Assignment / Explore Query Planning

Xuebao Zhao

Install the package

```r
# Package names
packages <- c("RMySQL", "RSQLite", "DBI")
# Install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}
# Packages loading
invisible(lapply(packages, library, character.only = TRUE))
```

```
## Loading required package: DBI

##
## Attaching package: 'RSQLite'

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

Connect database through SQLite

```r
fpath = ""
dbfile = "sakila.db"

# if database file already exists, we connect to it, otherwise
# we create a new database
scon <- dbConnect(RSQLite::SQLite(), paste0(fpath,dbfile))
```

```r
check_sqlite <- dbGetQuery(scon, "SELECT * FROM FILM")
head(check_sqlite)
```

```
##   film_id            title
## 1       1 ACADEMY DINOSAUR
## 2       2   ACE GOLDFINGER
## 3       3 ADAPTATION HOLES
## 4       4 AFFAIR PREJUDICE
## 5       5      AFRICAN EGG
## 6       6     AGENT TRUMAN
##
## 1                      A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in Th
## 2                  A Astounding Epistle of a Database Administrator And a Explorer who must Find a Ca
## 3                      A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack i
## 4                          A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monk
## 5 A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in
## 6                             A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestle
##   release_year language_id original_language_id rental_duration rental_rate
## 1         2006           1                   NA               6        0.99
## 2         2006           1                   NA               3        4.99
## 3         2006           1                   NA               7        2.99
## 4         2006           1                   NA               5        2.99
## 5         2006           1                   NA               6        2.99
## 6         2006           1                   NA               3        2.99
##   length replacement_cost rating                 special_features
## 1     86            20.99     PG Deleted Scenes,Behind the Scenes
## 2     48            12.99      G          Trailers,Deleted Scenes
## 3     50            18.99  NC-17          Trailers,Deleted Scenes
## 4    117            26.99      G  Commentaries,Behind the Scenes
## 5    130            22.99      G                   Deleted Scenes
## 6    169            17.99     PG                   Deleted Scenes
##           last_update
## 1 2006-02-15 05:03:42
## 2 2006-02-15 05:03:42
## 3 2006-02-15 05:03:42
## 4 2006-02-15 05:03:42
## 5 2006-02-15 05:03:42
## 6 2006-02-15 05:03:42
```

Connect database through MySQL

```
# Settings
db_user <- 'root'
db_password <- 'zxb0285zxb'
db_name <- 'sakila'

db_host <- 'localhost'
db_port <- 3306 # always this port unless you change it during installation

# 3. Connect to DB
mcon <-  dbConnect(MySQL(), user = db_user, password = db_password,
                 dbname = db_name, host = db_host, port = db_port)
```

```
check_mysql <- dbGetQuery(mcon, "SELECT * FROM FILM")
```

```
## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 0 imported as
## numeric

## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 4 imported as
## numeric
## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 5 imported as
## numeric

## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 6 imported as
## numeric

## Warning in .local(conn, statement, ...): Decimal MySQL column 7 imported as
## numeric

## Warning in .local(conn, statement, ...): Unsigned INTEGER in col 8 imported as
## numeric

## Warning in .local(conn, statement, ...): Decimal MySQL column 9 imported as
## numeric

## Warning in .local(conn, statement, ...): unrecognized MySQL field type 7 in
## column 12 imported as character
```

```
head(check_mysql)
```

```
##   film_id            title
## 1       1 ACADEMY DINOSAUR
## 2       2   ACE GOLDFINGER
## 3       3 ADAPTATION HOLES
## 4       4 AFFAIR PREJUDICE
## 5       5      AFRICAN EGG
## 6       6     AGENT TRUMAN
##
## 1                      A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in Th
## 2                  A Astounding Epistle of a Database Administrator And a Explorer who must Find a Ca
## 3                      A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack i
## 4                          A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monk
## 5 A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in
## 6                             A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestle
##   release_year language_id original_language_id rental_duration rental_rate
## 1         2006           1                   NA               6        0.99
## 2         2006           1                   NA               3        4.99
## 3         2006           1                   NA               7        2.99
## 4         2006           1                   NA               5        2.99
## 5         2006           1                   NA               6        2.99
## 6         2006           1                   NA               3        2.99
##   length replacement_cost rating                 special_features
## 1     86            20.99     PG Deleted Scenes,Behind the Scenes
## 2     48            12.99      G          Trailers,Deleted Scenes
## 3     50            18.99  NC-17          Trailers,Deleted Scenes
## 4    117            26.99      G   Commentaries,Behind the Scenes
## 5    130            22.99      G                   Deleted Scenes
## 6    169            17.99     PG                   Deleted Scenes
##           last_update
## 1 2006-02-15 05:03:42
## 2 2006-02-15 05:03:42
## 3 2006-02-15 05:03:42
## 4 2006-02-15 05:03:42
## 5 2006-02-15 05:03:42
## 6 2006-02-15 05:03:42
```

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

Find the indexes which is created by user

```
index_name <- dbGetQuery(scon,
"SELECT `name`
  FROM sqlite_master
WHERE `type` = 'index' AND `name` NOT LIKE '%auto%';")

index_name
```

```
##         name
## 1 TitleIndex
```

Drop the indexes which is created by user

```
for (name in index_name$name) {
  query <- paste0("DROP INDEX IF EXISTS ", name)
  dbExecute(scon, query)
}
```

Find the number of films per category

```
number_cate <- dbGetQuery(scon,
"SELECT category.name, COUNT(film_category.film_id) as num_films
  FROM category
    JOIN film_category ON category.category_id = film_category.category_id
  GROUP BY category.name")

number_cate
```

```
##           name num_films
## 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 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)

Find the indexes which is created by user

```r
query <- paste0("SELECT TABLE_NAME, INDEX_NAME FROM INFORMATION_SCHEMA.STATISTICS WHERE TABLE_SCHEMA =
                 db_name, "' AND `INDEX_TYPE` = NULL")
mindex <- dbGetQuery(mcon,query)
mindex
```

```
## [1] TABLE_NAME INDEX_NAME
## <0 rows> (or 0-length row.names)
```

Drop the indexes which is created by user

```r
# Loop through the table IDs and get the table names
if (length(mindex$TABLE_NAME) != 0) {
# If there is index then do the below
  for (index in mindex) {
  query <- paste0("DROP ", mindex$INDEX_NAME, " ON ", mindex$TABLE_NAME)
  result <- dbGetQuery(mcon, query)
  print(result$table_name)
  }
}
```

Find the number of films per category

```r
number_cate_my <- dbGetQuery(mcon,
"SELECT category.name, COUNT(film_category.film_id) as num_films
  FROM category
    JOIN film_category ON category.category_id = film_category.category_id
  GROUP BY category.name")

number_cate_my
```

```
##           name num_films
## 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)

The query plans for the query executions in (1)

```
bt_3 <- Sys.time()
splan <- dbGetQuery(scon, "EXPLAIN SELECT category.name, COUNT(film_category.film_id) as num_films
  FROM category
    JOIN film_category ON category.category_id = film_category.category_id
  GROUP BY category.name")

et_3 <- Sys.time()
t.loop <- et_3 - bt_3

cat("Time elapsed: ", round((t.loop),3), " sec")
```

```
## Time elapsed:  0.002  sec
```

```
print(splan)
```

```
##    addr       opcode p1 p2 p3     p4 p5 comment
## 1     0         Init  0 45  0   <NA>  0      NA
## 2     1   SorterOpen  2  2  0 k(1,B)  0      NA
## 3     2      Integer  0  5  0   <NA>  0      NA
## 4     3         Null  0  8  8   <NA>  0      NA
## 5     4        Gosub  7 41  0   <NA>  0      NA
## 6     5     OpenRead  3 13  0 k(2,,)  0      NA
## 7     6     OpenRead  0  5  0      2  0      NA
## 8     7       Rewind  3 15 10      0  0      NA
## 9     8       Column  3  1 10   <NA>  0      NA
## 10    9    SeekRowid  0 14 10   <NA>  0      NA
## 11   10       Column  0  1 11   <NA>  0      NA
## 12   11       Column  3  0 12   <NA>  0      NA
## 13   12   MakeRecord 11  2 13   <NA>  0      NA
## 14   13 SorterInsert  2 13  0   <NA>  0      NA
## 15   14         Next  3  8  0   <NA>  1      NA
## 16   15   OpenPseudo  4 13  2   <NA>  0      NA
## 17   16   SorterSort  2 44  0      <NA>  0       NA
## 18   17   SorterData  2 13  4      <NA>  0       NA
## 19   18       Column  4  0  9      <NA>  0       NA
## 20   19      Compare  8  9  1    k(1,B)  0       NA
## 21   20         Jump 21 25 21      <NA>  0       NA
## 22   21         Move  9  8  1      <NA>  0       NA
## 23   22        Gosub  6 35  0      <NA>  0       NA
## 24   23        IfPos  5 44  0      <NA>  0       NA
## 25   24        Gosub  7 41  0      <NA>  0       NA
## 26   25       Column  4  1 14      <NA>  0       NA
## 27   26      AggStep  0 14  2  count(1)  1       NA
## 28   27           If  4 29  0      <NA>  0       NA
## 29   28       Column  4  0  1      <NA>  0       NA
## 30   29      Integer  1  4  0      <NA>  0       NA
## 31   30   SorterNext  2 17  0      <NA>  0       NA
## 32   31        Gosub  6 35  0      <NA>  0       NA
## 33   32         Goto  0 44  0      <NA>  0       NA
## 34   33      Integer  1  5  0      <NA>  0       NA
## 35   34       Return  6  0  0      <NA>  0       NA
## 36   35        IfPos  4 37  0      <NA>  0       NA
## 37   36       Return  6  0  0      <NA>  0       NA
## 38   37     AggFinal  2  1  0  count(1)  0       NA
## 39   38         Copy  1 15  1      <NA>  0       NA
## 40   39    ResultRow 15  2  0      <NA>  0       NA
## 41   40       Return  6  0  0      <NA>  0       NA
## 42   41         Null  0  1  3      <NA>  0       NA
## 43   42      Integer  0  4  0      <NA>  0       NA
## 44   43       Return  7  0  0      <NA>  0       NA
## 45   44         Halt  0  0  0      <NA>  0       NA
## 46   45  Transaction  0  0 50         0  1       NA
## 47   46         Goto  0  1  0      <NA>  0       NA
```

The query plans for the query executions in (2)

```r
bt_3_2 <- Sys.time()

mplan <- dbGetQuery(mcon, "EXPLAIN SELECT category.name, COUNT(film_category.film_id) as num_films
  FROM category
    JOIN film_category ON category.category_id = film_category.category_id
  GROUP BY category.name")

bt_3_2 <- Sys.time()
et_3_2 <- Sys.time()
t.loop <- et_3_2 - bt_3_2

cat("Time elapsed: ", round((t.loop),3), " sec")
```

```
## Time elapsed:  0.001  sec
```

```r
print(mplan)
```

```
##   id select_type         table partitions type             possible_keys
## 1  1      SIMPLE      category       <NA>  ALL                   PRIMARY
## 2  1      SIMPLE film_category       <NA>  ref fk_film_category_category
##                        key key_len                        ref rows filtered
## 1                     <NA>    <NA>                       <NA>   16      100
## 2 fk_film_category_category       1 sakila.category.category_id   62      100
##             Extra
## 1 Using temporary
## 2     Using index
```

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?

The query plans for the same query executed in SQLite and MySQL are different, as expected. Here are some observations on the differences:

SQLite uses a sorter to group the results by category name, while MySQL does not need to do this because it uses a different grouping strategy based on the join order and index access method.

SQLite scans the film table first, while MySQL scans the film_category table first. This is likely because SQLite chooses a different join order compared to MySQL.

SQLite uses a seek operation to access the film_category table, while MySQL uses a ref operation. This is because MySQL has an index on the category_id column of the film_category table, which allows it to perform an index lookup rather than a full table scan.

SQLite does not show any information about the number of rows returned by each operation, while MySQL shows the estimated number of rows and the percentage of rows that are filtered by each operation.

In terms of performance, Here we used "Sys.time" to measure run-time performance of R code. Query plans in (1) takes more longer time than in (2) in most cases.

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"

```r
find_zorro <- dbGetQuery(scon, "SELECT title, language.name AS language, length
  FROM film
    JOIN language ON film.language_id = language.language_id
  WHERE title = 'ZORRO ARK'")

print(find_zorro)
```

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

Question 6 For the query in (5), display the query plan

```r
bt_6 <- Sys.time()

splan_6 <- dbGetQuery(scon, "EXPLAIN SELECT title, language.name AS language, length
  FROM film
```

```
    JOIN language ON film.language_id = language.language_id
  WHERE title = 'ZORRO ARK'")

et_6 <- Sys.time()
t.loop <- et_6 - bt_6

cat("Time elapsed: ", round((t.loop),3), " sec")
```

```
## Time elapsed:  0.002  sec
```

```
print(splan_6)
```

```
##    addr      opcode p1 p2 p3        p4 p5 comment
## 1     0        Init  0 14  0      <NA>  0      NA
## 2     1    OpenRead  0  9  0         9  0      NA
## 3     2    OpenRead  1 18  0         2  0      NA
## 4     3      Rewind  0 13  0      <NA>  0      NA
## 5     4      Column  0  1  1      <NA>  0      NA
## 6     5          Ne  2 12  1  BINARY-8 82      NA
## 7     6      Column  0  4  3      <NA>  0      NA
## 8     7   SeekRowid  1 12  3      <NA>  0      NA
## 9     8      Column  0  1  4      <NA>  0      NA
## 10    9      Column  1  1  5      <NA>  0      NA
## 11   10      Column  0  8  6      NULL  0      NA
## 12   11   ResultRow  4  3  0      <NA>  0      NA
## 13   12        Next  0  4  0      <NA>  1      NA
## 14   13        Halt  0  0  0      <NA>  0      NA
## 15   14 Transaction  0  0 50         0  1      NA
## 16   15     String8  0  2  0 ZORRO ARK  0      NA
## 17   16        Goto  0  1  0      <NA>  0      NA
```

Question 7 In the SQLite database, create a user-defined index called "TitleIndex" on the column TITLE in the table FILM

```
dbExecute(scon, "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

```
find_zorro2 <- dbGetQuery(scon, "SELECT title, language.name AS language, length
  FROM film
    JOIN language ON film.language_id = language.language_id
  WHERE title = 'ZORRO ARK'")

print(find_zorro2)
```

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

```
bt_8 <- Sys.time()
splan_8 <- dbGetQuery(scon, "EXPLAIN SELECT title, language.name AS language, length
  FROM film
    JOIN language ON film.language_id = language.language_id
  WHERE title = 'ZORRO ARK'")

et_8 <- Sys.time()
t.loop <- et_8 - bt_8

cat("Time elapsed: ", round((t.loop),3), " sec")
```

```
## Time elapsed:  0.002  sec
```

```
print(splan_8)
```

```
##    addr       opcode p1  p2 p3        p4 p5 comment
## 1     0         Init  0  16  0      <NA>  0      NA
## 2     1     OpenRead  0   9  0         9  0      NA
## 3     2     OpenRead  2 791  0    k(2,,)  2      NA
## 4     3     OpenRead  1  18  0         2  0      NA
## 5     4      String8  0   1  0 ZORRO ARK  0      NA
## 6     5       SeekGE  2  15  1         1  0      NA
## 7     6        IdxGT  2  15  1         1  0      NA
## 8     7 DeferredSeek  2   0  0      <NA>  0      NA
## 9     8       Column  0   4  2      <NA>  0      NA
## 10    9    SeekRowid  1  14  2      <NA>  0      NA
## 11   10       Column  2   0  3      <NA>  0      NA
## 12   11       Column  1   1  4      <NA>  0      NA
## 13   12       Column  0   8  5      NULL  0      NA
## 14   13    ResultRow  3   3  0      <NA>  0      NA
## 15   14         Next  2   6  1      <NA>  0      NA
## 16   15         Halt  0   0  0      <NA>  0      NA
## 17   16  Transaction  0   0 51         0  1      NA
## 18   17         Goto  0   1  0      <NA>  0      NA
```

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?

The query plans for (6) and (8) are different. In (6), the query plan involves using an index seek operation with a binary comparison to retrieve the rows that match the specified title. In contrast, (8) uses an index seek operation to find the first row with a title greater than or equal to "ZORRO ARK" and then performs an index greater than operation to retrieve the remaining rows. Additionally, (8) involves deferred seeks to skip over any duplicate rows with the same title, whereas (6) does not.

Here we used "Sys.time" to measure run-time performance of R code. Query plans in (6) takes more longer time than in (8) in most cases.

In the query plan, we can determine whether an index is used or not by checking for the presence of operations like "OpenRead" and "IdxGT" that indicate an index is being accessed. Additionally, the use of specific indexes can be identified by their names in the "comment" column of the query plan.

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)

```
find_gold <- dbGetQuery(scon, "SELECT title, language.name AS language, length
  FROM film
    JOIN language ON film.language_id = language.language_id
  WHERE title LIKE '%GOLD'")

print(find_gold)
```

```
##        title language length
## 1 OSCAR GOLD  English    115
## 2 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?

```
splan_11 <- dbGetQuery(scon, "EXPLAIN SELECT title, language.name AS language, length
  FROM film
    JOIN language ON film.language_id = language.language_id
  WHERE title LIKE '%GOLD'")

print(splan_11)
```

```
##    addr      opcode p1 p2 p3      p4 p5 comment
## 1     0        Init  0 15  0    <NA>  0      NA
## 2     1    OpenRead  0  9  0       9  0      NA
## 3     2    OpenRead  1 18  0       2  0      NA
## 4     3      Rewind  0 14  0    <NA>  0      NA
## 5     4      Column  0  1  3    <NA>  0      NA
## 6     5    Function  1  2  1 like(2)  0      NA
## 7     6       IfNot  1 13  1    <NA>  0      NA
## 8     7      Column  0  4  4    <NA>  0      NA
## 9     8   SeekRowid  1 13  4    <NA>  0      NA
## 10    9      Column  0  1  5    <NA>  0      NA
## 11   10      Column  1  1  6    <NA>  0      NA
## 12   11      Column  0  8  7    NULL  0      NA
## 13   12   ResultRow  5  3  0    <NA>  0      NA
## 14   13        Next  0  4  0    <NA>  1      NA
## 15   14        Halt  0  0  0    <NA>  0      NA
## 16   15 Transaction  0  0 51       0  1      NA
## 17   16     String8  0  2  0   %GOLD  0      NA
## 18   17        Goto  0  1  0    <NA>  0      NA
```

The query plan for (10) does not use the index we created on the TITLE column. The LIKE function in line 6 does not use an index, as it performs a full table scan to find matching rows. We can see that there is no mention of an index in the query plan

```
dbDisconnect(scon)
dbDisconnect(mcon)
```

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