DBMS Models and Implementation
Instructor: Abhishek Santra
Project 3: Data Analysis using Map/Reduce + Query Processing

Made available on: 3/19/2024
Submit by: 4/23/2024 (11:59 PM) No extension for this project
Submit to: Canvas (1 zipped folder containing all the files/sub-folders)
Weight: 15% of total
Total Points: 100

One of the advantages of cloud computing is its ability to deal with very large data sets and still have a reasonable response time. Typically, the map/reduce paradigm is used for these types of problems in contrast to the RDBMS approach for storing, managing, and manipulating this data. An immediate analysis of a large data set does not require designing a schema and loading the data set into an RDBMS. Hadoop is a widely used open-source map/reduce platform. Hadoop Map/Reduce is a software framework for writing applications which process vast amounts of data in parallel on large clusters. In this project, you will use the IMDB (International Movies) dataset and develop programs to get interesting insights into the dataset using Hadoop map/reduce paradigm.
Please use the following links for a better understanding of Hadoop and Map/Reduce (https://hadoop.apache.org/docs/r3.3.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)

We have the entire IMDb data set installed as a Relational database on Oracle (on UTA's omega computer). It is a large database with Millions of rows as can be seen from the schema given at the end of this description. The IMDb dataset is publicly available and stores world-wide information about movies, TV episodes, actor, directors, ratings and genres of the movies, etc. This allows us to write SQL queries as well and see whether we can compute the same things as we do on Map/Reduce and also understand query plans using the EXPLAIN statement of Oracle. We can also drill down further on the results of Hadoop analysis.

1. **Installation: There are two options**
   A. **RECOMMENDED - Hadoop Single Node Cluster Setup (Hadoop 3.3.2):** We advise you to use a Linux installation such as Ubuntu 16.04 on your system in order to breeze through the steps of the Hadoop cluster set up. You can use a virtual machine for this purpose.

      NOTE: If you are on Windows, you may also use the Windows Subsystem for Linux (WSL). It allows users to install a complete Ubuntu terminal environment in minutes on a Windows machine, allowing cross-platform application development without leaving Windows. You may follow the instruction provided here: https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10

      The steps to install a *single node cluster in the pseudo-distributed mode* are given here – https://hadoop.apache.org/docs/r3.3.2/hadoop-project-dist/hadoop-common/SingleCluster.html

      Some important points that must be considered while setting up the cluster are:

- Preferred site for download: https://archive.apache.org/dist/hadoop/common/hadoop-3.3.2/hadoop-3.3.2.tar.gz
- After completing the prerequisites and running the command ($ bin/hadoop), jump to https://hadoop.apache.org/docs/r3.3.2/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation, and follow the steps for Configuration, Setup passphraseless ssh, Execution (Only steps 1 – 4) and YARN on single node (Steps 1-3)
- The cluster will start up by running the commands,
  $ sbin/start-dfs.sh
  $ sbin/start-yarn.sh
  which you must have done while following the above steps.

  If you face an error stating "localhost: rcmd: socket: Permission denied" on running start-dfs.sh, you have to set the default rcmd type to ssh, which you can do by running the following command:

  ```
  export PDSH_RCMD_TYPE=ssh
  ```

  This should resolve the error and allow start-dfs.sh to run correctly.

  Also, if you have installed ssh previously but it is not running, there will be an error stating:

  ```
  ssh: connect to host localhost port 22: Connection refused
  ```

  You can check whether ssh is running by running the following command:

  ```
  service ssh status
  ```

  If it states "`* sshd is not running`", then run the following command:

  ```
  sudo service ssh start
  ```

  After this, it should say -

  ```
  * Starting OpenBSD Secure Shell server sshd
  ```

  And on running `service ssh status` again, you should get the following message:

  ```
  * sshd is running
  ```

- To check if the Namenode, Datanode, Secondary Namenode, ResourceManager, NodeManager are running as separate processes, run the command
  $ jps
- Once the cluster is up, run the most basic code, WordCount, that counts the number of occurrences of each word in the input files
  - Download the WordCount.java code from canvas into the updated Hadoop folder

- Make a directory (example, inputFiles) on the HDFS (Hadoop Distributed File System) to store the input files
  $ **bin/hdfs dfs -mkdir** /inputFiles
- Copy a file from the local file system to the HDFS using the command
  $ **bin/hdfs dfs -put** <path_of_file_on_local_system> /inputFiles
- To execute the code, follow the steps given here,
  https://hadoop.apache.org/docs/r3.3.2/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0
- If you face errors like "`HADOOP_COM.SUN.TOOLS.JAVAC.MAIN_USER: invalid variable name`" during the compilation step, do not do the following two steps:

```
$ bin/hadoop com.sun.tools.javac.Main WordCount.java

$ jar cf wc.jar WordCount*.class
```

Instead, do

```
$ mkdir -p classes

$ javac -d classes -cp classes:`hadoop classpath` WordCount.java

$ jar cf wc.jar -C classes .

$ chmod -R 750 classes

$ chmod -R 750 wc.jar
```

- This will create a folder called "classes" and compile the java code and place the classes in that folder. Please note that the jar file will be created in the present working directory, not in the classes folder.
- Then, follow the steps to run the program as mentioned in the guide.

- To stop the cluster, run the commands
  $ sbin/stop-dfs.sh
  $ sbin/stop-yarn.sh
- Some useful shell commands for the HDFS can be found here,
  https://hadoop.apache.org/docs/r3.3.2/hadoop-project-dist/hadoop-common/FileSystemShell.html

B. **Cloud Services (self-exploration is required for this option):** The second option is to use Amazon Elastic Map/Reduce (or equivalent). Amazon EMR (Elastic Map/Reduce) is a web service provided by Amazon that uses Hadoop and distributes large datasets and processes them into multiple EC2 instances. You have to sign up for AWS. Please make sure you read carefully your AWS agreements/contracts/free use. You may have sufficient free services to complete the projects, but you should monitor and understand your use. Don't leave processes running when not necessary. Note that if you exceed your monthly quota, you will get charged. Also, a

credit/debit card is necessary for signing up for this service. You may use other cloud environments like IBM cloud, Google cloud or Microsoft Azure. There should be online help available for different cloud providers regarding how to set up a cluster and run the first basic map/reduce code. ***Note that if you use Amazon/ECS, you can use multiple mappers and reducers and will be able to better understand and appreciate distributed aspect of map/reduce and how the response time changes (should decrease) with more resources. You may not be able to do this on your laptop installation of Hadoop or on your desktop.***

2. IMDB Dataset (Available on Canvas)

IMDB is a dataset containing information about movies (international) and TV episodes from their beginnings. The information includes movie titles, directors, actors, genre, and year produced/started. Some rating information is also included. The same is true for TV episodes and includes a number of seasons in terms of start and end years as well as episodes in each season. It is quite large and should not require additional information to understand the data set. The data of this database that you will use is given below.

Spring2024-Project3-IMDbData.txt: This dataset contains the information about the various **imdb titles** (movies, tv episodes, documentaries etc.), produced across the world. Each field in this dataset is *separated by a semi-colon*. A random sample from this file has been shown below

tt0108778;tvSeries;Friends;1994;8.9;Comedy,Romance
tt0468569;movie;The Dark Knight;2008;9;Action,Crime,Drama
tt2388725;short;Paperman;2012;8.2;Animation,Comedy,Family

The description of the various fields has been given below

Field Number	Field Name	Field Description (fields are separated by ";", within a field (e.g., genre) items are separated by ",")
1	TITLE ID	The 9-digit unique IMDB title identifier attached to every entry, example: *tt0468569*
2	TITLE TYPE	Every IMDB title in the file is categorized as one of the following TITLETYPEs • *tvMovie* • *tvSpecial* • *tvPilot* • *tvMiniSeries* • *tvEpisode* • *tvSeries* • *movie* • *video* • *videoGame* • *short* • *tvShort*
3	TITLE NAME	The name of the IMDB Title, example: *The Dark Knight*
4	YEAR	The year of release, example: *2008*
5	RATING	The IMDb rating of the title, example: *9*
6	GENRE LIST	Multiple genres can be attached to a particular title, and they are separated by **commas,** example: *Action,Crime,Drama*

3. **Project 3 Problem Specification:** You need to compute the following for the given data using the map/reduce paradigm. Try to compare and understand how you would do it using RDBMS if these files were stored as relations. This may help you understand when map/reduce is meaningful and when to use a RDBMS.

[Task 1] Write a Map/Reduce program to compute the *number of highly rated movies* with *different genre combinations* for *pre-defined periods*.

i) 3 disjoint 10-year periods are being explored as a part of this project – [1991-2000], [2001-2010], and [2011-2020].

ii) Genre Combinations to be explored

- Action, Thriller
- Adventure, Drama
- Comedy, Romance

IMPORTANT: *Remember, when exploring a genre combination like Comedy, Romance you need to consider the movies with a rating of at least 7.5 and which have at least these two genres present in their corresponding list of genres. For example, movies with "Comedy,Drama,Romance", "Comedy,Family,Romance", "Comedy,Romance, etc. satisfy the requirement.*

Example M/R code output for the above

[1991-2000],Comedy;Romance,<numberOfHighlyRatedMovies>
[1991-2000],Action;Thriller,<numberOfHighlyRatedMovies>

iii) Using the generated result, analyze and **plot 2** interesting inferences. *For example, you can plot a histogram of each genre combination for different periods and perform a trend analysis. You can come up with other types of analysis, such as the genre combination that is declining across periods, or the genre combination that is going strong across periods.* You can use spreadsheet to post-process the results obtained from the Map/Reduce program to perform the analysis. You can plot histograms, line graphs, scatter plots etc., in order to justify your inferences. **Think out of the box!**

You need to design and develop a map program (including a combiner if needed) and a reduce program to solve the above problems. The most important aspects of this design will be to identify the <key, value> pairs to be output by the mapper and computations in the reducer to produce the desired final output.

[TASK 2] Use the IMDb database on omega whose s**chema details are given at the end of this description to** write following SQL queries. **Be careful and aware that this is NOT a toy database as it has millions of rows in each table and hence your queries can produce tens of thousands of rows as output! Hence, you need to think before posing a query.**

iv) For ***one of the 3 periods and one of the genre combinations given above***, write an SQL query to identify the **top 5 movies**, based on **movie *rating***. *Consider only those movies that have received at least 150000 votes*. Output the ***movie names, ratings***, and optionally other details, such as ***lead actor/actress names***. *You are free to choose any period and genre combination.*

IMPORTANT: *Remember, when exploring a genre combination like Comedy, Romance you need to consider the movies which have at least these two genres present in their corresponding list of genres. For example, movies with "Comedy,Drama,Romance", "Comedy,Family,Romance", "Comedy,Romance", etc. satisfy the requirement.*

Also, someone is a lead actor/actress if the ordering attribute in the title_principals table is 1.

v) Using the EXPLAIN statement of Oracle, generate the query plan used in the query in **(iv)** and discuss how the query is evaluated.
Refer: https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i16971

4. Project Report: Please include (at least) the following sections in a **REPORT.{pdf, doc, docx}** file that you will turn in with your code:

i. **Overall Status**
Give a *brief* overview of how you implemented the major components. If you were unable to finish any portion of the project, please give details about what is completed and your understanding of what is not. (This information is useful when determining partial credit.)

ii. **Analysis Results:**
- [TASK 1 – M/R] Explain all the results and related inferences (with graphs if needed) that were drawn.
- [TASK 2 - SQL] Give the SQL query and NEATLY FORMATTED output along with the query plan. *All English queries, SQL equivalents, and their output should be in one output file* (named **4331-5331_Proj3Spring24_team_<teamNo>.sql**)

For this, you should use the comment capability (--) in SQL to write English queries and use set echo on and spool <filename> and spool off for getting everything in one file.

iii. **File Descriptions**
List the files you have created and *briefly* explain their major functions and/or data structures.

iv. **Division of Labor**
Describe how you divided the work, i.e. which group member did what. Please also include how much time each of you spent on this project.

5. What to submit:

- After you are satisfied that your code does exactly what the project requires, you may turn it in for grading. Please submit your project report with your project.
- You will turn in one zipped file containing you're
  a. **Source code (M/R code, English queries, SQL queries, their output)**
  b. **Outputs from the M/R code**
  c. **Raw spreadsheets that were used for analysis.**
  d. **Report that includes analysis results for both as specified above**
- All the above files should be placed in a single zipped folder named as '**4331-5331_Proj3Spring24_team_<teamNo>**'
  Only one zipped folder should be uploaded using CANVAS. No Email Submissions
- You can submit your zip file multiple times. The latest one (based on timestamp) will be used for grading. So, be careful about what you turn in and when!
- **Only one person per group should turn in the zip file!**
- **[IMPORTANT] To discourage late submissions, a penalty of 25% per day (no partial penalty) will be imposed. This means that no submission will be accepted if it is delayed by more than 3 days.**

6. Coding style:

Be sure to observe the following standard Java naming conventions and style. These will be used across all projects for this course; hence it is necessary that you understand and follow them correctly. You can look this up on the web. Remember the following:

i. Class names begin with an upper-case letter, as do any subsequent words in the class name.
ii. Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter.
iii. Class, instance, and local variables begin with a lower-case letter, and any subsequent words in the name of that variable begin with an upper-case letter.
iv. No hardwiring of constants. Constants should be declared using all upper-case identifiers with _ as separators.
v. All user prompts (if any) must be clear and understandable.
vi. Give meaningful names for classes, methods, and variables even if they seem to be long. The point is that the names should be easy to understand for a new person looking at your code.
vii. Your program is properly indented to make it understandable. Proper matching of if … then … else and other control structures is important and should be easily understandable
viii. Do not put multiple statements in a single line.

In addition, ensure that your code is properly documented in terms of comments and other forms of documentation for generating meaningful Javadoc.

7. Grading Scheme:

i. The **TASK 1 (Map Reduce)** will be graded using the following scheme (*100 Points*):

a.	Set Up	**20**
b.	Correctness of the Map Code	**20**
c.	Correctness of the Reduce Code	**20**
d.	Correctness of the Output	**20**
e.	Analyzing the results obtained *(2 Analysis Results with graphs, plots etc., wherever necessary)*	**20**

ii. The **TASK 2 (SQL Queries)** will be graded using the following scheme (*50 Points*):

f.	Correctness of the SQL Query and Output	**20**
g.	Query Plan generation using EXPLAIN statement	**20**
h.	Explanation/analysis of query plan	**10**

iii. Overall Completeness of Report **20**
(Status, File Descriptions, Division of Labor)

iv. Answering questions during **MANDATORY demo** **30**

TOTAL **200**

8. Mandatory Project 3 Demonstrations: Mandatory Team-wise demos will be scheduled after the due date. A sign-up sheet for the demo Schedule will be provided.

Accessing Oracle on Omega

All of you have access to Omega using your Netid and password. You should also have access to Oracle whose password is pre-defined for you. In case you do not have access to Oracle, please check with OIT to get access. Following are the steps for logging into Omega and Oracle

i) Log into Omega using Putty. You will also need to use the Ivanti Secure Client vpn.
ii) Type sqlplus {username{/password}}
iii) Prompts (with SQL>) for username and password if not given in the above statement
iv) You have access to the IMDb database created in the database imdb00. You can only access it but not modify.

Some useful commands:

i) SQL> Select * From cat; //note the semicolon

Lists all the tables and views defined by the current user.

ii) SQL> Describe imdb00.title_basics // no semicolon for describe statement

//you have to prefix all table names with imdb00

Describes the details (attributes and types) of the title_basics table in the database imdb00

iii) help command: Command is the name for which you want to get help!

E.g., SQL>help column

iv) You can either enter sql commands at the prompt or put it in a file and submit it.

v) If you put it in a file, use the command

SQL>@<filename> or start <filename> to execute statements in the file.

If the file name has .sql extension, you need not specify the extension

Since you are experimenting, please use a file that you can change and try again!

vi) SQL> Set echo on

Sets echo on to echo all the input; especially important if you are submitting a file

vii) SQL> Spool <filename>

viii) SQL> spool off

Is useful to redirect your output to a file for analysis. Created in the current directory.
You need to set spool off to see its contents. Subsequent runs will append to the same file!

IMDB Database on Omega

At ITLab, we have created a large database and populated it with millions of rows of International Movies and TV episodes information. It is known as the IMDb database by the community (publicly available data set, but not as a relational DBMS) and used by researchers in databases and other fields. The details of the tables are given below. This has all movie and TV episode information from the beginning (1890s approx.) until 2021 for US and international movies and TV episodes. You can query this database for looking up certain information of interest, finding aggregate and statistical information that you are interested in, and OLAP analysis queries as well to the extent possible using SQL. **This database has been populated with Oracle on Omega**.

The IMDb database includes the following information: movie title, year produced, genres a movie belongs to, actors, writers, directors, runtime, adult or non-adult classification, reviews in terms of votes on the movie, average rating, region, language etc. Similarly for TV series.

The purpose of setting up this database and this homework is to provide you with an understanding of the differences between a toy DBMS and large real-world DBMS, in terms of the kinds of queries you can ask, the response time, and appreciate the technology behind a DBMS (query optimization, concurrency control, simple relational abstraction, easy-to-use, non-procedural query language etc.)

Please make sure you do not write queries that produce large amounts of output. You need to think in terms of aggregate queries so you can extract the sliver of information that you are interested in. Also, as many fields contain strings with some delimiter, you need to include the LIKE operator with % and _ for picking out the correct string of interest (can also use string matching). For example, genres can be matched using LIKE 'Comedy' or LIKE 'Drama'. Note the first letter is capitalized. The other genres present are Horror, Short, Thriller, Sci-Fi, Music, Musical, to name a few. Explore other attributes as well. Some populated field values have a \N as their value. So, it is useful to have NOT LIKE '\N' to exclude those. *Type simple queries to explore and understand the schema and attribute domains.*

[MANDATORY] Note that you have to use the prefix "imdb00.", to access these tables on omega. For example, to access/query the table title_basics, use imdb00.title_basics
This is critical for the correctness of the queries and the evaluation process.

The following tables are populated in the database:

1. **TITLE_BASICS table:** *Contains the information for titles*

```
SQL> describe imdb00.TITLE_BASICS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TCONST                                    NOT NULL VARCHAR2(10)
 TITLETYPE                                          NVARCHAR2(1000)
 PRIMARYTITLE                                       NVARCHAR2(1000)
 ORIGINALTITLE                                      NVARCHAR2(1000)
 ISADULT                                            VARCHAR2(10)
 STARTYEAR                                          VARCHAR2(10)
 ENDYEAR                                            VARCHAR2(10)
 RUNTIMEMINUTES                                     VARCHAR2(20)
 GENRES                                             NVARCHAR2(1000)


SQL> select count(*) from imdb00.TITLE_BASICS;

  COUNT(*)
----------
   Total number of rows: 9099315 (~9.1 million rows)
```

****

Additional Attribute Description:

1. TCONST: *alphanumeric unique identifier of the title*
2. TITLETYPE: *the type/format of the title (e.g., movie, short, tvseries, tvepisode, video, etc.)*
3. PRIMARYTITLE: *the more popular title / the title used by the filmmakers on promotional materials at the point of release*
4. ORIGINALTITLE: original title, in the original language
5. ISADULT: *0 (on-adult title); 1 (adult title)*
6. STARTYEAR: *represents the release year of a title. In the case of TV Series, it is the series start year*
7. ENDYEAR: *TV Series end year. '\N' for all other title types*
8. RUNTIMEMINUTES: *primary runtime of the title, in minutes*
9. GENRES: *includes up to three genres associated with the title*

2. **NAME_BASICS table:** *Contains the information for names/people involved*

```
SQL> describe imdb00.NAME_BASICS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NCONST                                    NOT NULL VARCHAR2(10)
 PRIMARYNAME                                        NVARCHAR2(1000)
 BIRTHYEAR                                          VARCHAR2(10)
 DEATHYEAR                                          VARCHAR2(10)
 PRIMARYPROFESSION                                  NVARCHAR2(1000)
 KNOWNFORTITLES                                     NVARCHAR2(1000)

SQL> select count(*) from imdb00.NAME_BASICS;

  COUNT(*)
----------
      11804374 (~11 million rows)
```

****

Additional Attribute Description:

1. NCONST: *alphanumeric unique identifier of the name/person*
2. PRIMARYNAME: *name by which the person is most often credited*
3. BIRTHYEAR: *in YYYY format*
4. DEATHYEAR: *in YYYY format if applicable, else '\N'*
5. PRIMARYPROFESSION: *the top-3 professions of the person*
6. KNOWNFORTITLES: *titles (TCONSTs) the person is known for*

3. **TITLE_PRINCIPALS table:** *Contains the principal cast/crew for titles*

```
SQL> describe imdb00.TITLE_PRINCIPALS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TCONST                                    NOT NULL VARCHAR2(10)
 ORDERING                                           VARCHAR2(10)
 NCONST                                    NOT NULL VARCHAR2(10)
 CATEGORY                                           NVARCHAR2(1000)
 JOB                                                NVARCHAR2(1000)
 CHARACTERS                                         NVARCHAR2(1000)


 SQL> select count(*) from imdb00.TITLE_PRINCIPALS;

  COUNT(*)
----------
 Total number of row: 51316529 ( ~51 million rows)
```

****

Additional Attribute Description:

1. TCONST: *alphanumeric unique identifier of the title*
2. ORDERING: *a number to uniquely identify rows for a given titleId (TCONST)*
3. NCONST: *alphanumeric unique identifier of the name/person*
4. CATEGORY: *the category of job that person was in*
5. JOB: *the specific job title if applicable, else '\N'*
6. CHARACTERS: *the name of the character played if applicable, else '\N'*

4. TITLE_RATINGS tables: *Contains the IMDb rating and votes information for titles*

```
SQL> describe imdb00.TITLE_RATINGS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TCONST                                    NOT NULL VARCHAR2(10)
 AVERAGERATING                                      NUMBER(5,2)
 NUMVOTES                                           NUMBER(15)

SQL> select count(*) from imdb00.TITLE_RATINGS;

  COUNT(*)
----------
 Total number of rows:  1254579 ( ~1.2 million rows)
```

****

Additional Attribute Description:

1. TCONST: *alphanumeric unique identifier of the title*
2. AVERAGERATING: *weighted average of all the individual user ratings*
3. NUMVOTES: *number of votes the title has received*

5. **TITLE_AKAS table:** *Contains the following information for titles*

```
SQL> describe imdb00.TITLE_AKAS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TITLEID                                   NOT NULL VARCHAR2(10)
 ORDERING                                           VARCHAR2(10)
 TITLE                                              NVARCHAR2(1000)
 REGION                                             NVARCHAR2(1000)
 LANGUAGE                                           NVARCHAR2(1000)
 TYPES                                              NVARCHAR2(1000)
 ATTRIBUTES                                         NVARCHAR2(1000)
 ISORIGINALTITLE                                    VARCHAR2(10)

SQL> select count(*) from imdb00.TITLE_AKAS;

  COUNT(*)
----------
   Total number of rows: 32689621 (~32 million rows)
```

****

Additional Attribute Description:

1. TITLEID: *a TCONST, an alphanumeric unique identifier of the title*
2. ORDERING: *a number to uniquely identify rows for a given titleId*
3. TITLE: *the localized title*
4. REGION: *the region for this version of the title*
5. LANGUAGE: *the language of the title*
6. TYPES: *Enumerated set of attributes for this alternative title. One or more of the following: "alternative", "dvd", "festival", "tv", "video", "working", "original", "imdbDisplay".*
7. ATTRIBUTES: *Additional terms to describe this alternative title, not enumerated*
8. ISORIGINALTITLE: *0 (not original title); 1 (original title)*

6. **TITLE_EPISODE table:** *Contains the tv episode information*

```
SQL> describe imdb00.TITLE_EPISODE

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TCONST                                    NOT NULL VARCHAR2(10)
 PARENTTCONST                              NOT NULL VARCHAR2(10)
 SEASONNUMBER                                       VARCHAR2(10)
 EPISODENUMBER                                      VARCHAR2(10)

SQL> select count(*) from imdb00.TITLE_EPISODE;

  COUNT(*)
----------
   Total number of rows: 6846726 (~6.8 million rows)
```

****

Additional Attribute Description:

1. TCONST: *alphanumeric identifier of episode*
2. PARENTTCONST: *alphanumeric identifier of the parent TV Series*
3. SEASONNUMBER: *season number the episode belongs to*
4. EPISODENUMBER: *episode number of the TCONST in the TV series*