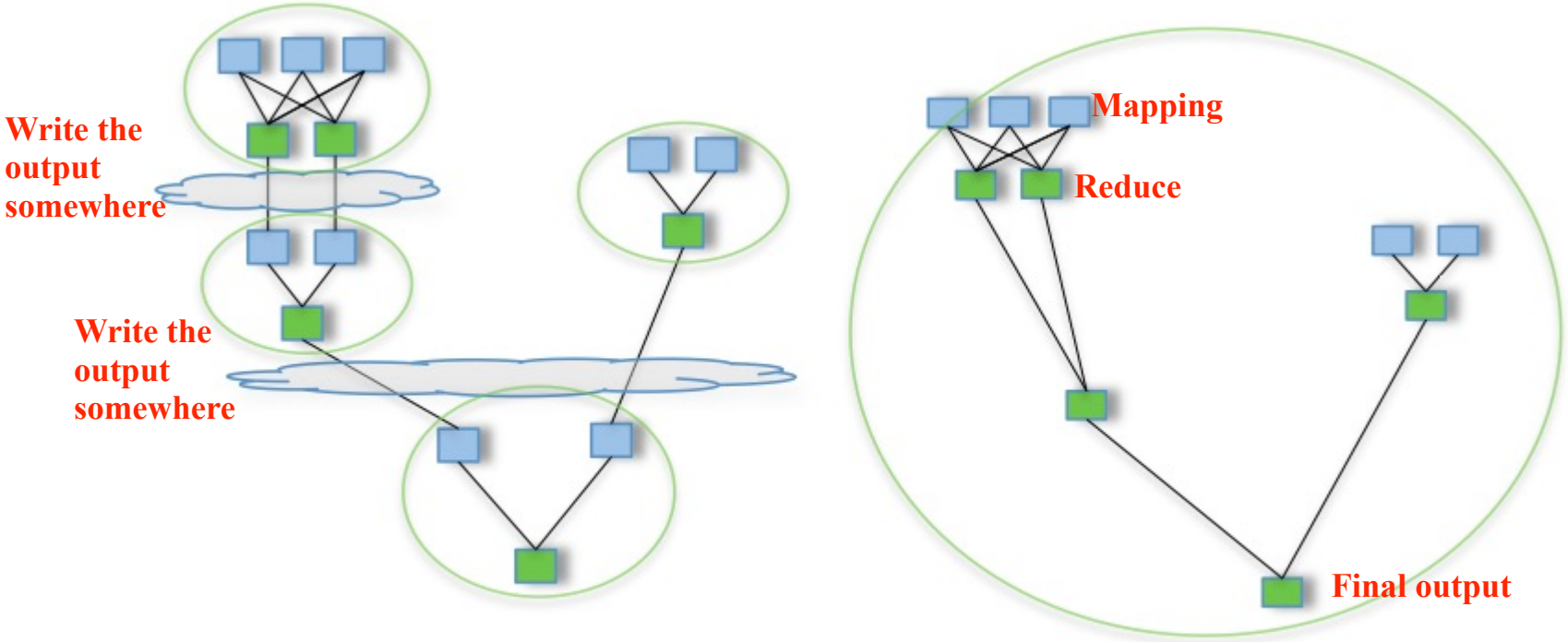
```
ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate:chararray, videoRelease:chararray, imdbLink:chararray);

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
    ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;

ratingsByMovie = GROUP ratings BY movieID;

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID, AVG(ratings.rating) AS avgRating;

fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;

fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;
```

TEZ

Once client execute a function, Tez will evaluate what's the final output; then work backward to get the most optimal pathway

- an application framework which allows for a complex directed-acyclic-graph (DAG) of tasks for processing data.

backward workflow



Pig/Hive - MR

MapReduce algorithm

Pig/Hive - Tez

Tez algorithm

Lazy evaluation

https://bit.ly/3vqVPmX

To run even faster (hopefully)!!

Let's time it -> Pig alone v.s. Pig + Tez !!!!

Challenge: Find DISTINCT records using PIG

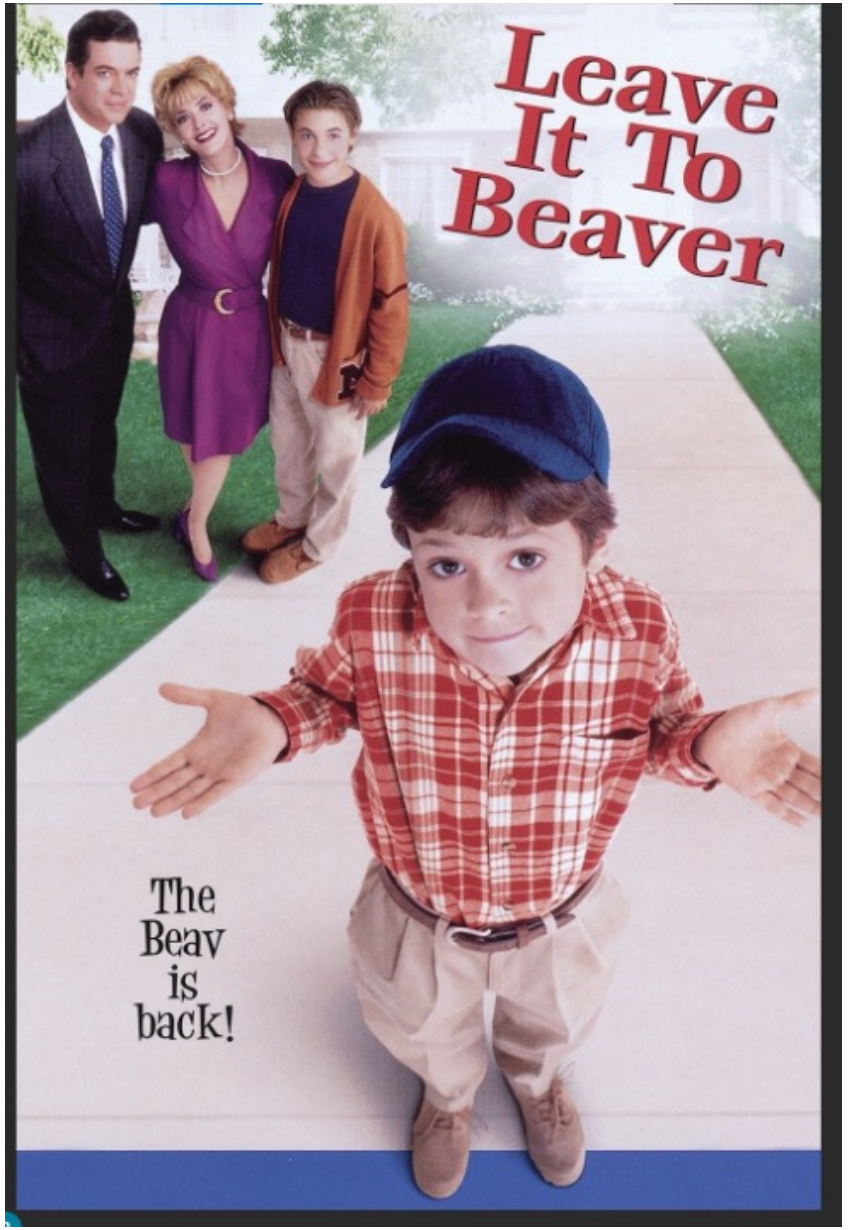
- Save this data into a csv file called "**student_details.csv**"
- Load the csv file from PuTTY to hadoop
- Find distinct records
- Save the results into your folder in hadoop

student_details.csv

```
001Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khana,9848022339,Delhi
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai
006,Archana,Mishra,9848022335,Chennai
```

Another Challenge

- Write a Pig Script that finds the ***most popular bad movies***
- The criteria:
  - Find all movies with an average *rating less than 2.0*
  - Sort them by *total number of ratings* [in descending order]
  - Save the results in a file name.
- Use the earlier example of finding old movies with ratings > 4.0
- The only new function is ***COUNT()***
- instead of ***AVG(ratings.rating)***, we can use ***COUNT(ratings.rating)*** to get the total number of ratings for a given group's bag



Most_rated_oneStar_movies - COMPLETED

Job ID	job_1712561115437_0044
Started	2024-04-08 19:42

Results

```
(Leave It to Beaver (1997),1.8409090909090908,44)
(Mortal Kombat: Annihilation (1997),1.9534883720930232,43)
(Crow: City of Angels, The (1996),1.9487179487179487,39)
(Bio-Dome (1996),1.903225806451613,31)
(Barb Wire (1996),1.9333333333333333,30)
(Free Willy 3: The Rescue (1997),1.7407407407407407,27)
(Showgirls (1995),1.9565217391304348,23)
(Lawnmower Man 2: Beyond Cyberspace (1996),1.7142857142857142,21)
(Children of the Corn: The Gathering (1996),1.3157894736842106,19)
(Home Alone 3 (1997),1.894736842105263,19)
(Ready to Wear (Pret-A-Porter) (1994),1.8333333333333333,18)
```

Learning more