# Database Technologies

# Database, DBMS, and DB Server

## Database

- A **collection of data** organized for easy access and management
- Stores facts, records, and information (tables, rows, columns, etc.)
- Example: Student grades, customer records, product catalog

Think of it as the **library of data**

## DBMS

- **Database Management System**
- Software that lets users **create, read, update, and delete** data in the database
- Provides query language (like SQL), transactions, security, and backups

Think of it as the **librarian who manages the library**

## DB Server

- The **computer/system** where the DBMS runs
- Handles requests from multiple clients and ensures reliable access to the database
- Provides processing power, storage, and network services

Think of it as the **library building that hosts both the books and the librarian**

## Summary Table

| Term | What it is | Analogy |
|---|---|---|
| **Database** | Data itself | Books in a library |
| **DBMS** | Software to manage data | Librarian |
| **DB Server** | Machine hosting DBMS & database | Library building |

# Three-layer Data Model Architecture

## Overview

A **three-layer data model** separates how data is represented, stored, and exchanged.
This improves **clarity, safety, and flexibility** in modern applications.

## 1. Dart Class (or any class)

- Data represented as **Dart objects**
- Provides **type safety** (compiler checks)
- Ensures **clear, maintainable code**

Example:

```dart
class Todo {
  final int id;
  final String title;
  final bool completed;
}
```

## 2. Database

- Data stored as **records** (tables, rows, columns)
- Ensures **persistence** (survives restarts)
- Provides **reliability** (transactions, indexing, backups)

Example:

```sql
INSERT INTO todos (id, title, completed)
VALUES (1, 'Learn Flutter', false);
```

# 3. API Communication

- Data transferred as **JSON**
- Lightweight, human-readable format
- Ensures **portability** (works across platforms)
- Enables **interoperability** (clients & servers)

Example:

```json
{
  "id": 1,
  "title": "Learn Flutter",
  "completed": false
}
```

## Summary Table

| Layer | Format | Purpose |
|---|---|---|
| **Dart Class** | Dart objects | Type Safety & Code Clarity |
| **Database** | Records (SQL) | Persistence & Reliability |
| **API** | JSON | Portability & Interoperability |

# Database Structure

## 1. Database 📚

- A **container** that stores and organizes data
- Can hold many tables (SQL) or collections (NoSQL)
- Provides persistence and security

### SQL Example

```sql
CREATE DATABASE SchoolDB;
```

### NoSQL Example (MongoDB)

```javascript
use("SchoolDB");
```

## 2. Table

- A **structured set** of related data
- In SQL: tables with rows & columns
- In NoSQL: collections with documents

### SQL Example

```
CREATE TABLE Students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT
);
```

### NoSQL Example (MongoDB)

```
db.createCollection("Students");
```

## 3. Row & Column

- **Row (Record/Document)** → one entry in the table/collection
- **Column (Field/Attribute)** → defines type of data stored

### SQL Example

```sql
INSERT INTO Students (id, name, age)
VALUES (1, 'Alice', 20);
```

### **NoSQL Example (MongoDB)** One File

```javascript
db.Students.insertOne({
  id: 1,
  name: "Alice",
  age: 20
});
```

## 4. CRUD Actions

**CRUD** = Create, Read, Update, Delete

Used to manipulate database data.

### SQL Example

```sql
-- Create
INSERT INTO Students (id, name, age) VALUES (2, 'Bob', 22);

-- Read
SELECT * FROM Students WHERE id = 2;

-- Update
UPDATE Students SET age = 23 WHERE id = 2;

-- Delete
DELETE FROM Students WHERE id = 2;
```

## NoSQL Example (MongoDB)

```
// Create
db.Students.insertOne({id: 2, name: "Bob", age: 22});

// Read
db.Students.find({id: 2});

// Update
db.Students.updateOne({id: 2}, {$set: {age: 23}});

// Delete
db.Students.deleteOne({id: 2});
```

## Summary

| Concept | SQL (Relational) | NoSQL (Document) |
|---|---|---|
| **Database** | Database | Database |
| **Table** | Table | Collection |
| **Row** | Record | Document |
| **Column** | Field | Attribute |
| **CRUD** | SQL Statements | JSON-like ops |

# SQL Database

## Key Idea

- SQL databases use **tables, rows, and columns**
- Data is stored in a **structured, relational format**
- Ensures consistency, integrity, and powerful querying with SQL

### Underlying Data Structures

SQL databases organize data using **specialized data structures**:

- **Tables** → stored as **files on disk** (rows & columns)
- **Indexes** → often implemented as **B-Trees** for fast search
- **Relations** → connections between tables using **foreign keys**
- **Transactions** → ensure **ACID properties** (Atomicity, Consistency, Isolation, Durability)

## Organization Diagram

```
SchoolDB (Database)
 ├── Index files
 ├── Relations files
 ├── Students (Table)
 │     ├── Row (Record) -> {id: 1, name: Alice, age: 20}
 │     └── Row (Record) -> {id: 2, name: Bob, age: 22}
 ├── Courses (Table)
 └── Teachers (Table)
```

## 1. Row = Record

A **row** in a table represents a **record**.

Each row stores data for one entity.

```sql
-- Row in Students table
INSERT INTO Students (id, name, age)
VALUES (1, 'Alice', 20);
```

## 2. Table

A **table** is a structured set of rows and columns.

This is equivalent to a **collection** in NoSQL.

```sql
CREATE TABLE Students (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  age INT
);
```

Example content:

| id | name | age |
|---|---|---|
| 1 | Alice | 20 |
| 2 | Bob | 22 |

## 3. Database

A **database** contains multiple tables.

This is equivalent to a **directory of collections** in NoSQL.

```
CREATE DATABASE SchoolDB;
```

Inside `SchoolDB`:

- Students table
- Courses table
- Teachers table

## Summary

| SQL Concept | SQL Meaning |
|---|---|
| Database | Container of tables |
| Table | Structured dataset (rows & columns) |
| Row | Record (one entry) |
| Column | Field (attribute of data) |

# Understanding NoSQL Database

## Key Idea

- NoSQL databases often use **JSON-like objects**
- Data is stored in a flexible, hierarchical structure
- Easier to scale and adapt compared to relational databases

# 1. Document = Row

A **JSON object** stored in a file represents a **document**.

This is equivalent to a **row** in SQL.

```json
{
  "id": 1,
  "name": "Alice",
  "age": 20
}
```

## 2. Collection = Table

A **directory with documents** is called a **collection**.

This is equivalent to a **table** in SQL.

```
// Students collection
{
  "id": 1, "name": "Alice", "age": 20
}
{
  "id": 2, "name": "Bob", "age": 22
}
```

## 3. Database = Set of Collections

A **directory of collections** makes up a **NoSQL database**.

This is equivalent to a **database** in SQL.

```
SchoolDB/                    <-- Database
  └── Students/              <-- Collection (Table)
        ├── alice.json       <-- Document (Row)
        └── bob.json
  └── Courses/               <-- Collection (Table)
        ├── math.json
        └── science.json
```

## Summary

| SQL Concept | NoSQL Equivalent |
|---|---|
| Database | Database (folder of collections) |
| Table | Collection (folder of documents) |
| Row | Document (JSON object) |
| Column | Field (key-value pair in JSON) |

# Four Databases (ASE 456)

PocketBase • IndexedDB • SQLite • Firebase

## PocketBase

- **Lightweight DB server** that runs locally or on a server
- **Backed by SQLite** (relational engine)
- Provides a **NoSQL-like API** (collections & documents)
- Useful for rapid prototyping and small apps

**Match:** Hybrid (SQL engine + NoSQL API)

## IndexedDB

- Built into **web browsers**
- Stores data as **key-value pairs**
- Schema-less, works with **objects** (often JSON)
- Best for **offline web apps** and caching

**Match:** NoSQL (object store)

## SQLite

- **Embedded relational database**
- Stores data in a single **file**
- Full **SQL support** (tables, rows, columns)
- Popular in **mobile, desktop, and IoT apps**

**Match:** SQL (relational)

# Firebase

- **Cloud NoSQL database** by Google
- Two options:
  - **Realtime Database** (tree structure)
  - **Firestore** (collections & documents)
- Scales easily, syncs across devices

**Match:** NoSQL (document store)

## Summary Table

| Database | Type | Characteristics | Match | DB / DB Server |
|---|---|---|---|---|
| **PocketBase** | Hybrid | SQLite engine, NoSQL-style API | SQL + NoSQL | **Database Server** |
| **IndexedDB** | Client-side | Browser storage, JSON-like objects | NoSQL | **Database** |
| **SQLite** | Embedded | File-based, relational, SQL support | SQL | **Database** |
| **Firebase** | Cloud | Realtime sync, collections & documents | NoSQL | **Database Server** |

Dart access Database and Database Server using API, so there is no difference in terms of usage.

## Database vs Database Server in Dart

### Key Idea

- Dart always uses an **API/driver** to access data
- CRUD code looks the **same**

### Difference

- **Database** = the data itself
- **Database Server** = software hosting databases, handling security, concurrency, transactions

## Summary

In Dart:

- Usage feels the same
- Deployment & management differ