1. Sure thing! Here's a simple SQL query that combines first\_name and last\_name into a single alias called full\_name:

SELECT CONCAT(first\_name, ' ', last\_name) AS full\_name FROM your\_table\_name;

🧠 Breakdown:

* CONCAT() joins the first and last names with a space in between.
* AS full\_name gives the resulting column the alias "full\_name".
* Replace your\_table\_name with the actual name of your table.

If you're using a database like **PostgreSQL**, you can also use the || operator:

SELECT first\_name || ' ' || last\_name AS full\_name FROM your\_table\_name;

Let me know what database you're working with, and I can tailor it even more precisely!

1. To retrieve only unique rows or unique values from specific columns in an SQL SELECT statement, the DISTINCT keyword is used.

Using DISTINCT for unique rows:

To retrieve unique combinations of all selected columns, place DISTINCT immediately after SELECT.

Код

SELECT DISTINCT column1, column2, ...

FROM table\_name;

WITH RankedFilms AS (

SELECT

film\_id,

title,

rental\_rate,

ROW\_NUMBER() OVER (ORDER BY rental\_rate ASC, title ASC) AS rank

FROM film

)

SELECT

film\_id,

title,

rental\_rate

FROM RankedFilms

WHERE rank BETWEEN 11 AND 20;

Why this works:

* ROW\_NUMBER() assigns a unique rank to each movie based on its rental rate.
* The ORDER BY rental\_rate ASC, title ASC ensures consistent ordering.
* The outer query filters for ranks 11 through 20—your “next 10 cheapest.”

This approach is super flexible and works across most SQL engines like PostgreSQL, SQL Server, MySQL 8+, and Oracle.

URL: URL: https://github.com/EgineFox/DI-Bootcamp/blob/main/Week5/Day4/DCH-EX2.sql suggestions for improvement: - Complete the missing queries for Exercise 1 (all parts). - Complete the missing queries for Exercise 2, specifically questions 12, 13, 14, and 15 (bonus). - Add comments to each query explicitly linking it to the corresponding exercise and question number for better clarity. - Consider using OFFSET X FETCH NEXT Y ROWS ONLY as an alternative to ROW\_NUMBER() for pagination if the database version supports it, as it can sometimes be more concise, especially if LIMIT is allowed (though the bonus specifically asked to avoid LIMIT). - For searches with leading wildcards (e.g., ILIKE '%Harry Potter%'), consider if full-text search capabilities are needed for performance on very large datasets. Correctness: 58%

URL: URL: https://github.com/EgineFox/DI-Bootcamp/blob/main/Week5/Day4/DCH-Ex1.sql suggestions for improvement: - Consider consistent indentation for multi-line SELECT statements and JOIN clauses to enhance readability, especially for more complex queries involving multiple columns and joins. - While SELECT \* is acceptable for exercises, in production code or when only specific columns are truly needed, it's generally better practice to explicitly list required columns. This improves clarity and can offer minor performance benefits by reducing data transfer. - For the advanced query using ROW\_NUMBER() (Exercise 2, Q11), adding a brief comment explaining the approach or the reason for using ROW\_NUMBER() instead of LIMIT OFFSET could be beneficial for educational purposes. Correctness: 100%

URL: URL: https://github.com/EgineFox/DI-Bootcamp/blob/main/Week5/Day4/DCh-SQL\_puzzle.sql suggestions for improvement: - The current code does not address any of the specific exercises (Exercise 1: Items and customers, Exercise 2: dvdrental database) outlined in the chapter content. Please remove this code and instead provide SQL queries that directly solve the problems presented in Exercise 1 and Exercise 2. - For Exercise 1, ensure your queries target tables like 'items' and 'customers' within the 'public' database, assuming they exist as per the instructions. - For Exercise 2, ensure your queries are written against the 'dvdrental' database and its tables (e.g., 'customer', 'film', 'address', 'payment', 'inventory', 'city', 'country') as detailed in the setup and subsequent questions. Correctness: 0%

**LEFT JOIN**

**CONSTRAINT fk\_film**

**FOREIGN KEY (film\_id)**

**REFERENCES new\_film (ID)**

**ON DELETE CASCADE,**

**CONSTRAINT fk\_language**

**FOREIGN KEY (language\_id)**

**REFERENCES language (language\_id)**

**);**

**SELECT film.title, film.rental\_rate**

**FROM film**

**JOIN inventory ON film.film\_id = inventory.film\_id**

**JOIN rental ON inventory.inventory\_id = rental.inventory\_id**

**WHERE rental.return\_date IS NULL**

**ORDER BY film.rental\_rate DESC**

**LIMIT 30;**

Fixes and Notes:

* **Join Tables**: You need to join film, inventory, and rental to connect movies to their rental status.
* **Correct Column Name**: You had a typo in rentel\_rate — it should be rental\_rate.
* **Return Condition**: rental.return\_date IS NULL is sufficient to find outstanding rentals. The return\_date > CURRENT\_DATE condition isn't necessary unless you're tracking future returns (which is rare).
* **Ordering**: Sorting by film.rental\_rate ensures you're getting the most expensive ones.

**db\_env\Script\activate**