Assignment / Explore Query Planning

Vasumathi Narayanan

```
library(RSQLite)
library(sqldf)
```

```
## Loading required package: gsubfn
```

```
## Loading required package: proto
```

```
dbfile = "sakila.db"
# if database file already exists, we connect to it, otherwise
# we create a new database
dbcon <- dbConnect(RSQLite::SQLite(), dbfile)
```

```
library(RMySQL)
```

```
## Loading required package: DBI
```

```
##
## Attaching package: 'RMySQL'
```

```
## The following object is masked from 'package:RSQLite':
##
##     isIdCurrent
```

```
sakilaDBCon = dbConnect(RMySQL::MySQL(),
                            dbname='sakila',
                            host='localhost',
                            port=3306,
                            user='ecommerceapp',
                            password='ecommerceapp')
```

Question 1.

Ensuring that no user-defined indexes exist (delete all user-defined indexes, if there are any), find the number of films per category. The query should return the category name and the number of films in each category. Show us the code that determines if there are any indexes and the code to delete them if there are any.

```
# query to check for user-defined indexes
dbExecute(dbcon,"DROP INDEX IF EXISTS TitleIndex")
```

```
## [1] 0
```

```
query_indexes <- "
  SELECT *
  FROM sqlite_master
  WHERE type = 'index' AND name NOT LIKE 'sqlite_%';
"

# execute the query
indexes <- dbGetQuery(dbcon, query_indexes)

# check if any results were returned
if(nrow(indexes) > 0) {
  # generate SQL statements to drop indexes
  drop_indexes <- paste0("DROP INDEX ", indexes$name, ";")
  # execute the SQL statements to drop indexes
  for (i in 1:length(drop_indexes)) {
    dbExecute(dbcon, drop_indexes[i])
  }
}

# query to find number of films per category
query_films_per_category <- "
 SELECT CATEGORY.NAME,COUNT(FILM.FILM_ID) FROM CATEGORY INNER JOIN FILM_CATEGORY ON CATEGORY.CATEGORY_I
"

start_time_sqlite <- Sys.time()
# execute the query and store the results in a data frame
films_per_category <- dbGetQuery(dbcon, query_films_per_category)
end_time_sqlite <- Sys.time()

# print the results
print(films_per_category)
```

```
##           name COUNT(FILM.FILM_ID)
## 1       Action                  64
## 2    Animation                  66
## 3     Children                  60
## 4     Classics                  57
## 5       Comedy                  58
## 6  Documentary                  68
## 7        Drama                  62
## 8       Family                  69
## 9      Foreign                  73
## 10       Games                  61
## 11      Horror                  56
## 12       Music                  51
## 13         New                  63
## 14      Sci-Fi                  61
## 15      Sports                  74
## 16      Travel                  57
```

to check if indices are deleted

```
# query to check for user-defined indexes
query_indexes <- "
  SELECT *
  FROM sqlite_master
  WHERE type = 'index' AND name NOT LIKE 'sqlite_%';
"

# execute the query
indexes <- dbGetQuery(dbcon, query_indexes)
indexes
```

```
## [1] type     name     tbl_name rootpage sql
## <0 rows> (or 0-length row.names)
```

Question 2.

Ensuring that no user-defined indexes exist (delete all user-defined indexes, if there are any), execute the same query (same SQL) as in (1) but against the MySQL database. Make sure you reuse the same SQL query string as in (1).

To delete User-defined index

```
index_exists <- dbGetQuery(sakilaDBCon, "SELECT COUNT(*) as index_count FROM information_schema.statisti
if (index_exists$index_count > 0)
{
  dbExecute(sakilaDBCon, "DROP INDEX `TitleIndex` ON `film`;")
}
```

To delete pre-defined indices

```
indexes <- dbGetQuery(sakilaDBCon, "SELECT DISTINCT(TABLE_NAME), INDEX_NAME FROM INFORMATION_SCHEMA.STA
# Drop all user-defined indexes
if (nrow(indexes) > 0) {
  for (i in 1:nrow(indexes)) {
    index_name <- indexes$INDEX_NAME[i]
    table_name <- indexes$TABLE_NAME[i]
    dbExecute(sakilaDBCon, paste0("DROP INDEX ", index_name, " ON ", table_name))
  }
}
```

```
start_time_mysql <- Sys.time()
# execute the query and store the results in a data frame
films_per_category <- dbGetQuery(sakilaDBCon, query_films_per_category)
end_time_mysql <- Sys.time()
```

```
# print the results
print(films_per_category)
```

```
##           NAME COUNT(FILM.FILM_ID)
## 1       Action                  64
## 2    Animation                  66
## 3     Children                  60
## 4     Classics                  57
## 5       Comedy                  58
## 6  Documentary                  68
## 7        Drama                  62
## 8       Family                  69
## 9      Foreign                  73
## 10       Games                  61
## 11      Horror                  56
## 12       Music                  51
## 13         New                  63
## 14      Sci-Fi                  61
## 15      Sports                  74
## 16      Travel                  57
```

Question 3.

Find out how to get the query plans for SQLite and MySQL and then display the query plans for each of the query executions in (1) and (2).

```
dbGetQuery(dbcon, paste("EXPLAIN QUERY PLAN ", query_films_per_category))
```

```
##   id parent notused
## 1  8      0       0
## 2 10      0       0
## 3 13      0       0
## 4 16      0       0
##                                                                detail
## 1 SCAN FILM_CATEGORY USING COVERING INDEX sqlite_autoindex_film_category_1
## 2                     SEARCH CATEGORY USING INTEGER PRIMARY KEY (rowid=?)
## 3                         SEARCH FILM USING INTEGER PRIMARY KEY (rowid=?)
## 4                                          USE TEMP B-TREE FOR GROUP BY
```

```
dbGetQuery(sakilaDBCon, paste("EXPLAIN ", query_films_per_category))
```

```
##   id select_type         table partitions   type
## 1  1      SIMPLE      CATEGORY       <NA>  index
## 2  1      SIMPLE FILM_CATEGORY       <NA>    ref
## 3  1      SIMPLE          FILM       <NA> eq_ref
##                         possible_keys                      key key_len
## 1                            PRIMARY                  PRIMARY       1
## 2 PRIMARY,fk_film_category_category fk_film_category_category       1
## 3                            PRIMARY                  PRIMARY       2
##                            ref rows filtered       Extra
```

```
## 1                          <NA>   16      100        <NA>
## 2  sakila.CATEGORY.category_id   62      100 Using index
## 3 sakila.FILM_CATEGORY.film_id    1      100 Using index
```

#Execution time comparison for sqlite and MYSQL

```
# Calculate the execution time for SQLite and MySQL
execution_time_sqlite <- end_time_sqlite - start_time_sqlite
execution_time_mysql <- end_time_mysql - start_time_mysql

# Display the execution times for SQLite and MySQL
print(paste("Execution time in SQLite:", execution_time_sqlite))
```

```
## [1] "Execution time in SQLite: 0.00353598594665527"
```

```
print(paste("Execution time in MySQL:", execution_time_mysql))
```

```
## [1] "Execution time in MySQL: 0.00909900665283203"
```

Question 4.

Comment on the differences between the query plans? Are they the same? How do they differ? Why do you think they differ? Do both take the same amount of time?

Access method: The access method used by the database engine to retrieve data from the tables can be different. In SQLite, the query plan may show the use of an "SCAN, SEARCH and GROUP BY a Binary Tree" to retrieve the required data, while in MySQL, it may use a "Full Table Scan (as it shows 16 rows)" or "Using index" strategy.

Execution Time: Execution time for SQLite is more than the Execution time for MySQL to execute the query_films_per_category. This will also depend on the underlying architecture the servers are running on and will change everytime. we write with different query (as in without JOIN)

Index usage: The index usage can also be different. SQLite may use different indexes (FILM_CATEGORY,FILM, CATEGORY) for each table in the join, while MySQL may use one (PRIMARY KEY) or more indexes for the join.

Question 5.

Write a SQL query against the SQLite database that returns the title, language and length of the film with the title "ZORRO ARK".

```
dbExecute(dbcon,"DROP INDEX IF EXISTS TitleIndex")
```

```
## [1] 0
```

```
getQuery <-"SELECT film.title, language.name, film.length from film inner join language on language.lang
start_time_getquery <- Sys.time()
resultgetquery <- dbGetQuery(dbcon, getQuery)
end_time_getquery <- Sys.time()
print(resultgetquery)
```

```
##       title    name length
## 1 ZORRO ARK English     50
```

Question 6.

For the query in (5), display the query plan.

```
dbGetQuery(dbcon, paste("EXPLAIN QUERY PLAN ", getQuery))
```

```
##   id parent notused                                           detail
## 1  3      0       0                                        SCAN film
## 2  7      0       0 SEARCH language USING INTEGER PRIMARY KEY (rowid=?)
```

Question 7. (Index Creation)

In the SQLite database, create a user-defined index called "TitleIndex" on the column TITLE in the table FILM

```
dbExecute(dbcon,"CREATE INDEX TitleIndex on film(title)")
```

```
## [1] 0
```

Question 8.

Re-run the query from (5) now that you have an index and display the query plan.

```
start_time_getquery_index <- Sys.time()
resultgetquery <- dbGetQuery(dbcon, getQuery)
end_time_getquery_index <- Sys.time()
print(resultgetquery)
```

```
##       title    name length
## 1 ZORRO ARK English     50
```

```
dbGetQuery(dbcon, paste("EXPLAIN QUERY PLAN ", getQuery))
```

```
##   id parent notused                                             detail
## 1  4      0       0          SEARCH film USING INDEX TitleIndex (title=?)
## 2  9      0       0 SEARCH language USING INTEGER PRIMARY KEY (rowid=?)
```

```
exeTime_without_index <-end_time_getquery - start_time_getquery
exeTime_with_index <-end_time_getquery_index  - start_time_getquery_index
print(paste("Execution time in SQLite without index: ", exeTime_without_index))
```

```
## [1] "Execution time in SQLite without index:  0.00249695777893066"
```

```
print(paste("Execution time in SQLite with inidex: ", exeTime_with_index))
```

```
## [1] "Execution time in SQLite with inidex:  0.00293493270874023"
```

Question 9

Are the query plans the same in (6) and (8)? What are the differences? Is there a difference in execution time? How do you know from the query plan whether it uses an index or not?

No, the films were scanned using the primary key, but after the index was created on film table, the query plan shows that TitleIndex is used to retrieve the film details. We can see the difference in the execution time of (6) and (8). The execution time of query without index (6) takes a bit less seconds to run when compared with (8). Inner Joins on tables takes time as it will create joined tables for storing intermediate result sets (rows), and applying index over it, might also add up to the execution time. So thats why (8) takes more time to run than (6).

Question 10

Write a SQL query against the SQLite database that returns the title, language and length of all films with the word "GOLD" with any capitalization in its name, i.e., it should return "Gold Finger", "GOLD FINGER", "THE GOLD FINGER", "Pure GOLD" (these are not actual titles).

```
get_name_like_gold<- "SELECT film.title, language.name AS language, film.length
FROM film
JOIN language ON film.language_id = language.language_id
WHERE film.title LIKE '%gold%' ;"
result <- dbGetQuery(dbcon, get_name_like_gold)
result
```

```
##                    title language length
## 1          ACE GOLDFINGER  English     48
## 2   BREAKFAST GOLDFINGER  English    123
## 3              GOLD RIVER  English    154
## 4 GOLDFINGER SENSIBILITY  English     93
## 5         GOLDMINE TYCOON  English    153
## 6              OSCAR GOLD  English    115
## 7   SILVERADO GOLDFINGER  English     74
## 8              SWARM GOLD  English    123
```

Question 11

Get the query plan for (10). Does it use the index you created? If not, why do you think it didn't?

```
dbGetQuery(dbcon, paste("EXPLAIN QUERY PLAN ", get_name_like_gold))
```

```
##   id parent notused                                              detail
## 1  3      0       0                                           SCAN film
## 2  8      0       0 SEARCH language USING INTEGER PRIMARY KEY (rowid=?)
```

Reasons why the index was not used for the Above query:

The cardinality of the title column: If the title column has a low cardinality (i.e., a small number of unique values - 8 values here), the query planner may choose not to use the index because it may not provide a significant benefit. In this case, a full table scan may be faster than using the index.

The query predicate: If the query predicate (i.e., the WHERE clause) is not selective enough, the query planner may choose not to use the index. For example, if the predicate matches a large proportion of the

rows in the table, a full table scan may be faster than using the index. The like pattern was not selective as it used %gold% which is different from "gold" so a full scan was perfromed.

```
dbDisconnect(dbcon)
dbDisconnect(sakilaDBCon)
```

```
## [1] TRUE
```