Assignment / Explore Query Planning

Bhavitha kandru

Spring 2024

Load library

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

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

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

```
## Warning in doTryCatch(return(expr), name, parentenv, handler): unable to load shared object '/Library
##   dlopen(/Library/Frameworks/R.framework/Resources/modules//R_X11.so, 0x0006): Library not loaded: /
##   Referenced from: <B3716E5A-BF4D-3CA3-B8EB-89643DB72A04> /Library/Frameworks/R.framework/Versions/4
##   Reason: tried: '/opt/X11/lib/libSM.6.dylib' (no such file), '/System/Volumes/Preboot/Cryptexes/OS/
```

```
## tcltk DLL is linked to '/opt/X11/lib/libX11.6.dylib'
```

```
## Could not load tcltk.  Will use slower R code instead.
```

```
library(RMySQL)
```

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

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

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

load database

```
con <- dbConnect(RSQLite::SQLite(), dbname="sakila.db")
```

Question 1:

Ensuring that no user-defined indexes exist (delete all user-defined indexes that you may have created, if there are any; remember that your program can be run more than once), find the number of films per language. The query should return the language name and the number of films in each language.

```r
# Get a list of all user-defined indexes (excluding auto-created ones for primary keys)
indexes <- dbGetQuery(con, "SELECT name FROM sqlite_master WHERE type = 'index' AND sql IS NOT NULL;")

# Drop each user-defined index
for (index_name in indexes$name) {
  dbExecute(con, paste0("DROP INDEX IF EXISTS ", index_name, ";"))
}


# Number of films:
films <- "
SELECT lang.name AS language_name, Count(film.film_id) AS number_of_films
FROM film
JOIN language lang ON film.language_id = lang.language_id
GROUP BY lang.name
ORDER BY number_of_films DESC;
"

# Execute the query
films_per_language <- dbGetQuery(con, films)

# View the result
print(films_per_language)
```

```
##   language_name number_of_films
## 1       English            1000
```

Question 2:

Find out how to get the query plans for SQLite then display the query plans for the query executions in (1).

```r
# Execute the EXPLAIN QUERY PLAN
query_plan <- dbGetQuery(con,paste("EXPLAIN QUERY PLAN",films))

# Display the query plan
print(query_plan)
```

```
##   id parent notused                                        detail
## 1  8      0       0                                     SCAN film
## 2 10      0       0 SEARCH lang USING INTEGER PRIMARY KEY (rowid=?)
## 3 13      0       0                    USE TEMP B-TREE FOR GROUP BY
## 4 50      0       0                    USE TEMP B-TREE FOR ORDER BY
```

Question 3:

Write a SQL query against the SQLite database that returns the title, category name and length of the film with the title “ZORRO ARK”.

```r
title <-"SELECT f.title, c.name AS category_name, f.length
FROM film f
JOIN film_category fc ON f.film_id = fc.film_id
```

```
JOIN category c ON fc.category_id = c.category_id
WHERE f.title = 'ZORRO ARK';
"
ark <- dbGetQuery(con,title)
print(ark)
```

```
##       title category_name length
## 1 ZORRO ARK        Comedy     50
```

Question 4:

For the query in (3), display the query plan.

```
# Execute the EXPLAIN QUERY PLAN
plan <- dbGetQuery(con,paste("EXPLAIN QUERY PLAN", title))

# Display the query plan
print(plan)
```

```
##   id parent notused
## 1  4      0       0
## 2  6      0       0
## 3  9      0       0
##                                                    detail
## 1 SCAN fc USING COVERING INDEX sqlite_autoindex_film_category_1
## 2                  SEARCH c USING INTEGER PRIMARY KEY (rowid=?)
## 3                  SEARCH f USING INTEGER PRIMARY KEY (rowid=?)
```

Question 5:

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

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

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

Question 6:

Re-run the query from (3) now that you have an index and display the query plan.

```
result <- dbGetQuery(con, title)
print(result)
```

```
##       title category_name length
## 1 ZORRO ARK        Comedy     50
```

Question 7:

Are the query plans the same in (4) and (6)? What are the differences? How do you know from the query plan whether it uses an index or not? Comment on the differences.

No the queries in question 4 and 6 are not same beacuse the query plans in questions 4 and 6 as the latter uses the TitleIndex for efficient data retrieval, resulting in the use of the index being explicitly mentioned in the plan. Typically, indexes improve query performance by reducing the need for full table scans, although the actual impact on execution time can vary based on query complexity and data size.

Question 8:

Measure the execution time of your queries (within R) for (3) and (6), i.e., the query with and without an index. Is there a difference in execution time? What is the difference? Comment on the differences.

Indexes can speed up query execution time by optimizing lookup efficiency, particularly for large datasets. If an indexed query takes longer, it could be due to overhead or specific query characteristics. In such cases, it is essential to examine query design and index utilization more closely.

Question 9:

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", "Goldmine", "Marigold", "The Baumgold Files", "GOLD FINGER", "THE GOLD FINGER", "Pure Gold", "goldfish" (these are not actual titles). But it should not return "G.O.L.D.", "Goolders", or "Gol Depression". In other words, look for the occurrence of the four character sequence [Gg][Oo][Ll][Dd]

```
names_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%' ;"
results <- dbGetQuery(con,names_like_gold)
print(results)
```

```
##                    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 10:

Get the query plan for (9). Does it use the index you created? If not, why do you think it didn't?

```
query_10<-dbGetQuery(con,paste("EXPLAIN QUERY PLAN",names_like_gold))
print(query_10)
```

```
##   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: 1. Low cardinality of title column (only 8 unique values). 2. Query predicate not selective enough. 3. Non-selective LIKE pattern ("%gold%") used in WHERE clause.

```
dbDisconnect(con)
```