

How to request a random row in SQL?

Asked 16 years, 4 months ago Modified 10 months ago

Viewed 604k times



How can I request a random row (or as close to truly random as possible) in pure SQL?

618



sql

random



Share

edited Apr 28, 2023 at 22:11



Improve this question

Follow

asked Aug 21, 2008 at 6:28



sverre joh

16.8k ● 13 ● 42 ● 29

i used to always do this in php after query results from sql... this is probably a lot faster for processing as per the solution's limit 1 appendage – [CheeseConQueso](#) Dec 23, 2009 at 20:11

[If SQL Server you can use a CLR aggregate to avoid unnecessary sorts](#) – [Martin Smith](#) Mar 20, 2011 at 15:30

- 4 It seems that there is no "pure SQL" solution which run on every dbms... there is a solution for each of them. – [Manu](#) Aug 5, 2014 at 16:11

Performance version:

stackoverflow.com/questions/4329396/...

– [Ciro Santilli OurBigBook.com](#) Sep 7, 2015 at 19:38

The most maintainable and readable solution for selecting a random row is very different depending on SQL dialect unfortunately. – [moo](#) Apr 28, 2023 at 18:48

32 Answers

Sorted by:

Highest score (default)



1

2

Next



886



See this post: [SQL to Select a random row from a database table](#). It goes through methods for doing this in MySQL, PostgreSQL, Microsoft SQL Server, IBM DB2 and Oracle (the following is copied from that link):

Select a random row with MySQL:

```
SELECT column FROM table
ORDER BY RAND()
LIMIT 1
```

Select a random row with PostgreSQL:

```
SELECT column FROM table
ORDER BY RANDOM()
LIMIT 1
```

Select a random row with Microsoft SQL Server:

```
SELECT TOP 1 column FROM table
ORDER BY NEWID()
```

Select a random row with IBM DB2

```
SELECT column, RAND() as IDX
FROM table
ORDER BY IDX FETCH FIRST 1 ROWS ONLY
```

Select a random record with Oracle:

```
SELECT column FROM
( SELECT column FROM table
ORDER BY dbms_random.value )
WHERE rownum = 1
```

Share Improve this answer

edited Aug 21, 2013 at 16:39

Follow

answered Aug 21, 2008 at 6:32



Yaakov Ellis

41.4k ● 27 ● 134 ● 184


38 -1 for relying on `order by rand()` or equivalents in all dbs :|. [also mentioned here](#). – AD7six May 26, 2014 at 9:27

28 Ten years ago [some guy said](#) that using `ORDER BY RAND()` is wrong... – trejder Jun 23, 2015 at 6:42

2 `ORDER BY NEWID()` seems to be markedly slower on SQL Server. My query looks like: select top 1000 C.CustomerId, CL.LoginName from Customer C inner join LinkedAccount LA on C.CustomerId=LA.CustomerId inner join

CustomerLogin CL on C.CustomerId=CL.CustomerId group by C.CustomerId, CL.LoginName having count(*)>1 order by NEWID() Removing the "order by NEWID()" line returns results much faster. – [Ben Power](#) Aug 26, 2015 at 23:02

5 For SQLite use RANDOM() function. – [Volodymyr Chumak](#) Oct 22, 2015 at 22:34

14 These solutions don't scale. They are $O(n)$ with n being the number of records in the table. Imagine you have 1 million records, do you really want to generate 1 million random numbers or unique ids? I'd rather use `COUNT()` and involve that in a new `LIMIT` expression with a single random number. – [Christian Hujer](#) Oct 2, 2016 at 11:35 



Solutions like Jeremies:

186

```
SELECT * FROM table ORDER BY RAND() LIMIT 1
```



work, but they need a sequential scan of all the table (because the random value associated with each row needs to be calculated - so that the smallest one can be determined), which can be quite slow for even medium sized tables. My recommendation would be to use some kind of indexed numeric column (many tables have these as their primary keys), and then write something like:

```
SELECT * FROM table WHERE num_value >= RAND() *  
    ( SELECT MAX (num_value ) FROM table )  
ORDER BY num_value LIMIT 1
```

This works in logarithmic time, regardless of the table size, if `num_value` is indexed. One caveat: this assumes that `num_value` is equally distributed in the range `0..MAX(num_value)`. If your dataset strongly deviates from this assumption, you will get skewed results (some rows will appear more often than others).

Share Improve this answer

edited Dec 31, 2014 at 13:07

Follow

answered Aug 21, 2008 at 6:37




Grey Panther

13.1k ● 7 ● 49 ● 67

8 The second suggestion is not random. You can't predict the row that's going to be picked, but if you had to bet, you'd bet on the second row. And you'd never bet on the last row, it's the less likely to be picked whatever is the distribution of your `num_value` and how big your table is. – [Etienne Racine](#) Nov 14, 2010 at 17:38

1 I know that usually `RAND()` functions are not of very high quality, but other than that can you please elaborate why the selection wouldn't be random? – [Grey Panther](#) Nov 16, 2010 at 11:43

13 The first one is WRONG in SQL Server. The `RAND()` function is invoked only once per query not once per row. So it always selects the first row (try it). – [Jeff Walker Code Ranger](#) Feb 8, 2012 at 21:49 

3 The second one also assumes that all of the rows are accounted for: it's possible it will choose a row that has been deleted. – [Sam Rueby](#) Feb 20, 2012 at 13:56

- 3 @Sam.Rueby Actually, `num_value >= RAND()` ... limit 1 ensures that empty rows will be skipped until it finds existing row. – [ghord](#) Jul 22, 2012 at 13:51
-



I don't know how efficient this is, but I've used it before:

72

```
SELECT TOP 1 * FROM MyTable ORDER BY newid()
```



Because GUIDs are pretty random, the ordering means you get a random row.



Share Improve this answer

edited Apr 30, 2015 at 8:52

Follow



[Jason Van Der Meijden](#)

155 ● 15

answered Aug 21, 2008 at 6:30



[Matt Hamilton](#)

204k ● 61 ● 392 ● 321

-
- 1 I'm using MS SQL server, `SELECT TOP 1 * FROM some_table_name ORDER BY NEWID()` worked great for me, thanks for the advice guys! – [user471414](#) Oct 10, 2010 at 8:12

That's exactly the same thing as `ORDER BY RAND() LIMIT 1` – [Ken Bloom](#) Dec 2, 2010 at 5:04

-
- 6 This is also very database specific since it uses `TOP 1` and `newid()`. – [Gray](#) Feb 8, 2011 at 15:02

-
- 13 This is a bad idea. This method will not use an index unless each column is indexed individually. Table with 100million

records could take a very long time to get one record.

– user1853517 Dec 13, 2012 at 19:14

3 @Switch and what solution would you propose?

– Akmal Salikhov Aug 9, 2018 at 8:02 



```
ORDER BY NEWID()
```

40

takes 7.4 milliseconds



```
WHERE num_value >= RAND() * (SELECT MAX(num_value) FROM
```



takes 0.0065 milliseconds!

I will definitely go with latter method.

Share Improve this answer

Follow

edited Jul 28, 2013 at 18:51



Rob Hruska

120k ● 32 ● 169 ● 192

answered Dec 21, 2010 at 7:57



Neel

409 ● 4 ● 2

4 The second option won't pick the last row. I don't know why - just pointing it out. – Saturn Oct 8, 2014 at 23:31

11 @Voldemort: `rand()` returns a floating-point number `n` where `0 < n < 1`. Assuming `num_value` is an integer, the return value of `rand() * max(num_value)` will also be coerced to an integer, thus truncating anything after the decimal point. Hence, `rand() * max(num_value)` will

always be less than `max(num_value)` , which is why the last row will never be selected. – [Ian Kemp](#) Feb 18, 2015 at 14:57

I will not be efficient if my data are deleted often - if I find a gap, I'll have to rerun the whole query. – [Loic Coenen](#) May 22, 2017 at 5:26

- 1 @IanKemp Stupid question, so then why not simply use `SELECT MAX(num_value) + 1` ?? Since `rand` (or `RANDOM` in most cases) returns `[0,1)`, you'll get the full range of values. Also, yeah, you're right, got to fix a query. – [tekHedd](#) Dec 25, 2018 at 0:32 ✎

@IanKemp @Neel `rand()` returns `0 <= v < 1.0` .
dev.mysql.com/doc/refman/8.0/en/... It also says how to properly pick an integer from `0..max_val`:
`FLOOR(RAND() * max_val)` – [Curtis Yallop](#) Apr 10, 2021 at 16:24



You didn't say which server you're using. In older versions of SQL Server, you can use this:

17



```
select top 1 * from mytable order by newid()
```



In SQL Server 2005 and up, you can use `TABLESAMPLE` to get a random sample that's repeatable:



```
SELECT FirstName, LastName  
FROM Contact  
TABLESAMPLE (1 ROWS) ;
```


Follow



shA.t

16.9k ● 5 ● 57 ● 119

answered Aug 21, 2008 at 6:30



Jon Galloway

53.1k ● 25 ● 127 ● 194

-
- 9 MSDN says newid() is preferred over tablesample for truly random results: msdn.microsoft.com/en-us/library/ms189108.aspx – Andrew Hedges Nov 10, 2008 at 23:02
-
- 9 @Andrew Hedges : ORDER BY NEWID() is too costful – Andrei Rînea Nov 4, 2010 at 14:56
-
- 1 Don't use TABLESAMPLE - it returns non-precise number of rows and doesn't guarantee non-empty result when ised with (1 ROWS) – [gukoff](#) Mar 31, 2021 at 21:27
-



For SQL Server

16



newid()/order by will work, but will be very expensive for large result sets because it has to generate an id for every row, and then sort them.



TABLESAMPLE() is good from a performance standpoint, but you will get clumping of results (all rows on a page will be returned).



For a better performing true random sample, the best way is to filter out rows randomly. I found the following code sample in the SQL Server Books Online article [Limiting Results Sets by Using TABLESAMPLE](#):

If you really want a random sample of individual rows, modify your query to filter out rows randomly, instead of using TABLESAMPLE. For example, the following query uses the NEWID function to return approximately one percent of the rows of the Sales.SalesOrderDetail table:

```
SELECT * FROM Sales.SalesOrderDetail
WHERE 0.01 >= CAST(CHECKSUM(NEWID(), SalesOrderID)
                  / CAST (0x7fffffff AS int))
```

The SalesOrderID column is included in the CHECKSUM expression so that NEWID() evaluates once per row to achieve sampling on a per-row basis. The expression CAST(CHECKSUM(NEWID(), SalesOrderID) & 0x7fffffff AS float / CAST (0x7fffffff AS int) evaluates to a random float value between 0 and 1.

When run against a table with 1,000,000 rows, here are my results:

```
SET STATISTICS TIME ON
SET STATISTICS IO ON

/* newid()
   rows returned: 10000
   logical reads: 3359
   CPU time: 3312 ms
   elapsed time = 3359 ms
*/
SELECT TOP 1 PERCENT Number
```

```

FROM Numbers
ORDER BY newid()

/* TABLESAMPLE
   rows returned: 9269 (varies)
   logical reads: 32
   CPU time: 0 ms
   elapsed time: 5 ms
*/
SELECT Number
FROM Numbers
TABLESAMPLE (1 PERCENT)

/* Filter
   rows returned: 9994 (varies)
   logical reads: 3359
   CPU time: 641 ms
   elapsed time: 627 ms
*/
SELECT Number
FROM Numbers
WHERE 0.01 >= CAST(CHECKSUM(NEWID(), Number) & 0x7ffff
                  / CAST (0x7fffffff AS int))

SET STATISTICS IO OFF
SET STATISTICS TIME OFF

```

If you can get away with using TABLESAMPLE, it will give you the best performance. Otherwise use the newid()/filter method. newid()/order by should be last resort if you have a large result set.

Share Improve this answer

answered May 28, 2009 at 18:23

Follow



Rob Boek

1,963 ● 16 ● 21

This is interesting. If I'm reading this right, it starts evaluating rows, and picks off the bottom 1% of the values (≥ 0.01), so if you have 1,000,000 rows, and you pick up 10,000 of those

that match, that will be a good random sample set. However, ideally, you want to look at your count() and number of rows you want, and divide those to get the % to pick. (I.E. if you wanted just 100 rows to allow a distribution over all of them, you could do $100/1000000$ or $1/10000$ or 0.0001 if you selected 0.01 you would hit your 100 early and never get later rows. – [Traderhut Games](#) Nov 14, 2023 at 19:13



4

If possible, use stored statements to avoid the inefficiency of both indexes on RND() and creating a record number field.



```
PREPARE RandomRecord FROM "SELECT * FROM table
LIMIT ?,1";
SET @n=FLOOR(RAND()*(SELECT COUNT(*) FROM table));
EXECUTE RandomRecord USING @n;
```

Share Improve this answer

answered Jan 9, 2011 at 6:49

Follow



ldrut

3,917 ● 1 ● 18 ● 4

This solution also takes care of returning random rows when the indexed numeric value used in the where clause above is not equally distributed; so even if it takes almost the same (constant) time as using where id_value >= RAND() * MAX(id_value), it's better. – [guido](#) Feb 8, 2011 at 22:33

As far as I can tell this does not run in constant time, it runs in linear time. In the worst case, @n is equal to the number of rows in the table, and "SELECT * FROM table LIMIT ?,1" evaluates @n - 1 rows until it gets to the last one. – [Andres Riofrio](#) Sep 21, 2014 at 5:05



4

Random function from the sql could help. Also if you would like to limit to just one row, just add that in the end.



```
SELECT column FROM table
ORDER BY RAND()
LIMIT 1
```



Share Improve this answer

answered Jul 6, 2018 at 7:59

Follow



Nitin

456 ● 8 ● 20



3

Best way is putting a random value in a new column just for that purpose, and using something like this (pseude code + SQL):



```
randomNo = random()  
execSql("SELECT TOP 1 * FROM MyTable WHERE MyTable.Ran
```



This is the solution employed by the MediaWiki code. Of course, there is some bias against smaller values, but they found that it was sufficient to wrap the random value around to zero when no rows are fetched.

newid() solution may require a full table scan so that each row can be assigned a new guid, which will be much less performant.

rand() solution may not work at all (i.e. with MSSQL) because the function will be evaluated just once, and every row will be assigned the same "random" number.

Share Improve this answer

answered Aug 21, 2008 at 6:36

Follow



Ishmaeel

14.4k ● 9 ● 73 ● 84

- 1 Wrapping around when you get 0 results provides a provably random sample (not just "good enough"). This solution

almost scales to multi-row queries (think "party shuffle"). The problem is that results tend to be selected in the same groups repeatedly. To get around this, you would need to re-distribute the random numbers you have just used. You could cheat by keeping track of randomNo and setting it to max(randomness) from results, but then $p(\text{row } i \text{ on query } 1 \text{ AND row } i \text{ on query } 2) == 0$, which isn't fair. Let me do some maths, and I'll get back to you with a truly fair scheme.

– [alsuren](#) Oct 29, 2009 at 9:25



For SQL Server 2005 and 2008, if we want a random sample of individual rows (from [Books Online](#)):

3



```
SELECT * FROM Sales.SalesOrderDetail
WHERE 0.01 >= CAST(CHECKSUM(NEWID()), SalesOrderID) & 0
/ CAST (0x7fffffff AS int)
```



Share Improve this answer

answered Oct 8, 2008 at 12:56

Follow



[Santiago Cepas](#)

4,094 ● 2 ● 27 ● 31



In late, but got here via Google, so for the sake of posterity, I'll add an alternative solution.

3



Another approach is to use TOP twice, with alternating orders. I don't know if it is "pure SQL", because it uses a variable in the TOP, but it works in SQL Server 2008.



Here's an example I use against a table of dictionary words, if I want a random word.



```
SELECT TOP 1
    word
FROM (
    SELECT TOP(@idx)
        word
    FROM
        dbo.DictionaryAbridged WITH(NOLOCK)
    ORDER BY
        word DESC
) AS D
ORDER BY
    word ASC
```

Of course, @idx is some randomly-generated integer that ranges from 1 to COUNT(*) on the target table, inclusively. If your column is indexed, you'll benefit from it too. Another advantage is that you can use it in a function, since NEWID() is disallowed.

Lastly, the above query runs in about 1/10 of the exec time of a NEWID()-type of query on the same table. YYMV.

Share Improve this answer

answered Jul 20, 2010 at 2:03

Follow



[alphadogg](#)

12.9k ● 11 ● 59 ● 88



Insted of [using RAND\(\), as it is not encouraged](#), you may simply get max ID (=Max):

3



```
SELECT MAX(ID) FROM TABLE;
```




get a random between 1..Max



(=My_Generated_Random)

```
My_Generated_Random = rand_in_your_programming_lang_fu
```

and then run this SQL:

```
SELECT ID FROM TABLE WHERE ID >= My_Generated_Random 0
```

Note that it will check for any rows which Ids are EQUAL or HIGHER than chosen value. It's also possible to hunt for the row down in the table, and get an equal or lower ID than the My_Generated_Random, then modify the query like this:

```
SELECT ID FROM TABLE WHERE ID <= My_Generated_Random 0
```

Share Improve this answer

edited Mar 10, 2017 at 16:33

Follow

answered Mar 10, 2017 at 16:19



forsberg

1,893 ● 2 ● 22 ● 28

What would happen if generated random ID does not exist in table anymore? Deleted or passive rows that you don't want to show to user would cause a trouble. – Ebleme Mar 1, 2019 at 12:25

Nothing. You get the CLOSEST, not exact, id number. If you consider id=1 to be removed, exchange 1 with minimum.



3



For SQL Server and needing "a single random row"..

If not needing a true sampling, generate a random value `[0, max_rows)` and use the [ORDER BY..OFFSET..FETCH from SQL Server 2012+](#).

This is *very fast* if the `COUNT` and `ORDER BY` are over appropriate indexes - such that the data is 'already sorted' along the query lines. If these operations are covered it's a quick request and does not suffer from the *horrid scalability* of using `ORDER BY NEWID()` or similar. Obviously, this approach won't scale well on a non-indexed HEAP table.

```
declare @rows int
select @rows = count(1) from t

-- Other issues if row counts in the bigint range..
-- This is also not 'true random', although such is li
declare @skip int = convert(int, @rows * rand())

select t.*
from t
order by t.id -- Make sure this is clustered PK or IX/
offset (@skip) rows
fetch first 1 row only
```

Make sure that the appropriate transaction isolation levels are used and/or account for 0 results.

For SQL Server and needing a "general row sample" approach..

Note: This is an adaptation of the answer as found [on a SQL Server specific question about fetching a sample of rows](#). *It has been tailored for context.*

While a general sampling approach should be used with caution here, it's still potentially useful information in context of other answers (and the repetitious suggestions of non-scaling and/or questionable implementations). Such a sampling approach is less efficient than the first code shown and is error-prone if the goal is to find a "single random row".

Here is *an updated and improved form of **sampling a percentage of rows***. It is based on the same concept of some other answers that use CHECKSUM / BINARY_CHECKSUM and modulus.

- ***It is **relatively fast over huge data sets** and can be efficiently used in/with derived queries.*** Millions of pre-filtered rows can be sampled in seconds *with no tempdb usage* and, if aligned with the rest of the query, the overhead is often minimal.
- ***Does not suffer from `CHECKSUM(*)` / `BINARY_CHECKSUM(*)` issues with runs of data.*** When using the `CHECKSUM(*)` approach, the rows can be selected in "chunks" and not "random" at all! This is because ***CHECKSUM prefers speed over distribution.***

- **Results in a *stable/repeatable* row selection** and can be trivially changed to produce different rows on subsequent query executions. Approaches that use `NEWID()` can never be stable/repeatable.
- **Does not use `ORDER BY NEWID()` of the entire input set, as ordering can become a significant bottleneck with large input sets.** Avoiding unnecessary sorting also reduces memory and tempdb usage.
- **Does not use `TABLESAMPLE` and thus works with a `WHERE` pre-filter.**

Here is the gist. [See this answer for additional details and notes.](#)

Naïve try:

```
declare @sample_percent decimal(7, 4)
-- Looking at this value should be an indicator of why
-- general sampling approach can be error-prone to sel
select @sample_percent = 100.0 / count(1) from t

-- BAD!
-- When choosing appropriate sample percent of "approx
-- it is very reasonable to expect 0 rows, which defin
-- If choosing a larger sample size the distribution i
-- and is very much NOT 'true random'.
select top 1
    t.*
from t
where 1=1
    and ( -- sample
        @sample_percent = 100
        or abs(
            convert(bigint, hashbytes('SHA1', convert(
t.rowguid)))
```

```

    ) % (1000 * 100) < (1000 * @sample_percent)
)

```

This can be largely remedied by a hybrid query, by mixing sampling and `ORDER BY` selection from the *much smaller sample set*. This limits the sorting operation to the sample size, not the size of the original table.

```

-- Sample "approximately 1000 rows" from the table,
-- dealing with some edge-cases.
declare @rows int
select @rows = count(1) from t

declare @sample_size int = 1000
declare @sample_percent decimal(7, 4) = case
    when @rows <= 1000 then 100
    when (100.0 * @sample_size / @rows) < 0.0001 then
percent
    else 100.0 * @sample_size / @rows
end

-- There is a statistical "guarantee" of having sample
number of rows.
-- The limited rows are then sorted randomly before th
select top 1
    t.*
from t
where 1=1
    and ( -- sample
        @sample_percent = 100
    or abs(
        convert(bigint, hashbytes('SHA1', convert(
t.rowguid)))
    ) % (1000 * 100) < (1000 * @sample_percent)
    )
-- ONLY the sampled rows are ordered, which improves s
order by newid()

```

Follow

answered Feb 12, 2021 at 22:10



[user2864740](#)

61.8k ● 15 ● 157 ● 226



3



In [drizzle orm](#) if you want to do this then you can follow this code to show `random order`

```
db
  .select()
  .from(users)
  .orderBy(sql.raw("RANDOM()"))
  .limit(itemPerPage)
  .offset(offset)
```

This is the orderby line

```
orderBy(sql.raw("RANDOM()"))
```

Share Improve this answer

answered Jan 27 at 7:23

Follow



[Developer Sabbir](#)

363 ● 3 ● 9



2

As pointed out in @BillKarwin's comment on @cnu's answer...

When combining with a LIMIT, I've found that it performs much better (at least with PostgreSQL 9.1) to JOIN with a



random ordering rather than to directly order the actual rows: e.g.



```
SELECT * FROM tbl_post AS t
JOIN ...
JOIN ( SELECT id, CAST(-2147483648 * RANDOM() AS integer
      FROM tbl_post
      WHERE create_time >= 1349928000
    ) r ON r.id = t.id
WHERE create_time >= 1349928000 AND ...
ORDER BY r.rand
LIMIT 100
```

Just make sure that the 'r' generates a 'rand' value for every possible key value in the complex query which is joined with it but still limit the number of rows of 'r' where possible.

The CAST as Integer is especially helpful for PostgreSQL 9.2 which has specific sort optimisation for integer and single precision floating types.

Share Improve this answer

Follow

edited Oct 26, 2012 at 20:02



Blake B.

295 ● 1 ● 5 ● 13

answered Oct 12, 2012 at 4:01



karmakaze

36.1k ● 1 ● 33 ● 33



For MySQL to get random record

2



```
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
```

More detail <http://jan.kneschke.de/projects/mysql/order-by-rand/>

Share Improve this answer

answered Jul 21, 2013 at 0:34

Follow



Sophy

9,265 ● 6 ● 37 ● 31

After testing many of the answers I believe that this is the best one. It seems to be fast and picks a good random number each time. It seems similar to @GreyPanther's second suggestion above, but this answer picks more random numbers. – Jeff Baker Apr 28, 2017 at 5:30



With SQL Server 2012+ you can use the [OFFSET FETCH query](#) to do this for a single random row

2



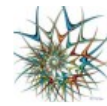
```
select * from MyTable ORDER BY id OFFSET n ROW FETCH
```

where id is an identity column, and n is the row you want - calculated as a random number between 0 and count()-1 of the table (offset 0 is the first row after all)

This works with holes in the table data, as long as you have an index to work with for the ORDER BY clause. Its also very good for the randomness - as you work that out yourself to pass in but the niggles in other methods are not present. In addition the performance is pretty good, on a smaller dataset it holds up well, though I've not tried serious performance tests against several million rows.

Share Improve this answer
Follow

answered Mar 28, 2018 at 19:41



[gbjaanb](#)

52.6k ● 12 ● 110 ● 154



```
SELECT * FROM table ORDER BY RAND() LIMIT 1
```

1

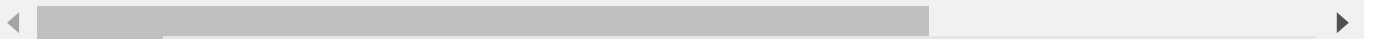
Share Improve this answer
Follow

answered Aug 21, 2008 at 6:30



[Paige Ruten](#)

176k ● 37 ● 182 ● 199



Ten years ago (2005) some guy [said](#) that using `ORDER BY RAND()` is wrong... – [trejder](#) Jun 23, 2015 at 7:09



Most of the solutions here aim to avoid sorting, but they still need to make a sequential scan over a table.

1



There is also a way to avoid the sequential scan by switching to index scan. If you know the index value of your random row you can get the result almost instantially. The problem is - how to guess an index value.

The following solution works on PostgreSQL 8.4:

```
explain analyze select * from cms_refs where rec_id in
(select (random()*(select last_value from cms_refs_r
from generate_series(1,10))
limit 1;
```

I above solution you guess 10 various random index values from range 0 .. [last value of id].

The number 10 is arbitrary - you may use 100 or 1000 as it (amazingly) doesn't have a big impact on the response time.

There is also one problem - if you have sparse ids **you might miss**. The solution is to **have a backup plan** :) In this case an pure old order by random() query. When combined id looks like this:

```
explain analyze select * from cms_refs where rec_id in
(select (random()*(select last_value from cms_refs
from generate_series(1,10))
union all (select * from cms_refs order by random(
limit 1;
```

Not the **union ALL** clause. In this case if the first part returns any data the second one is NEVER executed!

Share Improve this answer

Follow

answered Jul 2, 2009 at 13:12



hegemon

6,754 ● 2 ● 33 ● 30



1

You may also try using `new id()` function.

Just write a your query and use order by `new id()` function. It quite random.



Share Improve this answer

Follow

edited Feb 24, 2012 at 20:55



Taryn

247k ● 57 ● 370 ● 408

answered Jul 18, 2011 at 6:11



Jai - gotaninterviewcall

11 ● 1



1

Didn't quite see this variation in the answers yet. I had an additional constraint where I needed, given an initial seed, to select the same set of rows each time.



For MS SQL:

Minimum example:



```
select top 10 percent *  
from table_name  
order by rand(checksum(*))
```

Normalized execution time: 1.00

NewId() example:

```
select top 10 percent *  
from table_name  
order by newid()
```

Normalized execution time: 1.02

`NewId()` is insignificantly slower than `rand(checksum(*))`, so you may not want to use it against large record sets.

Selection with Initial Seed:

```
declare @seed int  
set @seed = Year(getdate()) * month(getdate()) /* any  
*/  
  
select top 10 percent *  
from table_name  
order by rand(checksum(*) % seed) /* any other math fu
```

If you need to select the same set given a seed, this seems to work.

Share Improve this answer

Follow

answered Jul 29, 2014 at 17:36



klyd

3,979 ● 3 ● 28 ● 35



In MSSQL (tested on 11.0.5569) using

1

```
SELECT TOP 100 * FROM employee ORDER BY CRYPT_GEN_RANDOM
```



is significantly faster than



```
SELECT TOP 100 * FROM employee ORDER BY NEWID()
```

Share Improve this answer

answered Apr 16, 2015 at 13:36

Follow



David Knight

761 ● 6 ● 12

I guess "It Depends". I did a test of both on a 10 Million row, 5.2GB table where the average row size was 530 Bytes and the NEWID() method was 51% faster and used 40% less CPU than the CRYPT_GEN_RANDOM(10) method.

– Jeff Moden Jun 17, 2023 at 18:18



For Firebird:

1

```
Select FIRST 1 column from table ORDER BY RAND()
```



Share Improve this answer

answered Nov 28, 2016 at 8:23

Follow



Luigi04

457 ● 1 ● 5 ● 14



In SQL Server you can combine TABLESAMPLE with NEWID() to get pretty good randomness and still have

1

speed. This is especially useful if you really only want 1, or a small number, of rows.



```
SELECT TOP 1 * FROM [table]
TABLESAMPLE (500 ROWS)
ORDER BY NEWID()
```

Share Improve this answer

answered Apr 18, 2017 at 0:51

Follow



[Chris Arbogast](#)

123 ● 1 ● 7

The large drawback with TABLESAMPLE is that it applies before any filtering. – [user2864740](#) Feb 12, 2021 at 23:10 ✎



I have to agree with CD-MaN: Using "ORDER BY RAND()" will work nicely for small tables or when you do your SELECT only a few times.

0



I also use the "num_value >= RAND() * ..." technique, and if I really want to have random results I have a special "random" column in the table that I update once a day or so. That single UPDATE run will take some time (especially because you'll have to have an index on that column), but it's much faster than creating random numbers for every row each time the select is run.

Share Improve this answer

answered Aug 21, 2008 at 7:20

Follow



[BlaM](#)

28.8k ● 33 ● 93 ● 106



0



Be careful because TableSample doesn't actually return a random sample of rows. It directs your query to look at a random sample of the 8KB pages that make up your row. Then, your query is executed against the data contained in these pages. Because of how data may be grouped on these pages (insertion order, etc), this could lead to data that isn't actually a random sample.

See: <http://www.mssqltips.com/tip.asp?tip=1308>

This MSDN page for TableSample includes an example of how to generate an actually random sample of data.

<http://msdn.microsoft.com/en-us/library/ms189108.aspx>

Share Improve this answer

answered May 13, 2009 at 2:52

Follow



Sean Turner

1,178 ● 2 ● 10 ● 14



0



It seems that many of the ideas listed still use ordering

However, if you use a temporary table, you are able to assign a random index (like many of the solutions have suggested), and then grab the first one that is greater than an arbitrary number between 0 and 1.

For example (for DB2):

```
WITH TEMP AS (  
  SELECT COMLUMN, RAND() AS IDX FROM TABLE)
```

```
SELECT COLUMN FROM TABLE WHERE IDX > .5  
FETCH FIRST 1 ROW ONLY
```

Share Improve this answer

answered Jan 31, 2011 at 22:35

Follow



DAVID

1

- 2 After considering this solution, I have found a fundamental flaw in my logic. This would consistently return the same small set up values, near the beginning of the table, because I assume that if there was a even distribution between 0 and 1, there is a 50% chance that the first row will meet that criteria. – DAVID Jan 31, 2011 at 22:59



A simple and efficient way from

http://akinas.com/pages/en/blog/mysql_random_row/

0



```
SET @i = (SELECT FLOOR(RAND() * COUNT(*)) FROM table);  
'SELECT * FROM table LIMIT ?, 1'; EXECUTE get_stmt USI
```



Share Improve this answer

answered Dec 18, 2013 at 14:45

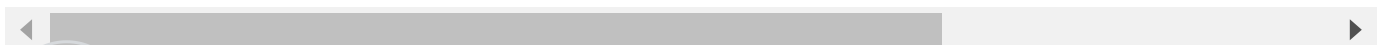


Follow



Mmmh mmh

5,460 ● 3 ● 22 ● 29



There is better solution for Oracle instead of using dbms_random.value, while it requires full scan to order rows by dbms_random.value and it is quite slow for large tables.

0



Use this instead:



```
SELECT *
FROM employee sample(1)
WHERE rownum=1
```

Share Improve this answer

answered Sep 3, 2014 at 11:00

Follow



sev3ryn

1,077 ● 1 ● 10 ● 17



0

For SQL Server 2005 and above, extending @GreyPanther's answer for the cases when `num_value` has not continuous values. This works too for cases when we have not evenly distributed datasets and when



`num_value` is not a number but a unique identifier.



```
WITH CTE_Table (SelRow, num_value)
AS
(
    SELECT ROW_NUMBER() OVER(ORDER BY ID) AS SelRow, n
)

SELECT * FROM table Where num_value = (
    SELECT TOP 1 num_value FROM CTE_Table WHERE SelRo
MAX(SelRow) FROM CTE_Table)
)
```

Share Improve this answer

answered May 28, 2018 at 8:37

Follow



Endri

804 ● 15 ● 36

1

2

Next



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.