Database Modeling

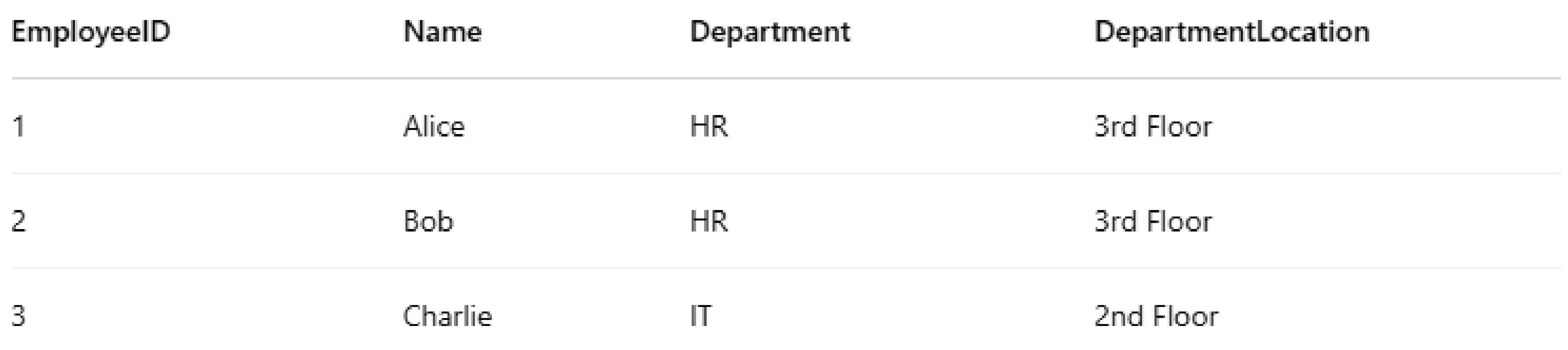
Database modeling is the process of designing and structuring data efficiently so that it can be stored, managed, and retrieved optimally. It defines how different pieces of data relate to each other and ensures the following:

1. Data Integrity – Ensures data is accurate and consistent.
2. Redundancy Reduction – Prevents storing the same data multiple times.
3. Efficient Queries – Helps in retrieving data quickly.
4. Scalability – Allows databases to grow as new features or data are added.

1. Prevents Redundant (Duplicate) Data

Redundant data means storing the same information multiple times, leading to wasted storage and data inconsistency.

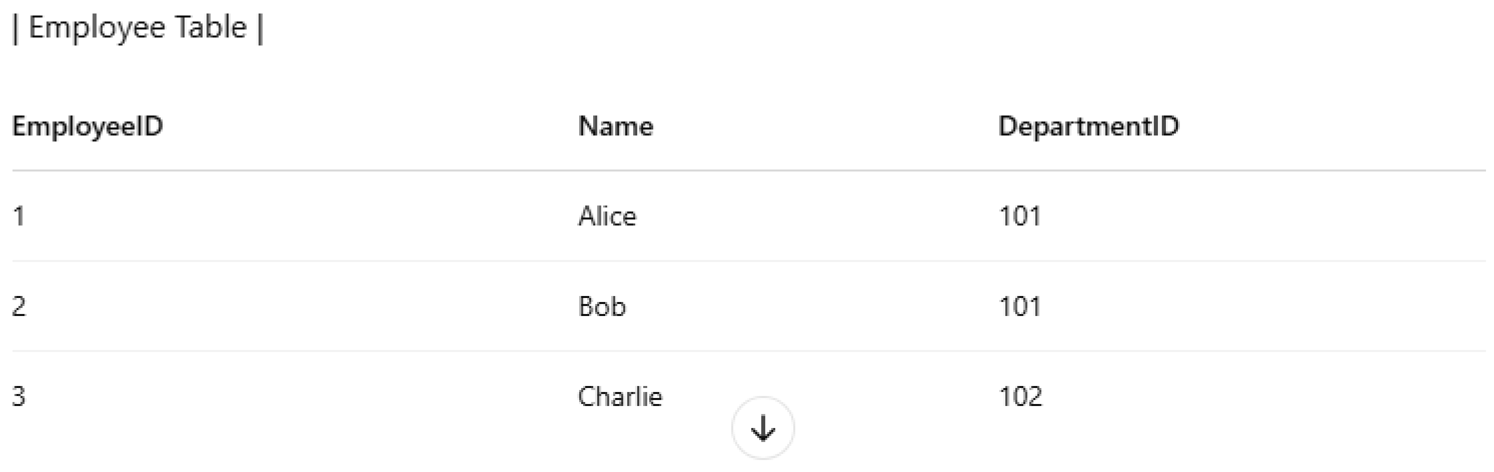
Example: Without Proper Database Modeling (Bad Design)



EmployeeID	Name	Department	DepartmentLocation
1	Alice	HR	3rd Floor
2	Bob	HR	3rd Floor
3	Charlie	IT	2nd Floor

🚫 Problem: The "HR" department's location is stored twice. If the HR department moves, you need to update multiple rows, leading to inconsistencies.

Solution: Normalize the Database (Good Design)



| Employee Table |

EmployeeID	Name	DepartmentID
1	Alice	101
2	Bob	101
3	Charlie	102

| Department Table |

DepartmentID	DepartmentName	Location
101	HR	3rd Floor
102	IT	2nd Floor

Now, if the HR department moves, you only update one row.

2. Enforce Rules

Enforcing rules in a database ensures data accuracy, security, and consistency. This is done using constraints, relationships, and automation.

2.1 Constraints in SQL

Constraints define rules for valid data entry.

Constraint	Purpose	Example
Primary Key (PK)	Ensures unique identification of each row	`UserID` in `Users` table
Foreign Key (FK)	Links tables and enforces relationships	`CustomerID` in `Orders` table references `Customers`
Not Null	Prevents empty (NULL) values	`Email` must not be NULL
Unique	Prevents duplicate values	Each `Username` must be unique
Check	Restricts values based on a condition	`Age` must be ≥ 18

Example: Applying Constraints in SQL

```
CREATE TABLE Users (
UserID INT PRIMARY KEY,
Name VARCHAR(100) NOT NULL,
Email VARCHAR(100) UNIQUE,
Age INT CHECK (Age >= 18)
);
```

This ensures users have a name, unique email, and are at least 18 years old.

2.2 Enforcing Business Rules with Triggers

Triggers automate rule enforcement when data is inserted or updated.

Example: Prevent Negative Account Balance

```
CREATE TRIGGER prevent_negative_balance
BEFORE UPDATE ON Accounts
FOR EACH ROW
WHEN (NEW.Balance < 0)
BEGIN
RAISE ABORT 'Balance cannot be negative!';
END;
```

This ensures that no account can have a negative balance.

2.3 Referential Integrity (Maintaining Data Consistency)

Foreign keys enforce relationships and prevent orphaned data.

Example: Preventing Orders Without a Customer

```
CREATE TABLE Orders (
OrderID INT PRIMARY KEY,
CustomerID INT,
Amount DECIMAL(10,2),
FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

A new order cannot be placed unless a valid CustomerID exists.

3. Organize Data Logically

Organizing data logically ensures efficient storage, easy retrieval, and scalability. A well-structured database minimizes redundancy, improves performance, and makes data easier to manage.

- Normalization reduces redundancy and improves consistency.
- Indexing speeds up searches and enhances performance.
- Data Warehouses help analyze large-scale historical data.

Fundamental Concepts

1. Relations (Tables)

A relation represents a table in a database. It consists of rows and columns where each row is a record (tuple), and each column is an attribute.

2. Tuple (Record)

A single row in a table that represents one entity instance.

Example: A row in a users table containing id, name, and email.

3. Attribute (Column)

A column represents a specific property of an entity.

Example: In a users table, name and email are attributes

4. Keys (Identifiers for Uniqueness & Relationships)

Primary Key (PK): Uniquely identifies each row. Example: id in users.

Foreign Key (FK): Establishes a relationship between two tables. Example: user_id in orders references id in users.

Types of Relationships

Relationships define how tables interact with each other. Here are the primary types:

1. One-to-One (1-1) Relationship

Each row in Table A relates to only one row in Table B.

Example: A user can have only one profile and vice versa.

```
CREATE TABLE users (
 id INT PRIMARY KEY,
name VARCHAR(255)
          );
```

```
CREATE TABLE profiles (
  id INT PRIMARY KEY,
  user_id INT UNIQUE,
  bio TEXT,
FOREIGN KEY (user_id) REFERENCES users(id)
);
```

2. One-to-Many (1-M) Relationship

A row in Table A relates to multiple rows in Table B.

Example: A customer can place multiple orders.

```
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(255)
);

CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT,
  amount DECIMAL(10,2),
FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

3. Many-to-Many (M-M) Relationship

Multiple rows in Table A relate to multiple rows in Table B.

Example: A student can enroll in multiple courses, and a course can have multiple students.

Implemented via a junction table.

```
CREATE TABLE students (
  id INT PRIMARY KEY,
  name VARCHAR(255)
);

CREATE TABLE courses (
  id INT PRIMARY KEY,
  title VARCHAR(255)
);

CREATE TABLE enrollments (
  student_id INT,
  course_id INT,
  PRIMARY KEY (student_id, course_id),
  FOREIGN KEY (student_id) REFERENCES students(id),
  FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

Normalization Forms (NF) in Databases

Normalization is the process of structuring a database to reduce redundancy and improve data integrity. It involves breaking down tables into smaller, related ones while preserving relationships.

There are different Normalization Forms (NFs), each ensuring a higher level of data consistency.

1. First Normal Form (1NF) – Atomicity & Uniqueness

A table is in 1NF if:

- All columns contain atomic (indivisible) values.
- Each row is unique (has a primary key).
- There are no repeating groups or arrays.

Example – Non-1NF (Repeating Groups)

OrderID	Customer	Product(s)	Price
1	Alice	Laptop, Mouse	1000
2	Bob	Phone, Earbuds	1200

🚫 Problem: "Product(s)" contains multiple values (not atomic).

Converted to 1NF (Atomic Values in Rows)

OrderID	Customer	Product	Price
1	Alice	Laptop	1000
1	Alice	Mouse	50
2	Bob	Phone	1200
2	Bob	Earbuds	100

2. Second Normal Form (2NF) – Removing Partial Dependencies

A table is in 2NF if:

- It is already in 1NF.

- Every non-key column is fully dependent on the primary key (No partial dependencies).

Example – Non-2NF (Partial Dependency)

OrderID	Product	CustomerName	Address
1	Laptop	Alice	NYC
2	Phone	Bob	LA

🚫 Problem: CustomerName & Address depend only on OrderID, not Product.

Converted to 2NF (Separate Related Data into Tables)

Customers Table

CustomerID	Name	Address
1	Alice	NYC
2	Bob	LA

Orders Table

OrderID	CustomerID
1	1
2	2

OrderDetails Table

OrderID	Product
1	Laptop
2	Phone

3. Third Normal Form (3NF) – Removing Transitive Dependencies

A table is in 3NF if:
- It is already in 2NF.
- All columns depend only on the primary key, not on other non-key columns (No transitive dependency).

Example – Non-3NF (Transitive Dependency)

StudentID	Name	Course	Instructor	InstructorPhone
1	Alice	Math	Prof. Smith	123-456-7890
2	Bob	Science	Prof. Lee	987-654-3210

🚫 Problem: InstructorPhone depends on Instructor, not directly on StudentID.

Converted to 3NF (Separate Related Data into Tables)

Students Table

StudentID	Name
1	Alice
2	Bob

Courses Table

CourseID	Course	InstructorID
101	Math	1
102	Science	2

Instructors Table

InstructorID	Name	Phone
1	Prof. Smith	123-456-7890
2	Prof. Lee	987-654-3210