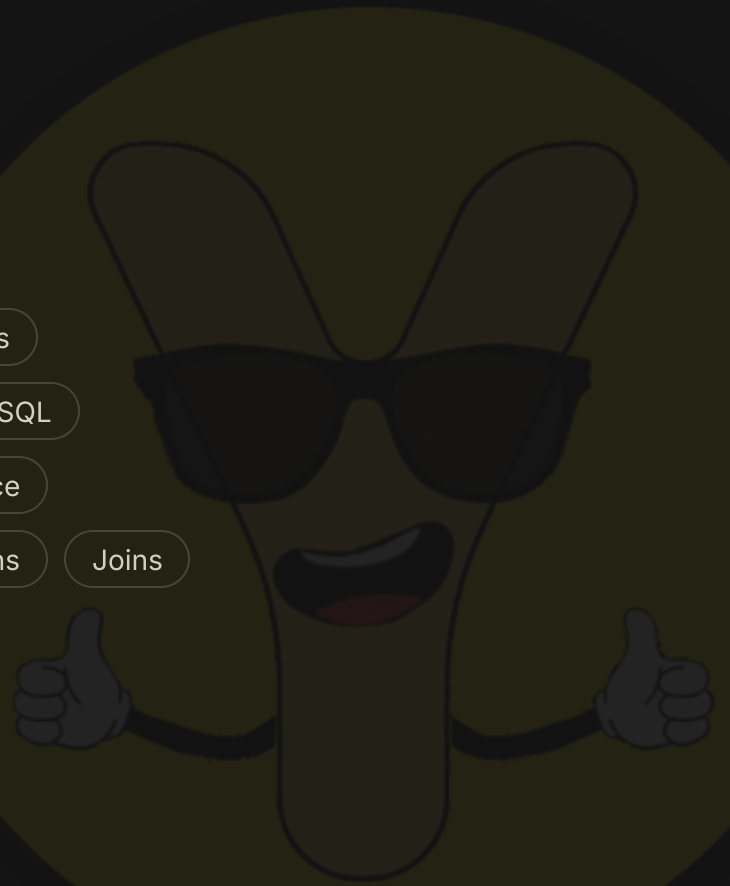
AKTU · 3RD YEAR · SEMESTER 5 · BCS501

Database Management System

Unit 2 — Relational Data Model and SQL

- Relational Data Model Concepts
- Integrity Constraints
- Entity Integrity
- Referential Integrity
- Keys Constraints
- Domain Constraints
- Relational Algebra
- Relational Calculus
- Tuple and Domain Calculus
- Characteristics of SQL
- Advantages of SQL
- SQL Data Type and Literals
- Types of SQL Commands
- SQL Operators and Their Precedence
- Tables, Views and Indexes
- Queries and Sub Queries
- Aggregate Functions
- Insert, Update and Delete Operations
- Joins



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

Relational Data Model Concepts

The Relational Data Model organizes data into one or more tables (relations) of rows and columns.

- Relation (Table): A two-dimensional table with rows and columns.
- Tuple (Row): A single record in a relation, representing a set of related data values.
- Attribute (Column): A named column in a relation, representing a specific characteristic or property.
- Domain: The set of all possible values for an attribute.
- Relation Schema: The logical design of a relation, including its name and attributes, e.g., $R(A_1, A_2, ..., A_n)$.
- Relation Instance: The actual data stored in a relation at a specific point in time.
- Degree: The number of attributes (columns) in a relation.
- Cardinality: The number of tuples (rows) in a relation.

Integrity Constraints

Rules enforced on data to ensure its accuracy, consistency, and reliability within a relational database.

- Purpose: To maintain the quality and validity of data, preventing erroneous entries and ensuring data relationships are correctly preserved.
- Types:
  - Domain Constraints: Define the permissible values for an attribute.
  - Entity Integrity Constraints: Ensure that primary key values are unique and not NULL.
  - Referential Integrity Constraints: Maintain consistency between related tables.
  - Key Constraints: Define unique identifiers for tuples within a relation.
- Enforcement: Applied automatically by the DBMS or explicitly defined by the database designer.

Entity Integrity Constraint

ENTITY INTEGRITY

A fundamental rule in the relational model stating that the primary key of a base relation cannot contain NULL values.

- Rationale: The primary key uniquely identifies each tuple (row) in a relation. If it were NULL, we wouldn't be able to uniquely identify a specific tuple.
- Uniqueness: Each primary key value must also be unique across all tuples in the relation.
- Example: In a `Students` table, `StudentID` is the primary key. If `StudentID` could be NULL, two different students might not have a unique identifier, leading to ambiguity.
- Implementation: Enforced by defining a column or set of columns as `PRIMARY KEY` during table creation, which implicitly sets `NOT NULL` and `UNIQUE`.

Referential Integrity Constraint

REFERENTIAL INTEGRITY

A rule that ensures that a foreign key value in a referencing relation either matches an existing primary key value in the referenced relation or is completely NULL.

- Relationship: It maintains consistency between two related tables, preventing orphaned records.
- Foreign Key: An attribute (or set of attributes) in one relation (the referencing relation) that refers to the primary key of another relation (the referenced relation).
- Actions on Deletion/Update:
  - CASCADE: Delete/update referencing rows when the referenced row is deleted/updated.
  - SET NULL: Set foreign key values to NULL when the referenced row is deleted/updated.
  - SET DEFAULT: Set foreign key values to a default value.
  - RESTRICT/NO ACTION: Prevent deletion/update of the referenced row if referencing rows exist.

Key Constraints

Keys are attributes or sets of attributes that uniquely identify tuples within a relation and establish relationships between relations.

- Super Key: Any set of attributes that uniquely identifies a tuple in a relation. It can contain redundant attributes.
- Candidate Key: A minimal super key; it uniquely identifies a tuple, and no proper subset of its attributes can uniquely identify tuples.
- Primary Key: A candidate key chosen by the database designer to uniquely identify tuples in a relation. It must be unique and NOT NULL.
- Alternate Key: Any candidate key that is not chosen as the primary key.
- Foreign Key: An attribute (or set of attributes) in one relation that refers to the primary key of another relation, establishing a link.
- Composite Key: A key that consists of two or more attributes to uniquely identify a tuple.

Domain Constraints

DOMAIN CONSTRAINT

Rules that specify the permissible set of values for an attribute, ensuring that data entered into a column is valid according to its defined type and range.

- Data Type: Specifies the type of data an attribute can hold (e.g., `INTEGER`, `VARCHAR(50)`, `DATE`, `BOOLEAN`).
- Length: Defines the maximum number of characters or digits allowed for an attribute.
- Value Range: Restricts the values an attribute can take (e.g., `AGE > 0 AND AGE < 150`).
- Default Values: Specifies a default value for an attribute if no value is explicitly provided during insertion.
- `NOT NULL` Constraint: Ensures that an attribute cannot store a NULL value.
- `CHECK` Constraint: Allows defining custom conditions that attribute values must satisfy.
- Example: An `Age` attribute must be an integer and greater than 0. A `Gender` attribute might be restricted to 'M', 'F', or 'O'.

Relational Algebra

RELATIONAL ALGEBRA

A procedural query language that takes one or two relations as input and produces a new relation as output. It forms the theoretical foundation for SQL.

- Procedural: Specifies *how* to retrieve data by providing a sequence of operations.
- Operations: Divided into two main groups:
  - Set Operations: Based on mathematical set theory (Union, Intersection, Difference, Cartesian Product).
  - Relational Operations: Specific to the relational model (Select, Project, Rename, Join, Division).
- Closure Property: The result of any relational algebra operation is always a relation, allowing operations to be nested.
- Use: Essential for understanding query optimization and the underlying logic of database systems.

Relational Algebra: Set Operations

These operations combine or compare tuples from two relations, requiring them to be union-compatible (same number and compatible types of attributes).

- Union ($\cup$): Combines all unique tuples from two relations. $R \cup S = \{t \mid t \in R \text{ or } t \in S\}$.
- Intersection ($\cap$): Returns tuples that are present in both relations. $R \cap S = \{t \mid t \in R \text{ and } t \in S\}$.
- Difference ($-$): Returns tuples present in the first relation but not in the second. $R - S = \{t \mid t \in R \text{ and } t \notin S\}$.
- Cartesian Product ($\times$): Combines every tuple from the first relation with every tuple from the second relation. If R has n_R tuples and S has n_S tuples, $R \times S$ will have $n_R \times n_S$ tuples. $R \times S = \{(t_R, t_S) \mid t_R \in R \text{ and } t_S \in S\}$.

Relational Algebra: Relational Operations

These operations manipulate the structure and content of individual relations.

- Select (σ): Filters tuples (rows) based on a specified condition. $\sigma_{condition}(R)$. Example: $\sigma_{DeptID='CS'}(Students)$ selects all students from the CS department.
- Project (π): Selects specific attributes (columns) from a relation, eliminating duplicate resulting tuples. $\pi_{A_1,A_2,...,A_k}(R)$. Example: $\pi_{Name,Email}(Students)$ selects only the Name and Email attributes.
- Rename (ρ): Renames a relation or its attributes. $\rho_{S(B_1,...,B_k)}(R)$ renames relation R to S and its attributes.
- Join ($\bowtie$): Combines tuples from two relations based on a common attribute or condition.
  - Theta Join: $(R \bowtie_\theta S)$ where θ is a join condition.
  - Equijoin: A theta join where θ uses only equality conditions.
  - Natural Join ($\bowtie$): An equijoin on all common attributes, removing duplicate columns.
- Division ($\div$): Used for queries involving "for all" or "every" conditions. $R \div S$ returns tuples from R that match all tuples in S on common attributes.

Relational Calculus

RELATIONAL CALCULUS

A non-procedural (declarative) query language that describes *what* data to retrieve without specifying *how* to retrieve it.

- Declarative: Users specify the properties of the desired result, and the DBMS determines the execution strategy.
- Based on Predicate Logic: Uses mathematical logic to define the conditions that the desired tuples must satisfy.
- Types:
  - Tuple Relational Calculus (TRC): Variables range over tuples.
  - Domain Relational Calculus (DRC): Variables range over domain values of attributes.
- Expressiveness: Both TRC and DRC are equivalent in expressive power to Relational Algebra.
- Foundation: Provides a theoretical basis for declarative query languages like SQL.

Tuple Relational Calculus (TRC) vs. Domain Relational Calculus (DRC)

Feature	Tuple Relational Calculus (TRC)	Domain Relational Calculus (DRC)
Variables	Range over tuples (rows) of a relation.	Range over domain values of attributes.
Syntax	Uses tuple variables. Example: $\{t \mid P(t)\}$	Uses domain variables. Example: $\{x_1, x_2, ..., x_n \mid P(x_1, ..., x_n)\}$
Focus	Focuses on selecting entire tuples that satisfy a condition.	Focuses on selecting specific attribute values that satisfy a condition.
Quantifiers	Uses existential ($\exists$) and universal ($\forall$) quantifiers.	Also uses existential ($\exists$) and universal ($\forall$) quantifiers.
Example (TRC)	$\{t \mid Employee(t) \land t.Salary > 50000\}$	N/A
Example (DRC)	N/A	$\{< N, A, S > \mid \exists I(Employee(I, N, A, S) \land S > 50000)\}$ (where I,N,A,S are domain variables for ID, Name, Age, Salary)

Characteristics of SQL (Structured Query Language)

SQL is a standard language for managing and manipulating relational databases.

- Declarative: Users specify *what* data they need, not *how* to retrieve it.
- Standardized: An ANSI/ISO standard, ensuring portability across different DBMS.
- Data Definition Language (DDL): Used to define and modify database schema (e.g., `CREATE`, `ALTER`, `DROP`).
- Data Manipulation Language (DML): Used to retrieve, insert, update, and delete data (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`).
- Data Control Language (DCL): Used to manage user permissions and access control (e.g., `GRANT`, `REVOKE`).
- Transaction Control Language (TCL): Used to manage transactions (e.g., `COMMIT`, `ROLLBACK`, `SAVEPOINT`).
- Interactive: Allows users to directly interact with the database.
- Embedded: Can be embedded within other programming languages (e.g., Java, Python, C++).

Advantages of SQL

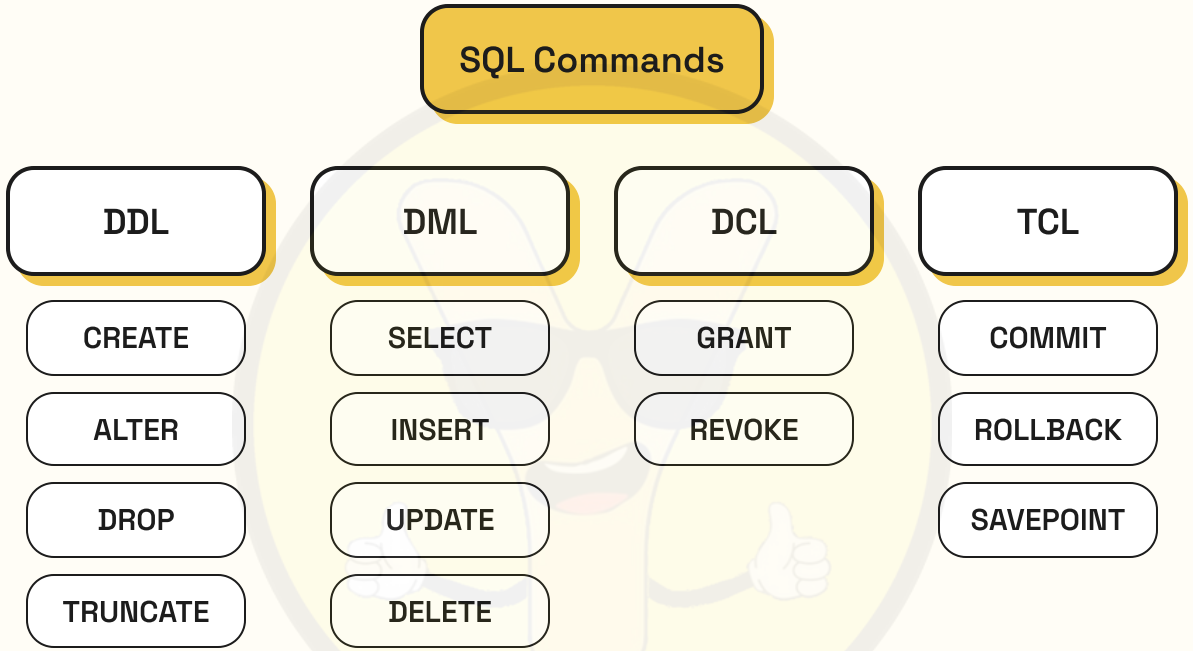
SQL offers numerous benefits that make it the industry standard for relational database management.

- High Performance: Optimized for efficient data retrieval and manipulation, especially for large datasets.
- Standard Language: Widely adopted and supported by almost all relational database systems (Oracle, MySQL, SQL Server, PostgreSQL).
- Easy to Learn: Relatively simple syntax, resembling natural English, making it accessible for beginners.
- Portability: SQL code is largely portable across different database systems, reducing vendor lock-in.
- Data Integrity: Supports various integrity constraints (primary key, foreign key, check) to ensure data accuracy and consistency.
- Client-Server Architecture: Facilitates communication between client applications and database servers.
- Scalability: Can handle large volumes of data and a high number of concurrent users.
- Powerful Querying: Capable of handling complex queries, aggregations, and data transformations.

SQL Data Types and Literals

Category	Data Type (Examples)	Description	Literals (Examples)
Numeric	INT, INTEGER	Whole numbers	123, -45
Numeric	DECIMAL(p,s), NUMERIC	Fixed-point numbers (precision, scale)	123.45, -0.01
Numeric	FLOAT, REAL, DOUBLE	Floating-point numbers	3.14e+2, 0.005
String	VARCHAR(n), TEXT	Variable-length character string	'Hello World', 'SQL'
String	CHAR(n)	Fixed-length character string	'A', 'XYZ'
Date/Time	DATE	Date (YYYY-MM-DD)	'2023-10-27'
Date/Time	TIME	Time (HH:MM:SS)	'14:30:00'
Date/Time	DATETIME, TIMESTAMP	Date and time	'2023-10-27 14:30:00'

Types of SQL Commands



SQL commands are categorized based on their function in managing and manipulating database objects and data.

SQL Operators and Their Precedence

SQL operators are used to perform operations on data values, and precedence rules determine the order of evaluation.

- Arithmetic Operators: `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulo).
- Comparison Operators: `=`, `≠` (or `<>`), `>`, `<`, `≥`, `≤` (for comparing values).
- Logical Operators: `AND`, `OR`, `NOT` (for combining conditions).
- Set Operators: `UNION`, `INTERSECT`, `EXCEPT` / `MINUS` (for combining query results).
- Special Operators: `BETWEEN`, `LIKE`, `IN`, `IS NULL`, `EXISTS`, `ALL`, `ANY`.
- Operator Precedence (Highest to Lowest):
  - 1. Parentheses `()`
  - 2. Arithmetic operators (`*`, `/`, `%` then `+`, `-`)
  - 3. Comparison operators (`=`, `>`, `<`, etc.)
  - 4. `NOT`
  - 5. `AND`
  - 6. `OR`
- Example: `WHERE (Age > 20 AND Salary < 50000) OR Department = 'IT'`

Tables, Views and Indexes

These are fundamental database objects used for storing, presenting, and optimizing data access.

- Tables:
  - The primary storage units in a relational database, organizing data into rows and columns.
  - Defined using `CREATE TABLE` statement with specified columns, data types, and constraints.
  - Example: `CREATE TABLE Students (ID INT PRIMARY KEY, Name VARCHAR(50), Age INT);`
- Views:
  - Virtual tables based on the result set of a SQL query. They do not store data themselves.
  - Provide a customized, simplified, or security-enhanced representation of data from one or more tables.
  - Example: `CREATE VIEW YoungStudents AS SELECT ID, Name FROM Students WHERE Age < 20;`
- Indexes:
  - Special lookup tables that the database search engine can use to speed up data retrieval.
  - Created on one or more columns of a table to quickly locate data without scanning the entire table.
  - Example: `CREATE INDEX idx_student_name ON Students (Name);`
  - Types: Clustered (determines physical order) and Non-clustered (separate structure).

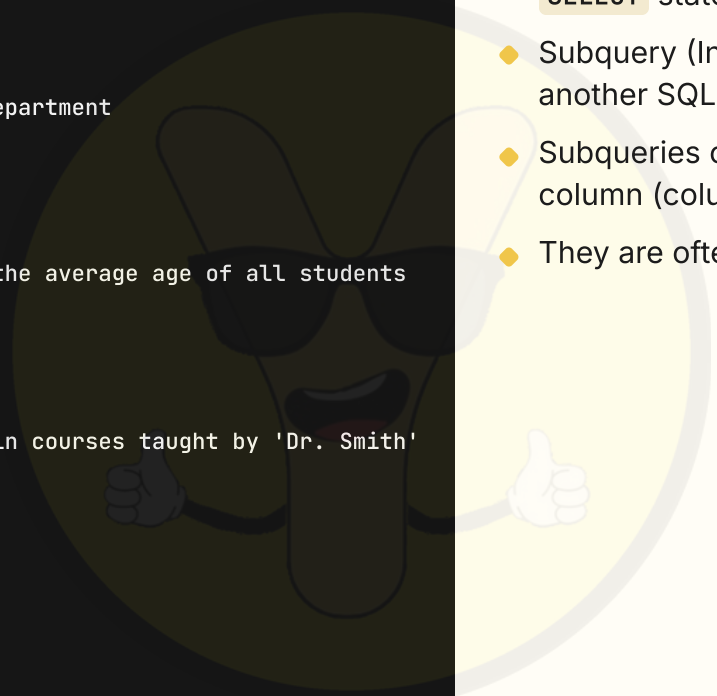
Queries and Sub Queries

```sql
-- Basic Query: Select all students
SELECT *
FROM Students;

-- Query with WHERE clause: Select students from 'CS' department
SELECT ID, Name, Department
FROM Students
WHERE Department = 'CS';

-- Subquery: Select students whose age is greater than the average age of all students
SELECT ID, Name, Age
FROM Students
WHERE Age > (SELECT AVG(Age) FROM Students);

-- Subquery with IN operator: Select students enrolled in courses taught by 'Dr. Smith'
SELECT S.Name
FROM Students S
WHERE S.ID IN (
    SELECT E.StudentID
    FROM Enrollments E
    JOIN Courses C ON E.CourseID = C.CourseID
    WHERE C.Instructor = 'Dr. Smith'
);
```



- Query: A request for data or information from a database. The `SELECT` statement is used for querying.
- Subquery (Inner Query/Nested Query): A query nested inside another SQL query.
- Subqueries can return a single value (scalar subquery), a single column (column subquery), or a table (row subquery).
- They are often used in `WHERE`, `HAVING`, `FROM`, or `SELECT` clauses.

Aggregate Functions

SQL aggregate functions perform calculations on a set of rows and return a single summary value.

- COUNT(): Returns the number of rows that match a specified criterion.
  - `COUNT(*)`: Counts all rows.
  - `COUNT(column_name)`: Counts non-NULL values in a column.
  - `COUNT(DISTINCT column_name)`: Counts unique non-NULL values.
- SUM(column_name): Calculates the sum of all values in a numeric column.
- AVG(column_name): Calculates the average of all values in a numeric column.
- MIN(column_name): Returns the minimum value in a column.
- MAX(column_name): Returns the maximum value in a column.
- GROUP BY Clause: Groups rows that have the same values in specified columns into summary rows.
- HAVING Clause: Filters groups based on a specified condition, similar to `WHERE` but for groups.
- Example: `SELECT Department, AVG(Salary) FROM Employees GROUP BY Department HAVING AVG(Salary) > 60000;`

Insert, Update, and Delete Operations (DML)

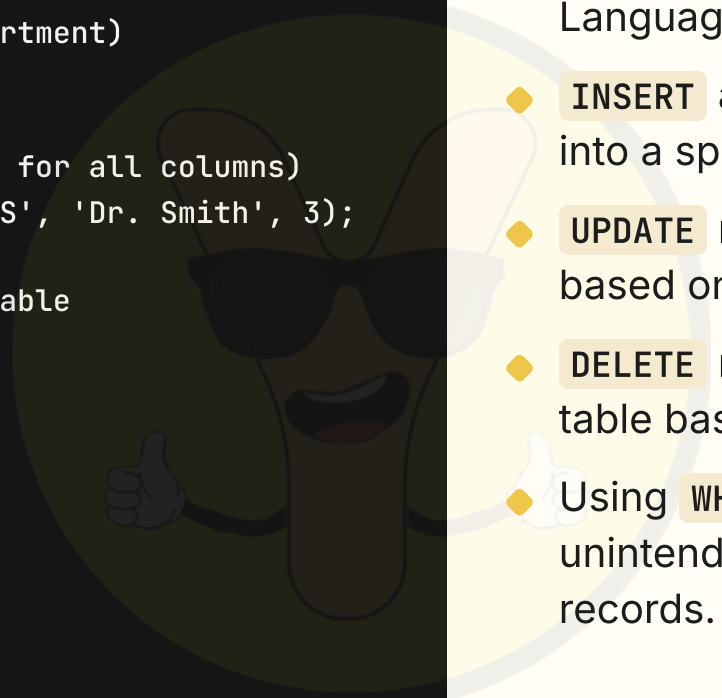
```
-- INSERT: Adding new rows to a table
INSERT INTO Students (ID, Name, Age, Department)
VALUES (101, 'Alice', 20, 'CS');

-- INSERT with specific columns (optional for all columns)
INSERT INTO Courses VALUES ('CS101', 'DBMS', 'Dr. Smith', 3);

-- UPDATE: Modifying existing data in a table
UPDATE Students
SET Age = 21, Department = 'IT'
WHERE ID = 101;

-- DELETE: Removing rows from a table
DELETE FROM Students
WHERE Age > 25;

-- DELETE all rows (without WHERE clause)
-- DELETE FROM Students;
```



- These are fundamental Data Manipulation Language (DML) commands.
- `INSERT` adds one or more new records (rows) into a specified table.
- `UPDATE` modifies existing records in a table based on a specified condition.
- `DELETE` removes one or more records from a table based on a specified condition.
- Using `WHERE` clause is crucial to avoid unintended modifications or deletions of all records.

Joins

Joins combine rows from two or more tables based on a related column between them, allowing retrieval of data from multiple sources.

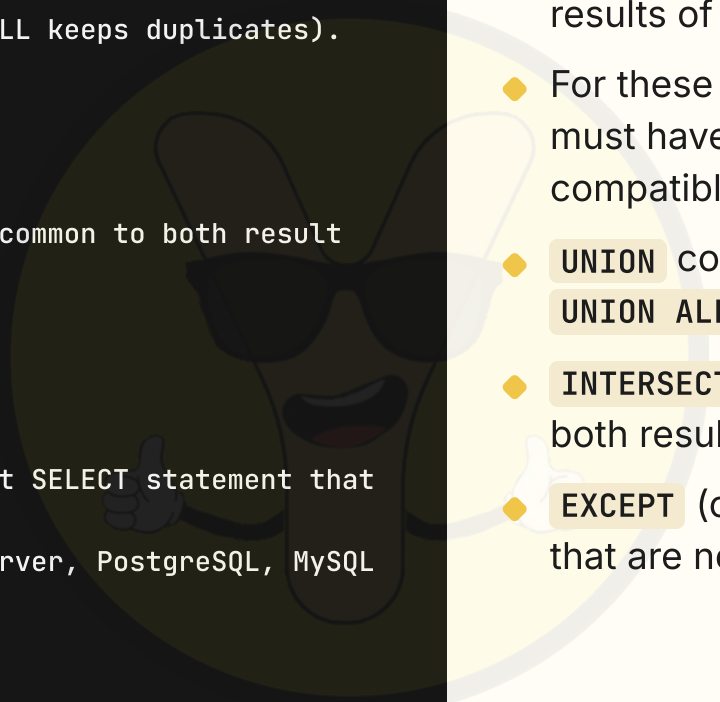
- INNER JOIN: Returns only the rows that have matching values in both tables. Most common type of join. Example: `SELECT S.Name, C.CourseName FROM Students S INNER JOIN Enrollments E ON S.ID = E.StudentID JOIN Courses C ON E.CourseID = C.CourseID;`
- LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table, and the matching rows from the right table. If no match, NULLs are returned for the right table's columns.
- RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table, and the matching rows from the left table. If no match, NULLs are returned for the left table's columns.
- FULL JOIN (or FULL OUTER JOIN): Returns all rows when there is a match in one of the tables. Returns rows from both tables, with NULLs for non-matching sides.
- CROSS JOIN: Produces the Cartesian product of the two tables, combining every row from the first table with every row from the second. (Similar to Relational Algebra's Cartesian Product).
- SELF JOIN: A table is joined with itself, typically using aliases to treat it as two separate tables.

UNION, INTERSECT, EXCEPT/MINUS

```
-- UNION: Combines the result sets of two or more SELECT statements.
-- Removes duplicate rows by default (UNION ALL keeps duplicates).
SELECT Name, City FROM Employees
UNION
SELECT Name, City FROM Customers;

-- INTERSECT: Returns only the rows that are common to both result
sets.
SELECT EmployeeID FROM Employees
INTERSECT
SELECT StudentID FROM Students;

-- EXCEPT / MINUS: Returns rows from the first SELECT statement that
are not present in the second.
-- (MINUS is used in Oracle, EXCEPT in SQL Server, PostgreSQL, MySQL
8+)
SELECT Name FROM AllPeople
EXCEPT
SELECT Name FROM Employees;
```



- These are SQL set operators used to combine the results of two or more `SELECT` statements.
- For these operators to work, the `SELECT` statements must have the same number of columns and compatible data types in corresponding columns.
- `UNION` combines results, removing duplicates (use `UNION ALL` to keep them).
- `INTERSECT` returns only the common rows found in both result sets.
- `EXCEPT` (or `MINUS`) returns rows from the first query that are not found in the second query.

Cursors, Triggers, and Procedures in SQL/PL SQL

These advanced SQL features provide more control and automation for database operations.

- Cursors:
  - A database object that allows traversal over the rows of a result set, one row at a time.
  - Used in procedural SQL (like PL/SQL, T-SQL) to process individual rows returned by a `SELECT` statement.
  - Steps: DECLARE, OPEN, FETCH, CLOSE, DEALLOCATE.
  - Useful for row-by-row processing where set-based operations are not suitable.
- Triggers:
  - Stored procedural code that is automatically executed (triggered) in response to certain events on a table or view.
  - Events: `INSERT`, `UPDATE`, `DELETE`.
  - Timing: `BEFORE` or `AFTER` the event.
  - Use cases: Enforcing complex business rules, auditing changes, maintaining derived data.
- Procedures (Stored Procedures):
  - A named block of SQL statements (and procedural logic) that is stored in the database and can be executed on demand.
  - Advantages: Improved performance (pre-compiled), reduced network traffic, enhanced security, reusability.
  - Can accept input parameters and return output parameters.
  - Often written in PL/SQL (Oracle), T-SQL (SQL Server), or similar procedural extensions.

Exam corner

- **Q1** Explain the fundamental concepts of the Relational Data Model, including relation, tuple, attribute, domain, degree, and cardinality.
- **Q2** Differentiate between Entity Integrity and Referential Integrity constraints with suitable examples.
- **Q3** Describe the various types of keys used in a relational database. Explain the difference between a Candidate Key and a Primary Key.
- **Q4** Explain the basic set operations (Union, Intersection, Difference, Cartesian Product) and relational operations (Select, Project, Join) of Relational Algebra with examples and their notations.
- **Q5** Compare and contrast Relational Algebra and Relational Calculus. Briefly explain Tuple Relational Calculus and Domain Relational Calculus.
- **Q6** List and explain the different types of SQL commands (DDL, DML, DCL, TCL) with at least two examples for each category.
- **Q7** Write SQL queries to demonstrate the use of aggregate functions (COUNT, SUM, AVG, MIN, MAX) along with GROUP BY and HAVING clauses.
- **Q8** Explain the concept of Joins in SQL. Describe INNER JOIN, LEFT JOIN, and FULL JOIN with suitable examples.

Unit 2 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 3 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

