Assignment / Explore Query Planning and Indexing

Summer Full 2023

Setup

Creating a connection variable to the SQlite databse to perform all the query operation

```r
library(RSQLite)

#provide path to database folder


#provied path to database file

dbfile = "sakila.db"

dbcon <- dbConnect(RSQLite :: SQLite() , dbfile)

tables <- dbListTables(dbcon)

# Print the table names
print(tables)
```

```
##  [1] "actor"                      "actor_info"
##  [3] "address"                    "category"
##  [5] "city"                       "country"
##  [7] "customer"                   "customer_list"
##  [9] "film"                       "film_actor"
## [11] "film_category"              "film_list"
## [13] "film_text"                  "inventory"
## [15] "language"                   "nicer_but_slower_film_list"
## [17] "payment"                    "rental"
## [19] "sales_by_film_category"     "sales_by_store"
## [21] "sqlite_sequence"            "staff"
## [23] "staff_list"                 "store"
```

Ques 1

Below is the R chunk to perform deletion of all user defined indexes and query to find the number of films per language

```r
# TODO : Delete user defined indexes

drop_ind <- "DROP INDEX IF EXISTS TitleIndex"

dbExecute(dbcon , drop_ind)
```

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

```
query <- "SELECT l.name AS language_name , COUNT(*) AS Total_films
FROM film as f
JOIN language as l ON f.language_id = l.language_id
GROUP BY l.name"


df <- dbGetQuery(dbcon, query)

print(df)
```

```
##   language_name Total_films
## 1       English        1000
```

Ques 2

Below is the R chunk to find the query plan for above query

```
query2  <- " EXPLAIN QUERY PLAN
SELECT l.name AS language_name , COUNT(*) AS Total_films
FROM film as f
JOIN language as l ON f.language_id = l.language_id
GROUP BY l.name"


result2 <- dbGetQuery(dbcon, query2)

print(result2)
```

```
##   id parent notused                                       detail
## 1  7      0       0                                       SCAN f
## 2  9      0       0 SEARCH l USING INTEGER PRIMARY KEY (rowid=?)
## 3 12      0       0                 USE TEMP B-TREE FOR GROUP BY
```

Ques 3

Below is the R chunk to find the title , category name and length of the film, titled “ZORRO ARK”

```
query3  <- "SELECT f.title , f.length , c.name
  FROM film as f
  JOIN film_category as fc ON f.film_id = fc.film_id
  JOIN category as c ON fc.category_id = c.category_id
  where f.title = 'ZORRO ARK'
  "

# calculating time taken here for the purpose of ques 8
s_t <- system.time ({
  result3 <- dbGetQuery(dbcon, query3)
})
```

```
print(s_t)
```

```
##    user  system elapsed 
##   0.001   0.001   0.001
```

```
print(result3)
```

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

Ques 4

Below is the R chunk to explain the query plan of above query

```
query4  <- " EXPLAIN QUERY PLAN
 SELECT f.title , f.length , c.name
  FROM film as f
  JOIN film_category as fc ON f.film_id = fc.film_id
  JOIN category as c ON fc.category_id = c.category_id
  where f.title = 'ZORRO ARK'
  "


result4 <- dbGetQuery(dbcon, query4)

print(result4)
```

```
##   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=?)
```

Ques 5

Below is the R chunk to create index film table

```
dbExecute(dbcon, "CREATE INDEX TitleIndex ON FILM (TITLE)")
```

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

Ques 6

Below is the R chunk to create index film table

```
query6_plan  <- " EXPLAIN QUERY PLAN
 SELECT f.title , f.length , c.name
  FROM film as f
  JOIN film_category as fc ON f.film_id = fc.film_id
  JOIN category as c ON fc.category_id = c.category_id
  where f.title = 'ZORRO ARK'
  "


result6_plan <- dbGetQuery(dbcon, query6_plan)

print(result6_plan)
```

```
##   id parent notused
## 1  5      0       0
## 2 10      0       0
## 3 14      0       0
##                                                                     detail
## 1                                    SEARCH f USING INDEX TitleIndex (title=?)
## 2 SEARCH fc USING COVERING INDEX sqlite_autoindex_film_category_1 (film_id=?)
## 3                                 SEARCH c USING INTEGER PRIMARY KEY (rowid=?)
```

Ques 7

Below is the R chunk to explain the index working :

Difference : (6) searches the films table by the index of TitleIndex where as (4) uses primary key (rowid) to search

How to identify : The search from index is specified by index name, here in case of (6) we have "SEARCH f USING INDEX TitleIndex (title=?)"

Comments : where there is a search by title query having an index in title column of the table ensures the db engine improve the efficiency of data retrieval operations by a sorted copy of the indexed column(s) and corresponding data rows.

QUES 8

We can see that after creating index the operation happened relatively fast, as it incorporated efficient search with sorted index rather than linear scan

```
time <- system.time({
  film_info <- dbGetQuery(dbcon, "
    SELECT f.title, c.name AS category, 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'
  ")
})
```

```
print(time)
```

```
##    user  system elapsed 
##       0       0       0
```

QUES 9

```
# as per readings using lower case because MySQL queries are not case-sensitive
# by default. If this works different it might be because of the internals of
# engine

query9 <-  "
  SELECT f.title , f.length , l.name
  FROM film as f
  JOIN language as l ON f.language_id = l.language_id
  where f.title LIKE '%gold%'
  "

result9 <- dbGetQuery(dbcon, query9)

print(result9)
```

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

```
# as per readings using lower case because MySQL queries are not case-sensitive
# by default. If this works different it might be because of the internals of
# engine

query10 <-  "
  EXPLAIN QUERY PLAN
  SELECT f.title , f.length , l.name
  FROM film as f
  JOIN language as l ON f.language_id = l.language_id
  where f.title LIKE '%gold%'
  "

result10 <- dbGetQuery(dbcon, query10)

print(result10)
```

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

The query doesnt use indexes because : Indexes are not useful when pattern matching searches are performed with LIKE. And in the case of searches using LIKE theengines do linear scan anyways.