Social

twitter (https://twitter.com/weigon/)

github (https://github.com/weigon/)

ORDER BY RAND() (//jan.kneschke.de/projects/mysql/order-by-rand/)

Date Do 15 Februar 2007 **Tags** mysql (//jan.kneschke.de/tags/mysql/)

If you read the MySQL manual you might have seen the ORDER BY RAND() to randomize the the rows and using the LIMIT 1 to just take one of the rows.

```
SELECT name
  FROM random
 ORDER BY RAND()
 LIMIT 1;
```

This example works fine and is fast if you only when let's say 1000 rows. As soon as you have 10000 rows the overhead for sorting the rows becomes important. Don't forget: we only sort to throw nearly all the rows away.

I never liked it. And there are better ways to do it. Without a sorting. As long as we have a numeric primary key.

For the first examples we assume the be ID is starting at 1 and we have no holes between 1 and the maximum value of the ID.

move the work into the application

First idea: We can simplify the whole job if we calculate the ID beforehand in the application.

```
SELECT MAX(id) FROM random;
## generate random id in application
SELECT name FROM random WHERE id = <random-id>
```

As MAX(id) == COUNT(id) we just generate random number between 1 and MAX(id) and pass it into the database to retrieve the random row.

The first SELECT is a NO-OP and is optimized away. The second is a `eq_ref` against a constant value and also very fast.

move the job into the database

But is it really necessary to do it in the application ? Can't we do it in the database ?

```
# generating a random ID
> SELECT RAND() * MAX(id) FROM random;
+-------------------+
| RAND() * MAX(id) |
+-------------------+
|  689.37582507297 |
+-------------------+
# oops, this is a double, we need an int

> SELECT CEIL(RAND() * MAX(id)) FROM random;
+-------------------------+
| CEIL(RAND() * MAX(id)) |
+-------------------------+
|                  1000000  |
+-------------------------+
# better. But how is the performance:

> EXPLAIN
   SELECT CEIL(RAND() * MAX(id)) FROM random;
+----+-------------+-------+-------+------+-------------+
| id | select_type | table | type  | rows | Extra       |
+----+-------------+-------+-------+------+-------------+
|  1 | SIMPLE      | random  | index | 1000000  | Using index |
+----+-------------+-------+-------+------+-------------+
## a index scan ? we lost our optimization for the MAX()

> EXPLAIN
   SELECT CEIL(RAND() * (SELECT MAX(id) FROM random));
+----+-------------+-------+------+------+-------------------------
| id | select_type | table | type | rows | Extra
+----+-------------+-------+------+------+-------------------------
|  1 | PRIMARY     | NULL  | NULL | NULL | No tables used
|  2 | SUBQUERY    | NULL  | NULL | NULL | Select tables optimized a
+----+-------------+-------+------+------+-------------------------
## a simple Sub-Query is bringing us the performance back.
```

Ok, now we know how to generate the random ID, but how to get the row ?

```
> EXPLAIN
SELECT name
  FROM random
 WHERE id = (SELECT CEIL(RAND() *
                         (SELECT MAX(id)
                            FROM random));
+----+-------------+--------+------+---------------+------+---------
| id | select_type | table  | type | possible_keys | key  | key_len
+----+-------------+--------+------+---------------+------+---------
|  1 | PRIMARY     | random | ALL  | NULL          | NULL | NULL
|  3 | SUBQUERY    | NULL   | NULL | NULL          | NULL | NULL
+----+-------------+--------+------+---------------+------+---------
> show warnings;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+-----------------------------------------+
```

NO, NO, NO. Don't go this way. This is the most obvious, but also the most wrong way to do it. The reason: the SELECT in the WHERE clause is executed for every row the outer SELECT is fetching. This leads to 0 to 4091 rows, depending on your luck.

We need a way to make sure that the random-id is only generated once:

```
SELECT name
  FROM random JOIN
       (SELECT CEIL(RAND() *
                    (SELECT MAX(id)
                       FROM random)) AS id
        ) AS r2
       USING (id);
+----+-------------+-----------+--------+------+-------------------
| id | select_type | table     | type   | rows | Extra
+----+-------------+-----------+--------+------+-------------------
|  1 | PRIMARY     | <derived2> | system |    1 |
|  1 | PRIMARY     | random    | const  |    1 |
|  2 | DERIVED     | NULL      | NULL   | NULL | No tables used
|  3 | SUBQUERY    | NULL      | NULL   | NULL | Select tables opti
+----+-------------+-----------+--------+------+-------------------
```

The inner SELECT is generating a constant TEMPORARY table and the JOIN is selecting just on single row. Perfect.

No Sorting, No Application, Most parts of the query optimized away.

adding holes to the numbers

To generalize the last solution we add the possibility of holes, like when you `DELETE` rows.

```
SELECT name
  FROM random AS r1 JOIN
       (SELECT (RAND() *
                     (SELECT MAX(id)
                        FROM random)) AS id)
        AS r2
 WHERE r1.id >= r2.id
 ORDER BY r1.id ASC
 LIMIT 1;
+----+-------------+------------+--------+------+------------------
| id | select_type | table      | type   | rows | Extra
+----+-------------+------------+--------+------+------------------
|  1 | PRIMARY     | <derived2> | system |    1 |
|  1 | PRIMARY     | r1         | range  |  689 | Using where
|  2 | DERIVED     | NULL       | NULL   | NULL | No tables used
|  3 | SUBQUERY    | NULL       | NULL   | NULL | Select tables opti
+----+-------------+------------+--------+------+------------------
```

The JOIN now adds all the IDs which are greater or equal than our random value and selects only the direct neighboor if a direct match is not possible. BUT as soon as one row is found, we stop (the `LIMIT 1`). And we read the rows according to the index (`ORDER BY id ASC`). As we are using `>=` instead of a `=` we can get rid of a the `CEIL` and get the same result with a little less work.

Equal Distribution

As soon as the distribution of the IDs is not equal anymore our selection of rows isn't really random either.

```
> select * from holes;
+----+----------------------------------+----------+
| id | name                             | accesses |
+----+----------------------------------+----------+
|  1 | d12b2551c6cb7d7a64e40221569a8571 |      107 |
|  2 | f82ad6f29c9a680d7873d1bef822e3e9 |       50 |
|  4 | 9da1ed7dbbdcc6ec90d6cb139521f14a |      132 |
|  8 | 677a196206d93cdf18c3744905b94f73 |      230 |
| 16 | b7556d8ed40587a33dc5c449ae0345aa |      481 |
+----+----------------------------------+----------+
```

The `RAND` function is generating IDs like 9 to 15 which all lead to the id 16 to be selected as the next higher number.

There is no real solution for this problem, but your data is mostly constant you can add a mapping table which maps the row-number to the id:

```
> create table holes_map ( row_id int not NULL primary key, random_i
> SET @id = 0;
> INSERT INTO holes_map SELECT @id := @id + 1, id FROM holes;
> select * from holes_map;
+--------+-----------+
| row_id | random_id |
+--------+-----------+
|      1 |         1 |
|      2 |         2 |
|      3 |         4 |
|      4 |         8 |
|      5 |        16 |
+--------+-----------+
```

The `row_id` is now free of holes again and we can run our random query again:

```
SELECT name FROM holes
  JOIN (SELECT r1.random_id
         FROM holes_map AS r1
         JOIN (SELECT (RAND() *
                      (SELECT MAX(row_id)
                         FROM holes_map)) AS row_id)
               AS r2
        WHERE r1.row_id >= r2.row_id
        ORDER BY r1.row_id ASC
        LIMIT 1) as rows ON (id = random_id);
```

After 1000 fetches we see a equal distribution again:

```
> select * from holes;
+----+----------------------------------+----------+
| id | name                             | accesses |
+----+----------------------------------+----------+
|  1 | d12b2551c6cb7d7a64e40221569a8571 |      222 |
|  2 | f82ad6f29c9a680d7873d1bef822e3e9 |      187 |
|  4 | 9da1ed7dbbdcc6ec90d6cb139521f14a |      195 |
|  8 | 677a196206d93cdf18c3744905b94f73 |      207 |
| 16 | b7556d8ed40587a33dc5c449ae0345aa |      189 |
+----+----------------------------------+----------+
```

Maintaining the holes

Let's take the tables as before:

```
DROP TABLE IF EXISTS r2;
CREATE TABLE r2 (
  id SERIAL,
  name VARCHAR(32) NOT NULL UNIQUE
);

DROP TABLE IF EXISTS r2_equi_dist;
CREATE TABLE r2_equi_dist (
  id SERIAL,
  r2_id bigint unsigned NOT NULL UNIQUE
);
```

When ever we change something in **r2** we want to that **r2_equi_dist** is updated too.

```
DELIMITER $$
DROP TRIGGER IF EXISTS tai_r2$$
CREATE TRIGGER tai_r2
 AFTER INSERT ON r2 FOR EACH ROW
BEGIN
  DECLARE m BIGINT UNSIGNED DEFAULT 1;

  SELECT MAX(id) + 1 FROM r2_equi_dist INTO m;
  SELECT IFNULL(m, 1) INTO m;
  INSERT INTO r2_equi_dist (id, r2_id) VALUES (m, NEW.id);
END$$
DELIMITER ;

DELETE FROM r2;

INSERT INTO r2 VALUES ( NULL, MD5(RAND()) );
INSERT INTO r2 VALUES ( NULL, MD5(RAND()) );
INSERT INTO r2 VALUES ( NULL, MD5(RAND()) );
INSERT INTO r2 VALUES ( NULL, MD5(RAND()) );

SELECT * FROM r2;
+----+----------------------------------+
| id | name                             |
+----+----------------------------------+
|  1 | 8b4cf277a3343cdefbe19aa4dabc40e1 |
|  2 | a09a3959d68187ce48f4fe7e388926a9 |
|  3 | 4e1897cd6d326f8079108292376fa7d5 |
|  4 | 29a5e3ed838db497aa330878920ec01b |
+----+----------------------------------+
SELECT * FROM r2_equi_dist;
+----+-------+
| id | r2_id |
+----+-------+
|  1 |     1 |
|  2 |     2 |
|  3 |     3 |
|  4 |     4 |
+----+-------+
```

INSERT is quite simple, on **DELETE** we have to update the equi-dist-id to maintain the hole-free setup:

```
DELIMITER $$
DROP TRIGGER IF EXISTS tad_r2$$
CREATE TRIGGER tad_r2
 AFTER DELETE ON r2 FOR EACH ROW
BEGIN
  DELETE FROM r2_equi_dist WHERE r2_id = OLD.id;
  UPDATE r2_equi_dist SET id = id - 1 WHERE r2_id > OLD.id;
END$$
DELIMITER ;

DELETE FROM r2 WHERE id = 2;

SELECT * FROM r2;
+----+----------------------------------+
| id | name                             |
+----+----------------------------------+
|  1 | 8b4cf277a3343cdefbe19aa4dabc40e1 |
|  3 | 4e1897cd6d326f8079108292376fa7d5 |
|  4 | 29a5e3ed838db497aa330878920ec01b |
+----+----------------------------------+
SELECT * FROM r2_equi_dist;
+----+-------+
| id | r2_id |
+----+-------+
|  1 |     1 |
|  2 |     3 |
|  3 |     4 |
+----+-------+
```

UPDATE is straight-forward again. We only have to maintain the Foreign Key constraint:

```
DELIMITER $$
DROP TRIGGER IF EXISTS tau_r2$$
CREATE TRIGGER tau_r2
 AFTER UPDATE ON r2 FOR EACH ROW
BEGIN
  UPDATE r2_equi_dist SET r2_id = NEW.id WHERE r2_id = OLD.id;
END$$
DELIMITER ;

UPDATE r2 SET id = 25 WHERE id = 4;

SELECT * FROM r2;
+----+----------------------------------+
| id | name                             |
+----+----------------------------------+
|  1 | 8b4cf277a3343cdefbe19aa4dabc40e1 |
|  3 | 4e1897cd6d326f8079108292376fa7d5 |
| 25 | 29a5e3ed838db497aa330878920ec01b |
+----+----------------------------------+
SELECT * FROM r2_equi_dist;
+----+-------+
| id | r2_id |
+----+-------+
|  1 |     1 |
|  2 |     3 |
|  3 |    25 |
+----+-------+
```

Multiple Rows at once

If you want to get more than one row returned, you can:

- execute the Query several times
- write a stored procedure which is executing the query and stores the result in a temp-table
- make a UNION (/projects/mysql/analyzing-complex-queries)

a stored procedure

Stored procedures provide you with the structures you know from your favourite programming language:

- loops
- control structures
- procedures
- ...

For this task we only need a LOOP:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS get_rands$$
CREATE PROCEDURE get_rands(IN cnt INT)
BEGIN
  DROP TEMPORARY TABLE IF EXISTS rands;
  CREATE TEMPORARY TABLE rands ( rand_id INT );

loop_me: LOOP
    IF cnt < 1 THEN
      LEAVE loop_me;
    END IF;

    INSERT INTO rands
       SELECT r1.id
         FROM random AS r1 JOIN
              (SELECT (RAND() *
                            (SELECT MAX(id)
                               FROM random)) AS id)
               AS r2
        WHERE r1.id >= r2.id
        ORDER BY r1.id ASC
        LIMIT 1;

    SET cnt = cnt - 1;
  END LOOP loop_me;
END$$
DELIMITER ;

CALL get_rands(4);
SELECT * FROM rands;
+---------+
| rand_id |
+---------+
|  133716 |
|  702643 |
|  112066 |
|  452400 |
+---------+
```

I leave to the reader to fix the issues:

- use dynamic SQL and pass in the name of the temporary table
- use a UNIQUE index on the table and catch the UNIQUE key violation to remove the possible duplicates in the result-set

Performance

Now let's see what happends to our performance. We have 3 different queries for solving our problems.

- Q1. ORDER BY RAND()
- Q2. RAND() * MAX(ID)
- Q3. RAND() * MAX(ID) + ORDER BY ID

Q1 is expected to cost N * log2(N), Q2 and Q3 are nearly constant.

The get real values we filled the table with N rows (one thousand to one million) and executed each query 1000 times.

```
   100        1.000      10.000     100.000    1.000.000
Q1  0:00.718s  0:02.092s  0:18.684s  2:59.081s  58:20.000s
Q2  0:00.519s  0:00.607s  0:00.614s  0:00.628s   0:00.637s
Q3  0:00.570s  0:00.607s  0:00.614s  0:00.628s   0:00.637s
```

As you can see the plain ORDER BY RAND() is already behind the optimized query at only 100 rows in the table.

A more detailed analysis of those queries is at analyzing-complex-queries (/projects/mysql/analyzing-complex-queries).

Comments

22 comments

Aleix
December 06 2017, 15:17
Hi!

Thank you for the explanation, it was very helpful.

But I want to raise a doubt, the RAND function can return a value between 0 (inclusive) and 1 (exclusive). In your code, if RAND returns 0 there will be an error because CEIL(0) is 0 and there is no row with id = 0.

To solve this, I think that will be better to use FLOOR and a +1 to counter the 0 instead of CEIL:

```
SELECT name
  FROM random JOIN
   (SELECT FLOOR(RAND() *
                (SELECT MAX(id)
                   FROM random)+1) AS id
    ) AS r2
   USING (id);
```

Reply to this comment

Sugi
March 27 2017, 09:49

How to select a row from random and it has condition?

for ex i have 4 columns, i want to select value $column*a if $clolumn*c = 'grabed' ? thanks

Reply to this comment

Anonymous
October 23 2015, 04:01
Guillem sola wrote:

> You have a point but, obviously this only works with numeric sequences with no gaps.
>
> No very useful then as this doesn't work for tables with string codes, guid, geographic positions...

Just add an id column

Reply to this comment

Rick James
June 06 2015, 18:20
8 techniques for picking random row(s). Each may or may not be usable in any given situation.

```
http://mysql.rjweb.org/doc.php/random
```

(There is some overlap with this blog.)

Reply to this comment

Guillem sola
July 19 2014, 08:06
You have a point but, obviously this only works with numeric sequences with no gaps.

No very useful then as this doesn't work for tables with string codes, guid, geographic positions...

Reply to this comment

Anonymous
April 28 2014, 19:13
Why This works:

```
 SET @t = CEIL(RAND()*(SELECT MAX(id) FROM logo));
 SELECT id FROM logo WHERE logo_type_id=4 AND id>=@t ORDER BY id L
IMIT 1;
```

and gives random results, but this always returns the same 4 or 5 results ?

```
SELECT id FROM logo WHERE logo_type_id=4 AND id>=CEIL(RAND()*(SELE
CT MAX(id) FROM logo)) ORDER BY id LIMIT 1;
```

Reply to this comment

Anonymous
February 19 2014, 17:08
You say in section 'Equal Distributions' that 'There is no real solution for this problem' and then you present a real solution, or am I wrong?

Reply to this comment

mab
November 22 2013, 01:34
Teo wrote:

> One thing I don't get. Why do we need to generate a random ID at all??? While can't we get the number of rows (rather than max(id)), generate a random POSITION (rather than a random id), and then use that position as the first argument of LIMIT (being 1 the second argument) ??? Am I missing something?

Did that once. Works quite well, even on not uniformly distibuted ID values, but only if you want to fetch a single row. Still, you need to generate one random number per row either way.

Reply to this comment

mab
November 22 2013, 01:28
First of all, sry I could not get the comment syntax right.

Thank you so much for this post! Really appreciate it. I wanted to point out that there has been some variation of your code at http://stackoverflow.com/questions/4329396/mysql-select-10-random-rows-from-600k-rows-fast (http://stackoverflow.com/questions/4329396/mysql-select-10-random-rows-from-600k-rows-fast)

The improvement in dealing with gaps in the IDs mentioned there (by bogdan) looks like this:

```
 DELIMITER $$ DROP PROCEDURE IF EXISTS get_rands$$ CREATE PROCEDURE
get_rands(IN cnt INT) BEGIN
```

```
`DROP TEMPORARY TABLE IF EXISTS rands;
CREATE TEMPORARY TABLE rands ( rand_id INT );

loop_me: LOOP
    IF cnt < 1 THEN
      LEAVE loop_me;
    END IF;

    SET @no_gaps_id := 0;

    INSERT INTO rands
       SELECT r1.id
         FROM (SELECT id, @no_gaps_id := @no_gaps_id + 1 AS no_gap
s_id FROM random) AS r1 JOIN
              (SELECT (RAND() *
                            (SELECT COUNT(*)
                               FROM random)) AS id)
               AS r2
        WHERE r1.no_gaps_id >= r2.id
        ORDER BY r1.no_gaps_id ASC
        LIMIT 1;

    SET cnt = cnt - 1;
END LOOP loop_me;`
```

END$$ DELIMITER ;

Reply to this comment

netsearch
October 16 2013, 15:10
Hi

I implemented this solution but it is not working for my case:

I run several instances of the same software that process db rows.

Not it seems every instance loads the same row as the others - so it is not random for processes running at the same time.

Each instance should row a different random row than the other instances - if not they all work on the same row.

Do you have a solution for that?

Thanks

Reply to this comment

BamDaa
October 02 2013, 06:51
My Version.

SET @foo = (SELECT CEIL(RAND() * COUNT(id)) FROM table) ; PREPARE STMT FROM 'SELECT * FROM table LIMIT ?, 1'; EXECUTE STMT USING @foo;

Reply to this comment

Anonymous
June 27 2013, 04:13
Just stumbled on this post, and it gave me...

a) the answer to my question (fast way to pick a random row)

b) In depth explanation of how the queries work

c) Breakdowns of WHY certain things are more efficient

d) Details proving the different efficiency claims

As a coder it just made my day to find this (within only a minute of searching!), something so helpful and informative. I wish all my questions could be answered so quickly and thoroughly. Thanks Jan!

Reply to this comment

holo
June 19 2013, 18:16
Hi Is it really necessary to create a table to map holes and keep randomness equally distributed? Can't we just use the "virtual" column row_id and select a random position from there ?

Cheers

Reply to this comment

Matt
June 08 2013, 08:41
@Miltos Tereres You're right, triggers might be performance nightmare on frequently updated tables. If You need only approximately random values consider using MySQL Event scheduler/similar solution in other DBMS/cron job to maintain the holes table every X minutes/hours/days/months.

Reply to this comment

jack

April 26 2013, 00:31

@Anonymous, let's see if this works for you: create a table "selected" having a single primary key column "id". Then add a subquery at the end: ... and r1.id not in (select id from selected.id)

Reply to this comment

Teo

March 30 2013, 20:13

One thing I don't get. Why do we need to generate a random ID at all??? While can't we get the number of rows (rather than max(id)), generate a random POSITION (rather than a random id), and then use that position as the first argument of LIMIT (being 1 the second argument) ??? Am I missing something?

Reply to this comment

Teo

March 30 2013, 20:12

One thing I don't get. Why do we need to generate a random ID at all??? While can't we get the number of rows (rather than max(id)), generate a random POSITION (rather than a random id), and then use that position as the first argument of LIMIT (being 1 the second argument) ??? Am I missing something?

Reply to this comment

Anonymous

March 19 2013, 13:54

These seem like interesting solutions, however, my goal is not only to select the row, but to also mark it as "selected", so the next time I get a random row, I want to filter out all "selected" rows. I tried adding this to the where clause, but if I understand correctly, this suffers from "holes". I can't use triggers like in most hosted DBMS solutions, so is there a way to address this?

Reply to this comment

roger
March 06 2013, 20:43
I suppose you could keep some "statistics" about your table, like the stddev of the gap between entries, and then use that to come up with some formula that could do a reasonably equal distribution somehow with a few passes?

Reply to this comment

Miltos Tereres
October 30 2012, 11:08
This seems nice but i worry about delete. If a have 500k rows and i have some deletes it would take forever to complete and would that lock the table or something ?

Reply to this comment

fenway
February 21 2007, 16:23

Why isn't it possible to simply pick a bunch of random leaf nodes from the primary key index, for example? Wouldn't that be *MUCH* less expensive, at least for MyISAM tables?

Reply to this comment

dudus

February 21 2007, 13:56

How hard is it to make the query preprocessor rewrite the 'order by rand() limit 1' query using one of this solutions?

Reply to this comment

Add comment

Your name (optional)

Your email (optional) [?]

Submit

Preview

Comments can be styled with Markdown (http://daringfireball.net/projects/markdown/basics). Type something and a preview will show up here.
This commenting system is powered by Juvia (https://github.com/phusion/juvia).

Back to top