



## SQL Lesson 12: Order of execution of a Query

Now that we have an idea of all the parts of a query, we can now talk about how they all fit together in the context of a complete query.

Complete SELECT query

```
SELECT DISTINCT column, AGG_FUNC(column_or_expression), ...
FROM mytable
    JOIN another_table
    ON mytable.column = another_table.column
WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
LIMIT count OFFSET COUNT;
```

Each query begins with finding the data that we need in a database, and then filtering that data down into something that can be processed and understood as quickly as possible. Because each part of the query is executed sequentially, it's important to understand the order of execution so that you know what results are accessible where.

### Query order of execution

#### 1. FROM and JOIN s

The **FROM** clause, and subsequent **JOIN** s are first executed to determine the total working set of data that is being queried. This includes subqueries in this clause, and can cause temporary tables to be created under the hood containing all the columns and rows of the tables being joined.

#### 2. WHERE

Once we have the total working set of data, the first-pass **WHERE** constraints are applied to the individual rows, and rows that do not satisfy the constraint are discarded. Each of the constraints can only access columns directly from the tables requested in the **FROM** clause. Aliases in the **SELECT** part of the query are not accessible in most databases since they may include expressions dependent on parts of the query that have not yet executed.

#### 3. GROUP BY

The remaining rows after the **WHERE** constraints are applied are then grouped based on common values in the column specified in the **GROUP BY** clause. As a result of the grouping, there will only be as many rows as there are unique values in that column. Implicitly, this means that you should only need to use this when you have aggregate functions in your query.

#### 4. HAVING

If the query has a **GROUP BY** clause, then the constraints in the **HAVING** clause are then applied to the grouped rows, discard the grouped rows that don't satisfy the constraint. Like the **WHERE** clause, aliases are also not accessible from this step in most databases.

#### 5. SELECT

Any expressions in the **SELECT** part of the query are finally computed.

#### 6. DISTINCT

Of the remaining rows, rows with duplicate values in the column marked as **DISTINCT** will be discarded.

#### 7. ORDER BY

If an order is specified by the **ORDER BY** clause, the rows are then sorted by the specified data in either ascending or descending order. Since all the expressions in the **SELECT** part of the query have been computed, you can reference aliases in this clause.

8. **LIMIT / OFFSET**

Finally, the rows that fall outside the range specified by the **LIMIT** and **OFFSET** are discarded, leaving the final set of rows to be returned from the query.

Conclusion

Not every query needs to have all the parts we listed above, but a part of why SQL is so flexible is that it allows developers and data analysts to quickly manipulate data without having to write additional code, all just by using the above clauses.

Exercise

Here ends our lessons on **SELECT** queries, congrats of making it this far! This exercise will try and test your understanding of queries, so don't be discouraged if you find them challenging. Just try your best.

Table: Movies (Read-Only)

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117

Table: Boxoffice (Read-Only)

Movie_id	Rating	Domestic_sales	International_sales
5	8.2	380843261	555900000
14	7.4	268492764	475066843
8	8	206445654	417277164
12	6.4	191452396	368400000
3	7.9	245852179	239163000
6	8	261441092	370001000
7	6.7	200000000	200000000

Query Results

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116
7	Cars	John Lasseter	2006	117

```
SELECT * FROM movies;
```

RESET

Exercise 12 — Tasks

1. Find the number of movies each director has directed
2. Find the total domestic and international sales that can be attributed to each director

Stuck? Read this task's Solution.

Solve all tasks to continue to the next lesson.

Finish above Tasks (/lesson/inserting\_rows)

Next – SQL Lesson 13: Inserting rows (/lesson/inserting\_rows)

Previous – SQL Lesson 11: Queries with aggregates (Pt. 2) (/lesson/select\_queries\_with\_aggregates\_pt\_2)

Find SQLBolt useful? Please consider  
Donating (\$4) via Paypal to support our site.



## Remote U.S Software Jobs

U.S Software Developers Jobs. Work  
Remotely. No Visa needed. Full time jobs.  
High salary.

Turing.com

Like 1.2K

Share

Tweet

2019 © SQLBolt

Email (mailto:sqlbolt@gmail.com) | Twitter (http://twitter.com/sqlbolt)

Additional Courses

Interactive Regular Expressions Lessons (https://regexone.com)