

Practical assignment 1 - Query optimization (Oracle)

COMP7104-DASC7104

This assignment is a concrete application of the concepts, structures, algorithms presented in class, on query execution and query optimization, with the ORACLE DBMS. This system provides a good example of a sophisticated optimizer based on **index structures** and comprehensive **evaluation algorithms**. All the **join** algorithms described in class are indeed implemented in ORACLE. In addition, the system offers simple and practical tools (**AUTOTRACE** and **EXPLAIN**) to understand the execution plan chosen by the optimizer, as well as to obtain performance statistics (I/O cost and CPU cost, among others).

Starting the Oracle DBMS and connecting to it with SQL Developer

I can launch the Oracle server (its Docker image) by running the following command:

```
docker run -d --name oracle-ee -v
/Users/Bogdan/data/OracleDBData:/opt/oracle/oradata -p 1521:1521 -p
5500:5500 -e ORACLE_PWD='BoGDdanC1!e!' container-
registry.oracle.com/database/enterprise:21.3.0.0
```

In your command, you must replace

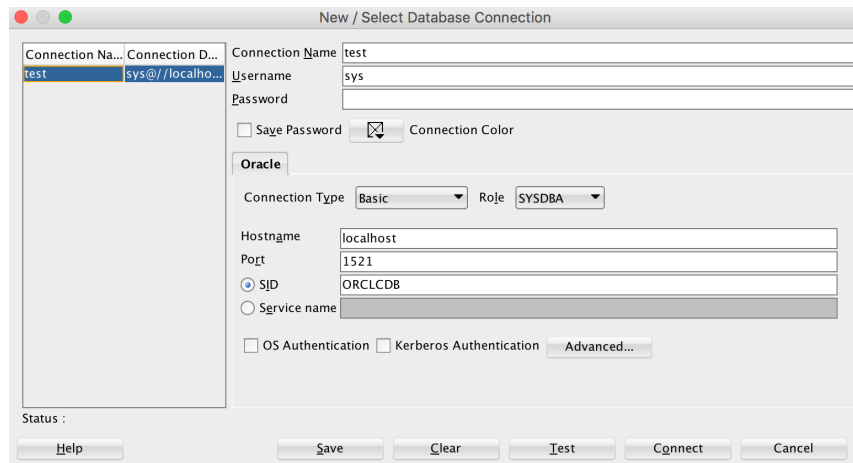
- 1) /Users/Bogdan/data/OracleDBData with a directory that you created beforehand on your machine. This is the directory where the Oracle database will be saved whenever you turn off the Oracle server (its Docker image); in this way, after each restart you can reuse the database.
- 2) 'BoGDdanC1!e!' by your own password (feel free to use mine too...), recall that you need a not-too-simple password, at least 8 characters long, with at least one upper-case, at least one digit, and at least one special character.

The -v option does the directory “sharing” between your machine the Docker image, and it works in both directions; your directory will be mapped to the ORCL directory on the Docker image. It is very important to use this option, so that the database is saved on your machine each time you may turn off the Oracle docker image or the Docker desktop altogether.

The -p option indicates that the container’s port 1521 (the one usually used by the Oracle server) is mapped to the same port 1521 of the Docker (the one we will use); same for 5500.

Connecting to the Oracle server is done via an Oracle client, such as SQL Developer, as instructed in the Docker setup document.

You can launch SQL Developer and connect to the Oracle server by clicking on the **+** icon, then filling the connection form as follows (using your chosen password):



If everything went well you can connect and run a test query for asking the time, such as

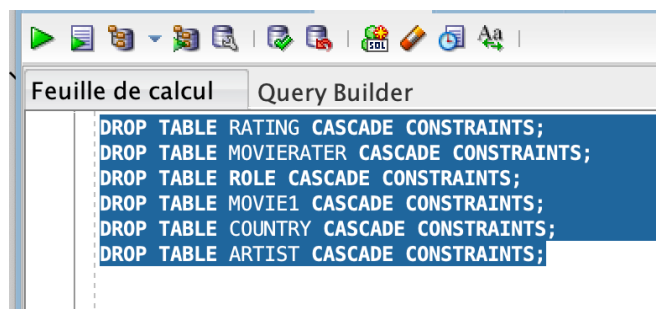
```
SELECT SYSDATE FROM DUAL;
```

Alternatively, instead of SQL Developer you can use the command line Oracle client SQL*Plus.

The database: creating the tables & indexes, populating the database

Follow in SQL Developer the following steps:

Step 1: **Execute all the commands from schema_stage1.sql**, as follows: copy/paste and execute in SQL Developer each block of commands delimited by (----), one block at a time, and observe whether all is ok. You can simply select in SQL Developer the block of commands you want to execute and click on the large green rectangle button (as illustrated below); in this way, all the selected commands will be executed one after the other. Note: only the drop commands, at a first execution, should raise errors. This stage of execution should be rather fast, creating the tables.



Step 1': Before Step 2 (inserting the data), test that the -v option worked for you. Just create the tables (Step 1), stop and restart the Oracle docker image, and see if the tables are still there after restarting. That is, if the database has been saved and then reloaded from the directory you indicated with the -v option. Only when this is the case proceed to Step 2.

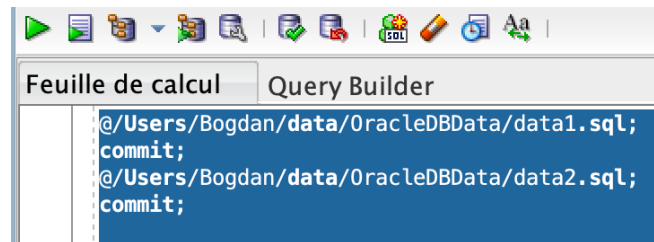
Step 2: Execute the following commands in SQL Developer, which will populate the tables we created at the previous stage. **This stage is slow, may take hours (on my machine it takes roughly 12 hours),** make sure you disable sleep mode on your computer and that it is not running on battery but on wall electricity; **the advice is to run this stage overnight.**

```
@/Users/Bogdan/data/OracleDBData/data1.sql;
```

```
commit;
```

```
@/Users/Bogdan/data/OracleDBData/data2.sql;
```

```
commit;
```



Note: Instead of `/Users/Bogdan/data/OracleDBData/` you must put the path where you saved the `data1.sql` and `data2.sql` scripts.

Note: these two files are large, do not try to open them in a text editor (it will likely crash), you can take a peek and see a few lines from them with the “more” or “tail” commands instead.

Step 3: Like at stage 1, execute all the commands from `schema_stage2.sql`: copy/paste and execute in SQL Developer each block of commands delimited by (---), one block at a time, and observe whether all is ok. Some of the steps here may take a little while (few minutes), be patient. **You must execute all the commands, no exceptions.**

The schema of the database we will use is the following:

ARTIST (IDARTIST, LASTNAME, FIRSTNAME, DOB)

COUNTRY (CODE, NAME, LANGUAGE)

MOVIE (IDMOVIE, TITLE, YEAR, IDMES, GENRE, SUMMARY, CODECOUNTRY)

- IDMES is the identifier of the movie director
- We will have four movie tables, called `Moviei`, for $i=1,2,3,4$

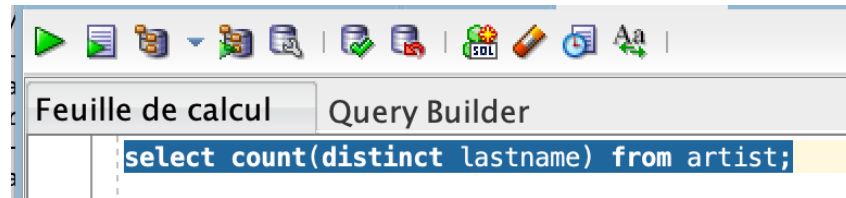
ROLE (IDMOVIE, IDARTIST, ROLENAME)

MOVIERATER (EMAIL, LASTNAME, FIRSTNAME, PROVINCE)

RATING (IDMOVIE, EMAIL, RATE)

Oracle's AUTOTRACE tool

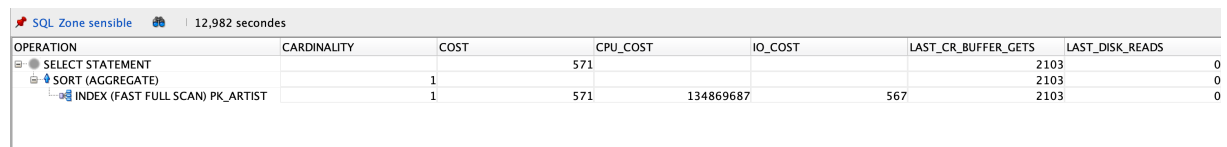
We can observe the execution plan Oracle chooses for a given query by using the menu AUTOTRACE, by pressing F6. For that, you must select with the mouse the query to be executed in autotrace mode



And then hit the AUTOTRACE button below (or F6):



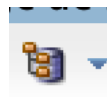
The plan that was actually executed (the trace thereof) will be displayed as follows:



OPERATION	CARDINALITY	COST	CPU_COST	IO_COST	LAST_CR_BUFFER_GETS	LAST_DISK_READS
SELECT STATEMENT			571		2103	0
SORT (AGGREGATE)	1				2103	0
INDEX (FAST FULL SCAN) PK_ARTIST	1		571	134869687	567	2103

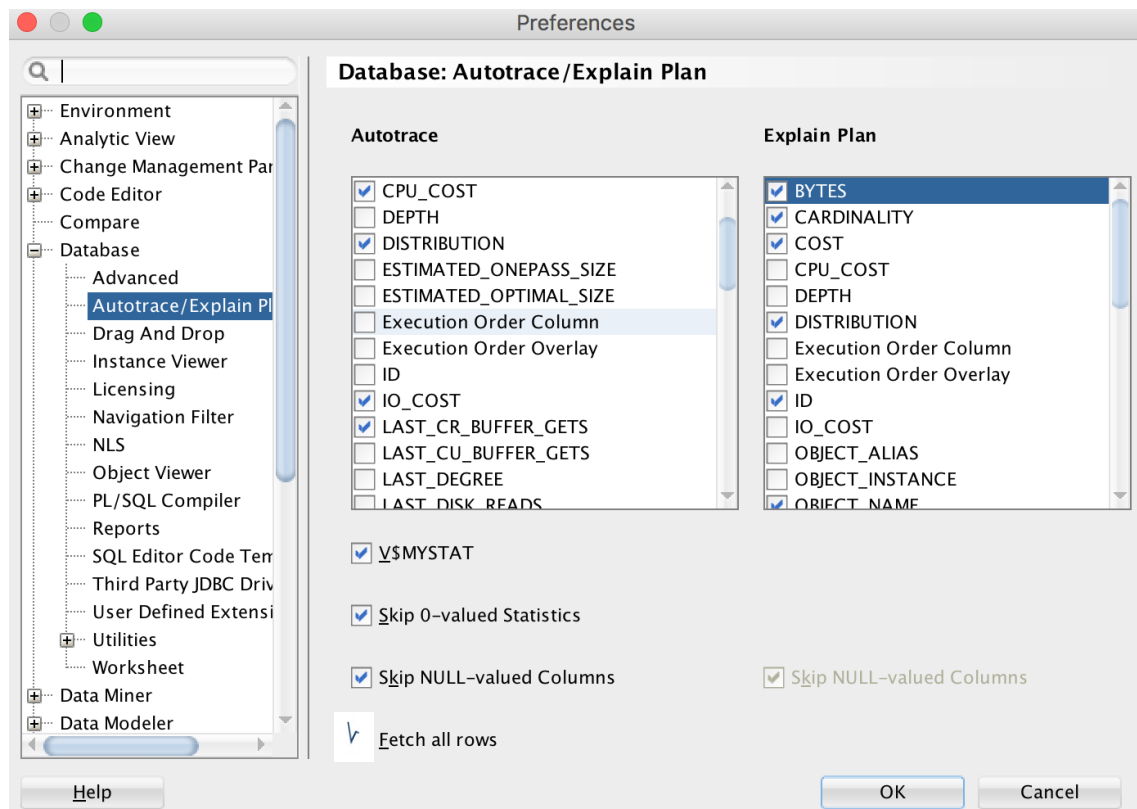
There are many available statistics we can see along with a plan (in the trace). We will focus mainly on **LAST_CR_BUFFER_GETS** (number of pages from the RAM buffer) and **LAST_DISK_READS** (number of pages read from disk).

Similarly, you can EXPLAIN the plan that Oracle intends to execute (before execution), by selecting with the mouse the query to be explained and then hitting the button below (or F10):



So the difference between **AUTOTRACE** and **EXPLAIN** is that the former **runs the query** and shows the executed plan and its main statistics, while the latter shows an estimated best plan and its estimated statistics **without running the query**.

Note: what is displayed in the trace of the execution plan, by pressing F6, is configurable in SQL Developer / Preferences / Database / Autotrace Explain Plan (see screenshot below).



Note: It is important to make sure that in SQL Developer -> Preferences->Database -> Autotrace Explain Plan the option **Fetch All Rows** is **checked**. This is an option that, if not checked, allows query processing to be faster. In some cases, it is easier to fetch only a certain number of results (for instance if no order or grouping is required, etc), and only later, if necessary (if the user scrolls down on the result), other results (all of them if necessary) will be fetched; we do not want this behavior. ✓

Here are some operations and options in Oracle query execution plans (list by no means exhaustive).

OPERATORS	OPTION	MEANING
AGGREGATE	GROUP BY	Computing a one-line result by grouping and aggregating
AND-EQUAL		An operation that has as input a set of rowIds and returns their intersection (used in accesses based on an index)
COUNTING		Counting the number of lines
FILTER		Applies a filter on a set of rows.
INDEX	UNIQUE SCAN	Finding one rowId in an index.
INDEX	RANGE SCAN	Finding one or several rowIds in an index.
MERGE JOIN		Doing a merge join.
NESTED LOOPS		Doing an index nested loops join.
SORT	UNIQUE	Sorting for duplicate elimination
SORT	GROUP BY	Sorting for grouping
SORT	JOIN	Sorting for joining (merge-join)
SORT	ORDER BY	Sorting for ORDER BY
TABLE ACCESS	FULL	Getting all the rows of a table
TABLE ACCESS	CLUSTER	Getting rows by a search key in a clustered index
TABLE ACCESS	BY ROW ID	Getting rows by rowId

Note: rowId and recordId are synonyms.

Oracle implements 3 join algorithms: Nested Loops Join (**when there is at least one index**), Sort-Merge Join and Hash Join when there is no index.

Environment – Cache and page size

We will work with a buffer cache (RAM) of limited size (on purpose, in order to make Oracle's optimizer more "creative"), for that you need to execute the following command:

```
alter system set db_cache_size = 400M;
```

Note: the cache size by default is already 400M, so no need to do this systematically.

You can first test if this is indeed the case by

```
show parameter db_cache_size;
```

The page size is 8KB. You can obtain this value (8KB is the default) by

```
show parameter db_block_size ;
```

SECTION 1

The goal of this first section is to discover and analyze the database that will be used for studying query optimization in Oracle. You will have to gather the necessary information for a good understanding of the execution plans and related statistics.

Bitmap indexes:

A few words on *bitmap indexes*: remember an index provides pointers to the rows in a table that contain a given key value. We saw that a B+-tree index stores a rowId or a list of rowIds for each key, corresponding to the rows (i.e., records) with that key value.

In a bitmap index, a bitmap for each key value replaces the list of rowIds. *Each bit* in the bitmap corresponds to a *rowId*, and if the bit is set, it means that the row with the corresponding rowId contains the key value.

A *mapping function* is necessary to convert the bit positions to actual rowId in a heap file; in this way, that the bitmap index provides the same functionality as a regular index.

By this, bitmap indexes can be very effective for queries that contain *multiple conditions* in the `WHERE` clause, since they can *be tested by logical and/or on bitmaps*: in this way, rows that satisfy some, but not all, conditions are *filtered* out before the table itself is accessed.

Bitmap indexes are very efficient especially for low-cardinality attributes, where the ratio between the *number of distinct values* of the attribute and *the number of records* is very low (less than 1<%). Gender would be for example one such low-cardinality attribute in a table listing persons.

Question 1. Fill the following table:

Tables and indexes:

Table name	Nb of records	Nb of pages in heap file (approximative is fine)	Index (type, field/column, alternative 1/2/3, clustered / unclustered)
ARTIST			
MOVIE1			
MOVIE2			
MOVIE3			
MOVIE4			
MOVIERATER			
RATING			
COUNTRY			
ROLE			

Question 2. Fill the following tables:

Attribute name	Cardinality (number of distinct values)	Minimal value	Maximal value
MOVIE1.IDMOVIE			
MOVIE1.YEAR			
MOVIE1.CODECOUNTRY			

SECTION 2 – INSTRUCTIONS

Before executing queries, make sure the following system command is executed (note: after each restart of the Oracle Docker image you must re-execute this command):

```
ALTER SYSTEM SET optimizer_features_enable = "11.2.0.4";
```

(you are encouraged to figure out what this might achieve)

For each SQL query to be executed, here are the instructions that will be used in order to understand its optimization and execution:

- **RULE:** Optimization mode without statistics over the tables
ALTER SESSION SET OPTIMIZER_MODE=RULE;
- **CHOOSE:** optimization mode with statistics, allows to obtain a more efficient plan
ALTER SESSION SET OPTIMIZER_MODE=CHOOSE;
(by default, we will use CHOOSE, and only CHOOSE unless RULE is also requested explicitly)
- **CACHE:** You must flush (empty) the RAM cache before each execution.

```
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

Note: most likely, with the Docker Oracle image, the cache flush may not work for you; you can check if this is the case or not by running the same (any) query twice, and in between doing a buffer flush. If at the second query run the number of disk reads is 0, it means the cache has not been flushed... In this case, the alternative to cache flush (same effect) is to **restart the Oracle database systematically** (each time), between any two queries on the same table(s), using the maintenance scripts: /home/oracle/shutDown.sh and /home/oracle/startUp.sh. This will take approx. 10 seconds each time but **is necessary and important to do**.

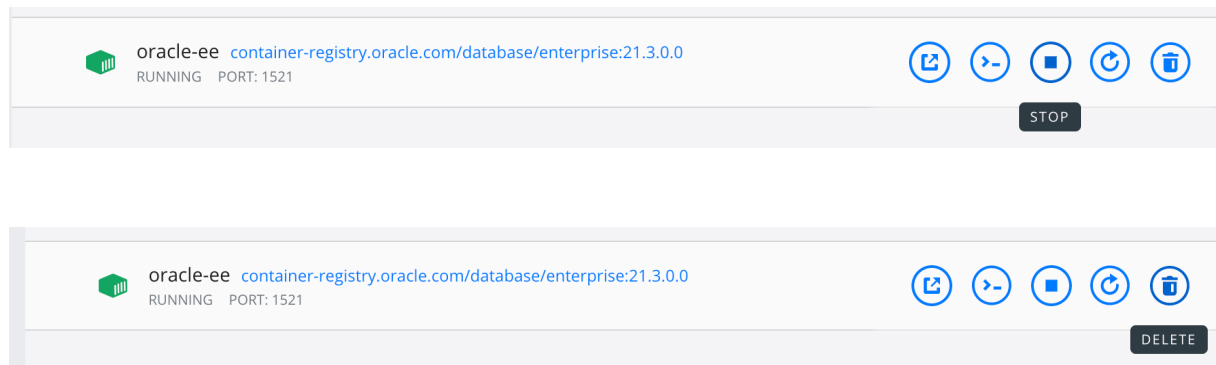
For that, just login to the cmd terminal of the Oracle Docker container:

```
docker exec -it oracle-ee /bin/bash
```

There, you can restart just the database system within the container, using the shutdown/startup scripts:

- ./shutDown.sh immediate (use mode immediate)
- ./startUp.sh

Another alternative (albeit slow) would be to **shutdown & restart the docker container** entirely, again systematically (each time). This may take approx. 2-3 minutes each time. If you go for this alternative, the advice is to run a query, take a snapshot of the execution details, then restart. While the container restarts, you can use the snapshot to draft your answer; in this way little time will be wasted by the systematic restarting. You can restart the docker container from the Docker Desktop main menu, by selecting Dashboard and then stop, then garbage collecting the Oracle container.



- MOVIEi: A query on table MOVIE can be executed on one of the MOVIE copies (1-4). Each time, which MOVIE copy to use will be indicated in text.

Here is one overall example:

```
ALTER SESSION SET OPTIMIZER_MODE=RULE;
ALTER SYSTEM FLUSH BUFFER_CACHE;
SELECT title, genre FROM MOVIE1 m WHERE IDMOVIE=50273 ;
```

```
ALTER SESSION SET OPTIMIZER_MODE=CHOOSE;
ALTER SYSTEM FLUSH BUFFER_CACHE;
SELECT title, genre FROM MOVIE1 m WHERE IDMOVIE=50273 ;
```


For each query, you will complete a form like the following, by looking at the LAST_CR_BUFFER_GETS (corresponding roughly to the number of pages from the RAM buffer cache, a.k.a. “consistent gets” or “logical reads”) and LAST_DISK_READS (corresponding roughly to the number of pages read from disk, a.k.a. “physical reads”):

Here is an answer form example (numbers are entirely made up the sake of the example):

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
MOVIE1	RULE	INDEX RANGE SCAN + ACCESS ROWIDS	123456789	123456789
MOVIE1	CHOOSE	TABLE ACCESS FULL	123456789	123456789

Besides explaining what the query does / computes, explain also by one phrase, with your own words, for each query, what can be observed in the execution plan.

Your query explanation: *This query gives the title and genre of movie 50273.*

Optional: you can also describe the result of the query.

Your plan explanation: *in the form above, in RULE mode we use the index, since it is there. However, in CHOOSE mode the optimizer understands that using the index would be too expensive, thus a full scan is done instead.*

SECTION 3 – SINGLE-TABLE QUERIES

Question 3.1 On MOVIE1 and MOVIE4 :

SELECT TITLE FROM MOVIEi WHERE IDMOVIE=50273 ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
MOVIE1	CHOOSE			
MOVIE4	CHOOSE			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 3.2 On MOVIE2 and MOVIE4 :

SELECT COUNT(*) FROM MOVIEi WHERE IDMOVIE BETWEEN 55273 AND 60000 ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
MOVIE2	CHOOSE			
MOVIE4	CHOOSE			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 3.3 On MOVIE2, with RULE, CHOOSE :

SELECT TITLE FROM MOVIEi WHERE YEAR=1999 ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>RULE</i>			
<i>MOVIE2</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 3.4 On MOVIE2, MOVIE3, MOVIE4 :

SELECT COUNT(*) FROM MOVIEi WHERE YEAR=1999 ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>CHOOSE</i>			
<i>MOVIE3</i>	<i>CHOOSE</i>			
<i>MOVIE4</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 3.5 On MOVIE2, MOVIE3, with RULE, CHOOSE :

SELECT TITLE FROM MOVIEi WHERE CODECOUNTRY='aaej' ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>RULE</i>			
<i>MOVIE2</i>	<i>CHOOSE</i>			
<i>MOVIE3</i>	<i>RULE</i>			
<i>MOVIE3</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

SECTION 4 – MULTI-TABLE (JOIN) QUERIES

Question 4.1 On MOVIE2 and MOVIE3, with RULE, CHOOSE :

SELECT TITLE, LASTNAME FROM MOVIEi M, ARTIST A WHERE CODECOUNTRY='aaej' AND M.IDMES=A.IDARTIST ;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>CHOOSE</i>			
<i>MOVIE2</i>	<i>RULE</i>			
<i>MOVIE3</i>	<i>CHOOSE</i>			

<i>MOVIE3</i>	<i>RULE</i>			
---------------	-------------	--	--	--

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.2 On MOVIE2 :

```
SELECT lastname, title
FROM MOVIEi M, MOVIERATER MR, RATING R
WHERE R.email='aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' AND
MR.email= R.email AND M.IDMOVIE=R.IDMOVIE;
```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.3 On MOVIE1 :

```
SELECT lastname, title
FROM MOVIEi M, MOVIERATER MR, RATING R
WHERE MR.email= R.email AND M.IDMOVIE=R.IDMOVIE AND R.IDMOVIE=
367856;
```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE1</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.4 On MOVIE3 :

```
SELECT LASTNAME, COUNT(*)
FROM MOVIEi M, ARTIST A
WHERE YEAR=1999 AND M.IDMES=A.IDARTIST
GROUP BY A.LASTNAME;
```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE3</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.5 On MOVIE3 :

```
SELECT LASTNAME, COUNT(*)
```

```

FROM ARTIST
WHERE IDARTIST IN (SELECT DISTINCT IDMES
                   FROM MOVIEi
                   WHERE YEAR=1999)
GROUP BY LASTNAME;

```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
MOVIE3	CHOOSE			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.6 On Movie3 :

```

SELECT COUNT(*)
FROM MOVIEi M, ROLE R, ARTIST A
WHERE M.IDMOVIE=R.IDMOVIE AND R.IDACTOR=A.IDARTIST AND
      M.IDMES=A.IDARTIST AND CODECOUNTRY='aaej' AND YEAR=1999;

```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
MOVIE3	CHOOSE			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.7 On MOVIE2 and MOVIE4 :

```

SELECT TITLE
FROM MOVIEi M, RATING R
WHERE M.IDMOVIE = R.IDMOVIE
GROUP BY TITLE
HAVING AVG(RATE) > 15;

```

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>CHOOSE</i>			
<i>MOVIE4</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan:

Question 4.8 On MOVIE2 and MOVIE4 :

```

SELECT TITLE
FROM MOVIEi M, (SELECT IDMOVIE, AVG(RATE) as AVG_RATE

```

FROM RATING
GROUP BY IDMOVIE) N
WHERE M.IDMOVIE = N.IDMOVIE AND AVG_RATE > 15;

TABLE	MODE	OPERATORS	BUFFER READS	DISK READS
<i>MOVIE2</i>	<i>CHOOSE</i>			
<i>MOVIE4</i>	<i>CHOOSE</i>			

Your explanation of the SQL query (in plain English) and of the execution plan: