Joins and Sub-queries in MySQL

1. Introduction to Joins

- **What are Joins?**
  - Joins are used to combine rows from two or more tables based on a related column between them.
  - They are essential for querying data from multiple tables in a relational database.
- **Why Use Joins?**
  - To retrieve data that is spread across multiple tables.
  - To establish relationships between tables using foreign keys.
  - To avoid data redundancy and maintain normalization.

2. Types of Joins

1. **INNER JOIN**:
   - Returns only the rows that have matching values in both tables.
   - Syntax:

     ```
     SELECT columns
     FROM table1
     INNER JOIN table2 ON table1.column = table2.column;
     ```

   - Example:

     ```
     SELECT students.name, enrollments.course_id
     FROM students
     INNER JOIN enrollments ON students.id = enrollments.student_id;
     ```

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 is found, `NULL` values are returned for columns from the right table.
   - Syntax:

     ```
     SELECT columns
     FROM table1
     LEFT JOIN table2 ON table1.column = table2.column;
     ```

- Example:

```
SELECT students.name, enrollments.course_id
FROM students
LEFT JOIN enrollments ON students.id = enrollments.student_id;
```

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 is found, `NULL` values are returned for columns from the left table.
   - Syntax:

   ```
   SELECT columns
   FROM table1
   RIGHT JOIN table2 ON table1.column = table2.column;
   ```

   - Example:

   ```
   SELECT students.name, enrollments.course_id
   FROM students
   RIGHT JOIN enrollments ON students.id = enrollments.student_id;
   ```

4. **FULL JOIN (or FULL OUTER JOIN)**:

   - Returns all rows when there is a match in either the left or right table.
   - If no match is found, `NULL` values are returned for missing sides.
   - Note: MySQL does not support `FULL JOIN` directly, but it can be emulated using `UNION` of `LEFT JOIN` and `RIGHT JOIN`.

5. **CROSS JOIN**:

   - Returns the Cartesian product of the two tables (all possible combinations of rows).
   - Syntax:

   ```
   SELECT columns
   FROM table1
   CROSS JOIN table2;
   ```

   - Example:

   ```
   SELECT students.name, courses.course_name
   FROM students
   CROSS JOIN courses;
   ```

3. Introduction to Sub-queries

- **What are Sub-queries?**
  - A sub-query is a query nested inside another query.
  - It is used to perform operations that require multiple steps.
  - Sub-queries can return a single value, a single row, multiple rows, or multiple columns.
- **Why Use Sub-queries?**
  - To break down complex queries into simpler parts.
  - To perform calculations or filtering based on intermediate results.
  - To compare a value against a set of values returned by another query.

4. Types of Sub-queries

1. **Single-Row Sub-query**:
   - Returns a single row with a single column.
   - Used with comparison operators like `=`, `>`, `<`, etc.
   - Example:

   ```
   SELECT name
   FROM students
   WHERE age = (SELECT MAX(age) FROM students);
   ```

2. **Multi-Row Sub-query**:
   - Returns multiple rows with a single column.
   - Used with operators like `IN`, `ANY`, `ALL`.
   - Example:

   ```
   SELECT name
   FROM students
   WHERE id IN (SELECT student_id FROM enrollments WHERE grade = 'A');
   ```

3. **Correlated Sub-query**:
   - A sub-query that depends on the outer query for its values.
   - Executed repeatedly, once for each row processed by the outer query.
   - Example:

   ```
   SELECT name
   FROM students s
   ```

```
WHERE EXISTS (SELECT 1 FROM enrollments e WHERE e.student_id = s.id);
```

4. **Scalar Sub-query**:

    - Returns a single value (one row and one column).
    - Can be used in `SELECT`, `WHERE`, or `HAVING` clauses.
    - Example:

    ```
    SELECT name, (SELECT COUNT(*) FROM enrollments WHERE student_id =
    students.id) AS total_courses
    FROM students;
    ```

---

5. Practical Use Cases

1. **Joins**:

    - Retrieve data from multiple related tables (e.g., students and their enrolled courses).
    - Combine tables to analyze relationships (e.g., departments and their instructors).

2. **Sub-queries**:

    - Find the maximum or minimum value in a table (e.g., the highest salary in the instructors table).
    - Filter data based on conditions from another table (e.g., students who scored an 'A' in any course).
    - Perform calculations or comparisons (e.g., students older than the average age).

---

6. Key Differences Between Joins and Sub-queries

Feature	Joins	Sub-queries
Purpose	Combine rows from multiple tables.	Perform intermediate calculations.
Performance	Generally faster for large datasets.	Can be slower due to nested execution.
Readability	Easier to read for simple queries.	Better for breaking down complex logic.
Use Case	Retrieving related data from tables.	Filtering or calculations based on results.

---

7. Best Practices

- Use **joins** when you need to combine data from multiple tables.
- Use **sub-queries** when you need to perform intermediate calculations or filtering.
- Avoid deeply nested sub-queries for better readability and performance.
- Use **aliases** to make queries more readable.

---

8. Common Mistakes to Avoid

- Forgetting to use proper join conditions, leading to Cartesian products.
- Using sub-queries where a join would be more efficient.
- Writing overly complex sub-queries that are hard to debug.

---

9. Summary

- **Joins** are used to combine data from multiple tables based on related columns.
- **Sub-queries** are used to perform intermediate calculations or filtering.
- Both are essential tools for querying relational databases effectively.
