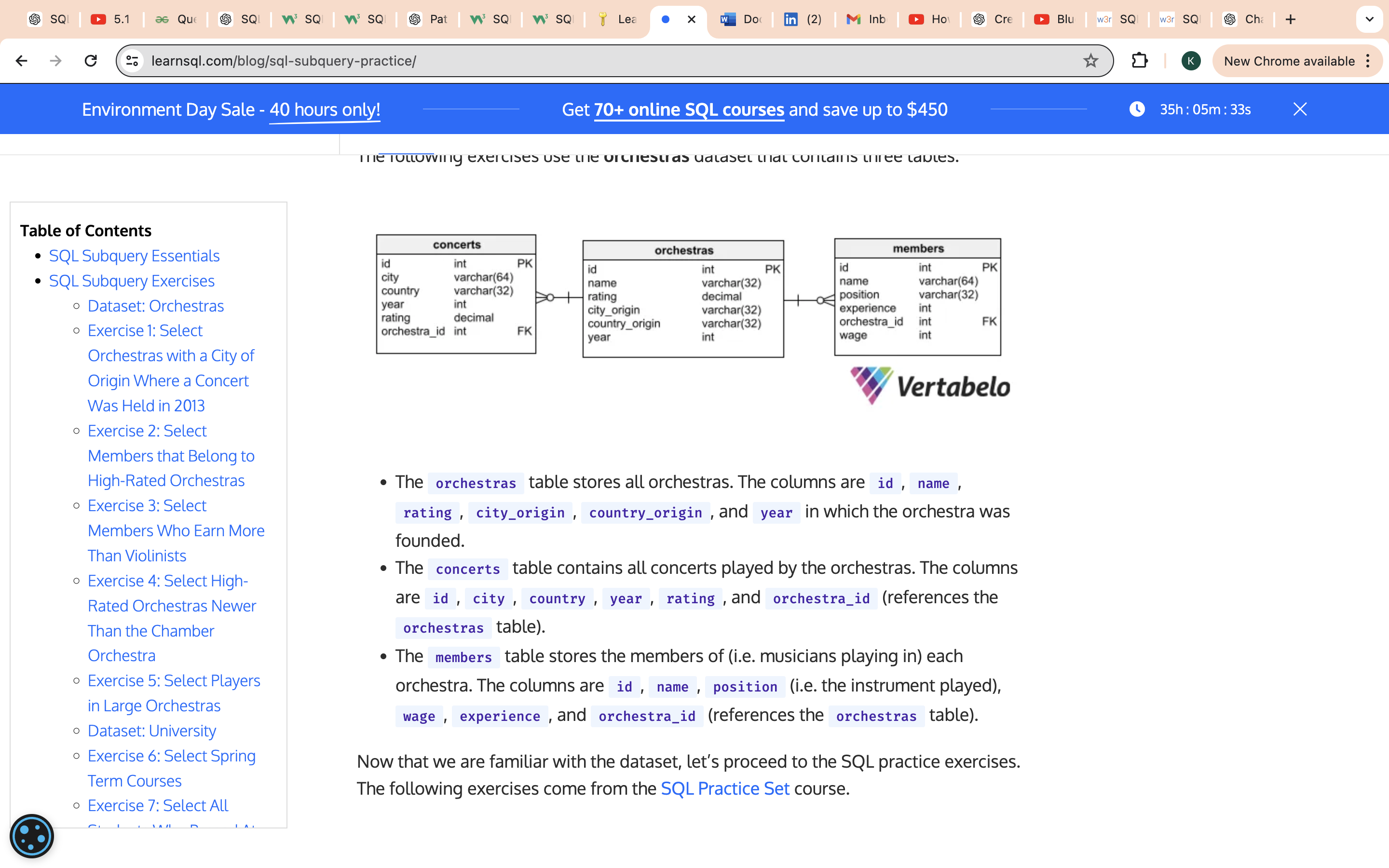
The **orchestras** table stores all orchestras. The columns are **id**, **name**, **rating**, **city\_origin**, **country\_origin**, and **year** in which the orchestra was founded.

* The **concerts** table contains all concerts played by the orchestras. The columns are **id**, **city**, **country**, **year**, **rating**, and **orchestra\_id** (references the **orchestras** table).
* The **members**table stores the members of (i.e. musicians playing in) each orchestra. The columns are **id**, **name**, **position** (i.e. the instrument played), **wage**, **experience**, and **orchestra\_id** (references the **orchestras** table).

### Exercise 1: Select Orchestras with a City of Origin Where a Concert Was Held in 2013

My code:

Select name From orchestras

where (Select city From concerts Where year = 2013 As city);

correct code:

SELECT name

FROM orchestras

WHERE city\_origin IN (SELECT city FROM concerts WHERE year = 2013);

**Solution explanation:**

We aim to select names of orchestras that fulfill a certain condition, so we start with **SELECT name FROM orchestras**. Then, the condition is going to be imposed on the **city\_origin** column, as mentioned in the instructions.

We want to select only the orchestras whose city of origin belongs to the group of cities where concerts were played in the year 2013. To create this condition in the **WHERE** clause, we use SQL subquery.

Let’s create a (sub)query that selects all cities where concerts were played in 2013: S**ELECT city FROM concerts WHERE year = 2013**. It returns a column containing city names.

To ensure that the city of origin belongs to the cities returned by the subquery, we use the **IN** operator.

Select name and position from members

where experience > 10 AND (Select rating from orchestras) < 8

### Exercise 2: Select Members that Belong to High-Rated Orchestras

**Exercise:**

Select the names and positions (i.e. instrument played) of all orchestra members that have above 10 years of experience and do not belong to orchestras with a rating below 8.0.

My code:

Select name, position From members

where experience > 10 AND (Select rating from orchestras where rating >8)

Correct code:

SELECT

  name,

  position

FROM members

WHERE experience > 10

AND orchestra\_id NOT IN (SELECT id FROM orchestras WHERE rating < 8.0);

### Exercise 3: Select Members Who Earn More Than Violinists

**Exercise:**

Show the name and position of orchestra members who earn more than the average wage of all violinists.

My code:

Select name, position from

where Avg(wage) >

Actual code:

SELECT name, position

FROM members

WHERE wage > (SELECT AVG(wage)

              FROM members

              WHERE position = 'violin');

### Exercise 4: Select High-Rated Orchestras Newer Than the Chamber Orchestra

**Exercise:**

Show the names of orchestras that were created after the 'Chamber Orchestra' and have a rating greater than 7.5.

My code:

Select names from orchestras

|  |
| --- |
| SELECT name  FROM orchestras  WHERE year > (SELECT year FROM orchestras                WHERE name = 'Chamber Orchestra')  AND rating > 7.5; |

**Solution explanation:**

First, we select names from the **orchestras** table.

Then, we create a subquery that returns the year when the Chamber Orchestra was created.

As we want to list orchestras created after the Chamber Orchestra, we impose a condition on the year column to be greater than whatever is returned by this subquery.

Finally, we define the condition on the **rating** column to be greater than 7.5.

### Exercise 5: Select Players in Large Orchestras

**Exercise:**

Show the name and number of members for each orchestra that has more members than the average membership of all orchestras in the table.

Select name, count(id)

Both subqueries and joins are used in SQL to retrieve data from multiple tables, but they do so in different ways and are used in different situations. Here's a detailed explanation with examples:

### Subqueries

A subquery (or inner query or nested query) is a query within another query. The result of the subquery is used by the outer query.

#### When to use Subqueries:

* When you need to filter results based on a separate query.
* When you need to use an aggregate function in a condition (e.g., in the WHERE or HAVING clause).
* When the query logic is easier to understand or maintain using subqueries.
* When you need to perform a calculation or comparison on a set of results.

#### Example of Subquery:

Let's say we have two tables: employees and departments.

**employees:**

| employee\_id | employee\_name | department\_id |
| --- | --- | --- |
| 1 | Alice | 1 |
| 2 | Bob | 2 |
| 3 | Charlie | 1 |
| 4 | David | 3 |

**departments:**

| department\_id | department\_name |
| --- | --- |
| 1 | HR |
| 2 | Finance |
| 3 | IT |

**Query: Find the names of employees who work in the HR department.**

sql

Copy code

SELECT employee\_name

FROM employees

WHERE department\_id = (SELECT department\_id

FROM departments

WHERE department\_name = 'HR');

### Joins

A join combines rows from two or more tables based on a related column between them.

#### When to use Joins:

* When you need to combine columns from multiple tables into a single result set.
* When you need to retrieve related data spread across multiple tables.
* When the query logic requires fetching data from multiple tables simultaneously.

#### Types of Joins:

1. **INNER JOIN**: Returns rows when there is a match in both tables.
2. **LEFT JOIN (or LEFT OUTER JOIN)**: Returns all rows from the left table, and the matched rows from the right table. If no match, NULL values are returned for columns from the right table.
3. **RIGHT JOIN (or RIGHT OUTER JOIN)**: Returns all rows from the right table, and the matched rows from the left table. If no match, NULL values are returned for columns from the left table.
4. **FULL JOIN (or FULL OUTER JOIN)**: Returns rows when there is a match in one of the tables. It returns all rows from the left table and the right table, with NULLs in places where there is no match.

#### Example of Join:

**Query: Find the names of employees and their corresponding department names.**

sql

Copy code

SELECT e.employee\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

### Comparison and Use Cases

#### Subqueries:

* **Use Case**: When you need to filter data based on aggregate results or complex conditions.
* **Example**: Find employees who have a salary higher than the average salary in the company.

sql

Copy code

SELECT employee\_name

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

#### Joins:

* **Use Case**: When you need to combine data from multiple tables to create a comprehensive result set.
* **Example**: Retrieve a list of employees along with their department names and manager names.

sql

Copy code

SELECT e.employee\_name, d.department\_name, m.employee\_name AS manager\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

LEFT JOIN employees m ON e.manager\_id = m.employee\_id;

### Summary

* **Subqueries** are useful for filtering and comparing results, especially when using aggregate functions.
* **Joins** are essential for combining related data from multiple tables into a single result set.
* The choice between subqueries and joins depends on the specific requirements of your query and which approach makes the logic clearer and more efficient.

4o

I couldn't understand when and where to use subquery and joins explain me in a better way.

Certainly. Let's delve deeper into the distinctions between subqueries and joins, elucidating when and where to employ each, accompanied by precise examples to solidify understanding.

### Subqueries

**Definition**: A subquery, also referred to as an inner query or nested query, is a query embedded within another SQL query. The result generated by the subquery is subsequently utilized by the outer query.

**Usage Scenarios**:

1. **Filtering Results**: Subqueries are advantageous when you need to filter results based on a condition derived from a separate query.
2. **Complex Conditions**: When conditions involve aggregate functions or intricate logic, subqueries can simplify the structure.
3. **Readability**: For certain complex queries, subqueries can enhance readability and maintainability.

**Example**: Consider the tables employees and departments.

* **employees**:

| employee\_id | employee\_name | department\_id |
| --- | --- | --- |
| 1 | Alice | 1 |
| 2 | Bob | 2 |
| 3 | Charlie | 1 |
| 4 | David | 3 |

* **departments**:

| department\_id | department\_name |
| --- | --- |
| 1 | HR |
| 2 | Finance |
| 3 | IT |

**Query**: Retrieve the names of employees in the HR department.

sql

Copy code

SELECT employee\_name

FROM employees

WHERE department\_id = (SELECT department\_id

FROM departments

WHERE department\_name = 'HR');

In this scenario, the subquery retrieves the department\_id for the 'HR' department, which is then used by the outer query to filter employees.

### Joins

**Definition**: A join operation combines rows from two or more tables based on a related column, enabling a comprehensive result set encompassing data from multiple tables.

**Types of Joins**:

1. **INNER JOIN**: Returns rows where there is a match in both tables.
2. **LEFT JOIN (LEFT OUTER JOIN)**: Returns all rows from the left table and matched rows from the right table; unmatched rows from the right table result in NULLs.
3. **RIGHT JOIN (RIGHT OUTER JOIN)**: Returns all rows from the right table and matched rows from the left table; unmatched rows from the left table result in NULLs.
4. **FULL JOIN (FULL OUTER JOIN)**: Returns rows when there is a match in one of the tables; includes all rows from both tables with NULLs where there is no match.

**Usage Scenarios**:

1. **Combining Data**: When data needs to be retrieved from multiple tables and displayed together, joins are optimal.
2. **Retrieving Related Data**: Joins are ideal for fetching related data across tables, providing a unified result set.
3. **Performance**: Joins are often more performant than subqueries, especially when handling large datasets.

**Example**: **Query**: Retrieve the names of employees and their corresponding department names.

sql

Copy code

SELECT e.employee\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

This join operation combines data from employees and departments based on the department\_id, yielding a result set that includes both employee names and their department names.

### Comparative Analysis

**Subqueries**:

* **Ideal For**: Filtering and comparison operations, particularly when involving aggregate functions or complex conditions.
* **Example**: Determining employees with salaries higher than the company average.

sql

Copy code

SELECT employee\_name

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

**Joins**:

* **Ideal For**: Merging data from multiple tables into a single comprehensive result set, especially when related data needs to be displayed together.
* **Example**: Listing employees with their department and manager names.

sql

Copy code

SELECT e.employee\_name, d.department\_name, m.employee\_name AS manager\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id

LEFT JOIN employees m ON e.manager\_id = m.employee\_id;

### Conclusion

The decision to use subqueries or joins hinges on the specific requirements of the query. Subqueries are beneficial for complex filtering and aggregate comparisons, while joins are indispensable for combining related data from multiple tables. Understanding the nuances of each method ensures efficient and effective data retrieval in SQL.