Database Fundamentals

What is a Database?

- A **database** is a structured collection of data stored in a computer system.
- A **Database Management System (DBMS)** is software that interacts with the database and performs:
  - Querying (retrieving) data
  - Inserting, updating, and deleting data
  - Organizing how data is stored
- **Types of databases:**
  - **Simple**: File-based (e.g., text files, spreadsheets)
  - **Complex**: Systems with multiple tables and billions of records
- The most common type in use: **Relational Databases**

Relational Databases (RDBMS)

- Store data in **tables** made up of **columns (fields)** and **rows (records)**
- Each table represents a single entity (e.g., Customers, Orders)
- Tables are connected through **keys**:
  - **Primary Key**: Uniquely identifies a record
  - **Foreign Key**: References a primary key in another table
- Enable efficient cross-referencing and complex queries using **SQL**

RDBMS Features

- Manage relational databases using CRUD operations
- Enforce:
  - Data types and constraints
  - Query optimization
  - Data consistency and security
- Include:
  - Transaction management
  - Concurrency control
  - User permissions
  - Backup and recovery
- Examples: **MySQL, PostgreSQL, Oracle, SQL Server, SQLite**

SQL: Structured Query Language

- Used to access and manipulate data in relational databases
- Standardized and portable across most RDBMS
- **Types of SQL Commands:**
  - **DDL**: Data Definition (CREATE, ALTER)
  - **DML**: Data Manipulation (SELECT, INSERT, UPDATE, DELETE)
  - **DCL**: Data Control (GRANT, REVOKE)
  - **TCL**: Transaction Control (COMMIT, ROLLBACK)
- SQL is **declarative**: specify *what* you want, not *how* to do it

Naming Conventions

- **Clarity**: Descriptive names (e.g., `last_name`, `Orders`)
- **Preventing Errors**:
  - Avoid reserved SQL words (e.g., use `Orders`, not `Order`)
  - Use alphanumeric characters and underscores only

- **Consistency**:
  - Use either `snake_case` or `CamelCase` consistently
  - Choose singular or plural for table names and be consistent
- **Primary & Foreign Keys**:
  - Primary: `id` or `table_name_id` (e.g., `employee_id`)
  - Foreign: `referenced_table_id` (e.g., `department_id`)

Database Design Process

1. **Identify Entities and Attributes** (e.g., Books → title, ISBN)
2. **Determine Relationships** (e.g., one-to-many, many-to-many)
3. **Define Tables and Keys**
   - Assign primary and foreign keys
4. **Apply Normalization**
   - Reduce redundancy; follow 1NF, 2NF, 3NF
5. **Specify Data Types and Constraints**
   - Examples: `NOT NULL`, `UNIQUE`, `CHECK`, valid types like `INT`, `DATE`
6. **Outcome**: Schema or ER diagram to map structure

Data Integrity

Ensures data is accurate, consistent, and reliable.

Types:

1. **Entity Integrity**:
   - Primary key must be unique and not null
   - Prevents duplicate or missing records
2. **Referential Integrity**:
   - Foreign key values must match a valid primary key
   - Prevents orphaned records
3. **Domain Integrity**:
   - Enforced by:
     - Data types
     - Constraints (`NOT NULL`, `CHECK`, `UNIQUE`)
     - Lookup tables
   - Ensures values are valid and within range

Example:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL CHECK (total_amount >= 0)
);

ALTER TABLE orders
ADD FOREIGN KEY (customer_id) REFERENCES customers(customer_id);
```

Key Database Terms

- **Table**: Collection of data organized in rows and columns
- **Row (Record/Tuple)**: Single data entry
- **Column (Field/Attribute)**: Describes one aspect of all records
- **Primary Key (PK)**: Unique, not null
- **Foreign Key (FK)**: Refers to a PK in another table
- **Schema**: Database blueprint (tables, columns, constraints)

- **Index**: Speeds up data retrieval
- **View**: Virtual table based on a query
- **Stored Procedure**: Precompiled SQL routine
- **Trigger**: Executes automatically on table events
- **Constraint**: Rule enforcing valid data
- **Normalization**: Eliminates redundancy (1NF, 2NF, 3NF)
- **Denormalization**: Adds redundancy to optimize reads
- **Transaction**: Group of operations with ACID properties
- **Query**: Command to retrieve data (e.g., `SELECT`)
- **Query Optimizer**: Chooses efficient execution plan

Atomic Values

- **Atomic value**: Cannot be divided further
- Requirement for **1NF**: No lists or groups in a field

Why Atomic?

- Easier querying, filtering, and sorting
- Better integrity and indexing

Examples:

- ✅ `first_name = 'John'`, `last_name = 'Doe'`
- ✅ Normalize phone numbers into a separate table
- ✅ Split addresses into components (`street`, `city`, `state`)

Relationships in Databases

Types:

- **One-to-One (1:1)**:
  - One record in Table A matches one in Table B
  - Example: `Person` ↔ `Passport`
- **One-to-Many (1:N)**:
  - One record in Table A relates to many in Table B
  - Example: `Customer` → `Orders`
- **Many-to-Many (M:N)**:
  - Requires a **junction table**
  - Example: `Students` ↔ `Courses` via `Enrollments`

Design Tips:

- Use foreign keys to enforce relationships
- Use cascading actions (`ON DELETE`, `ON UPDATE`)
- Use indexes for query performance

Parent and Child Tables:

- **Parent**: Referenced table (e.g., `Orders`)
- **Child**: Table with foreign key (e.g., `OrderItems`)

Keys in Relational Databases

Types:

- **Primary Key**: Unique, not null; one per table
- **Foreign Key**: Refers to primary key in another table
- **Candidate Key**: Any column combo that can be a PK
- **Alternate Key**: A candidate key not chosen as PK
- **Composite Key**: Uses multiple columns (e.g., `order_id`, `product_id`)
- **Surrogate Key**: System-generated (e.g., auto-increment ID)
- **Natural Key**: Real-world value (e.g., SSN)

Best Practices:

- Prefer **surrogate keys** for simplicity and performance
- Use **natural keys** only when stable and meaningful
- Always index PKs and frequently queried FKs
- Make FK `NOT NULL` if required; allow `NULL` if optional

Schema Modeling & Lookup Tables

- **Lookup Tables**: Store valid values (e.g., `status`, `country`)
  - Enforce domain integrity
- **Join Tables (Junction Tables)**:
  - Used for M:N relationships
  - Composite PK of FKs from related tables
- Normalize for integrity; denormalize for performance **only when needed**

Referential Integrity, Indexing, and Cascades

- Use **foreign key constraints** for valid references
- Use **CASCADE**, **SET NULL**, or **RESTRICT** for update/delete behavior
- Index foreign keys manually if used in joins or WHERE clauses
- Match NULLability and constraints to business logic

Example: Order System Schema

- **Customers**: `customer_id` (PK)
- **Products**: `product_id` (PK)
- **Orders**:
  - `order_id` (PK)
  - `customer_id` (FK, NOT NULL)
  - `status_id` (FK to `OrderStatus`)
- **OrderItems**:
  - Composite PK: (`order_id`, `product_id`)
  - FKs to `Orders` and `Products`
- **OrderStatus**: Lookup table for order states
- **Addresses**: Optional FK from `Orders` to `Addresses`