VG

# MCA-105

## DATABASE SYSTEMS

# PREFACE

Dear Reader,

It is a privilege to present this comprehensive collection of **RTU MCA Semester Examination Notes**, designed to cover the complete syllabus with clarity and academic accuracy. Every concept, definition, explanation, and example has been organized to support thorough understanding and effective exam preparation.

The purpose of these notes is to simplify complex topics and provide a reliable study resource that can assist in both detailed learning and quick revision. Continuous effort has been made to ensure correctness and relevance; however, learning grows when readers engage, question, and explore further.

May these notes serve as a strong academic foundation and contribute meaningfully to your preparation and future growth.

Warm regards,
**Virendra Goura**
Author
www.virendragoura.com

# DISCLAIMER

This e-book has been created with utmost care, sincere effort, and extensive proofreading. However, despite all attempts to avoid mistakes, there may still be some errors, omissions, or inaccuracies that remain unnoticed.
This e-book is issued with the understanding that neither the author nor the publisher shall be held responsible for any loss, damage, or misunderstanding arising from the use of the information contained within.
All content provided is for educational and informational purposes only.

# MCA-105 Database System

## Unit-1

**Introduction:** Overview of DBMS, Database System v/s File System, Architecture of DBMS, Data models, Entity Relationship Diagram, Types of Keys, Integrity Rules, Data Dictionary, Normalization (1NF, 2 NF, 3NF, BCNF, 4NF, 5NF), inclusion dependencies, loss less join decompositions, Codd's Rules.

## Unit-2

**Transaction Management: Transactions:** Concepts, ACID Properties, States Of Transaction, Serializaibility, Conflict & View Serializable Schedule, Checkpoints, Deadlock Handling.

## Unit-3

**Database Querying & Concurrency Control:** Relational Algebra, Set Operations, Relational Calculus, Steps In Query Processing, Algorithms For Selection, Sorting And Join Operations, Understanding Cost Issues In Queries, Query Optimization, Transformation Of Relational Expressions, Query Evaluation Plans.

**Concurrency Control:** Locks Based Protocols, Time Stamp Based Protocols, Validation Based Protocol, Multiple Granularity, Multi-version Schemes.

## Unit-4

**Recovery System & Security:** Failure Classifications, Recovery & Atomicity, Log Base Recovery, Recovery with Concurrent Transactions, Shadow Paging, Failure with Loss of Non-Volatile Storage, Recovery From Catastrophic Failure, Introduction to Security & Authorization, Introduction to emerging Databases-OODBMS, ORDBMS, Distributed database, Multimedia database, Special database-limitations of conventional databases, advantages of emerging databases.

## Unit-5

**SQL and PL/SQL: Introduction to SQL:** Characteristics of SQL, Advantages of SQL, SQL data types and literals, Types of SQL commands, SQL operators, Tables, views and indexes, Constraints, Group By and Having Clause, Order By Clause, Queries and sub queries, Functions, PL/SQL basics, blocks, architecture, variables, constants, attributes, character set, PL/SQL control structure, data types, conditional and sequential control statements, cursors, exceptions, triggers, functions, procedures and packages.

# UNIT 1– Introduction

**Introduction:** Overview of DBMS, Database System v/s File System, Architecture of DBMS, Data models, Entity Relationship Diagram, Types of Keys, Integrity Rules, Data Dictionary, Normalization (INF, 2 NF, 3NF, BCNF, 4NF, 5NF), inclusion dependencies, loss less join decompositions, Codd's Rules.

## 1. Introduction

A **Database Management System (DBMS)** is software that enables users to **store, organize, retrieve, and manage data efficiently**.
It provides a systematic and controlled environment to handle large volumes of related data.

In simple words, **DBMS acts as an interface** between the user and the database.

**Example DBMS:** MySQL, Oracle, SQL Server, PostgreSQL, MongoDB.

### Need for DBMS

Before DBMS, organizations used **file-based systems**, which had major drawbacks:

- Data redundancy (same data stored multiple times)
- Lack of data security
- Difficulty in access and modification
- No concurrency control
- No data integrity

DBMS solves these issues through **centralized control**, **data independence**, and **query optimization**.

## 2. Database System vs File System

| Basis | File System | Database System (DBMS) |
|---|---|---|
| **Data Storage** | Data stored in separate files | Data stored in an integrated database |
| **Data Redundancy** | High redundancy | Controlled redundancy |
| **Data Access** | Complex (requires user-written programs) | Easy (through SQL) |
| **Security** | Minimal | High-level user access control |
| **Integrity** | Hard to maintain | Enforced by integrity constraints |
| **Backup & Recovery** | Manual | Automatic and managed by DBMS |
| **Concurrency** | Not supported | Multiple users can access data simultaneously |
| **Example** | Traditional file storage | MySQL, Oracle, SQL Server |

## 3. Architecture of DBMS

DBMS follows a **three-level architecture**, known as **ANSI/SPARC Architecture**.
It helps in achieving **data abstraction** and **data independence**.

### 1. Internal Level (Physical Level)

- Describes how data is physically stored on storage devices.
- Deals with indexes, pointers, and storage structures.

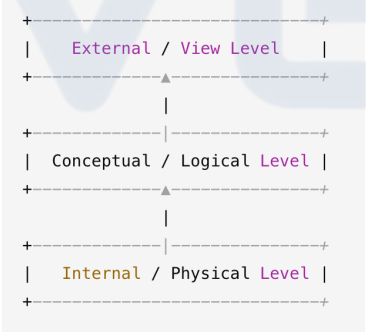
### 2. Conceptual Level (Logical Level)

- Describes **what data is stored** and the **relationships among data**.
- Independent of hardware or software.

### 3. External Level (View Level)

- The highest level of abstraction.
- Describes how individual users view the data (specific portions only).

**Diagram: DBMS Three-Level Architecture**




**Data Independence**

- **Logical Data Independence:** Changes in logical schema don't affect external schema.
- **Physical Data Independence:** Changes in physical storage don't affect logical schema.

## 4. Data Models

A **data model** defines how data is logically structured, organized, and manipulated in a database.

**Types of Data Models:**

| Model | Description | Example |
|---|---|---|
| **Hierarchical Model** | Data organized in tree structure; parent-child relationship | IBM IMS |
| **Network Model** | Many-to-many relationship using pointers | IDMS |
| **Relational Model** | Data stored in tables (relations) | MySQL, Oracle |
| **Entity-Relationship Model (ER Model)** | Uses diagrams to represent entities and relationships | Conceptual design |
| **Object-Oriented Model** | Combines objects with data and behavior | ObjectDB, Postgres-Object |

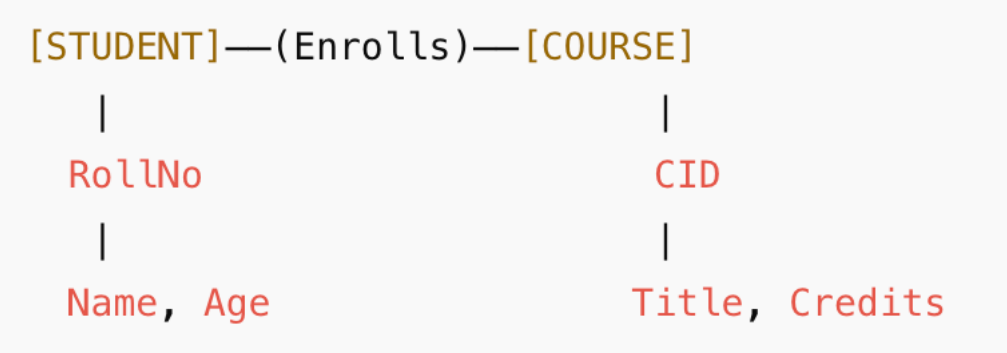
## 5. Entity Relationship Diagram (ER Diagram)

An **ER Diagram** is a visual representation of the entities, attributes, and relationships among them in a database.

**Basic Components of ER Diagram**

| Component | Symbol / Representation | Description |
|---|---|---|
| **Entity** | Rectangle | Represents an object (e.g., Student, Course) |
| **Attribute** | Oval | Property of an entity (e.g., Name, RollNo) |
| **Relationship** | Diamond | Association between entities (e.g., Enrolls) |
| **Primary Key** | Underlined Attribute | Uniquely identifies an entity |
| **Cardinality** | 1:1, 1:N, M:N | Defines participation between entities |

**Example ER Diagram**




**Meaning:**
A student *enrolls in* many courses; each course may have many students (Many-to-Many).

## 6. Types of Keys

Keys are attributes or sets of attributes that **identify records uniquely** in a relation.

| Key Type | Description | Example |
|---|---|---|
| **Super Key** | Any combination of attributes that uniquely identifies a row | (RollNo, Name) |
| **Candidate Key** | Minimal set of attributes uniquely identifying a record | RollNo |
| **Primary Key** | Chosen candidate key to identify records uniquely | RollNo |
| **Alternate Key** | Remaining candidate keys after selecting primary | EmailID |
| **Foreign Key** | Attribute that refers to a primary key in another table | CourseID in ENROLLS table |
| **Composite Key** | Combination of attributes to form a unique key | (StudentID, CourseID) |

## 7. Integrity Rules

Integrity rules ensure **accuracy, consistency, and validity** of data.

**1. Entity Integrity**

- Primary key value cannot be NULL.
  **Example:** Each student must have a unique RollNo.

**2. Referential Integrity**

- A foreign key must either match a primary key in another table or be NULL.
  **Example:** A course record must exist before a student enrolls in it.

**3. Domain Integrity**

- Values must be within valid range or domain.
  **Example:** Age must be between 0 and 120.

## 8. Data Dictionary

A **Data Dictionary** is a **centralized repository of metadata** (data about data).
It describes:

- Tables and columns
- Data types
- Constraints and relationships
- User privileges

**Purpose:**

- Acts as a reference guide for developers and administrators.
- Ensures consistency and documentation.

**Example Table:**

| Table Name | Column Name | Data Type | Key | Description |
|---|---|---|---|---|
| Student | RollNo | INT | PK | Unique Student ID |
| Student | Name | VARCHAR | - | Student Name |
| Course | CID | CHAR(5) | PK | Course ID |
| Course | Title | VARCHAR | - | Course Title |

## 9. Normalization

Normalization is a process of **organizing data to minimize redundancy** and **improve data integrity**.
It divides a large table into smaller related tables based on functional dependencies.

**1NF (First Normal Form)**

- Each column contains **atomic (indivisible)** values.
- No repeating groups allowed.

**Example:**

| RollNo | Name | Courses |
|---|---|---|
| 101 | Veer | DBMS, OS |

→ Convert to:

| RollNo | Name | Course |
|---|---|---|
| 101 | Veer | DBMS |
| 101 | Veer | OS |

**2NF (Second Normal Form)**

- Must be in 1NF.
- No **partial dependency** (non-key attributes depend on only part of a composite key).

**3NF (Third Normal Form)**

- Must be in 2NF.
- No **transitive dependency** (non-key attribute depends on another non-key).

**BCNF (Boyce–Codd Normal Form)**

- Stronger form of 3NF.
- For every dependency X → Y, X should be a **super key**.

**4NF (Fourth Normal Form)**

- Removes **multi-valued dependencies**.
- A table should not have two or more independent multi-valued facts about an entity.

**5NF (Fifth Normal Form)**

- Removes **join dependencies**.
- Every join dependency in the table is implied by candidate keys.

## 10. Inclusion Dependencies

Inclusion dependency ensures that **a set of values in one relation must appear in another relation** — often used for enforcing referential integrity.

**Example:**

Student(RollNo, Name)
Enroll(RollNo, CourseID)
Here, RollNo in **Enroll** must be included in RollNo of **Student**.

## 11. Lossless Join Decomposition

When a relation is decomposed into smaller relations, **data should not be lost** after joining them back.

**Condition for Lossless Join:**

If at least one of the decomposed relations contains a key of the original relation, the join is **lossless**.

**Example:**

R(A, B, C)
Decompose into: R1(A, B) and R2(B, C)
→ Common attribute B is a key → Lossless Join

## 12. Codd’s 12 Rules for RDBMS

Proposed by **E.F. Codd**, these rules define what qualifies as a **true relational database system**.

| Rule No. | Rule Name | Description |
|---|---|---|
| 1 | Information Rule | Data represented as values in tables |
| 2 | Guaranteed Access Rule | Each data item accessible by table name, primary key, column name |
| 3 | Systematic Null Values | Supports NULL to represent missing information |
| 4 | Dynamic Online Catalog | Metadata stored in data dictionary |
| 5 | Comprehensive Data Sub-Language | Must support at least one relational language (like SQL) |
| 6 | View Updating Rule | All theoretically updatable views must be updatable |
| 7 | High-Level Insert, Update, Delete | Should handle multiple rows at once |
| 8 | Physical Data Independence | Changes in storage do not affect schema |
| 9 | Logical Data Independence | Changes in logical structure do not affect application |
| 10 | Integrity Independence | Integrity constraints stored in catalog |
| 11 | Distribution Independence | Database distribution should be transparent |
| 12 | Non-Subversion Rule | Low-level access should not bypass integrity rules |

# UNIT 2– Transactions

**Transactions:** Concepts, ACID Properties, States Of Transaction, Serializaibility, Conflict & View Serializable Schedule, Checkpoints, Deadlock Handling.

## 1. Introduction

A **transaction** is a **logical unit of work** that performs a sequence of database operations such as read, write, update, or delete.
Each transaction transforms the database from one **consistent state** to another.

Transactions are essential to ensure **data consistency**, **isolation**, and **reliability**, especially in multi-user environments where multiple operations occur simultaneously.

**Example:**
A banking transaction that transfers ₹1000 from **Account A to Account B** involves:

```
T1:
  read(A)
  A = A – 1000
  write(A)
  read(B)
  B = B + 1000
  write(B)
```

If both parts execute successfully → transaction is complete.
If failure occurs midway → database must roll back to previous state.

## 2. Properties of Transactions (ACID Properties)

To maintain reliability, a transaction must satisfy the **ACID properties** — Atomicity, Consistency, Isolation, and Durability.

| Property | Meaning | Description |
|---|---|---|
| **Atomicity** | "All or Nothing" | Either the entire transaction executes or none of it does. |
| **Consistency** | Preserve Database Rules | The database must move from one valid state to another. |
| **Isolation** | Independence | Transactions must execute independently without interference. |
| **Durability** | Permanent Results | Once committed, results must survive system failures. |

**Detailed Explanation**

**1. Atomicity**

Ensures that a transaction is treated as a single, indivisible unit.
If one part fails, the entire transaction is rolled back.
**Example:** If debit succeeds but credit fails → rollback to restore both accounts.

**2. Consistency**

Every transaction must leave the database in a valid and consistent state.
All integrity constraints (like key or domain rules) must remain satisfied.

**3. Isolation**

Even if multiple transactions run concurrently, the intermediate results of one transaction should not be visible to others.
**Example:** Two users withdrawing money from the same account must see consistent balance.

**4. Durability**

After a transaction is committed, changes are permanently stored in the database — even if a crash occurs immediately after.

## 3. States of a Transaction

A transaction passes through multiple states during its execution.


| State | Description |
|---|---|
| **Active** | Transaction is being executed. |
| **Partially Committed** | All operations executed; waiting for commit. |
| **Committed** | Successful completion; all changes saved permanently. |
| **Failed** | Some error occurred (system crash, constraint violation, etc.). |
| **Aborted** | All changes undone using rollback; database restored to previous consistent state. |

**Commit and Rollback Commands**

- **COMMIT:** Makes all changes permanent.
- **ROLLBACK:** Undoes all changes of the current transaction.

**Example:**

```
BEGIN TRANSACTION
  UPDATE Account SET Balance = Balance - 500 WHERE AccNo = 101;
  UPDATE Account SET Balance = Balance + 500 WHERE AccNo = 202;
COMMIT;
```
If any update fails → ROLLBACK; restores both accounts.

## 4. Serializability

In a multi-user system, several transactions may execute **concurrently**.
DBMS must ensure that the final result is **equivalent to some serial (one-by-one) execution**.
This property is called **Serializability**.

It is the **primary concurrency control goal**.

**Types of Serializability**

1. **Conflict Serializability**
2. **View Serializability**

### 4.1 Conflict Serializability

Two operations are said to **conflict** if:

- They belong to different transactions.
- They access the same data item.
- At least one of them is a **write** operation.

**Example of Conflicting Operations:**

T1: Read(A)
T2: Write(A)

**Rules to Check Conflict Serializability**

1. Construct a **precedence (serialization) graph**:
    - Nodes → transactions
    - Edge Ti → Tj if Ti's operation precedes and conflicts with Tj
2. If the graph is **acyclic (no cycles)** → schedule is **conflict serializable**.

**Example:**

Schedule:
T1: R(A), W(A)
T2: R(A), W(A)

Graph:
T1 → T2 (since T1 writes A before T2 reads/writes A)
No cycles → **Conflict Serializable**.

### 4.2 View Serializability

Two schedules are **view-equivalent** if:

1. Each transaction reads the same initial value of data in both schedules.

2. Each transaction performs the same final write.
3. The final result of both schedules is identical.

If a schedule is **view-equivalent to a serial schedule**, it is **view serializable**.

**Difference Between Conflict & View Serializability**

| Aspect | Conflict Serializability | View Serializability |
|---|---|---|
| **Basis** | Order of conflicting operations | Read and write visibility |
| **Test** | Uses precedence graph | Uses view equivalence rules |
| **Complexity** | Easier to check | Harder to check |
| **Relation** | All conflict serializable schedules are view serializable | Reverse not always true |

## 5. Checkpoints

A **checkpoint** is a point in time where the DBMS saves the current state of the database into stable storage.
It acts as a **snapshot** to help the recovery system restore the database after failure.

**Purpose of Checkpoints**

- To **reduce recovery time** after a crash.
- To **avoid redoing** all transactions from the beginning.
- To **mark safe points** in the transaction log.

**Checkpoint Procedure**

1. Stop accepting new transactions.
2. Ensure all committed transactions are written to disk.
3. Record a **checkpoint entry** in the log.
4. Resume transaction execution.

**Example:**

<Checkpoint T1, T2>
<Transaction Commit T1>
<Crash>

Only transactions active after checkpoint (T2) are considered during recovery.

## 6. Deadlocks in DBMS

A **deadlock** occurs when two or more transactions are **waiting indefinitely** for resources held by each other, preventing further progress.

**Example**

T1 locks data A → waits for B
T2 locks data B → waits for A
→ Deadlock

**Conditions for Deadlock (Coffman Conditions)**

1. **Mutual Exclusion:** Resources cannot be shared.
2. **Hold and Wait:** A transaction holds one resource while waiting for another.
3. **No Preemption:** Resources cannot be forcibly taken away.
4. **Circular Wait:** Circular chain of waiting transactions exists.

**Deadlock Handling Methods**

**1. Deadlock Prevention**

Prevent occurrence by breaking one of the four conditions.

| Method | Description |
|---|---|
| **Wait-Die** | Older transaction waits; younger aborts. |
| **Wound-Wait** | Older transaction preempts (wounds) younger one. |
| **Timeouts** | Abort transaction if waiting too long. |

**2. Deadlock Detection**

- Maintain a **wait-for graph** where each node is a transaction.
- An edge Ti → Tj exists if Ti is waiting for a resource held by Tj.
- If the graph contains a cycle → **deadlock detected**.

**3. Deadlock Recovery**

- **Abort** one or more transactions to break cycle.
- **Choose victim** based on: cost, progress, and priority.
- **Rollback** victim transaction to previous checkpoint.

**Example of Wait-For Graph**

T1 → T2
T2 → T3
T3 → T1  ← Cycle indicates Deadlock

# UNIT 3– Database Querying & Concurrency Control

**Database Querying & Concurrency Control:** Relational Algebra, Set Operations, Relational Calculus, Steps In Query Processing, Algorithms For Selection, Sorting And Join Operations, Understanding Cost Issues In Queries, Query Optimization, Transformation Of Relational Expressions, Query Evaluation Plans.

**Concurrency Control:** Locks Based Protocols, Time Stamp Based Protocols, Validation Based Protocol, Multiple Granularity, Multi-version Schemes.

# DATABASE QUERYING

## 1. Introduction to Database Querying

Database querying is the process of **retrieving required data from a database** using formal query languages.
A query specifies:

- What data is required
- From where it is required
- Under what conditions

The DBMS converts user queries into **internal query representations** and executes them efficiently.

## 2. Relational Algebra

Relational Algebra is a **procedural query language** that uses a collection of **operations on relations** to retrieve data from a relational database.

It specifies **how the query is to be executed**.

### 2.2 Features of Relational Algebra

1. Procedural language
2. Based on set theory
3. Closed under operations
4. Input and output are relations
5. Used internally by DBMS for query optimization

### 2.3 Basic Relational Algebra Operations

#### 1. Selection ( σ )

Selection operation selects **rows (tuples)** from a relation that satisfy a given condition.

### Syntax

$$\sigma_{condition}(Relation)$$

### Example

EMP(EmpID, Name, Salary, Dept)

Select employees with salary > 40000:

$$\sigma_{Salary>40000}(EMP)$$

**Explanation**

- Does not change number of columns
- Filters rows based on condition

**2. Projection ( π )**

Projection operation selects **specific columns (attributes)** from a relation and removes duplicate tuples.

### Syntax

$$\pi_{Attribute1,Attribute2}(Relation)$$

### Example

$$\pi_{Name,Dept}(EMP)$$

**Explanation**

- Reduces number of columns
- Duplicate rows are eliminated

**3. Union ( ∪ )**

Union combines tuples of two relations into one relation.

**Conditions for Union**

1. Same number of attributes
2. Same domain

3. Same order of attributes

**R ∪ S**

**4. Set Difference ( − )**

Returns tuples that exist in **one relation but not in another**.

**R − S**

**5. Cartesian Product ( × )**

Combines each tuple of one relation with every tuple of another relation.

**R × S**

**Explanation**

- Produces large intermediate results
- Used before join operations

**6. Rename ( ρ )**

Rename operation changes the name of a relation or its attributes.

$$\rho_{NewName}(Relation)$$

**2.4 Derived Relational Algebra Operations**

**7. Join ( ⋈ )**

Join operation combines related tuples from two relations based on a condition.

**Types of Join**

1. **Theta Join** – uses comparison operators
2. **Equi Join** – uses equality condition
3. **Natural Join** – automatically joins common attributes

**Example**

STUDENT ⋈ ENROLLMENT

**8. Division ( ÷ )**

Division operation is used to answer **"for all" type queries**.

**Example**

Find students enrolled in all courses:

**ENROLL(Student, Course) ÷ COURSE(Course)**

## 3. Set Operations

Set operations treat relations as mathematical sets.

**Types of Set Operations**

1. Union
2. Intersection
3. Difference

**Properties**

- Relations must be union-compatible
- Duplicate tuples are removed automatically

## 4. Relational Calculus

Relational Calculus is a **non-procedural query language** that specifies **what data to retrieve**, not how to retrieve it.

### 4.1 Tuple Relational Calculus (TRC)

**Form**

$$\{t \mid P(t)\}$$

**Example**

$$\{t \mid t \in STUDENT \land t.Age > 20\}$$

---

### 4.2 Domain Relational Calculus (DRC)

**Form**

$$\{< x_1, x_2 > \mid P(x_1, x_2)\}$$

## 5. Steps in Query Processing

Query processing is the process by which a DBMS converts a user query into an efficient execution plan.

**Step 1: Parsing and Translation**

- Syntax checking
- Semantic checking
- SQL → Relational Algebra

**Step 2: Query Optimization**

- Generate alternative execution plans
- Choose plan with minimum cost

**Step 3: Query Evaluation**

- Execute optimized plan
- Return result to user

## 6. Algorithms for Selection Operation

**1. Linear Search Algorithm**

- Scan all blocks
- Cost = **b block accesses**
- Used when no index exists

**2. Binary Search Algorithm**

- File must be sorted
- Cost = $\log_2 b$

**3. Index-Based Selection**

- Uses primary or secondary index
- Fastest selection method

## 7. Algorithms for Sorting

**External Sorting (Merge Sort)**

**Working**

1. Create sorted runs in memory
2. Merge sorted runs

**Cost Formula**

$$2b \log_M b$$

## 8. Algorithms for Join Operations

### 1. Nested Loop Join

- For each tuple of outer relation, scan inner relation

$$Cost = b_r + n_r \times b_s$$

### 2. Block Nested Loop Join

- Uses blocks instead of tuples

$$Cost = b_r + \frac{b_r}{M} \times b_s$$

### 3. Sort-Merge Join

- Sort both relations
- Merge matching tuples

### 4. Hash Join

- Partition relations using hash function
- Join matching partitions

## 9. Understanding Cost Issues in Queries

### Major Cost Factors

1. Disk I/O (highest cost)
2. CPU processing time
3. Memory access
4. Communication overhead

**Cost Measurement**

- Number of disk seeks
- Number of block transfers

## 10. Query Optimization

Query optimization is the process of selecting the **most efficient execution plan** for a query.

**Types of Query Optimization**

**1. Heuristic Optimization**

- Apply selection early
- Apply projection early
- Reduce intermediate relations

**2. Cost-Based Optimization**

- Uses statistical information
- Chooses minimum cost plan

## 11. Transformation of Relational Expressions

Transformation rewrites relational algebra expressions into **equivalent but more efficient forms**.

**Important Rules**

1. **Selection Pushdown**

$$\sigma_{c1 \wedge c2}(R) = \sigma_{c1}(\sigma_{c2}(R))$$

2. **Join Commutativity**

$$R \bowtie S = S \bowtie R$$

3. **Join Associativity**

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

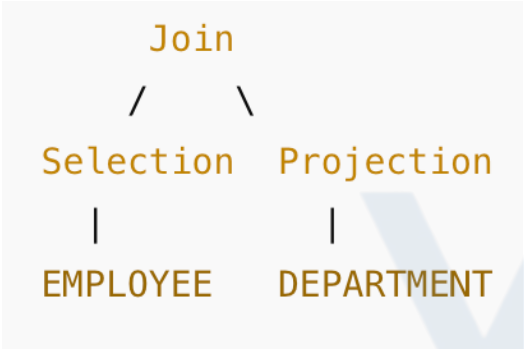
## 12. Query Evaluation Plans

A query evaluation plan describes **how a query will be executed** using specific algorithms.

### Types of Plans

1. **Logical Query Plan**
    - Based on relational algebra
2. **Physical Query Plan**
    - Specifies access paths and algorithms

### Query Tree Example



# CONCURRENCY CONTROL

## 1. Transaction Isolation and Concurrency Control

### 1.1 Transaction Isolation Property

Isolation is one of the **ACID properties** of a transaction.

**Isolation means:**
Each transaction should execute **as if it were the only transaction in the system**, even though many transactions may run concurrently.

Concurrency control mechanisms are responsible for enforcing **isolation**.

### 1.2 Schedules

A **schedule** is an ordering of read/write operations of multiple transactions.

**Types of Schedules**

1. **Serial Schedule**
    - Transactions execute one after another
    - No interleaving
    - Always correct but inefficient
2. **Concurrent (Non-Serial) Schedule**
    - Operations are interleaved
    - Improves performance
    - Needs concurrency control

### 1.3 Serializability

**Conflict Serializability**

A schedule is conflict serializable if it can be transformed into a serial schedule by swapping **non-conflicting operations**.

**Conflicting operations:**

- Read–Write
- Write–Read
- Write–Write

✔ Two-phase locking ensures conflict serializability

**View Serializability**

A schedule is view serializable if it produces the same **read-from relationships** and final writes as a serial schedule.

✔ More general than conflict serializability
✖ Hard to test in practice

## 2. Lock-Based Protocols

### 2.1 Lock Granularity Trade-Off

**Coarse Granularity Locks**

- Lock entire table or database
- Fewer locks
- Less concurrency

**Fine Granularity Locks**

- Lock rows or records
- More locks
- Higher concurrency

Multiple granularity locking balances this trade-off

### 2.2 Lock Conversion (Upgrading and Downgrading)

**Lock Upgrade**

- S-lock → X-lock
- Requires no other S-locks

**Lock Downgrade**

- X-lock → S-lock
- Allowed safely

✔ Lock upgrades may cause deadlock

### 2.3 Deadlock

**Necessary Conditions for Deadlock**

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait

If any condition is removed → deadlock cannot occur

**Deadlock Detection (Wait-for Graph)**

- Nodes represent transactions
- Edge Ti → Tj means Ti waits for Tj
- Cycle ⇒ deadlock exists

**Deadlock Recovery**

- Abort one transaction
- Release locks
- Restart transaction

### 2.4 Starvation

A transaction waits indefinitely because other transactions keep getting priority.

Occurs in:

- Timestamp protocols
- Deadlock prevention schemes

## 3. Timestamp-Based Protocols

### 3.1 Why Timestamps Work

- Enforces a **global order**
- Transactions behave as if executed in timestamp order
- Older transactions have higher priority

This eliminates **circular wait**, hence **no deadlock**.

### 3.2 Thomas Write Rule (Important Theory)

**Problem in Basic Timestamp Protocol**

Some writes are unnecessary and cause aborts.

**Thomas Write Rule**

If:

**TS(T) < W_TS(X)**

Then:

- Ignore the write instead of aborting

✔ Reduces aborts
✔ Still preserves serializability

### 3.3 Timestamp Protocol vs Locking

| Aspect | Timestamp | Locking |
|---|---|---|
| Deadlock | Impossible | Possible |
| Waiting | No | Yes |
| Abort | Frequent | Less |
| Overhead | Timestamp mgmt | Lock mgmt |

## 4. Validation-Based Protocol

### 4.1 Optimistic Philosophy

Validation protocol is based on the assumption:

"Most transactions do not conflict."

Instead of preventing conflicts, it **detects conflicts at commit time**.

### 4.2 Read and Write Sets

Each transaction maintains:

- **Read Set (RS)**: data items read
- **Write Set (WS)**: data items written

Validation ensures:

$$WS(T_j) \cap RS(T_i) = \emptyset$$

### 4.3 When Validation Protocol Works Best

✔ Read-heavy workloads
✔ Short transactions
✔ Low contention environments

✖ Not suitable for banking or reservation systems

## 5. Multiple Granularity Locking

### 5.1 Lock Hierarchy Tree

Database
  |
 Table
  |
 Page
  |
 Record

Locks must be acquired **top-down** and released **bottom-up**.

### 5.2 Intention Lock Compatibility

**Intention locks prevent:**

- A transaction from locking a higher-level object
- While another transaction holds a lower-level conflicting lock

**Compatibility Matrix (Expanded)**

| Held \ Requested | IS | IX | S | SIX | X |
|---|---|---|---|---|---|
| IS | ✔ | ✔ | ✔ | ✔ | ✖ |
| IX | ✔ | ✔ | ✖ | ✖ | ✖ |
| S | ✔ | ✖ | ✔ | ✖ | ✖ |
| SIX | ✔ | ✖ | ✖ | ✖ | ✖ |
| X | ✖ | ✖ | ✖ | ✖ | ✖ |

## 6. Multiversion Concurrency Control

### 6.1 Why MVCC Is Needed

Traditional locking:

- Blocks readers during writes

MVCC:

- Allows readers and writers to proceed **without blocking**

### 6.2 Version Visibility Rules

A transaction sees:

- The latest committed version

- With timestamp ≤ transaction's timestamp

### 6.3 Snapshot Isolation

**Each transaction sees a snapshot of the database**

- Prevents dirty and non-repeatable reads
- Used in PostgreSQL, Oracle

⚠ Does NOT guarantee full serializability (write skew possible)

### 6.4 Garbage Collection in MVCC

Old versions:

- Consume storage
- Must be removed when no transaction needs them

# UNIT 4– Recovery System & Security

**Recovery System & Security:** Failure Classifications, Recovery & Atomicity, Log Base Recovery, Recovery with Concurrent Transactions, Shadow Paging, Failure with Loss of Non-Volatile Storage, Recovery From Catastrophic Failure, Introduction to Security & Authorization, Introduction to emerging Databases-OODBMS, ORDBMS, Distributed database, Multimedia database, Special database-limitations of conventional databases, advantages of emerging databases.

## 1. Introduction to Recovery System

A **Recovery System** in a Database Management System (DBMS) is responsible for restoring the database to a **correct and consistent state** after a failure occurs. Failures may occur due to hardware crashes, software errors, power failures, or system crashes.

The main goal of recovery is to ensure:

- **Atomicity** – Transactions are either fully completed or fully undone.
- **Durability** – Once a transaction is committed, its changes are permanent.

Without a recovery system, data loss and inconsistency can occur, which may damage the reliability of the database.

## 2. Failure Classifications

Failures in a Database Management System (DBMS) are classified based on their **cause**, **scope**, and **impact on data consistency**. Understanding failure classification helps the recovery system decide the correct recovery action.

### 2.1 Transaction Failure

A **transaction failure** occurs when a transaction is unable to complete its execution successfully and must be terminated before completion.

**Causes of Transaction Failure**

1. **Logical Errors**
   These errors occur due to incorrect logic in the transaction.
   - Invalid input values
   - Violation of integrity constraints
   - Arithmetic errors such as division by zero
2. **System Errors**
   These errors occur due to system conditions.
   - Deadlock between transactions
   - Insufficient memory or CPU resources
   - Forced termination by the DBMS

**Effect of Transaction Failure**

- Only the **failed transaction** is rolled back.
- All operations performed by that transaction are **undone**.
- Other active or committed transactions are **not affected**.

### 2.2 System Failure

A **system failure** occurs when the entire database system crashes due to hardware or software problems.

**Causes of System Failure**

- Power failure
- Operating system crash
- CPU or main memory failure

**Effect of System Failure**

- All data stored in **main memory (RAM)** is lost.
- Data stored on **disk (secondary storage)** remains intact.
- Recovery is required using logs after system restart.

### 2.3 Media Failure

A **media failure** occurs when the storage media itself is damaged or corrupted.

**Causes of Media Failure**

- Disk head crash
- Disk corruption
- Physical damage due to fire, water, or hardware defects

**Effect of Media Failure**

- Loss of data stored on the disk.
- Normal recovery is not possible using logs alone.
- **Backup restoration** is required.

### 2.4 Communication Failure

Communication failure mainly occurs in **distributed database systems** where data is stored at multiple locations.

**Causes of Communication Failure**

- Network failure
- Message loss or delay
- Network partitioning

**Effect of Communication Failure**

- Incomplete or partial execution of distributed transactions.
- Data inconsistency across sites may occur.

## 3. Recovery and Atomicity

**Atomicity**

Atomicity is one of the **ACID properties** of transactions. It ensures that:

- A transaction is treated as a **single, indivisible unit**.
- Either **all operations are performed**, or **none are performed**.
- If a transaction fails, all its changes are **completely undone**.

**Need for Atomicity**

Without atomicity:

- Partial updates may remain in the database.
- Database consistency rules may be violated.
- Incorrect or unreliable data may be stored.

Atomicity ensures that the database always moves from one **consistent state** to another.

**Recovery Mechanism for Atomicity**

To maintain atomicity, the recovery system uses:

- **Log files** to record changes
- **Checkpoints** to reduce recovery time
- **Undo operations** for failed transactions
- **Redo operations** for committed transactions

These mechanisms ensure that incomplete transactions do not affect database correctness.

## 4. Log-Based Recovery

Log-based recovery is the most commonly used recovery technique in DBMS.

### 4.1 Transaction Log

A **transaction log** is a sequential file that records every modification made to the database.

**Each log record contains:**

- Transaction ID
- Name of the data item
- Old value (before update)
- New value (after update)

The log is stored on stable storage so that it survives system failures.

### 4.2 Write-Ahead Logging (WAL)

Write-Ahead Logging is a rule that ensures reliable recovery.

**Principle of WAL**

- Before any database change is written to disk, the corresponding **log record must be written first**.
- This guarantees that recovery information is always available.

**Benefit**

- Ensures correctness during system crash recovery.
- Supports both undo and redo operations.

### 4.3 Undo and Redo Operations

**Undo Operation**

- Used for **aborted or incomplete transactions**.
- Restores the database using **old values** from the log.

**Redo Operation**

- Used for **committed transactions**.
- Reapplies changes using **new values** from the log.

## 5. Recovery with Concurrent Transactions

When multiple transactions execute simultaneously, recovery becomes more complex.

**Problems in Concurrent Execution**

- Interleaved read/write operations

- Partial commits during system failure
- Difficulty in determining transaction status

**Solution**

- Each transaction is assigned a **unique transaction ID**.
- Logs record the exact order of operations.
- During recovery:
    - **Undo** all uncommitted transactions.
    - **Redo** all committed transactions.

This ensures **consistency** and **isolation** in concurrent environments.

## 6. Shadow Paging

Shadow paging is a recovery technique that **does not use log files**. Instead, it maintains two page tables.

**Working of Shadow Paging**

- A **shadow page table** points to original database pages.
- A **current page table** is used for updates.
- Updated pages are written to new disk locations.

**On Commit**

- Current page table becomes the new shadow page table.

**On Failure**

- The system uses the old shadow page table to restore the database.

**Advantages of Shadow Paging**

- No log file required
- Fast recovery
- Simple undo operation

**Disadvantages of Shadow Paging**

- Difficult to support concurrent transactions
- Inefficient for large databases

- Increased disk usage

## 7. Failure with Loss of Non-Volatile Storage

Non-volatile storage such as disks is assumed to be permanent, but failures can still occur.

### Causes

- Disk crash
- Fire, flood, or natural disasters
- Severe hardware damage

### Recovery Method

- Restore the database from the **most recent backup**.
- Apply log files if available to recover recent updates.

## 8. Recovery from Catastrophic Failure

A **catastrophic failure** destroys both:

- Main memory
- Secondary storage

### Recovery Steps

1. Restore the **last full backup**.
2. Apply **archived logs** to redo committed transactions.
3. Restart database services.

### Importance of Backups

- Regular backups reduce data loss.
- Off-site backups protect against natural disasters.
- Essential for disaster recovery planning.

## 9. Introduction to Security

### Database Security

Database security protects data from:

- Unauthorized access
- Data theft
- Accidental or intentional modification

**Security Goals**

1. **Confidentiality** – Data is accessible only to authorized users.
2. **Integrity** – Data remains accurate and consistent.
3. **Availability** – Data is accessible when needed.

## 10. Authorization

Authorization defines **who can access the database** and **what actions they can perform**.

**Types of Privileges**

- **SELECT** – Read data
- **INSERT** – Add new data
- **UPDATE** – Modify data
- **DELETE** – Remove data

**Authorization Mechanism**

- User accounts
- Roles
- Access control lists (ACLs)

## 11. Introduction to Emerging Databases

Traditional database systems such as **Relational Database Management Systems (RDBMS)** were designed mainly to store and process **simple, structured data** like numbers and text. These databases work very well for business applications such as banking, payroll, and inventory systems. However, with the growth of modern applications, traditional databases started facing many limitations.

Modern applications require handling **complex data types** such as images, audio, video, geographic data, and scientific data. They also require **high scalability**, **distributed access**, and **object-oriented support**, which conventional databases cannot efficiently provide.

To overcome these limitations, **Emerging Databases** were developed. Emerging databases are advanced database systems designed to manage complex, large-scale, and multimedia data efficiently.

**Key reasons for emerging databases:**

- Increase in multimedia and unstructured data
- Growth of distributed and internet-based applications
- Need for object-oriented data modeling
- Requirement of high performance and scalability

Examples of emerging databases include:

- Object-Oriented Databases (OODBMS)
- Object-Relational Databases (ORDBMS)
- Distributed Databases
- Multimedia Databases

## 12. Object-Oriented Database Management System (OODBMS)

An **Object-Oriented Database Management System (OODBMS)** is a database system that stores data in the form of **objects**, similar to the way data is represented in object-oriented programming languages such as Java and C++.

In OODBMS, data and its related operations are stored together as objects, unlike relational databases where data and operations are separate.

**Features of OODBMS**

**1. Encapsulation**

Encapsulation means combining **data and methods** into a single unit called an object.
The internal details of an object are hidden, and data can be accessed only through defined methods.

**Benefit:**
Improves data security and reduces complexity.

**2. Inheritance**

Inheritance allows a new class (child class) to acquire properties and methods of an existing class (parent class).

**Benefit:**
Promotes code reusability and reduces redundancy.

**3. Polymorphism**

Polymorphism allows the same method name to perform different operations based on the object that invokes it.

**Benefit:**
Provides flexibility and easier system expansion.

**Advantages of OODBMS**

**1. Handles Complex Data Easily**

OODBMS can store complex data types such as images, audio, video, and CAD data without breaking them into tables.

**2. Supports Multimedia Data**

It is well-suited for applications that require multimedia data storage, such as medical imaging and digital libraries.

**3. Better Representation of Real-World Objects**

Objects closely represent real-world entities, making database design more natural and intuitive.

## 13. Object-Relational Database Management System (ORDBMS)

An **Object-Relational Database Management System (ORDBMS)** is an advanced database system that combines the features of **relational databases** with **object-oriented concepts**.

It extends the traditional relational model by adding support for complex data types and object-oriented features while still using tables and SQL.

**Features of ORDBMS**

**1. User-Defined Data Types**

ORDBMS allows users to define their own data types, making it possible to store complex data such as images or structures.

**2. Inheritance**

Tables can inherit attributes from other tables, similar to class inheritance in object-oriented programming.

**3. Methods in Tables**

ORDBMS allows functions or methods to be associated with tables and data types.

**Advantages of ORDBMS**

**1. Better Than RDBMS for Complex Applications**

ORDBMS handles complex data and relationships more efficiently than traditional RDBMS.

**2. Compatible with SQL**

Since ORDBMS supports SQL, it is easier to migrate from RDBMS without rewriting entire applications.

**3. Combines Best of Both Worlds**

It offers relational simplicity along with object-oriented power.

## 14. Distributed Database

A **Distributed Database** is a collection of databases that are **physically stored at different locations** but logically appear as a **single database** to users.

The databases are connected through a network and managed by a Distributed Database Management System (DDBMS).

**Advantages of Distributed Databases**

**1. High Availability**

If one site fails, other sites can continue working, improving system reliability.

**2. Scalability**

New sites can be added easily without affecting existing systems.

**3. Fault Tolerance**

Failure of one node does not stop the entire database system.

**Challenges of Distributed Databases**

**1. Data Consistency**

Maintaining consistent data across multiple locations is complex and challenging.

**2. Network Dependency**

System performance depends heavily on network reliability and speed.

## 15. Multimedia Database

**Definition**

A **Multimedia Database** is a database system designed to store and manage **multimedia data** such as:

- Images
- Audio
- Video
- Graphics

These databases require special techniques for storage, indexing, and retrieval due to large data size.

**Applications of Multimedia Databases**

**1. Medical Systems**

Used for storing X-rays, MRI scans, and patient videos.

**2. Entertainment**

Used in movie streaming platforms, music applications, and gaming systems.

**3. Digital Libraries**

Used for storing digital books, videos, photographs, and archives.

## 16. Limitations of Conventional Databases

Traditional databases suffer from several limitations in modern environments:

**1. Poor Handling of Complex Data**

They are not designed to manage multimedia or unstructured data efficiently.

**2. No Multimedia Support**

Storing images, audio, and video is inefficient and slow.

**3. Limited Scalability**

Conventional databases struggle with large-scale distributed systems.

**4. Weak Performance for Unstructured Data**

Text, video, and graphical data cannot be efficiently indexed or queried.

## 17. Advantages of Emerging Databases

Emerging databases provide powerful solutions to modern data challenges:

**1. Support for Complex and Multimedia Data**

They can efficiently store and manage images, videos, and complex objects.

**2. Better Performance**

Optimized storage and retrieval techniques improve speed.

**3. Scalability**

They can easily handle large and distributed datasets.

**4. Flexibility**

Support multiple data models and application requirements.

**5. Support for Modern Applications**

Ideal for applications involving **Artificial Intelligence, Big Data, cloud computing, and multimedia systems**.

# UNIT 5– SQL and PL/SQL

**SQL and PL/SQL: Introduction to SQL:** Characteristics of SQL, Advantages of SQL, SQL data types and literals, Types of SQL commands, SQL operators, Tables, views and indexes, Constraints, Group By and Having Clause, Order By Clause, Queries and sub queries, Functions, PL/SQL basics, blocks, architecture, variables, constants, attributes, character set, PL/SQL control structure, data types, conditional and sequential control statements, cursors, exceptions, triggers, functions, procedures and packages.

## 1. Introduction to SQL

SQL (Structured Query Language) is a standard database language used to interact with relational databases. It allows users to define database structure, manipulate data, retrieve information, control access, and manage transactions. SQL hides internal complexity and provides a simple interface to users.
SQL is widely used because it is easy, powerful, and portable across different database systems.

SQL is a **declarative (non-procedural)** language, meaning users specify **what data is needed**, not **how it should be retrieved**. It works with **relational databases** where data is stored in tables consisting of rows and columns. SQL acts as a bridge between the user/ application and the database engine.

SQL is supported by almost all major DBMS such as MySQL, Oracle, PostgreSQL, SQL Server, etc., making it highly portable.

## 2. Characteristics of SQL

- SQL is a non-procedural language, users specify what data is required
- SQL commands are simple and readable
- SQL is case-insensitive
- SQL works on tables (relations)
- SQL supports DDL, DML, DCL, TCL, and DQL
- SQL allows multi-user access with security
- **Non-procedural:** Reduces complexity for users.
- **Readable syntax:** English-like commands improve usability.
- **Case-insensitive:** SELECT and select behave the same.
- **Table-based:** Data is always processed in relational form.
- **Command classification:** Makes database management organized.
- **Multi-user & secure:** Supports permissions, roles, and concurrency control.

## 3. Advantages of SQL

- Faster data retrieval
- Easy to write and maintain
- Handles large amount of data
- Supports data integrity and security
- Used in almost all DBMS products

SQL optimizes queries internally, ensuring **high performance**. Built-in constraints, keys, and permissions maintain **data accuracy and safety**. Because SQL is standardized, learning it once allows usage across multiple database platforms.

## 4. SQL Data Types and Literals

### 4.1 SQL Data Types

**Numeric Data Types**

INT
FLOAT
DECIMAL
Example:

salary INT;

**Character Data Types**

CHAR
VARCHAR
TEXT
Example:

name VARCHAR(50);

**Date and Time Data Types**

DATE
TIME
TIMESTAMP
Example:

dob DATE;

Data types define the **kind of data** a column can store.

- **INT:** Stores whole numbers.
- **FLOAT/DECIMAL:** Stores fractional values.
- **CHAR:** Fixed-length strings.
- **VARCHAR:** Variable-length strings.

- **DATE/TIME:** Used for temporal data.

Correct data type selection improves **storage efficiency** and **performance**.

### 4.2 SQL Literals

Literals are fixed constant values.
Examples:

'Database'
1000
'2025-01-01'

Literals represent **actual values** written directly in SQL queries.

- String literals are enclosed in **single quotes**
- Numeric literals are written directly
- Date literals follow standard date format

## 5. Types of SQL Commands

### 5.1 DDL (Data Definition Language)

**Used to define database structure.**

**CREATE**

```
CREATE TABLE student (
  roll INT PRIMARY KEY,
  name VARCHAR(30),
  marks INT
);
```

**ALTER**

```
ALTER TABLE student ADD age INT;
```

**DROP**

```
DROP TABLE student;
```

DDL commands define and modify database schema. Changes made by DDL are **permanent** and auto-committed.

**Output:**

- Table created
- Table altered
- Table dropped

### 5.2 DML (Data Manipulation Language)

**INSERT**

```
INSERT INTO student VALUES (1, 'Amit', 85);
```

**UPDATE**

```
UPDATE student SET marks = 90 WHERE roll = 1;
```

**DELETE**

```
DELETE FROM student WHERE roll = 1;
```

DML commands modify data inside tables. These operations can be **rolled back** if not committed.

**Output:**

- 1 row inserted
- 1 row updated
- 1 row deleted

### 5.3 DQL (Data Query Language)

```
SELECT * FROM student;
```

DQL retrieves data from database tables without modifying it.

**Output (example):**

| roll | name | marks |
|---|---|---|
| 1 | Amit | 90 |

### 5.4 DCL (Data Control Language)

```
GRANT SELECT ON student TO user1;
REVOKE SELECT ON student FROM user1;
```

DCL controls **user access and permissions**, ensuring database security.

### 5.5 TCL (Transaction Control Language)

```
COMMIT;
ROLLBACK;
SAVEPOINT sp1;
```

TCL manages transactions to maintain **data consistency** and **atomicity**.

## 6. SQL Operators

### Arithmetic Operators

SELECT marks + 5 FROM student;
**Output:** Marks increased by 5.

### Relational Operators

SELECT * FROM student WHERE marks > 60;
**Output:** Rows with marks greater than 60.

### Logical Operators

SELECT * FROM student WHERE marks > 60 AND age > 18;
**Output:** Rows satisfying both conditions.

### Special Operators

SELECT * FROM student WHERE name LIKE 'A%';
**Output:** Names starting with A.

## 7. Tables, Views, and Indexes

### 7.1 Tables

A table stores data in rows and columns.

Tables are the fundamental storage units in relational databases.

### 7.2 Views

CREATE VIEW topper AS
SELECT name, marks FROM student WHERE marks > 80;

Views are **virtual tables** that do not store data physically.

**Output:**
Shows only students scoring above 80.

### 7.3 Indexes

CREATE INDEX idx_marks ON student(marks);

Indexes improve **search speed** by minimizing disk access.

## 8. Constraints

```
CREATE TABLE employee (
  emp_id INT PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  salary INT CHECK (salary > 0),
  dept_id INT,
  UNIQUE(name)
);
```

**Types:**

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Constraints enforce **data integrity rules** and prevent invalid data entry.

## 9. GROUP BY and HAVING Clause

```
SELECT dept_id, COUNT(*)
FROM employee
GROUP BY dept_id
HAVING COUNT(*) > 3;
```

**GROUP BY groups rows**
**HAVING filters grouped data**

- GROUP BY forms groups.
- HAVING applies conditions on aggregated data.

**Output:** Departments having more than 3 employees.

## 10. ORDER BY Clause

```
SELECT * FROM student ORDER BY marks DESC;
```

Sorts result set in ascending or descending order.

**Output:** Students sorted by marks (highest first).

## 11. Queries and Subqueries

```
SELECT name
FROM employee
WHERE salary > (SELECT AVG(salary) FROM employee);
```

Subqueries are **nested queries** used for comparison and filtering.

**Output:** Employees earning above average salary.

## 12. SQL Functions

**Aggregate Functions**

SELECT AVG(marks) FROM student;
**Output:** Average marks.

**Scalar Functions**

SELECT UPPER(name) FROM student;
**Output:** Names in uppercase.

## 13. Introduction to PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is a procedural extension of SQL developed by Oracle. It allows programmers to write structured programs that include **procedural logic** such as conditions, loops, variables, cursors, and exception handling directly inside the database.

PL/SQL enhances the capabilities of SQL by combining SQL statements with programming constructs. This makes database applications more **powerful, efficient, secure, and modular**. Since PL/SQL code executes on the database server, it reduces network traffic and improves overall performance.

## 14. PL/SQL Block Structure

PL/SQL programs are written in the form of blocks. A block is the basic unit of a PL/SQL program.

**Example:**

```
DECLARE
  v_marks NUMBER;
BEGIN
  v_marks := 80;
  DBMS_OUTPUT.PUT_LINE(v_marks);
END;
```

**Structure of PL/SQL Block:**

1. **DECLARE** – Used for declaring variables, constants, cursors, and exceptions.
2. **BEGIN** – Contains executable SQL and PL/SQL statements.
3. **END** – Marks the end of the PL/SQL block.

**Output:**

80

## 15. Variables and Constants in PL/SQL

Variables are used to store data temporarily during program execution, while constants store fixed values that cannot be changed once assigned.

**Example:**

```
DECLARE
  pi CONSTANT NUMBER := 3.14;
  radius NUMBER := 5;
BEGIN
  DBMS_OUTPUT.PUT_LINE(pi * radius * radius);
END;
```
Here, pi is declared as a constant and radius is a variable.

**Output:**

78.5

## 16. PL/SQL Attributes

PL/SQL provides special attributes to maintain consistency between variables and database table columns.

**Example:**

```
v_salary employee.salary%TYPE;
emp_row employee%ROWTYPE;
```

- %TYPE assigns the data type of a table column to a variable.
- %ROWTYPE allows a variable to store an entire row of a table.

These attributes ensure **data type consistency** and reduce errors when table structures change.

## 17. PL/SQL Control Structures

Control structures are used to control the flow of execution in PL/SQL programs.

**IF Statement Example:**

```
IF marks >= 40 THEN
  result := 'PASS';
ELSE
  result := 'FAIL';
END IF;
```
This conditional statement checks whether the marks are greater than or equal to 40 and assigns the result accordingly.

**LOOP Example:**

```
FOR i IN 1..5 LOOP
  DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
```

**Output:**

```
1
2
3
4
5
```

## 18. Cursors

A cursor is a pointer that allows row-by-row processing of query results in PL/SQL. Cursors are especially useful when a query returns multiple rows.

**Example:**

```
DECLARE
  CURSOR c1 IS SELECT name FROM student;
  v_name student.name%TYPE;
BEGIN
  OPEN c1;
  FETCH c1 INTO v_name;
  CLOSE c1;
END;
```

**Explanation:**

- The cursor c1 stores the result of the SELECT query.
- Rows are fetched one at a time using FETCH.

Cursors enable **fine-grained control** over query result processing.

## 19. Exceptions

Exceptions are runtime errors that occur during the execution of a PL/SQL program. PL/SQL provides an exception handling mechanism to handle such errors gracefully.

**Example:**

```
BEGIN
  x := 10/0;
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Error');
END;
```

**Output:**

```
Error
```

Here, the division by zero error is handled using a predefined exception.

## 20. Triggers

A trigger is a special PL/SQL block that automatically executes when a specific database event occurs, such as INSERT, UPDATE, or DELETE.

**Example:**

```
CREATE OR REPLACE TRIGGER trg_before_insert
BEFORE INSERT ON student
FOR EACH ROW
BEGIN
  DBMS_OUTPUT.PUT_LINE('Insert Trigger Fired');
END;
```

Triggers are used to enforce business rules, maintain data integrity, and perform automatic actions.

## 21. Functions

A function is a named PL/SQL block that performs a specific task and **returns a value**.

**Example:**

```
CREATE FUNCTION square(n NUMBER)
RETURN NUMBER IS
BEGIN
  RETURN n*n;
END;
```

**Output:**

Returns the square of the given number.

Functions are commonly used in SQL queries and expressions.

## 22. Procedures

A procedure is a stored PL/SQL block that performs a specific task but **does not return a value**.

**Example:**

```
CREATE PROCEDURE show_msg IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello PL/SQL');
END;
```

**Output:**

Hello PL/SQL
Procedures are used to perform repetitive tasks and improve code reusability.

## 23. Packages

A package is a collection of related procedures, functions, variables, and cursors stored together as a single unit.

**Example:**

```
CREATE PACKAGE math_pkg IS
  FUNCTION add(a NUMBER, b NUMBER) RETURN NUMBER;
END;
```

Packages improve **modularity, performance, security, and maintainability** of PL/SQL programs.