Assignment / Explore Query Planning and Indexing

Agnibha Chatterjee

Fall Full 2023

Connecting to database

```
library(RSQLite)
library(DBI)

cwd <- getwd()
dbFile <- "sakila.db"
pathToDb <- paste0(cwd,  .Platform$file.sep, dbFile)
conn <- dbConnect(RSQLite::SQLite(), pathToDb)
```

Question 1

Dropping all user-defined indexes

```
sqlStmt <- "DROP INDEX IF EXISTS TitleIndex";
dbSendStatement(conn, sqlStmt)
```

```
## <SQLiteResult>
##   SQL  DROP INDEX IF EXISTS TitleIndex
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
query <- "SELECT language.name AS language, COUNT(film.film_id) AS no_of_films
          FROM film
          JOIN language ON language.language_id = film.language_id
          GROUP BY language.language_id;
          "
result <- dbGetQuery(conn, query)
```

```
## Warning: Closing open result set, pending rows
```

```
print(result)
```

```
##   language no_of_films
## 1  English        1000
```

Question 2

```
query <- "EXPLAIN QUERY PLAN
          SELECT language.name AS language, COUNT(film.film_id) AS no_of_films
          FROM film
          JOIN language ON language.language_id = film.language_id
          GROUP BY language.name;
          "
result <- dbGetQuery(conn, query)
print(result$detail)
```

```
## [1] "SCAN film"
## [2] "SEARCH language USING INTEGER PRIMARY KEY (rowid=?)"
## [3] "USE TEMP B-TREE FOR GROUP BY"
```

Question 3

```
query <- "SELECT film.title as film_name, category.name as category_name, film.length
          FROM film
          JOIN film_category ON film_category.film_id = film.film_id
          AND film.title = 'ZORRO ARK'
          JOIN category ON category.category_id = film_category.category_id;
         "
result <- dbGetQuery(conn, query)
print(result)
```

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

Question 4

```
query <- "EXPLAIN QUERY PLAN
          SELECT film.title as film_name, category.name as category_name, film.length
          FROM film
          JOIN film_category ON film_category.film_id = film.film_id
          AND film.title = 'ZORRO ARK'
          JOIN category ON category.category_id = film_category.category_id;
         "
result <- dbGetQuery(conn, query)
print(result$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=?)"
```

Question 5

```
sqlStmt <- "CREATE INDEX TitleIndex
          ON film (title);
         "
result <- dbSendQuery(conn, sqlStmt)
print(result)
```

```
## <SQLiteResult>
##   SQL  CREATE INDEX TitleIndex
##           ON film (title);
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

Question 6

```r
query <- "EXPLAIN QUERY PLAN
          SELECT film.title as film_name, category.name as category_name, film.length
          FROM film
          JOIN film_category ON film_category.film_id = film.film_id
          AND film.title = 'ZORRO ARK'
          JOIN category ON category.category_id = film_category.category_id;
         "
result <- dbGetQuery(conn, query)
```

```
## Warning: Closing open result set, pending rows
```

```r
print(result$detail)
```

```
## [1] "SEARCH film USING INDEX TitleIndex (title=?)"
## [2] "SEARCH film_category USING COVERING INDEX sqlite_autoindex_film_category_1 (film_id=?)"
## [3] "SEARCH category USING INTEGER PRIMARY KEY (rowid=?)"
```

Question 7

The query plan for q6 is different from the one used in q4 because the primary plan displayed above was to use `TitleIndex`. We can confirm that the index was considered because the output says "SEARCH film USING INDEX TitleIndex".

Question 8

```r
sqlStmt <- "DROP INDEX TitleIndex";
dbSendStatement(conn, sqlStmt)
```

```
## <SQLiteResult>
##   SQL  DROP INDEX TitleIndex
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```r
start.time <- Sys.time()
query <- "SELECT film.title as film_name, category.name as category_name, film.length
          FROM film
          JOIN film_category ON film_category.film_id = film.film_id
          AND film.title = 'ZORRO ARK'
          JOIN category ON category.category_id = film_category.category_id;
         "
result <- dbGetQuery(conn, query)
```

```
## Warning: Closing open result set, pending rows
```

```r
end.time <- Sys.time()
time.diff <- end.time - start.time
print(paste("Execution time (without index)", round((time.diff),3), "sec"))
```

```
## [1] "Execution time (without index) 0.005 sec"
```

```r
sqlStmt <- "CREATE INDEX TitleIndex
             ON film (title);
           "
result <- dbSendQuery(conn, sqlStmt)
print(result)
```

```
## <SQLiteResult>
##   SQL  CREATE INDEX TitleIndex
##             ON film (title);
##
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
start.time <- Sys.time()
query <- "SELECT f.title as film_name, c.name as category_name, length
          FROM film f
          JOIN film_category fc ON fc.film_id = f.film_id
          AND f.title = 'ZORRO ARK'
          JOIN category c ON c.category_id = fc.category_id;
         "
result <- dbGetQuery(conn, query)
```

```
## Warning: Closing open result set, pending rows
```

```
end.time <- Sys.time()
time.diff.index <- end.time - start.time
print(paste("Execution time (with index)", round((time.diff),3), "sec"))
```

```
## [1] "Execution time (with index) 0.005 sec"
```

```
print(paste("Time diff", time.diff - time.diff.index, "sec"))
```

```
## [1] "Time diff 0.0029909610748291 sec"
```

We can observe that the time difference between the query without an index and the query with an index is greater than 0, which indicates that fetching the results without an index took longer than fetching them with an index. Although the time difference is relatively small in this case due to the small number of rows in the film table (1,000 rows), the impact of the index would be significantly more noticeable when dealing with a larger table with more number of rows to be scanned.

Question 9

```
query <- "
      SELECT film.title, language.name, film.length
      FROM film
      JOIN language ON language.language_id = film.language_id
      AND LOWER(film.title) LIKE '%gold%';
         "
result <- dbGetQuery(conn, query)
print(result)
```

```
##                    title    name 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

```
query <- "
      EXPLAIN QUERY PLAN
      SELECT film.title, language.name, film.length
      FROM film
      JOIN language ON language.language_id = film.language_id
      AND LOWER(film.title) LIKE '%gold%';
         "
result <- dbGetQuery(conn, query)
print(result$detail)
```

```
## [1] "SCAN film"
## [2] "SEARCH language USING INTEGER PRIMARY KEY (rowid=?)"
```

The reason this query doesn't use `TitleIndex` is the use of the LIKE operator. The LIKE operator's ability to accommodate wildcards introduces complexity for the optimizer in pinpointing the index rows that match the query conditions. To illustrate, consider the query WHERE title LIKE '%GOLD.' In this case, an index scan cannot be used because the optimizer cannot determine which index rows commence with the term "GOLD." Similarly, a query like WHERE title LIKE 'GOLD%' cannot make use of an index scan because the optimizer cannot identify which index rows conclude with the term "GOLD". A full table scan is performed in this case.

Disconnecting from database

```
dbDisconnect(conn)
```