SQL JOIN - WHERE clause vs. ON clause

Asked 12 years, 1 month ago Active 25 days ago Viewed 721k times

728

299

After reading it, this is *not* a duplicate of [Explicit vs Implicit SQL Joins](). The answer may be related (or even the same) but the **question** is different.

What is the difference and what should go in each?

If I understand the theory correctly, the query optimizer should be able to use both interchangeably.

sql join where-clause on-clause

Share Improve this question Follow

edited Apr 7 '20 at 19:06
Vlad Mihalcea
95.8k 32 400 753

asked Dec 9 '08 at 20:14
BCS
65.7k 61 175 276

2 Just for future readers and your information you should read order of sql execution. This would help you more precisely to understand the underlying difference. – Rahul Neekhra Jan 20 '17 at 12:19

19 Answers

Active Oldest Votes

938

✓

They are not the same thing.

Consider these queries:

```
SELECT *
FROM Orders
LEFT JOIN OrderLines ON OrderLines.OrderID=Orders.ID
WHERE Orders.ID = 12345
```

and

```
SELECT *
FROM Orders
LEFT JOIN OrderLines ON OrderLines.OrderID=Orders.ID
    AND Orders.ID = 12345
```

The first will return an order and its lines, if any, for order number `12345`. The second will return all orders, but only order `12345` will have any lines associated with it.

With an `INNER JOIN`, the clauses are *effectively* equivalent. However, just because they are functionally the same, in that they produce the same results, does not mean the two kinds of clauses have the same semantic meaning.

Share Improve this answer Follow

edited Aug 16 '16 at 19:06

answered Dec 9 '08 at 20:21
Joel Coehoorn
357k 103 524 761

89 will you get better performance by putting the where clause in the "on" clause for an inner join? – FistOfFury Dec 7 '12 at 16:01

103 @FistOfFury Sql Server uses a query optimizer procedure that compiles and evaluates your code to produce the best execution plan it can. It's not perfect, but most of the time it won't matter and you'll get the same execution plan either way. – Joel Coehoorn Dec 7 '12 at 17:29

21 In Postgres I noted that they were NOT equivalent and resulted in different query plans. If you use ON, it resulted in the use of materialize. If you used WHERE, it used a hash. The materialize had a worse case that was 10x more costly than the hash. This was using a set of IDs rather than a single ID. – JamesHutchison Mar 29 '16 at 17:15

15 @JamesHutchison It's tough to make reliable performance generalizations based on observed behaviors like this. What was true one day tends to be wrong the next, because this is an implementation detail rather than documented behavior. Database teams are always looking for places to improve optimizer performance. I'll be surprised if the ON behavior doesn't improve to match the WHERE. It may not even show up anywhere in release notes from version to version other than something like "general performance improvements. – Joel Coehoorn Jun 22 '16 at 15:11

4 @FiHoran That's not how Sql Server works. It will agressively pre-filter based on items from the WHERE clause when statistics show it can be helpful. – Joel Coehoorn Mar 16 '18 at 1:02

421

- Does not matter for inner joins
- Matters for outer joins

  a. `WHERE` clause: **After** joining. Records will be filtered after join has taken place.

  b. `ON` clause - **Before** joining. Records (from right table) will be filtered before joining. This may end up as null in the result (since OUTER join).

Example: Consider the below tables:

```
1. documents:
 | id    | name        |
 --------|-------------|
 | 1     | Document1   |
 | 2     | Document2   |
 | 3     | Document3   |
 | 4     | Document4   |
 | 5     | Document5   |


2. downloads:
 | id   | document_id   | username |
 |------|---------------|----------|
 | 1    | 1             | sandeep  |
 | 2    | 1             | simi     |
 | 3    | 2             | sandeep  |
 | 4    | 2             | reya     |
 | 5    | 3             | simi     |
```

a) Inside `WHERE` clause:

```
 SELECT documents.name, downloads.id
   FROM documents
   LEFT OUTER JOIN downloads
     ON documents.id = downloads.document_id
   WHERE username = 'sandeep'

For above query the intermediate join table will look like this.

    | id(from documents) | name         | id (from downloads) | document_id | username
|
    |--------------------|--------------|---------------------|-------------|----------
|
    | 1                  | Document1    | 1                   | 1           | sandeep
|
    | 1                  | Document1    | 2                   | 1           | simi
|
    | 2                  | Document2    | 3                   | 2           | sandeep
|
    | 2                  | Document2    | 4                   | 2           | reya
|
    | 3                  | Document3    | 5                   | 3           | simi
|
    | 4                  | Document4    | NULL                | NULL        | NULL
|
    | 5                  | Document5    | NULL                | NULL        | NULL
|

 After applying the `WHERE` clause and selecting the listed attributes, the result
will be:

   | name         | id |
   |--------------|----|
   | Document1    | 1  |
   | Document2    | 3  |
```

b) Inside `JOIN` clause

```
 SELECT documents.name, downloads.id
 FROM documents
   LEFT OUTER JOIN downloads
     ON documents.id = downloads.document_id
       AND username = 'sandeep'

For above query the intermediate join table will look like this.

    | id(from documents) | name         | id (from downloads) | document_id | username
|
    |--------------------|--------------|---------------------|-------------|----------
|
    | 1                  | Document1    | 1                   | 1           | sandeep
|
    | 2                 | Document2    | 3                  | 2          | sandeep
|
    | 3                 | Document3    | NULL               | NULL       | NULL
|
    | 4                 | Document4    | NULL               | NULL       | NULL
|
    | 5                 | Document5    | NULL               | NULL       | NULL
|

Notice how the rows in `documents` that did not match both the conditions are populated
with `NULL` values.

After Selecting the listed attributes, the result will be:

   | name       | id   |
   |------------|------|
   |  Document1 | 1    |
   |  Document2 | 3    |
   |  Document3 | NULL |
   |  Document4 | NULL |
   |  Document5 | NULL |
```

Share Improve this answer Follow

edited Jun 20 '20 at 9:12
Community ♦
1 1

answered Jan 7 '14 at 20:54
Sandeep Jindal
11.2k 17 76 115

50 IMO this is the best answer because it clearly demonstrates what is going on 'under the hood' of the other popular answers. – psrpsrpsr May 15 '17 at 18:51

1 Excellent explanation .... well done! - Just curious what did you do to get that `intermediate join table` ?. Some 'Explain' command? – Manuel Jordan Oct 11 '19 at 21:27

1 @ManuelJordan No, this is just for explanation. A database may do something more performant than to create an intermediate table. – Sandeep Jindal Oct 18 '19 at 0:00

1 Understood, I assumed perhaps a third tool was used. – Manuel Jordan Oct 18 '19 at 0:01

3 This is good answer with correct explanation. Still I think that it is worth mention that most (if not all) SQL servers actually does not create full intermediate table like this before applying `WHERE` conditions. They all have optimizations! And it is very important to know, because when your query contains many JOINS of tables with millions of rows, but your `WHERE` condition restricts the result set to just few rows, thinking about performance of creating this big Cartesian-product-intermediate-table just to throw out 99.9% of the resulting rows can be scary. :) And misleading. – Ruslan Stelmachenko Jul 29 '20 at 20:50

149

On `INNER JOIN` s they are interchangeable, and the optimizer will rearrange them at will.

On `OUTER JOIN` s, they are not necessarily interchangeable, depending on which side of the join they depend on.

I put them in either place depending on the readability.

Share Improve this answer Follow

answered Dec 9 '08 at 20:21
Cade Roux
82.9k 38 170 259

5 In SQL Server there is one edge case where for inner joins it does make a difference – Martin Smith Nov 2 '11 at 21:32

It's probably a lot clearer in the Where clause, especially in Linq-To-Entities lambda expressions `Orders.Join( OrderLines, x => x.ID, x => OrderID, (o,l) => new {Orders = o, Lines = l}).Where( ol => ol.Orders.ID = 12345)` – Triynko Sep 10 '15 at 4:11

57

The way I do it is:

- Always put the join conditions in the `ON` clause if you are doing an `INNER JOIN`. So, do not add any WHERE conditions to the ON clause, put them in the `WHERE` clause.
- If you are doing a `LEFT JOIN`, add any WHERE conditions to the `ON` clause for the table in the ***right*** side of the join. This is a must, because adding a WHERE clause that references the right side of the join will convert the join to an INNER JOIN.

  The exception is when you are looking for the records that are not in a particular table. You would add the reference to a unique identifier (that is not ever NULL) in the RIGHT JOIN table to the WHERE clause this way: `WHERE t2.idfield IS NULL`. So, the only time you should reference a table on the right side of the join is to find those records which are not in the table.

Share Improve this answer Follow

edited Jan 18 '19 at 21:13
Marc.2377
5,476 5 40 71

answered Dec 9 '08 at 20:57
HLGEM
87.7k 11 104 169

9 This is the best answer I have read on this so far. Totally makes sense once your brain understands a left join is **going to** return all rows in the left table and you have to filter it later. – Nick Larsen Aug 10 '13 at 20:21

if you outer join a table with a nullable column, then you can still "where" that column being null without making it an inner join? That is not exactly looking for the records that are not in a particular table only. You can look for 1. not existed 2. does not have value at all. – Near Jan 7 '20 at 12:19

I mean you can look for both : "1. not existed 2. does not have value at all " together. And this applies to case where that field is not idfield. – Near Jan 7 '20 at 12:28

As I encountered this case: looking for participants including freshman (data not yet inputted) without an emergency contact. – Near Jan 7 '20 at 12:37

31

On an inner join, they mean the same thing. However you will get different results in an outer join depending on if you put the join condition in the WHERE vs the ON clause. Take a look at this related question and this answer (by me).

I think it makes the most sense to be in the habit of always putting the join condition in the ON clause (unless it is an outer join and you actually do want it in the where clause) as it makes it clearer to anyone reading your query what conditions the tables are being joined on, and also it helps prevent the WHERE clause from being dozens of lines long.

Share Improve this answer Follow

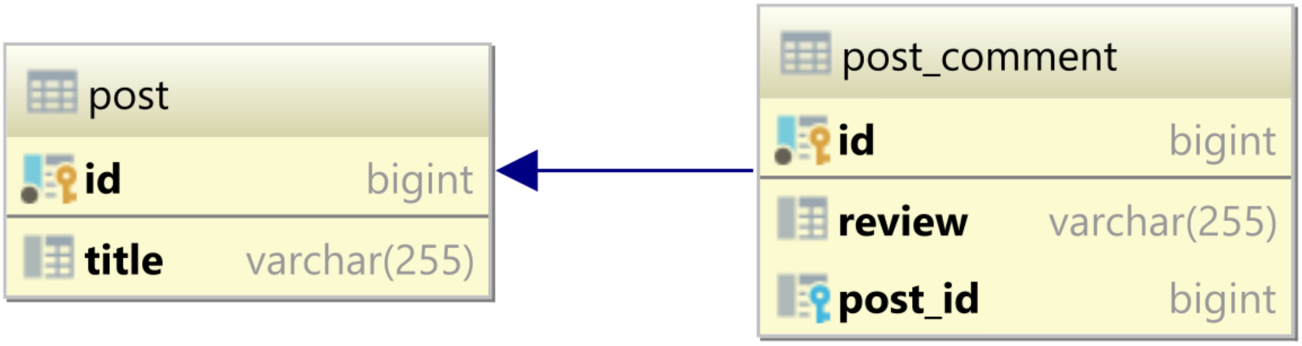
edited May 23 '17 at 12:26
Community ♦
1 1

answered Dec 9 '08 at 20:20
matt b
131k 62 267 333

28

Table relationship

Considering we have the following `post` and `post_comment` tables:



The `post` has the following records:

```
| id | title     |
|----|-----------|
| 1  | Java      |
| 2  | Hibernate |
| 3  | JPA       |
```

and the `post_comment` has the following three rows:

```
| id | review    | post_id |
|----|-----------|---------|
| 1  | Good      | 1       |
| 2  | Excellent | 1       |
| 3  | Awesome   | 2       |
```

SQL INNER JOIN

The SQL JOIN clause allows you to associate rows that belong to different tables. For instance, a CROSS JOIN will create a Cartesian Product containing all possible combinations of rows between the two joining tables.

While the CROSS JOIN is useful in certain scenarios, most of the time, you want to join tables based on a specific condition. And, that's where INNER JOIN comes into play.

The SQL INNER JOIN allows us to filter the Cartesian Product of joining two tables based on a condition that is specified via the ON clause.

SQL INNER JOIN - ON "always true" condition

If you provide an "always true" condition, the INNER JOIN will not filter the joined records, and the result set will contain the Cartesian Product of the two joining tables.

For instance, if we execute the following SQL INNER JOIN query:

```
SELECT
   p.id AS "p.id",
   pc.id AS "pc.id"
FROM post p
INNER JOIN post_comment pc ON 1 = 1
```

We will get all combinations of `post` and `post_comment` records:

```
| p.id    | pc.id      |
|---------|------------|
| 1       | 1          |
| 1       | 2          |
| 1       | 3          |
| 2       | 1          |
| 2       | 2          |
| 2       | 3          |
| 3       | 1          |
| 3       | 2          |
| 3       | 3          |
```

So, if the ON clause condition is "always true", the INNER JOIN is simply equivalent to a CROSS JOIN query:

```
SELECT
   p.id AS "p.id",
   pc.id AS "pc.id"
FROM post p
CROSS JOIN post_comment
WHERE 1 = 1
ORDER BY p.id, pc.id
```

SQL INNER JOIN - ON "always false" condition

On the other hand, if the ON clause condition is "always false", then all the joined records are going to be filtered out and the result set will be empty.

So, if we execute the following SQL INNER JOIN query:

```
SELECT
   p.id AS "p.id",
   pc.id AS "pc.id"
FROM post p
INNER JOIN post_comment pc ON 1 = 0
ORDER BY p.id, pc.id
```

We won't get any result back:

```
| p.id    | pc.id      |
|---------|------------|
```

That's because the query above is equivalent to the following CROSS JOIN query:

```
SELECT
   p.id AS "p.id",
   pc.id AS "pc.id"
FROM post p
CROSS JOIN post_comment
WHERE 1 = 0
ORDER BY p.id, pc.id
```

SQL INNER JOIN - ON clause using the Foreign Key and Primary Key columns

The most common ON clause condition is the one that matches the Foreign Key column in the child table with the Primary Key column in the parent table, as illustrated by the following query:

```
SELECT
   p.id AS "p.id",
   pc.post_id AS "pc.post_id",
   pc.id AS "pc.id",
   p.title AS "p.title",
   pc.review  AS "pc.review"
FROM post p
INNER JOIN post_comment pc ON pc.post_id = p.id
ORDER BY p.id, pc.id
```

When executing the above SQL INNER JOIN query, we get the following result set:

```
| p.id    | pc.post_id | pc.id      | p.title    | pc.review |
|---------|------------|------------|------------|-----------|
| 1       | 1          | 1          | Java       | Good      |
| 1       | 1          | 2          | Java       | Excellent |
| 2       | 2          | 3          | Hibernate  | Awesome   |
```

So, only the records that match the ON clause condition are included in the query result set. In our case, the result set contains all the `post` along with their `post_comment` records. The `post` rows that have no associated `post_comment` are excluded since they can not satisfy the ON Clause condition.

Again, the above SQL INNER JOIN query is equivalent to the following CROSS JOIN query:

```
SELECT
   p.id AS "p.id",
   pc.post_id AS "pc.post_id",
   pc.id AS "pc.id",
   p.title AS "p.title",
   pc.review  AS "pc.review"
```

```
FROM post p, post_comment pc
WHERE pc.post_id = p.id
```

The non-struck rows are the ones that satisfy the WHERE clause, and only these records are going to be included in the result set. That's the best way to visualize how the INNER JOIN clause works.

```
| p.id | pc.post_id | pc.id | p.title   | pc.review |
|------|------------|-------|-----------|-----------|
| 1    | 1          | 1     | Java      | Good      |
| 1    | 1          | 2     | Java      | Excellent |
~~| 1    | 2          | 3     | Java      | Awesome   |~~
~~| 2    | 1          | 1     | Hibernate | Good      |~~
~~| 2    | 1          | 2     | Hibernate | Excellent |~~
| 2    | 2          | 3     | Hibernate | Awesome   |
~~| 3    | 1          | 1     | JPA       | Good      |~~
~~| 3    | 1          | 2     | JPA       | Excellent |~~
~~| 3    | 2          | 3     | JPA       | Awesome   |~~
```

Conclusion

An INNER JOIN statement can be rewritten as a CROSS JOIN with a WHERE clause matching the same condition you used in the ON clause of the INNER JOIN query.

> Not that this only applies to INNER JOIN, not for OUTER JOIN.

Share Improve this answer Follow

edited Jan 8 at 22:35

answered May 25 '14 at 14:20

Vlad Mihalcea
95.8k 32 400 753

1 thanks for the answer. It's a good read but it talks about everything except the question asked here – pedram bashiri Jun 23 '20 at 18:24

12

There is great difference between ***where clause*** vs. ***on clause***, when it comes to left join.

Here is example:

```
mysql> desc t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| fid   | int(11)     | NO   |     | NULL    |       |
| v     | varchar(20) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

There fid is id of table t2.

```
mysql> desc t2;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | NO   |     | NULL    |       |
| v     | varchar(10) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Query on "on clause" :

```
mysql> SELECT * FROM `t1` left join t2 on fid = t2.id AND t1.v = 'K'
    -> ;
+----+-----+---+------+------+
| id | fid | v | id   | v    |
+----+-----+---+------+------+
|  1 |   1 | H | NULL | NULL |
|  2 |   1 | B | NULL | NULL |
|  3 |   2 | H | NULL | NULL |
|  4 |   7 | K | NULL | NULL |
|  5 |   5 | L | NULL | NULL |
+----+-----+---+------+------+
5 rows in set (0.00 sec)
```

Query on "where clause":

```
mysql> SELECT * FROM `t1` left join t2 on fid = t2.id where t1.v = 'K';
+----+-----+---+------+------+
| id | fid | v | id   | v    |
+----+-----+---+------+------+
|  4 |   7 | K | NULL | NULL |
+----+-----+---+------+------+
1 row in set (0.00 sec)
```

It is clear that, the first query returns a record from t1 and its dependent row from t2, if any, for row t1.v = 'K'.

The second query returns rows from t1, but only for t1.v = 'K' will have any associated row with it.

Share Improve this answer Follow

edited Jun 3 '17 at 15:36

answered Mar 13 '16 at 6:31

Hrishikesh Mishra

2,471 1 22 31

12

Let's consider those tables :

A

```
id | SomeData
```

B

```
id | id_A | SomeOtherData
```

`id_A` being a foreign key to table `A`

Writting this query :

```
SELECT *
FROM A
LEFT JOIN B
ON A.id = B.id_A;
```

Will provide this result :

```
/ : part of the result
                                         B
                      +---------------------------------+
            A         |                                 |
+---------------------+-------+                         |
|/////////////////////|///////|                         |
|/////////////////////|///////|                         |
|/////////////////////|///////|                         |
|/////////////////////|///////|                         |
|/////////////////////+-------+-------------------------+
|/////////////////////////////|
+-----------------------------+
```

What is in A but not in B means that there is null values for B.

Now, let's consider a specific part in `B.id_A`, and highlight it from the previous result :

```
/ : part of the result
* : part of the result with the specific B.id_A
                                         B
                      +---------------------------------+
            A         |                                 |
+---------------------+-------+                         |
|/////////////////////|///////|                         |
|/////////////////////|///////|                         |
|/////////////////////+---+///|                         |
|/////////////////////|***|///|                         |
|/////////////////////+---+---+-------------------------+
|/////////////////////////////|
+-----------------------------+
```

Writting this query :

```
SELECT *
FROM A
LEFT JOIN B
ON A.id = B.id_A
AND B.id_A = SpecificPart;
```

Will provide this result :

```
/ : part of the result
* : part of the result with the specific B.id_A
                                       B
                      +---------------------------------+
            A         |                                 |
+---------------------+-------+                         |
|/////////////////////|       |                         |
|/////////////////////|       |                         |
|/////////////////////+---+   |                         |
|/////////////////////|***|   |                         |
|/////////////////////+---+---+-------------------------+
|/////////////////////////////|
+-----------------------------+
```

Because this removes in the inner join the values that aren't in `B.id_A = SpecificPart`

Now, let's change the query to this :

```
SELECT *
FROM A
LEFT JOIN B
ON A.id = B.id_A
WHERE B.id_A = SpecificPart;
```

The result is now :

```
/ : part of the result
* : part of the result with the specific B.id_A
                                       B
                      +---------------------------------+
            A         |                                 |
+---------------------+-------+                         |
|                     |       |                         |
|                     |       |                         |
|                     +---+   |                         |
|                     |***|   |                         |
|                     +---+---+-------------------------+
|                             |
+-----------------------------+
```

Because the whole result is filtered against `B.id_A = SpecificPart` removing the parts `B.id_A IS NULL` , that are in the *A that aren't in B*

Share Improve this answer Follow

edited Aug 27 '20 at 7:10

answered Apr 12 '19 at 10:03

Cid
13.1k 4 22 42

In terms of the optimizer, it shouldn't make a difference whether you define your join clauses with

9

ON or WHERE.

However, IMHO, I think it's much clearer to use the ON clause when performing joins. That way you have a specific section of you query that dictates how the join is handled versus intermixed with the rest of the WHERE clauses.

Share Improve this answer Follow

answered Dec 9 '08 at 20:21
Grant Limberg
19.2k 10 59 83

4

Are you trying to join data or filter data?

For readability it makes the most sense to isolate these use cases to ON and WHERE respectively.

- join data in ON
- filter data in WHERE

It can become very difficult to read a query where the JOIN condition and a filtering condition exist in the WHERE clause.

Performance wise you should not see a difference, though different types of SQL sometimes handle query planning differently so it can be worth trying `¯\_(ツ)_/¯` (Do be aware of caching effecting the query speed)

Also as others have noted, if you use an outer join you will get different results if you place the filter condition in the ON clause because it only effects one of the tables.

I wrote a more in depth post about this here: https://dataschool.com/learn/difference-between-where-and-on-in-sql

Share Improve this answer Follow

answered Apr 29 '19 at 23:17
matthew david
73 8

2

In SQL, the 'WHERE' and 'ON' clause,are kind of Conditional Statemants, but the major difference between them are, the 'Where' Clause is used in Select/Update Statements for specifying the Conditions, whereas the 'ON' Clause is used in Joins, where it verifies or checks if the Records are Matched in the target and source tables, before the Tables are Joined

For Example: - 'WHERE'

```
SELECT * FROM employee WHERE employee_id=101
```

For Example: - 'ON'

There are two tables employee and employee_details, the matching columns are employee_id.

```
SELECT * FROM employee
INNER JOIN employee_details
ON employee.employee_id = employee_details.employee_id
```

Hope I have answered your Question. Revert for any clarifications.

Share Improve this answer Follow

edited Apr 12 '19 at 9:57
Onkar Musale
755 9 22

answered Feb 5 '14 at 10:57
Sharon Fernando
31 2

But you could use keyword `WHERE` in place of `ON`, couldn't you? sqlfiddle.com/#!2/ae5b0/14/0 – Qwerty Feb 26 '14 at 23:58

1

I think it's the join sequence effect. In the upper left join case, SQL do Left join first and then do where filter. In the downer case, find Orders.ID=12345 first, and then do join.

Share Improve this answer Follow

answered Jan 7 '14 at 3:49
Xing-Wei Lin
229 3 5

1

For an inner join, `WHERE` and `ON` can be used interchangeably. In fact, it's possible to use `ON` in a correlated subquery. For example:

```
update mytable
set myscore=100
where exists (
select 1 from table1
inner join table2
on (table2.key = mytable.key)
inner join table3
on (table3.key = table2.key and table3.key = table1.key)
...
)
```

This is (IMHO) utterly confusing to a human, and it's very easy to forget to link `table1` to anything (because the "driver" table doesn't have an "on" clause), but it's legal.

Share Improve this answer Follow

edited May 16 '14 at 23:05
John Paul
10.2k 6 52 68

answered May 16 '14 at 22:40
Austin Barry
11 1

1

for better performance tables should have a special indexed column to use for JOINS .

so if the column you condition on is not one of those indexed columns then i suspect it is better to keep it in WHERE .

so you JOIN using the indexed columns, then after JOIN you run the condition on the none indexed column .

Share Improve this answer Follow

answered Dec 12 '14 at 21:10
yakoub abaya
61 1 1 6

1

Normally, filtering is processed in the WHERE clause once the two tables have already been joined. It’s possible, though that you might want to filter one or both of the tables before joining them. i.e, the where clause applies to the whole result set whereas the on clause only applies to the join in question.

Share Improve this answer Follow

answered Feb 16 '18 at 5:29
sree
83 2 10

This is just not so since DBMSs "normally" optimize. – philipxy Apr 9 '19 at 14:24

@philipxy it's still an important distinction. While optimization can occur for inner joins, outer joins are semantically different and cannot be optimized this way, as they would yield different results. – Shirik Jun 24 '20 at 5:01

@Shirik It is not true that "filtering is processed in the WHERE clause once the two tables have already been joined"--unless you are talking about the "logical" "processing" that defines what a query returns rather than "processing" per optimization/implemenation--which is what the question asks about. The optimizer often evaluates parts of WHEREs in the parts of the implementation that more or less correspond to joining for both inner & outer joins. (Eg see the MySQL manual re "WHERE Clause Optimization".) – philipxy Jun 24 '20 at 5:12

1

I think this distinction can best be explained via the *logical order of operations in SQL*, which is, simplified:

- `FROM` (including joins)
- `WHERE`
- `GROUP BY`
- Aggregations
- `HAVING`
- `WINDOW`
- `SELECT`

- `DISTINCT`
- `UNION` , `INTERSECT` , `EXCEPT`
- `ORDER BY`
- `OFFSET`
- `FETCH`

Joins are not a clause of the select statement, but an operator inside of `FROM` . As such, all `ON` clauses belonging to the corresponding `JOIN` operator have "already happened" *logically* by the time logical processing reaches the `WHERE` clause. This means that in the case of a `LEFT JOIN` , for example, the outer join's semantics has already happend by the time the `WHERE` clause is applied.

I've explained the following example more in depth in this blog post. When running this query:

```
SELECT a.actor_id, a.first_name, a.last_name, count(fa.film_id)
FROM actor a
LEFT JOIN film_actor fa ON a.actor_id = fa.actor_id
WHERE film_id < 10
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY count(fa.film_id) ASC;
```

The `LEFT JOIN` doesn't really have any useful effect, because even if an actor did not play in a film, the actor will be filtered, as its `FILM_ID` will be `NULL` and the `WHERE` clause will filter such a row. The result is something like:

```
ACTOR_ID  FIRST_NAME  LAST_NAME  COUNT
--------------------------------------
194       MERYL       ALLEN      1
198       MARY        KEITEL     1
30        SANDRA      PECK       1
85        MINNIE      ZELLWEGER  1
123       JULIANNE    DENCH      1
```

I.e. just as if we inner joined the two tables. If we move the filter predicate in the `ON` clause, it now becomes a criteria for the outer join:

```
SELECT a.actor_id, a.first_name, a.last_name, count(fa.film_id)
FROM actor a
LEFT JOIN film_actor fa ON a.actor_id = fa.actor_id
  AND film_id < 10
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY count(fa.film_id) ASC;
```

Meaning the result will contain actors without any films, or without any films with `FILM_ID < 10`

```
ACTOR_ID  FIRST_NAME  LAST_NAME     COUNT
-----------------------------------------
3         ED          CHASE         0
4         JENNIFER    DAVIS         0
```

```
5         JOHNNY      LOLLOBRIGIDA  0
6         BETTE       NICHOLSON     0
...
1         PENELOPE    GUINESS       1
200       THORA       TEMPLE        1
2         NICK        WAHLBERG      1
198       MARY        KEITEL        1
```

In short

Always put your predicate where it makes most sense, logically.

Share Improve this answer Follow

answered Apr 9 '19 at 13:37
Lukas Eder
176k 109 584 1277

0

Regarding your question,

It is the same both 'on' or 'where' on an inner join as long as your server can get it:

```
select * from a inner join b on a.c = b.c
```

and

```
select * from a inner join b where a.c = b.c
```

The 'where' option not all interpreters know so maybe should be avoided. And of course the 'on' clause is clearer.

Share Improve this answer Follow

answered Jan 21 '20 at 13:23
Ezequiel
19 3

0

They **are equivalent**, literally.

In most open-source databases (most notable examples, in *MySql* and *postgresql*) the query planning is a variant of the classic algorithm appearing in *Access Path Selection in a Relational Database Management System (Selinger et al, 1979)*. In this approach, the conditions are of two types

- conditions referring to a single table (used for filtering)
- conditions referring to two tables (treated as *join conditions*, regardless of *where they appear*)

Especially in MySql, you can *see yourself*, by tracing the optimizer, that the `join .. on` conditions are *replaced during parsing* by the equivalent `where` conditions. A similar thing happens in postgresql (though there's no way to see it through a log, you have to read the source description).

Anyway, the main point is, the difference between the two syntax variants *is lost* during the parsing/query-rewriting phase, it does not even reach the query planning and execution phase. So, there's no question about whether they are equivalent in terms of performance, **they become identical long before they reach the execution phase**.

You can use `explain` , to verify that they produce identical plans. Eg, in postgres, **the plan will contain a `join` clause, even if you didn't use the `join..on` syntax anywhere**.

> Oracle and SQL server are not open source, but, as far as I know, they are based equivalence rules (similar to those in relational algebra), and they also produce identical execution plans in both cases.

> Obviously, the two syntax styles *are not* equivalent for outer joins, for those you have to use the `join ... on` syntax

Share Improve this answer Follow

edited Nov 14 '20 at 17:58

answered Nov 14 '20 at 17:42
blue_note
21.9k 6 44 63

-5

this is my solution.

```
SELECT song_ID,songs.fullname, singers.fullname
FROM music JOIN songs ON songs.ID = music.song_ID
JOIN singers ON singers.ID = music.singer_ID
GROUP BY songs.fullname
```

You *must have* the `GROUP BY` to get it to work.

Hope this help.

Share Improve this answer Follow

edited Apr 26 '10 at 16:11
BCS
65.7k 61 175 276

answered Apr 26 '10 at 7:53
Le Quang Chien
25

10 Grouping on only songs.fullname while you are also selecting song_id and singers.fullname is going to be a problem in most databases. – btilly Feb 9 '11 at 4:46