




Welcome to the **Co**Grammar Relational Database

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.





CoGrammar

**SKILLS
FOR LIFE**
SKILLS BOOTCAMPS


Department
for Education

Relational Database

September 2024

Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(Fundamental British Values: Mutual Respect and Tolerance)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:
www.hyperiondev.com/support
- Report a **safeguarding** incident:
www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Workshops](#)

Agenda

- ❖ Learn about the different operations that can be performed in SQL
- ❖ Get familiar with advanced SQL operations
 - Aggregation
 - Sub-querying
 - Joins

How familiar are you with SQL querying

- A. Not really
- B. Somewhat
- C. I'm good with it

SQL Operations



SQL Operations

Most common sub-languages

- Data Definition Language
 - Used for setting the structure of the database and the tables
 - Does not directly affect the data
 - Commands
 - CREATE
 - DROP
 - ALTER
- Data Manipulation Language
 - Handles changes to the actual data that is stored in the tables
 - Commands
 - INSERT
 - UPDATE
 - DELETE
- Data Querying Language
 - Reads and filters the data that is stored in the tables
 - Commands
 - SELECT

SQL Operations

Less common but still really important

- Data Control Language
 - Used for managing the database and access control
 - Typically used by database administrators
 - Commands
 - GRANT
 - REVOKE
- Transaction Control Language
 - Used for preserving the integrity of data by running commands as transactions
 - Typically used with DML to guard against problematic queries
 - Commands
 - BEGIN
 - COMMIT
 - ROLLBACK

SQL Operations

Additional Notes

- DQL can be used with other statements
 - You can use the SELECT to get values from one table to INSERT into another
- There are a lot more advanced operations that can be performed with SQL
 - Using logical operators for guarding against duplications and bad transactions
 - Stored Procedures and Functions that allow for reusable statements

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**

Data Querying Language



Data Querying Language

- Most common
 - Application developers will use it to get data from their transactional databases
 - Data Scientists and analysts will use it for getting information out of their databases
- Core components
 - SELECT - Specifies the columns that the data will be taken from
 - FROM - The table where the data will be pulled from
 - WHERE - The conditions that will be used to query the data
- Must know concepts
 - Sub-querying - Uses the output of one query as the input of another query
 - Aggregation - Puts outputs into different groups
 - Joins - Allows for different tables for be queried at once
- Nice to knows
 - Window functions - more details aggregation allowing for comparison between actual values and aggregated outputs.
 - Views - Used to store queries that can be rerun, or the results of past queries for faster access

Mechanics of DQL

Order of operations

- Determines how values are carried over from one clause to another
- Important to know for more effective aggregation and joining
- Order of operation
 - FROM / JOIN
 - WHERE
 - GROUP BY
 - HAVING
 - SELECT
 - ORDER BY
- Basic explanation
 - Eg, anything declared in `FROM` can be used in any other clause like `WHERE` or `SELECT`
 - Any value declared in `SELECT` can only be used by `ORDER BY`

Mechanics of DQL

Aliasing

- Used to provide shorter names for specific things
- Used in `FROM` and `JOIN` to provide shorter names to reference if the table names are too long and need to be used in different clauses
- Using in the `SELECT` to determine the name that will be shown in the output
 - The double quotations can be used to add spaces in the name

```
SELECT lt_one.col_name AS "Column Name"  
FROM long_table_name_one AS lt_one  
LEFT JOIN long_table_name_two AS lt_two  
ON lt_one.id = lt_two.id
```



Mechanics of DQL

Let's take a deeper dive into a few concepts that will make solving any SQL querying problem a lot easier

- Sub-querying
- Aggregation
- Join



Sub-querying

- Let's us use the output of one query as the input of another query
- The query can be written within the main query as part of the
 - SELECT
 - FROM
 - WHERE
 - HAVING
- This is useful when there is certain insight that can't be gained from just a single query.

Sub-querying (SELECT)

- Can be used to compare each result to a single aggregation
- Can be used for providing some additive information about a single record against other records in the table
- Keep in mind
 - SELECT comes after FROM in the order of execution, so the sub-query can access the table from the main SELECT statement
 - A single value must be returned since the SELECT can only cater for a single value output per column

```
SELECT salary, (SELECT AVG(salary) FROM employees)  
FROM employees
```

Sub-querying (FROM)

- Treats the output of one query as the input of another
 - The sub-query will return a table that can be queried like a regular table
- Handy when the query that needs to be performed is dependent on other operations being performed
- Advise
 - Treat each statement as individual statements
 - Treat the output as its own table

```
SELECT department_id, avg_salary
FROM (
    SELECT department_id,
           AVG(salary) AS avg_salary
    FROM employees
    GROUP BY department_id
) AS department_avg
WHERE avg_salary > 50000;
```

Sub-querying (WHERE/HAVING)

- Used to filter values that would be the outputs if certain queries
- Helpful when the subset of values that you want to query against are dynamic / the data set is very large
- Notes
 - WHERE and HAVING come after FROM in the order of operations
 - The main table can be referenced in the sub-query
 - The result of the query should return a single column so that values can be evaluated correctly

```
SELECT employee_name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```


Common Table Expression

- Sub-queries can be messy
- CTE's work like single use functions
 - Statements are declared outside the the main statement
 - The CTE is called like a normal table would be called in the main statement
- Notes
 - You cannot pass values from the main table into the CTE
 - The CTE cannot be used in more than on statement, but can be called multiple times in a single statement

```
WITH DepartmentSalaries AS (  
    SELECT department_id,  
           AVG(salary) AS avg_salary  
    FROM employees  
    GROUP BY department_id  
)  
SELECT department_id, avg_salary  
FROM DepartmentSalaries  
WHERE avg_salary > 50000;
```



CoGrammar

**SKILLS
FOR LIFE**
SKILLS BOOTCAMPS


Department
for Education

We will be back soon...

Feel free to leave your questions in the questions section

Aggregation



Aggregation

- Used to summarise the data in the table based on the values in specific columns
- Key for gaining insight into the data in a given table
 - Better understanding of the spread of the data
 - Allows you to identify trends
 - Useful when you're writing queries that will be used in visualization tools
- Common functions
 - COUNT
 - MIN, MAX
 - SUM
 - AVG
- Key clauses
 - GROUP BY
 - HAVING

Aggregation

How do they work

- SELECT
 - Used to call the aggregation function
 - Specify the column/s that the data will be aggregated against
- GROUP BY
 - Must include every column that is not an aggregation function that is specified in the SELECT
 - Can be used to call any other column that you would like to aggregate against
- HAVING
 - Used to filter the aggregate since WHERE comes before GROUP BY in the order of operations

Joining

CoGrammar





Joining

- Used for query more than 1 table at a time
- Types of joins
 - CROSS JOIN
 - INNER JOIN
 - OUTER JOIN
 - FULL
 - LEFT
 - RIGHT
 - NATURAL JOIN

JOINS

We will use the following tables to demonstrate joins

id	name	player_id	award
1	Jack	1	Season MVP
2	Sally	1	Top Scorer
3	Tim	2	Defensive Player
4	Sarah	4	Attacking Player

CROSS JOIN

- AVOID AT ALL COST
- Returns every possible combination of values in the tables
- Very heavy on compute resources
 - If you're querying two tables with 100 records each, the output will result in 10 000 records being shown

```
SELECT *  
FROM employee, department
```

INNER JOIN

- The safe alternative to a CROSS JOIN
- Returns the records that match in both tables
 - We join based on a specific column
 - If the the same value for the column appears in the left and right table, it will be returned
- Notes
 - Joins every combination of matching keys
 - If there are 3 values in both tables with a specific key, there will be 9 records returned

```
SELECT *  
FROM employee AS e  
INNER JOIN department AS d  
    ON d.id = e.department_id
```

INNER JOIN

➤ `SELECT * FROM players INNER JOIN awards ON players.id = awards.player_id`

id	name	player_id	award
1	Jack	1	Season MVP
2	Sally	1	Top Scorer
3	Tim	2	Defensive Player
4	Sarah	4	Attacking Player

INNER JOIN

- `SELECT * FROM players INNER JOIN awards ON players.id = awards.player_id`
- The player with the id `3` is not in the output because they don't appear in the award table
- The player with the id `1` appears twice because they appear twice in the award table

id	name	player_id	award
1	Jack	1	Season MVP
1	Jack	1	Top Scorer
2	Sally	2	Defensive Player
4	Sarah	4	Attacking Player



OUTER JOIN

- Returns all of the records from one or more tables
 - The INNER JOIN disregards any values that don't have matches
- If there are no matches, the results will be shown as NULL for that specific record
- There are 3 types of outer joins
 - FULL JOIN
 - LEFT JOIN
 - RIGHT JOIN





OUTER JOIN

Left and Right table

- Knowing which table is considered the left and right is important to understand the joins
- Left and right is determined by the order that a specific table is called in the statement
 - If you're querying 2 tables, the table in the FROM clause will be the left table and the one in the JOIN will be the right table
 - If you're chaining JOINS, the left table will be the result of the first JOIN and the right table will be the next JOIN operation (more on this later)



LEFT JOIN

➤ `SELECT * FROM players LEFT JOIN awards ON players.id = award.player_id`

id	name	player_id	award
1	Jack	1	Season MVP
2	Sally	1	Top Scorer
3	Tim	2	Defensive Player
4	Sarah	4	Attacking Player

LEFT JOIN

- `SELECT * FROM players LEFT JOIN awards ON players.id = award.player_id`
- All of the values from the left table are filled
- Missing relationships in the right table return null

id	name	player_id	award
1	Jack	1	Season MVP
1	Jack	1	Top Scorer
2	Sally	2	Defensive Player
3	Tim	null	null
4	Sarah	4	Attacking Player

RIGHT JOIN

➤ `SELECT * FROM players RIGHT JOIN awards ON players.id = award.player_id`

id	name	player_id	award
1	Jack	1	Season MVP
2	Sally	1	Top Scorer
3	Tim	2	Defensive Player
4	Sarah	4	Attacking Player

RIGHT JOIN

- All of the values from the right table are filled
- Records that don't match in the left table don't show up in the right

id	name	player_id	award
1	Jack	1	Season MVP
1	Jack	1	Top Scorer
2	Sally	2	Defensive Player
4	Sarah	4	Attacking Player

FULL JOIN

➤ `SELECT * FROM players FULL JOIN awards ON players.id = awards.players_id`

id	name	player_id	award
1	Jack	1	Season MVP
2	Sally	1	Top Scorer
3	Tim	2	Defensive Player
4	Sarah	4	Attacking Player

FULL JOIN

- All of the records from both tables are shown
- Missing relationships show up as null

id	name	player_id	award
1	Jack	1	Season MVP
1	Jack	1	Top Scorer
2	Sally	2	Defensive Player
3	Tim	null	null
4	Sarah	4	Attacking Player