Database Concepts: Joins, Sub-queries, Views & Indexes

1. JOINS and Their Types

What is a JOIN?

A JOIN is used to combine rows from two or more tables based on a related column between them.

Sample Tables for Examples:

Students Table:

StudentID	Name	Age	CourseID
1	Alice	20	101
2	Bob	21	102
3	Charlie	19	101
4	Diana	22	NULL

Courses Table:

CourseID	CourseName	Credits
101	Math	4
102	Physics	3
103	Chemistry	4

Types of JOINS:

1. INNER JOIN

Returns only matching records from both tables.

SQL Query:

```sql
SELECT Students.Name, Courses.CourseName
FROM Students
INNER JOIN Courses ON Students.CourseID = Courses.CourseID;
```

Result:

Name	CourseName
Alice	Math
Bob	Physics
Charlie	Math

Venn Diagram:

```
 Table A    Table B
  ___      ___
 |  |    |  |
 |  |____|  |
 |__|INNER|__|
    |JOIN |
    |____|
```

2. LEFT JOIN (LEFT OUTER JOIN)

Returns all records from the left table and matching records from the right table.

SQL Query:

```sql
SELECT Students.Name, Courses.CourseName
FROM Students
LEFT JOIN Courses ON Students.CourseID = Courses.CourseID;
```

Result:

Name	CourseName
Alice	Math
Bob	Physics
Charlie	Math
Diana	NULL

Venn Diagram:

```
Table A     Table B
  ___       ___
|###|     |  |
|###|____|  |
|###|LEFT |___|
|###|JOIN |
|###|____|
```

3. RIGHT JOIN (RIGHT OUTER JOIN)

Returns all records from the right table and matching records from the left table.

SQL Query:

```sql
SELECT Students.Name, Courses.CourseName
FROM Students
RIGHT JOIN Courses ON Students.CourseID = Courses.CourseID;
```

Result:

Name	CourseName
Alice	Math
Charlie	Math
Bob	Physics
NULL	Chemistry

Venn Diagram:

```
Table A     Table B
  ___       ___
|  |     |###|
|  |____|###|
|___|RIGHT|###|
    |JOIN |###|
    |____|###|
```

4. FULL OUTER JOIN

Returns all records when there's a match in either table.

SQL Query:

```sql
SELECT Students.Name, Courses.CourseName
FROM Students
FULL OUTER JOIN Courses ON Students.CourseID = Courses.CourseID;
```

Result:

Name	CourseName
Alice	Math
Bob	Physics
Charlie	Math
Diana	NULL
NULL	Chemistry

Venn Diagram:

```
  Table A     Table B
   ___      ___
  |###|     |###|
  |###|____|###|
  |###|FULL |###|
  |###|OUTER|###|
  |###|____|###|
```

5. CROSS JOIN

Returns the Cartesian product of both tables (all possible combinations).

SQL Query:

```sql
SELECT Students.Name, Courses.CourseName
FROM Students
CROSS JOIN Courses;
```

Result: (12 rows total - 4 students × 3 courses)

Name	CourseName
Alice	Math
Alice	Physics
Alice	Chemistry
Bob	Math
...	...

---

2. SUB-QUERIES

What is a Sub-query?

A sub-query is a query nested inside another query. It's also called an inner query or nested query.

Types of Sub-queries:

1. Single Row Sub-query

Returns exactly one row.

Example:

```sql
SELECT Name FROM Students
WHERE CourseID = (SELECT CourseID FROM Courses WHERE CourseName = 'Math');
```

Result:

Name
Alice
Charlie

2. Multiple Row Sub-query

Returns multiple rows.

Example:

```sql
SELECT Name FROM Students
WHERE CourseID IN (SELECT CourseID FROM Courses WHERE Credits > 3);
```

Result:

Name
Alice
Charlie

3. Correlated Sub-query

References columns from the outer query.

Example:

sql

```
SELECT Name FROM Students S1
WHERE Age > (SELECT AVG(Age) FROM Students S2 WHERE S2.CourseID = S1.CourseID);
```

Sub-query Locations:

In WHERE Clause:

sql

```
SELECT * FROM Students
WHERE CourseID = (SELECT CourseID FROM Courses WHERE CourseName = 'Physics');
```

In FROM Clause:

sql

```
SELECT AVG(Age) FROM
(SELECT Age FROM Students WHERE CourseID IS NOT NULL) AS ValidStudents;
```

In SELECT Clause:

sql

```
SELECT Name,
    (SELECT CourseName FROM Courses WHERE Courses.CourseID = Students.CourseID) AS Course
FROM Students;
```

---

3. VIEWS

What is a View?

A view is a virtual table based on the result of an SQL statement. It contains rows and columns just like a real table.

Creating Views:

Simple View:

```sql
CREATE VIEW StudentCourseView AS
SELECT Students.Name, Courses.CourseName, Courses.Credits
FROM Students
INNER JOIN Courses ON Students.CourseID = Courses.CourseID;
```

Using the View:

```sql
SELECT * FROM StudentCourseView;
```

Result:

Name	CourseName	Credits
Alice	Math	4
Bob	Physics	3
Charlie	Math	4

Advantages of Views:

- **Security**: Hide sensitive data
- **Simplicity**: Simplify complex queries
- **Consistency**: Ensure consistent data access
- **Abstraction**: Hide underlying table structure

View Diagram:

```
┌─────────────┐  ┌─────────────┐
│  Students   │  │   Courses   │
│  Table      │  │   Table     │
└─────────────┘  └─────────────┘
       │              │
       └──────────────────┘
              │
       ┌─────────────────┐
       │ StudentCourse   │
       │     View        │
       │   (Virtual)     │
       └─────────────────┘
```

---

4. INDEXES

What is an Index?

An index is a data structure that improves the speed of data retrieval operations on a database table.

Types of Indexes:

1. Clustered Index

- Physically reorders table data
- Only one per table
- Usually on primary key

```sql
CREATE CLUSTERED INDEX IX_Student_ID ON Students(StudentID);
```

2. Non-Clustered Index

- Separate structure pointing to table rows
- Multiple allowed per table

```sql
CREATE NONCLUSTERED INDEX IX_Student_Name ON Students(Name);
```

3. Unique Index

- Ensures uniqueness of values

```sql
```

```
CREATE UNIQUE INDEX IX_Student_Email ON Students(Email);
```

4. Composite Index

- Built on multiple columns

```sql
CREATE INDEX IX_Student_Name_Age ON Students(Name, Age);
```

Index Structure Diagram:

Without Index (Table Scan):

```
Query: Find "Bob"
┌─────────────────────────────────────┐
│ StudentID │ Name    │ Age │ CourseID │
├─────────────────────────────────────┤
│     1     │ Alice   │ 20 │   101    │ ← Check
│     2     │ Bob     │ 21 │   102    │ ← Check (Found!)
│     3     │ Charlie │ 19 │   101    │ ← Check
│     4     │ Diana   │ 22 │   NULL   │ ← Check
└─────────────────────────────────────┘
```

With Index (Index Seek):

```
Query: Find "Bob"
┌─────────────────┐  ┌─────────────────────────────────────┐
│ Name Index      │  │ StudentID │ Name    │ Age │ CourseID │
├─────────────────┤  ├─────────────────────────────────────┤
│ Alice    → 1    │  │     1     │ Alice   │ 20 │   101    │
│ Bob      → 2    │◄───►│   2     │ Bob     │ 21 │   102    │
│ Charlie  → 3    │  │     3     │ Charlie │ 19 │   101    │
│ Diana    → 4    │  │     4     │ Diana   │ 22 │   NULL   │
└─────────────────┘  └─────────────────────────────────────┘
```

When to Use Indexes:

Create Indexes When:

- Frequent WHERE clause conditions
- JOIN conditions
- ORDER BY clauses
- Large tables with many reads

Avoid Indexes When:

- Small tables
- Frequent INSERT/UPDATE/DELETE operations
- Limited storage space

Index Performance Impact:

```
Query Performance:
Without Index: O(n) - Linear search
With Index:    O(log n) - Tree search

Example with 1,000,000 rows:
Without Index: Up to 1,000,000 comparisons
With Index:    Up to 20 comparisons
```

Quick Reference Summary

JOIN Types:

- **INNER**: Only matching records
- **LEFT**: All from left + matching from right
- **RIGHT**: All from right + matching from left
- **FULL OUTER**: All records from both tables
- **CROSS**: Cartesian product

Sub-query Types:

- **Single Row**: Returns one row
- **Multiple Row**: Returns multiple rows
- **Correlated**: References outer query

View Benefits:

- Security, Simplicity, Consistency, Abstraction

Index Types:

- **Clustered**: Physical ordering
- **Non-Clustered**: Separate structure
- **Unique**: Ensures uniqueness
- **Composite**: Multiple columns

Remember: Practice with real databases to master these concepts!