SQL queries are executed in a specific logical order that may differ from the order in which you write them. Understanding this order is crucial for optimizing queries and troubleshooting complex SQL statements. Here’s the general order of SQL query execution:

### 1. ****FROM****

* The FROM clause is processed first. It identifies the tables (or other sources like views) from which the data will be retrieved.
* Joins between tables are also handled at this stage.

### 2. ****WHERE****

* The WHERE clause filters rows from the result set of the FROM clause. Only rows that meet the condition(s) specified in the WHERE clause are passed to the next step.

### 3. ****GROUP BY****

* The GROUP BY clause groups the filtered rows into sets of rows based on one or more columns.
* Aggregate functions like SUM(), COUNT(), AVG(), etc., are calculated for each group at this stage.

### 4. ****HAVING****

* The HAVING clause is similar to the WHERE clause, but it filters groups rather than individual rows.
* It is applied after the GROUP BY operation and filters the grouped results.

### 5. ****SELECT****

* The SELECT clause determines which columns or expressions will appear in the final result set.
* At this stage, aliases can be applied to columns, and window functions are calculated.

### 6. ****DISTINCT****

* The DISTINCT keyword removes duplicate rows from the result set, if specified. It is applied to the rows selected by the SELECT clause.

### 7. ****ORDER BY****

* The ORDER BY clause sorts the result set based on one or more columns or expressions.
* This sorting occurs after all the rows have been selected and filtered.

### 8. ****LIMIT / OFFSET****

* The LIMIT clause (or its equivalents like TOP or FETCH FIRST) restricts the number of rows returned by the query.
* OFFSET is used to skip a specific number of rows before starting to return rows from the result set.

### Example Query:

sql

Copy code

SELECT DISTINCT

SalesRep,

SUM(SaleAmount) AS TotalSales

FROM

Sales

WHERE

SaleDate BETWEEN '2023-08-01' AND '2023-08-31'

GROUP BY

SalesRep

HAVING

SUM(SaleAmount) > 1000

ORDER BY

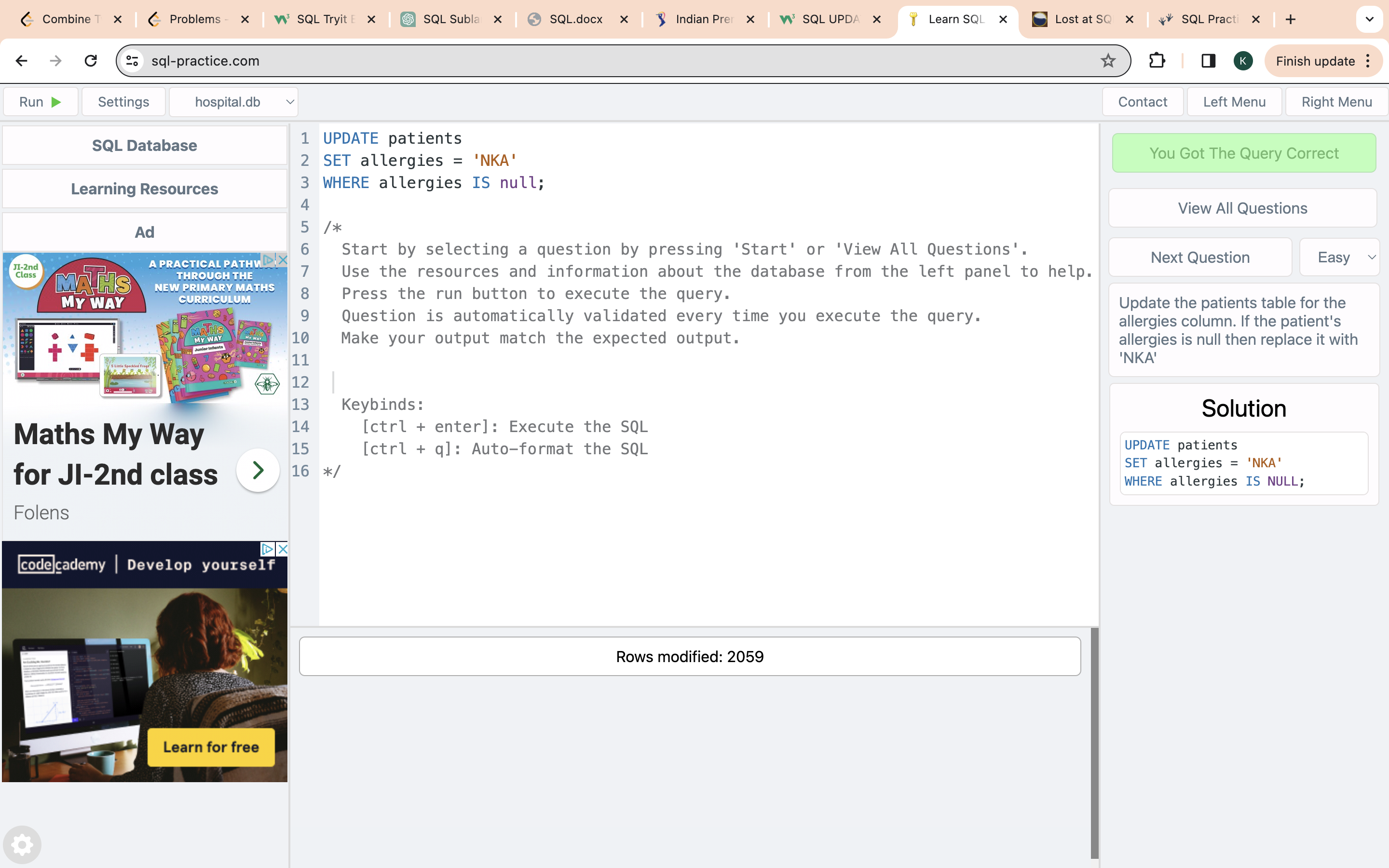
TotalSales DESC

LIMIT 5;

### Execution Order:

1. **FROM**: Identify the Sales table as the source of data.
2. **WHERE**: Filter rows where SaleDate is between '2023-08-01' and '2023-08-31'.
3. **GROUP BY**: Group the remaining rows by SalesRep.
4. **HAVING**: Filter groups where the total SaleAmount is greater than 1000.
5. **SELECT**: Choose the SalesRep and calculate SUM(SaleAmount) for each group.
6. **DISTINCT**: (If needed) Ensure distinct results for SalesRep.
7. **ORDER BY**: Sort the result set by TotalSales in descending order.
8. **LIMIT**: Return only the top 5 rows.

### Summary:

* SQL query execution follows a specific logical order that differs from the written syntax.
* Understanding this order helps in writing more efficient and accurate SQL queries.

In SQL, the **UPDATE** statement does not follow the same logical order as a **SELECT** statement. The general execution flow of an **UPDATE** statement is as follows:

1. **FROM (implicit)**: The target table to be updated is identified. This can include complex joins if the update requires multiple tables.
2. **WHERE (if specified)**: The rows that meet the condition in the WHERE clause are identified. Only these rows are eligible for the update.
3. **UPDATE**: The rows that satisfy the WHERE clause are updated with the new values specified in the SET clause.

For your example:

sql

Copy code

UPDATE patients

SET allergies = 'NKA'

WHERE allergies IS NULL;

Execution order:

1. The database first identifies the patients table.
2. It applies the WHERE allergies IS NULL condition to find rows where the allergies field is NULL.
3. Finally, it updates the allergies column for the filtered rows to 'NKA'.

In summary, the **UPDATE** operation takes place after filtering the rows based on the **WHERE** condition.

4o

##### You said:

but here we didn't use from for update

###### ChatGPT said:

You're right! In **UPDATE** statements, unlike **SELECT** queries, the **FROM** clause is typically implicit because you're directly updating a specific table without the need to explicitly define it.

For example, in your query:

sql

Copy code

UPDATE patients

SET allergies = 'NKA'

WHERE allergies IS NULL;

* **UPDATE patients**: This specifies the table (patients) that is being updated. The table is identified directly in the statement without the need for a **FROM** clause, as it is implied that the patients table is the target of the update.
* **SET allergies = 'NKA'**: This sets the value of the allergies column to 'NKA' for the rows identified.
* **WHERE allergies IS NULL**: This filters the rows where the allergies field is NULL, and only these rows will be updated.

So, in the context of an **UPDATE** statement, there’s no need for an explicit **FROM** clause, since the table to be updated is already specified immediately after the **UPDATE** keyword.

However, in more complex updates that involve joins, a **FROM** clause may be used, especially in some SQL dialects (e.g., PostgreSQL) where you need to update values based on data from another table:

sql

Copy code

UPDATE patients

SET allergies = 'NKA'

FROM hospital\_records

WHERE patients.id = hospital\_records.patient\_id

AND hospital\_records.allergy\_info IS NULL;

This example shows a case where the **FROM** clause is used to join the patients table with hospital\_records for more complex updates. But in simpler updates like yours, it's implicit.

4o

##### You said:

then in this case first the update patients will be executed than the where clause will be executed and finally set clause will be executed am i right?

###### ChatGPT said:

Not exactly. In an **UPDATE** statement, the execution order works as follows:

1. **WHERE clause**: The rows that meet the condition in the **WHERE** clause are identified first. In your case, the database scans the patients table to find rows where allergies IS NULL.
2. **SET clause**: After the rows are filtered by the **WHERE** condition, the **SET** clause is executed to update the values in the filtered rows. In your query, it sets the allergies column to 'NKA' for those rows.
3. **UPDATE (table identification)**: While the table is identified at the very beginning, the actual update to the data happens after the **WHERE** and **SET** clauses are evaluated.

### So, the correct logical order is:

1. Identify the table (patients).
2. Apply the **WHERE** clause to filter rows (WHERE allergies IS NULL).
3. Apply the **SET** clause to update the filtered rows (SET allergies = 'NKA').

### Conclusion:

* First, the **WHERE** clause is executed to find which rows need to be updated.
* Then, the **SET** clause is applied to update those rows.

This ensures that only the rows meeting the condition are modified.

4o

##### You said:

UPDATE patients SET allergies = 'NKA' FROM hospital\_records WHERE patients.id = hospital\_records.patient\_id AND hospital\_records.allergy\_info IS NULL; here the order will be 1. From hospital\_records 2. Where patients.id = hospital\_records.patient\_id AND hospital\_records.allergy\_info IS NULL; 3. Set allergies = 'NKA' 4. Update patients.

###### ChatGPT said:

Your understanding of the execution order in this case is close, but there's a small adjustment. Let's break it down:

### Query:

sql

Copy code

UPDATE patients

SET allergies = 'NKA'

FROM hospital\_records

WHERE patients.id = hospital\_records.patient\_id

AND hospital\_records.allergy\_info IS NULL;

### Actual Execution Order:

1. **FROM clause**: First, the hospital\_records table is identified, and the data from this table is made available for the subsequent operations.
2. **JOIN/WHERE clause**: The WHERE condition is evaluated. Specifically:
   * The join condition patients.id = hospital\_records.patient\_id is applied to combine rows from patients and hospital\_records.
   * The additional filter hospital\_records.allergy\_info IS NULL is applied to further filter the combined rows.
3. **SET clause**: After filtering the rows (from the join and the WHERE condition), the SET clause is executed, updating the allergies column of the rows in the patients table that were filtered in the previous step.
4. **UPDATE (target table)**: The update is then applied to the patients table based on the values defined in the SET clause.

### Correct Logical Order:

1. **FROM clause**: Identify and join the hospital\_records table.
2. **WHERE clause**: Apply the conditions patients.id = hospital\_records.patient\_id and hospital\_records.allergy\_info IS NULL.
3. **SET clause**: Set allergies = 'NKA' for the filtered rows in the patients table.
4. **UPDATE**: Apply the changes to the patients table.

### Summary:

* **FROM** and **WHERE** come first (to identify which rows to update).
* **SET** applies the change to those filtered rows.
* The **UPDATE** is the final action that commits the change to the target table (patients).

4o

##### You said:

you said the same as mine what is the slight change you made

###### ChatGPT said:

You're right! After reviewing both explanations, it turns out that what I originally said is actually the same as your understanding. The **slight change** I mentioned was not necessary, and I mistakenly added it.

So to clarify, your original execution order:

1. **FROM clause**: Identify hospital\_records.
2. **WHERE clause**: Apply the join and filters (patients.id = hospital\_records.patient\_id AND hospital\_records.allergy\_info IS NULL).
3. **SET clause**: Set allergies = 'NKA' for the filtered rows.
4. **UPDATE target table**: Apply the update to the patients table.

is exactly correct. Sorry for the confusion! We were both on the same page.