SQL lab #2

DBM1

October 20, 2015

Exercise 1 Query processing with the IMDB database

You will use the same database than for the previous lab, containing with real-world data coming from IMDB, and SQLDeveloper as an Oracle client.

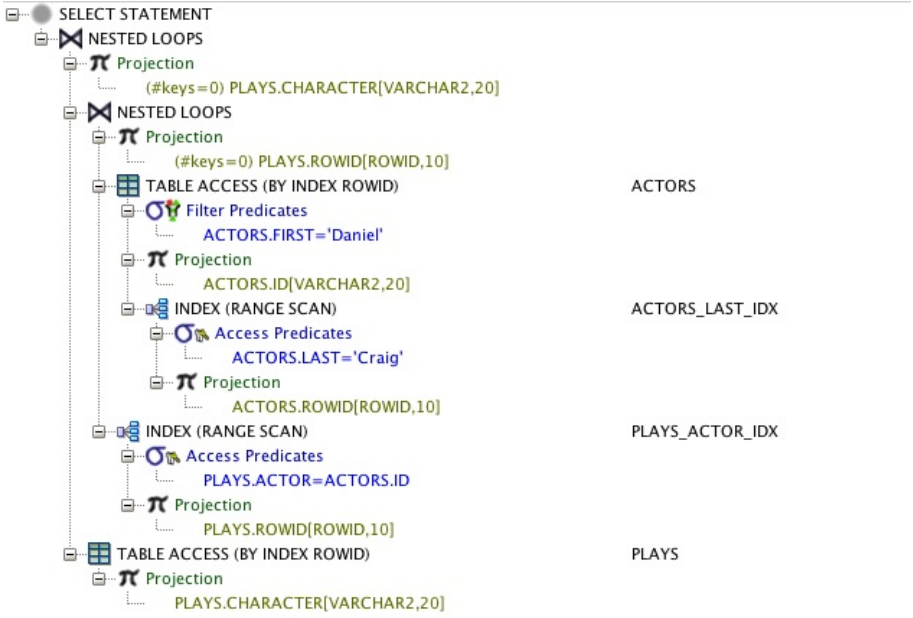
Throughout this part, you can use the following online resources to help you answer the questions:

- `http://use-the-index-luke.com/welcome`: A website explaining the internals of query processing in a simple way (available in english, french, german and chinese).
- `https://docs.oracle.com/database/121/TGSQL/toc.htm`: The Oracle documentation about query processing and optimization.

1. Let us start with an example. Consider the following query listing characters played by Daniel Craig:

```
select plays.character
from plays
join actors on plays.actor = actors.id
  and actors.first = 'Daniel' and actors.last = 'Craig'
```

We get the following execution plan:



We can now trace by hand what is happening inside each operator, to better understand the physical implementation of the plan. We start with the innermost operation, here the index range scan on ACTORS_LAST_IDX. After that step, we have the following data:

```
Craig, actors_rowid
Craig, actors_rowid
...
```

You have several actors whose last name is "Craig". Because you are using the index, you only retrieve the indexed value, along with a rowid (which is a unique id associated to each tuple, allowing to retrieve it efficiently). Next, by using this index, a table access occurs on ACTORS. After that step, we have the following data:

```
actors_rowid
actors_rowid
...
```

We get the rowids of the actors named Daniel Craig (still several of them). Internally, the first name has been checked but it is not returned, because of the projection.
Then, there is an index range scan on PLAYS_ACTOR_IDX within the first nested loop. Each previously retrieved actor rowid will be associated with a rowid inside the PLAYS table. After that step, we have the following data:

```
actors_rowid, plays_rowid
actors_rowid, plays_rowid
...
```

Finally, the next step is to pull out the data from the PLAYS relation. This occurs inside the outermost nested loop. After that step, we have the following data, which is the actual query's result:

```
7de4122fa443f3a994fd
74b9d3d780025c6b17b7
...
```

Trace the execution of the following queries (from previous lab):

a) Give the female actors whose first name has five letters ordered by their last name.

b) Give the list of non-suspended episodes (title, season, episode number) of your very own favorite series, ordered by season and episode.

2. Now, our objective is to get an overview of the most common physical operators. To do that, you will re-run queries of previous lab's exercise 2 and examine the execution plan of each. At the end, you should be able to explain, for each the following notions, its basic behavior and under which conditions it is likely to be used by the optimizer. Do not forget to use online resources at your disposal!

   a) Index unique, index range scan

   b) Table access full, table access by index rowid

   c) Nested loops join, hash join, merge join

   d) Access predicates and filter predicates

3. Consider now the very hypothetical case where you could be interested in the pairs of films in which Nicolas Cage plays and produced the same year. One solution is use six tables joined successively:

```
select s1.title, s1.year, s2.title, s2.year
from shows s1
join plays p1 on p1.show = s1.id
join actors a1 on a1.id = p1.actor and a1.first = 'Nicolas'
  and a1.last = 'Cage'
cross join shows s2
```

```
join plays p2 on p2.show = s2.id
join actors a2 on a2.id = p2.actor and a2.first = 'Nicolas'
  and a2.last = 'Cage'
where s1.kind <> 'episod' and s2.kind <> 'episod'
  and s1.id <> s2.id and s1.year = s2.year;
```

Another solution involves a subquery repeated for the two show table:

```
select s1.title, s1.year, s2.title, s2.year
from shows s1
cross join shows s2
where
  s1.id in (
    select show
    from plays
    join actors on plays.actor = actors.id
      and actors.first = 'Nicolas' and actors.last = 'Cage'
  )
  and s2.id in (
    select show
    from plays
    join actors on plays.actor = actors.id
      and actors.first = 'Nicolas' and actors.last = 'Cage'
  )
  and s1.kind <> 'episod' and s2.kind <> 'episod'
  and s1.id <> s2.id and s1.year = s2.year;
```

a) Explain the general idea of the query plan of each query.
b) Explain which one performs better and why. Support your answer with the cardinality and costs estimation provided by the query plan.

Exercise 2 Replicas placement

This part focuses on the reading and understanding of a research paper named "Where in the World is My Data?" (available on DBM1's webpage), published in VLDB, a top-conference in databases, in 2011. Start by reading the paper (you do not have to read the appendix) and then answer the following questions.

1. What is the goal of the master copy of each record in PNUTS?
2. Explain the difference between replication and caching with your own words (cf. Related work). You can use an example.
3. Why did the authors choose not to use the data access patterns to achieve dynamic placement of replicas? Does it seem justified given the experiments' results?
4. Why does the "Full" placement performs better than any other w.r.t. latency (Figure 4 and 7)?
5. Why does the "Dynamic" placement performs sometimes better than the "Bandwidth Optimal" placement (presented as a lower bound) w.r.t. to bandwidth (Figure 3 and 5)?
6. How do these solutions relate to horizontal fragmentation? vertical fragmentation? If yes, give the properties that are satisfied and those who are not.