# <u>UNIT-I</u>

## 1.1 Database-System Applications

Explains where database systems are used in real life:

- Banking (accounts, loans)
- Universities (student records, courses)
- Airlines (reservations)
- E-commerce, human resources, telecommunications, social media, scientific data, etc. Emphasizes that modern applications need to manage large, persistent, shared, and reliable data.

## 1.2 Purpose of Database Systems

Contrasts the traditional **file-processing approach** (data redundancy, inconsistency, difficult concurrency, weak security) with **database systems**: Advantages:

- Data abstraction and independence
- Reduced redundancy
- Consistency and integrity constraints
- Efficient querying
- Concurrent access and crash recovery
- Security and authorization

## 1.3 View of Data

Introduces levels of abstraction:

1. **Physical level** (how data is stored on disk)
2. **Logical level** (schema: tables, columns, relationships)
3. **View level** (external schemas or views for different users)

Also covers **data models**:

- Relational model
- Entity-Relationship (E-R) model
- Object-based, semi-structured (XML), NoSQL, etc.

## 1.4 Database Languages

- **Data-Definition Language (DDL)** → defines schema (CREATE TABLE, etc.)
- **Data-Manipulation Language (DML)** → querying and updates
    - Procedural (e.g., relational algebra)
    - Declarative (e.g., SQL: SELECT, INSERT, UPDATE, DELETE)
- Storage definition language (SDL), view definition language (VDL)

- Modern systems combine DDL and DML in SQL

(Exam-Ready Summary – Silberschatz Style)

| Language Type | Full Name | Purpose | Examples (SQL) |
|---|---|---|---|
| **DDL** | Data Definition Language | Define, alter, or drop schema (structure) of database objects | CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX |
| **DML** | Data Manipulation Language | Query and modify data (insert, update, delete, retrieve) | SELECT, INSERT, UPDATE, DELETE |
| **DCL** | Data Control Language | Security: grant/revoke permissions | GRANT, REVOKE |
| **TCL** | Transaction Control Language | Manage transactions | COMMIT, ROLLBACK, SAVEPOINT |
| **SDL / VDL** | Storage / View Definition Language | Rarely used separately in modern DBMS (merged into DDL) | (Historical – now part of SQL DDL) |

SQL

```
-- DDL
CREATE TABLE Student (
    id       INT PRIMARY KEY,
    name     VARCHAR(50) NOT NULL,
    dept     VARCHAR(10)
);

-- DML
INSERT INTO Student VALUES (101, 'Ali', 'CS');
UPDATE Student SET dept = 'EE' WHERE id = 101;
DELETE FROM Student WHERE id = 101;
SELECT * FROM Student WHERE dept = 'CS';

-- View (part of DDL)
CREATE VIEW CS_Students AS
SELECT id, name FROM Student WHERE dept = 'CS';
```

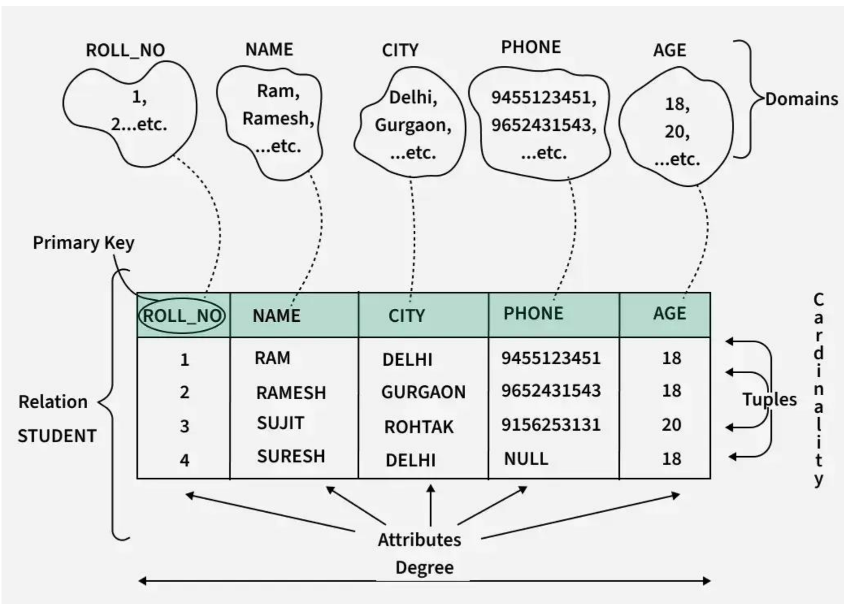
```sql
-- DCL
GRANT SELECT ON Student TO public;
REVOKE INSERT ON Student FROM user_bob;

-- TCL
BEGIN TRANSACTION;
UPDATE Account SET balance = balance - 100 WHERE id = 1;
UPDATE Account SET balance = balance + 100 WHERE id = 2;
COMMIT;    -- or ROLLBACK if something went wrong
```

## 1.5 Relational Databases

Core concepts:

- Relation = table
- Tuple = row
- Attribute = column
- Domain, keys (primary, candidate, foreign)
- Relational schema and instance
- Brief introduction to relational algebra and SQL as the standard language



**Types of Keys in the Relational Model**

**Primary Key**: A Primary Key uniquely identifies each tuple in a relation. It must contain unique values and cannot have NULL values.

**Candidate Key**: A Candidate Key is a set of attributes that can uniquely identify a tuple in a relation.

**Super Key**: A Super Key is a set of attributes that can uniquely identify a tuple.

**Foreign Key**: A Foreign Key is an attribute in one relation that refers to the primary key of another relation.

**Composite Key**: A Composite Key is formed by combining two or more attributes to uniquely identify a tuple.

## 1.6 Database Design

Design process:

1. Requirement analysis
2. Conceptual design using **E-R model** (entities, attributes, relationships, cardinality, participation, weak entities, specialization/generalization)
3. Logical design → mapping E-R to relational model
4. Schema refinement and normalization (1NF to BCNF, functional dependencies)
5. Physical design (indexes, file organization)

## 1.7 Data Storage and Querying

Storage hierarchy:

- Disk → buffer manager → files and access methods → indexes (B+-tree, hash)

Query processing:

- Parsing → query rewrite → optimization → evaluation plan → execution Cost-based optimization using statistics

## 1.8 Transaction Management

Transaction = unit of work that must be **atomic, consistent, isolated, durable (ACID)**

- Concurrency control (locks, timestamp ordering, multiversion)
- Recovery (log-based, shadow paging, ARIES)
- Deadlock handling

### ACID Properties (Must Memorize Exactly)

| Property | Meaning | How DBMS Guarantees It |
|---|---|---|
| **A**tomicity | All operations succeed or none (all-or-nothing) | Logging + Undo (ROLLBACK) |
| **C**onsistency | Takes DB from one consistent state to another | Constraints + Application logic + Atomicity |
| **I**solation | Partial effects of incomplete transactions invisible to others | Concurrency control (locks, MVCC, timestamps) |
| **D**urability | Once committed, changes survive crashes | Write-Ahead Logging (WAL) + Force-write to disk |

### Concurrency Control (How multiple transactions run safely)

| Technique | How it Works | Pros / Cons |
|---|---|---|
| **Locking (2PL)** | Shared (S) lock for read, Exclusive (X) lock for write | Simple, strict 2PL guarantees serializability |
| | Strict 2PL: hold X locks until commit | Prevents cascading rollback |
| **Timestamp Ordering** | Each transaction gets timestamp; enforce order on conflicting operations | Deadlock-free, but can abort many times |
| **Multiversion CC** | Keeps old versions → readers never block writers (used in PostgreSQL, Oracle) | High concurrency, no reader blocking |
| **Validation (Optimistic)** | Read → Work → Validate (no conflict?) → Commit | Good when conflicts rare |

### 1.9 Database Architecture

- Centralized vs. client-server vs. parallel vs. distributed databases
- Three-tier architecture (presentation, application/logic, database)
- Data servers, transaction servers

## Database Architecture

| Architecture Type | Description | Typical Use-Cases / Examples (2025) |
|---|---|---|
| **Centralized** | Single computer, single DBMS instance | Small university department, legacy systems |

| Architecture Type | Description | Typical Use-Cases / Examples (2025) |
|---|---|---|
| **Client–Server** | Clients (apps) connect via network to a dedicated database server | Most traditional web apps (PHP + MySQL, Java + Oracle) |
| **Two-Tier** | Client machine runs application + directly talks to DB server | Desktop apps, early web apps |
| **Three-Tier** (most common today) | 1. Presentation tier (browser/mobile app) 2. Application/logic tier (Node.js, Django, Spring, etc.) 3. Database tier | Netflix, Instagram, Uber, Amazon, almost every modern web/mobile app |
| **Parallel Databases** | Multiple CPUs/disks in one machine or cluster – shared-memory, shared-disk, shared-nothing | Google BigQuery, Snowflake, Teradata, Vertica |
| **Shared-Nothing** (winner for scale) | Each node has its own CPU + memory + disk; only network connects them | YouTube (Vitess), TikTok, Facebook (MyRocks), Uber (Schemaless) |
| **Distributed Databases** | Data spread across geographically different sites, still appears as one logical DB | Google Spanner, CockroachDB, YugabyteDB, Planet-scale (Vitess) |
| **Cloud-Native Databases** | Fully managed, auto-scaling, multi-region replication | AWS Aurora, DynamoDB Global Tables, Azure Cosmos DB, Neon (serverless Postgres) |

## 1.10 Data Mining and Information Retrieval

- **Data mining**: discovering patterns (classification, clustering, association rules)
- **Information retrieval**: text search, ranking (e.g., PageRank), handling unstructured data Briefly distinguishes traditional DBMS from these newer areas

## 1.12 Database Users and Administrators

User categories:

- End users (naive, sophisticated)
- Application programmers
- Database administrator (DBA): schema definition, storage and authorization, backup/recovery, performance tuning, security

## Database Users and Administrators

| User Type | Who They Are | How They Interact with the DBMS | Real-Life Examples (2025) |
|---|---|---|---|
| **Naive / Casual Users** | Non-technical end users, no SQL knowledge | Use mobile/web apps or ATMs – never write queries | You checking bank balance on PhonePe Passenger booking Uber Student viewing grades on portal |
| **Sophisticated Users** | Analysts, engineers, scientists | Write complex SQL / analytical queries, use BI tools | Data analyst at Amazon writing "top 100 products this month" Researcher querying genomic database |
| **Application Programmers** | Software developers | Write code in Python/Java/Go/Node that embeds SQL or ORM | Backend engineer at Swiggy writing INSERT INTO orders … Full-stack dev using Django ORM |
| **Database Administrators (DBA)** | The "god" of the database | Highest privileges – everything below + system-level tasks | Person who upgrades PostgreSQL at 3 AM, creates backups, tunes indexes, grants permissions |

| Section | Concept | Real-World Example |
|---|---|---|
| 1.1 Database-System Applications | Where databases are used | • Spotify: stores billions of songs, user playlists, listening history • Uber: matches riders and drivers in real time using location data • Amazon: product catalog, orders, reviews, recommendations • Hospital: patient records, appointments, prescriptions |
| 1.2 Purpose of Database Systems | Problems of file-processing vs. DBMS | Without DBMS (old way): HR stores employee data in Excel, Payroll stores it in another Excel → same employee has different salaries → inconsistency! With DBMS (modern way): Single Oracle/MySQL database → one source of truth, automatic consistency |
| 1.3 View of Data | Three levels of abstraction | University database: • Physical level: data stored in B+-tree indexes on SSD • Logical level: tables Student(ID, name, dept), Enrolled(sid, cid, grade) • View level: – Student sees only own grades: CREATE VIEW MyGrades AS SELECT … – Registrar sees everything – Professor sees only own classes |

| Section | Concept | Real-World Example |
|---|---|---|
| 1.4 Database Languages | DDL vs DML (SQL) | DDL (schema): CREATE TABLE Student (id INT PRIMARY KEY, name VARCHAR(50)); DML (data): INSERT INTO Student VALUES (12345, 'Ali'); SELECT name FROM Student WHERE id = 12345; |
| 1.5 Relational Databases | Tables, keys, foreign keys | Banking example: Customer(customer_id PK, name, address) Account(account_no PK, balance, customer_id FK → Customer) |
| 1.6 Database Design | E-R → Relational (simplified) | Library E-R: Entity: Book(ISBN, title), Borrower(card_no, name) Relationship: Borrows (many-to-many) → Relational tables: Book(ISBN PK, title) Borrower(card_no PK, name) Borrows(ISBN FK, card_no FK, borrow_date, due_date) |
| 1.7 Data Storage and Querying | Indexes speeding up queries | Amazon searching 300 million products by name: Without index → 5–10 seconds With B+-tree index on product_name → < 50 ms |
| 1.8 Transaction Management | ACID in action | ATM withdrawal of $100: 1. Check balance ≥ 100 2. Deduct $100 from account 3. Dispense cash If power fails after step 2, recovery system rolls back → money is not lost (Durability & Atomicity) |
| 1.9 Database Architecture | Three-tier architecture | Instagram (mobile app): • Tier 1 (Presentation): Your phone screen • Tier 2 (Application): Python/Django servers that apply filters, generate news feed • Tier 3 (Database): Thousands of MySQL + Cassandra shards storing posts, likes, comments |
| 1.10 Data Mining and Information Retrieval | Mining vs Retrieval | • Data mining: Netflix discovers “people who watched Stranger Things also watched Wednesday” → recommendation • Information retrieval: You type “best noise-cancelling headphones” in Google → ranks pages using PageRank + relevance |
| 1.12 Database Users and Administrators | Different users | • Naive user: You checking your bank balance on the mobile app • Sophisticated user: Data analyst writing complex SQL to find “top 10 customers by spending this month” • Application programmer: Developer writing the Python backend for the banking app • DBA: Person who upgrades PostgreSQL at 3 a.m. when no one is using the system |

| Company / App | What Happens in Real Time | Main Database(s) Used | Why This DB? / Special Techniques |
|---|---|---|---|
| **Uber** | Rider opens app → sees cars moving, gets ETA, live pricing | • **Google Spanner** (global rides) • **Schemaless (MySQL fork)** • **Redis** (real-time cache) | Spanner = globally consistent in milliseconds Redis for driver location (billions of writes/sec) |
| **WhatsApp** | 100+ billion messages per day, all delivered instantly | • **SQLite** (on phone) • **Custom Erlang-based DB** (backend) • **RocksDB** for message storage | Extremely high write throughput, low latency |
| **TikTok** | You scroll → infinite personalized video feed in <200 ms | • **TiDB + MySQL** (user data) • **Redis + SSDB** (feed cache) • **ClickHouse** (analytics) | Recommendation engine queries DB 1000s times per scroll |
| **Google Maps** | 1 billion users see live traffic, rerouting in seconds | • **Spanner** (core map data) • **Bigtable** (traffic layers) • **Memorystore (Redis)** | Spanner handles planet-scale consistent reads/writes |
| **Stock Exchanges** (NYSE, Binance, Nasdaq) | Millions of trades per second, matching buy/sell instantly | • **kdb+** (primary for high-frequency trading) • **Aerospike / Redis** (order book) • **PostgreSQL** (settlement) | kdb+ can do millions of ops/sec in memory |
| **Airbnb** | You search “Paris, 3 guests, Dec 24–27” → instant results with live availability | • **MySQL + Vitess** (sharded) • **Redis** (search cache) • **Elasticsearch** (full-text) | Vitess scales MySQL to millions of QPS |
| **Netflix** | 200+ million users streaming, recommendations update live | • **Cassandra** (user profiles, viewing history) • **DynamoDB** (some regions) • **Redis** (session & queue) | Cassandra handles massive write scaling |

| Company / App | What Happens in Real Time | Main Database(s) Used | Why This DB? / Special Techniques |
|---|---|---|---|
| **Swiggy / DoorDash / Zomato** | You place food order → restaurant + delivery partner get it instantly | • **PostgreSQL** (main data) • **Redis** (real-time order tracking) • **Kafka + Flink** (event streaming) | Real-time ETA calculation using live GPS + Redis |
| **Paytm / PhonePe (India)** | UPI payments settle in <1 second, 10M+ transactions/min | • **Custom sharded MySQL** • **Redis** (fraud detection) • **Aerospike** (real-time balance) | Sub-100ms latency required by government regulation |
| **Discord** | 150M monthly users chatting, voice, live screen sharing | • **Cassandra** (chat history) • **ScyllaDB** (newer clusters) • **Redis** (online presence) | ScyllaDB = drop-in Cassandra replacement, 5–10× faster |
| **Tesla (in-car system)** | Live navigation, Autopilot data upload, over-the-air updates | • **SQLite** (per car) • **Cassandra + Kafka** (fleet data) • **ClickHouse** (telemetry analytics) | Millions of cars uploading sensor data every minute |
| **Formula 1 Timing** | 20 cars, 300+ sensors each, data shown live on TV in <0.1s | • **Oracle Database** + **Berkeley DB** (real-time) • **Custom C++ in-memory engine** | Official timing partner uses Oracle for decades |