PHẦN 1

EXPLAIN trong SQL

I. Explain là gì

Explain là câu lệnh trong mysql giúp bạn biết được những gì xảy ra bên trong một câu lệnh khác. Sử dụng explain một cách thành thục sẽ giúp bạn tránh khỏi các câu query tồi, cũng giống như phát hiện ra các bottle neck của hệ thống như chưa dán index...

Giả sử mình có một câu sql như sau.

```
mysql> explain SELECT `branches`.`id` FROM `branches` INNER
JOIN `branch_accountings` ON `branches`.`id` =
`branch_accountings`.`branch_id` WHERE
`branch_accountings`.`deleted_at` IS NULL AND
`branch_accountings`.`user_id` = 2\G;
```

thì explain câu lệnh này sẽ có kết quả ví dụ như sau:

```
           id: 1
  select_type: SIMPLE
        table: branch_accountings
   partitions: NULL
         type: ref
possible_keys:
index_branch_accountings_on_branch_id,index_branch_accountings_o
n_user_id,index_branch_accountings_on_branch_id_and_user_id
          key: index_branch_accountings_on_user_id
      key_len: 5
```

```
          ref: const
         rows: 7
     filtered: 10.00
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: branches
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: fcfs_development.branch_accountings.branch_id
         rows: 1
     filtered: 100.00
        Extra: Using where; Using index
```

Từ ví dụ trên bạn có thể thấy explain cho chúng ta một vài nhận xét sau:

Một query to sẽ được "break" thành nhiều query nhỏ. Với mỗi query nhỏ đó, chúng ta sẽ có nhiều thông số để biết được là

query nhỏ đo thuộc loại nào, lấy từ index ra sao... Chúng ta sẽ đi vào chi tiết các thông số dưới đây.

II. Trường dữ liệu của EXPLAIN

a. select_type:

Về cơ bản thì bạn hiểu đây là tham số về "kiểu" query.

Trong trường hợp là Join query

Đầu tiên mình sẽ nói về một khái niệm trong join query gọi là Nested-Loop Join. Nested-Loop Join có nghĩa là đầu tiên là mysql sẽ lấy lần lượt data match điều kiện ở bảng 1, sau đó với mỗi data đó sẽ lấy lần lượt data match điều kiện ở bảng 2, sau đó JOIN vào kết quả cuối cùng. Bạn có thể hình dung đây là một cách match rất "trâu bò" là được.

Trong trường hợp query là join query, select_type sẽ luôn hiện là SIMPLE như bạn đã thấy ở hình trên. Do đó bạn đừng nhầm trong trường hợp này với nghĩa là query thuộc loại "đơn giản". Việc select_type là SIMPLE thể hiện là query được explain sẽ được xử lý bằng Nested-Loop Join, đơn giản là thế thôi.

Trong trường hợp là "subquery"

"subquery" tức là kiểu trong select lại có một select khác vậy, giống như ví dụ trên của mình. Khi đó thì select_type sẽ có những loại như sau:

PRIMARY .... khi mà query một field "nằm ngoài" subquery, truy vấn là câu SELECT ngoài cùng của một lệnh JOIN.

SUBQUERY .... query "đầu tiên" nằm trong subquery, không phụ thuộc vào query nào khác. Query này sẽ được execute đúng lần đầu tiên, sau đó kết quả sẽ được cache lại.

DEPENDENT SUBQUERY .... query mà phụ thuộc vào query nằm ngoài nó

UNCACHEABLE SUBQUERY .... query không cache được

DERIVED .... Truy vấn là một truy vấn con của truy vấn khác, nằm trong lệnh FROM.

Trong trường hợp là "union"

Khi mà query là union sẽ có những kiểu select_type sau:

PRIMARY ... bảng đầu tiên được fetch về khi union

UNION.... bảng thứ 2 được fetch về.

UNION RESULT .... kết quả union

DEPENDENT UNION.... khi mà trong subquery có union, và subquery đó thuộc loại DEPENDENT SUBQUERY

UNCACHEABLE UNION .... khi mà trong uncacheable subquery có chứa union

b. table

Đơn giản là tên của table mà query đó sử dụng, không có gì đáng chú ý cả

c. type

Biến này cho chúng ta một thông tin rất quan trọng, đó là "cách" access vào table sử dụng trong query đó. Tuỳ vào cách access chúng ta sẽ có những cách access "nhanh", và cách "access" chậm. Biến này là biến chính để giúp chúng ta tuning index cho database. Chúng ta hãy đi vào chi tiết nào:

system - Bảng không có hoặc chỉ có 1 dòng.

const ... Khi mà table được access sử dụng PRIMARY KEY, hoặc sử dụng index một field nào mà các giá trị của field đó là UNIQUE. Khi mà type là const chúng ta sẽ có tốc độ access vào table "nhanh nhất"! Bảng chỉ có duy nhất 1 dòng đã được đánh chỉ mục

mà khớp với điều kiện tìm kiếm. Đây là loại join nhanh nhất, bởi bảng chỉ cần đọc một lần duy nhất và giá trị của cột được xem như là hằng số khi join với các bảng khác.

eq_ref ... giống như const cơ mà field được sử dụng không đứng riêng mà nằm trong câu lệnh JOIN. Tất cả các thành phần của index được sử dụng bởi lệnh join và index thuộc loại PRIMARY KEY hoặc UNIQUE NOT NULL. Đây là loại join tốt thứ hai (chỉ sau const).

ref .... Khi mà field được tìm kiếm có được dán index , tuy nhiên field đó không phải là UNIQUE, và field đó được sử dụng trong phép so sánh Where = hoặc <=>.

range .... Khi mà field được tìm kiếm có dán index, và được sử dụng trong phép tìm theo range Where In hoặc là Where > hay Where < ..

index .... khi có dán index cơ mà để tìm ra kết quả thì mysql bắt buộc phải scan toàn bộ field, do đó mà sẽ rất chậm

ALL : Đây là khi mà không những field không dán index, mà lại còn phải scan toàn bộ field, đây chính là nơi mà bạn bắt buộc phải tuning , không thì sẽ chậm kinh hoàng :ohmygod:

d. possible_keys

List tất cả các key mà optimizer của mysql có thể sử dụng được index của chúng để tìm kiếm (nhớ là "có thể" thôi nhé, chưa chắc đã dùng). Trong thực tế, cột này đôi khi giúp cho việc tối ưu truy vấn, bởi nếu cột này trống (NULL), nó thường cho thấy không có chỉ mục liên quan được định nghĩa trong bảng.

e. key

Key được chính thức optimizer sử dụng để làm index để tìm kiếm. Cột này có thể chứa khóa không được liệt kê ở cột possible_keys. Trình tối ưu của MySQL luôn cố gắng tìm kiếm khóa tối ưu nhất cho truy vấn. Khi kết hợp nhiều bảng, nó có thể dùng khóa không

nằm trong danh sách possible_keys nhưng lại đem về hiệu quả cao hơn.

f. ref

Biến này thể hiện field mà key ở trên sẽ được đem ra so sánh với khi mà mysql tiến hành tìm kiếm. Tên cột hoặc hằng số được dùng để so sánh với chỉ mục được nêu ra ở cột key. MySQL có thể lấy ra một hằng số, hoặc một cột cho quá trình thực hiện truy vấn Trong trường hợp query là JOIN thì đây chính là giá trị của key ở bảng tương ứng mà được join cùng với bảng chính. Giả sử có ví dụ dưới đây:

```
mysql> EXPLAIN SELECT * FROM Country,City WHERE
Country.Code=City.CountryCode AND Country.Name LIKE 'A%';

+----+-------------+---------+--------+---------------+---
-----+---------+-----------------------+------+----------
--+

| id | select_type | table   | type   | possible_keys | key
| key_len | ref                     | rows | Extra       |

+----+-------------+---------+--------+---------------+---
-----+---------+-----------------------+------+----------
--+

|  1 | SIMPLE      | City    | ALL    | NULL          |
NULL    | NULL    | NULL                    | 4079 |
|

|  1 | SIMPLE      | Country | eq_ref | PRIMARY       |
PRIMARY | 3       | world.City.CountryCode |    1 | Using
where |
```

```
+----+-------------+---------+--------+---------------+----
-----+---------+-----------------------+------+----------
--+
```

2 rows in set (0.00 sec)

Trong trường hợp trên thì ref là CountryCode chính là field được đem ra để so sánh với Country.Code.

g. rows

Đây cũng là một thông số rất quan trọng. Nó thể hiện số dòng mà mysql "dự định" sẽ fetch ra từ bảng trong query đó. Như ở ví dụ ở trên thì trong query đầu tiên sẽ lấy ra 4079 dòng, trong query thứ 2 sẽ lấy ra 1 dòng từ 4079 dòng đó.

Có một điểm chú ý là khi query thuộc type là "DERIVED" tức là nó sẽ là một subquery nằm trong FROM statement, thì khi đó nếu không execute query thì mysql sẽ không thể nào "đoán" được số dòng cần lấy ra. Do vậy khi đó EXPLAIN sẽ khá là tốn thời gian nếu subquery đó nặng.

h. extra

Đây cũng là một thông số rất quan trọng để tuning mysql query. Biến này thể hiện optimizer sẽ thực hiện chiến lược thế nào để thực thi query đó. Chỉ cần nhìn qua extra thì bạn sẽ phần nào "đoán được" chuyện gì sẽ xảy ra đằng sau một query nào đó.

Biến này sẽ có những loại sau:

Using where .... Loại này có tần suất xuất hiện khá nhiều. Khi loại này xuất hiện thì tức là query của chúng ta có chỉ định điều kiện "Where", tuy nhiên để match điều kiện chỉ định thì chúng ta không thể chỉ nhìn vào index mà phải nhìn vào cả các

thông tin khác nữa. (do đó mà tất nhiên tốc độ xử lý sẽ chậm hơn khi mà chỉ cần nhìn vào index cũng có được thông tin)

Using index ... Khi mà chỉ cần nhìn vào index cũng có thể lấy về được thông tin cần thiết.

Using filesort ... Khi trong query của chúng ta có order by chẳng hạn thì thông tin lấy về sẽ cần phải sort. Using filesort nói lên điều đó. (filesort còn nói lên một vài điều nữa nhưng không nằm trong scope bài này nên mình sẽ tạm bỏ qua)

Using temporary ... Khi mà trong query chúng ta phải sort kết quả của JOIN , hay là trong query có sử dụng distinct thì mysql sẽ phải tạo "bảng tạm" để thực hiện việc này.

Using index for group by ... Khi trong query có MIN() , MAX(), GROUPBY() nhưng mà vẫn chỉ cần nhìn vào index là tìm được thông tin cần thiết.

Range checked for each record (index map: N) ... Khi trong query có JOIN mà range hoặc là index_merge được sử dụng

Not exists Khi query có LEFT JOIN, nhưng field của bảng bên phải của LEFT JOIN lại được quy định là NOT NULL.

h. key_len

Chiều dài của khóa mà trình tối ưu truy vấn (Query Optimizer) sử dụng. Ví dụ, key_len mang giá trị 4 có nghĩa là nó cần bộ nhớ để lưu 4 ký tự.

Nguồn:

1. Learning MySQL - Seyed M. M. Tahaghoghi, 2006
2. Ruby on Rails Tutorial - 6th edition - Michael Hartl