Assignment / Explore Query Planning and Indexing

Shubha, Adithya

2024-11-12

```r
# Cleans the environment before beginning the execution
# referredFrom:https://northeastern.instructure.com/courses/192346/assignments/2351524
rm(list = ls())
```

```r
# Install program required packages
# Installs RSQLite, DBI  packages
# referredFrom: http://artificium.us/lessons/06.r/l-6-104-r4progs/l-6-104.html#Install_Packages_on_Dema
installRequiredPackages <- function() {
  packages <- c("RSQLite", "DBI", "testthat")
  # Install packages that are not installed
  installed_packages <- packages %in% rownames(installed.packages())
  if (any(installed_packages == FALSE)) {
    install.packages(packages[!installed_packages])
  }
}

# Loads the required packages to the environment
# Loads RSQLite, DBI packages
loadRequiredPackages <- function() {
  suppressMessages({
    library(RSQLite)
    library(DBI)
    library(testthat)
  })
}
installRequiredPackages()
loadRequiredPackages()
```

```r
# Connects to the database
# referredFrom: http://artificium.us/lessons/06.r/l-6-301-sqlite-from-r/l-6-301.html#Connect_to_Databas
# @param dbName: Name of the database to connect to
connectToDatabase <- function(dbName) {
  return (dbConnect(RSQLite::SQLite(), dbname = dbName))
}

# Ensures the database connection is established
# by running a test query and asserting on the result.
# @param dbCon: Database connection object
ensureDatabaseConnection <- function(dbCon) {
  result <- dbGetQuery(dbCon, "SELECT * FROM film LIMIT 1")
  test_that("Result size is 1", {
    expect_equal(nrow(result), 1)
  })

}

# Function to clear databse cache to checking working of indexes
# @param dbCon: Database connection object
clearDatabaseCache <- function(dbCon) {
  # Free up cache memory in SQLite
  # referredFrom: https://stackoverflow.com/questions/4123196/sqlite-abnormal-memory-usage
  invisible(dbExecute(dbCon, "PRAGMA shrink_memory;"))

  # Garbage collection
  # referredFrom: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/gc
  invisible(gc())
}

# Global database connection object
dbCon <- connectToDatabase("sakila.db")
ensureDatabaseConnection(dbCon)
```

```
## Test passed
```

```r
clearDatabaseCache(dbCon)
```

Question 1

```r
# Removes the given user defined index
# @param indexName: Name of the index to remove
queryToRemoveUserDefinedIndexes <- function(indexName) {
  return (sprintf("DROP INDEX IF EXISTS %s", indexName))
}

# Gets all user defined indexes
# @param dbCon: Database connection object
getAllUserDefinedIndexes <- function(dbCon) {
  return (dbGetQuery(dbCon, "SELECT name FROM sqlite_master WHERE type='index'"))
}

# Removes all user defined indexes
# @param dbCon: Database connection object
removeUserDefinedIndexes <- function(dbCon) {
  indexes <- getAllUserDefinedIndexes(dbCon)
  for (index in indexes$name) {
    # Ignoring auto generated indexes
    if (!grepl("sqlite_autoindex", index)) {
      dbExecute(dbCon, queryToRemoveUserDefinedIndexes(index))
    }
  }
}
```

```r
# Query to get the film count per category
# Query assumes category_name and category_id have 1v1 mapping.
queryToGetFilmCountPerCategory <- function() {
  return (
    "SELECT C.name as category_name, count(FC.film_id) as film_count
FROM film_category as FC
         JOIN category as C ON FC.category_id = C.category_id
GROUP BY C.name
"
  )
}

# Gets the film count per category
# @param dbCon: Database connection object
getFilmCountPerCategory <- function(dbCon) {
  result <- dbGetQuery(dbCon, queryToGetFilmCountPerCategory())

  # Formatting the film count to have a readable format
  result$film_count <- format(result$film_count,
                              scientific = FALSE,
                              big.mark = ",")
  return (result)
}

# Displays information on Zorro Ark film
# @param result: Result of the query
displayInformationOnFilm <- function(result) {
  print(result,row.names = FALSE)
}

# Displays the film count per category
# @param dbCon: Database connection object
displayFilmCountPerCategory <- function(dbCon) {
  result <- getFilmCountPerCategory(dbCon)
  displayInformationOnFilm(result)
}

removeUserDefinedIndexes(dbCon)
displayFilmCountPerCategory(dbCon)
```

```
##  category_name film_count
##         Action         64
##      Animation         66
##       Children         60
##       Classics         57
##         Comedy         58
##    Documentary         68
##          Drama         62
##         Family         69
##        Foreign         73
##          Games         61
##         Horror         56
##          Music         51
##           New         63
##        Sci-Fi         61
##        Sports         74
##        Travel         57
```

Question 2

```r
# Gives the query plan for the given query
# @param dbCon: Database connection object
# @param query: Query for which the plan is required
getQueryPlan <- function(dbCon,query){
  return(dbGetQuery(dbCon, sprintf("EXPLAIN QUERY PLAN %s", query)))
}

# Displays the query plan for the given query
# Displays only the detail, other information can be added if required
# @param dbCon: Database connection object
# @param query: Query for which the plan is required
displayQueryPlan <- function(dbCon,query){
  result <- getQueryPlan(dbCon,query)
  print(result$detail,row.names = FALSE)
}

displayQueryPlan(dbCon,queryToGetFilmCountPerCategory())
```

```
## [1] "SCAN FC USING COVERING INDEX sqlite_autoindex_film_category_1"
## [2] "SEARCH C USING INTEGER PRIMARY KEY (rowid=?)"
## [3] "USE TEMP B-TREE FOR GROUP BY"
```

Question 3

```r
# Query to get title,length, rental_rate, release_year of Zorro Ark film
queryToGetInformationOnZorroArk <- function() {
  return (
    "SELECT  title,length,rental_rate,release_year FROM film
where title = 'ZORRO ARK'")
}

getInformationOnZorroArk <- function(dbCon) {
  return (dbGetQuery(dbCon, queryToGetInformationOnZorroArk()))
}

clearDatabaseCache(dbCon)
queryStartTime <- Sys.time()
informationOnZorroArk <- getInformationOnZorroArk(dbCon)
queryEndTime <- Sys.time()
# in milliseconds
timeToFetchInformationOnZorroArk <- round(((queryEndTime - queryStartTime)*1000),3)
displayInformationOnFilm(informationOnZorroArk)
```

```
##      title length rental_rate release_year
##  ZORRO ARK     50        4.99         2006
```

Question 4

```r
# display query plan on information of Zorro Ark
displayQueryPlan(dbCon,queryToGetInformationOnZorroArk())
```

```
## [1] "SCAN film"
```

Question 5

```r
# Query to create index on title column
queryToCreateIndexOnTitle <- function() {
  return ("CREATE INDEX IF NOT EXISTS TitleIndex
    ON film (title);")
}
createIndexOnTitle <- function(dbCon) {
  invisible(dbExecute(dbCon, queryToCreateIndexOnTitle()))
}

createIndexOnTitle(dbCon)
```

Question 6

```r
clearDatabaseCache(dbCon)
startTime <- Sys.time()
informationOnZorroArk <- getInformationOnZorroArk(dbCon)
endTime <- Sys.time()
# in milliseconds
timeToFetchInformationOnZorroArkAfterIndex <- round(((endTime - startTime)*1000),3)
displayInformationOnFilm(informationOnZorroArk)
```

```
##      title length rental_rate release_year
##  ZORRO ARK     50        4.99         2006
```

```r
displayQueryPlan(dbCon,queryToGetInformationOnZorroArk())
```

```
## [1] "SEARCH film USING INDEX TitleIndex (title=?)"
```

Question 7

The query plan for question 4 and question 6 are different. The difference is that query plan in question 6 uses the index to fetch the data while in question 4 the enitre film table is scanned to fetch the data. We know question 6 is using index because the query plan shows the keyword 'USING INDEX' with the index name 'TitleIndex' we created earlier.

Question 8

The time taken to fetch information on Zorro Ark before creating the index is 2.505 ms and after creating the index is 1.544 ms This shows that there is difference between both these times. The difference is 0.961 ms. Ideally it should be the case that the time to fetch the information on Zorro Ark is reduced after creating the index on the title column since the index helps in fetching the data faster.

But if the data set is small then the time taken to fetch the data is almost the same before and after creating the index(as the difference is really low in milliseconds). This is because the time taken to scan the entire table is almost the same as the time taken to scan the index and fetch the data. Here index might introduce an overhead where it has to scan the index and then fetch the data from the table. This overhead might be more than the time taken to scan the entire table and fetch the data and is not worth if it is a single row lookup in a small table. This is the reason why the time taken to fetch the data before and after creating the index is almost the same in-case of smaller dataset.

Question 9

```r
# Query to get information on films having title with gold
queryToGetInformationWithFilmTitleHavingGold <- function() {
  return (
    "SELECT
    title,
    length,
    rental_rate,
    release_year
FROM film
WHERE LOWER(title) LIKE '%gold%';")
}

# Displays information on films having title with gold
# @param dbCon: Database connection object
displayInformationWithFilmTitleHavingGold <- function(dbCon) {
  result <- dbGetQuery(dbCon, queryToGetInformationWithFilmTitleHavingGold())
  displayInformationOnFilm(result)
}
displayInformationWithFilmTitleHavingGold(dbCon)
```

```
##                    title length rental_rate release_year
##          ACE GOLDFINGER     48        4.99         2006
##    BREAKFAST GOLDFINGER    123        4.99         2006
##              GOLD RIVER    154        4.99         2006
##  GOLDFINGER SENSIBILITY     93        0.99         2006
##         GOLDMINE TYCOON    153        0.99         2006
##              OSCAR GOLD    115        2.99         2006
##    SILVERADO GOLDFINGER     74        4.99         2006
##              SWARM GOLD    123        0.99         2006
```

Question 10

```r
# Display query plan on information of films having title with gold
displayQueryPlan(dbCon,queryToGetInformationWithFilmTitleHavingGold())
```

```
## [1] "SCAN film"
```

The query plan shows that it is not using index. This happens because if an approximate search using LIKE is performed, then an index is not useful and the table is scanned row by row. The query plan above shows that the table is scanned and the index is ignored.

```r
# Disconnects from the database
# referredFrom: http://artificium.us/lessons/06.r/l-6-301-sqlite-from-r/l-6-301.html#Disconnect_from_Da
dbDisconnect(dbCon)
```