`RDBMS` (Relational Database Management System)

> Relational Database Management Systems (RDBMS) organize data into tables, which consist of rows and columns. Tables in an RDBMS are related to each other through keys. Let's create an example with two relational tables, `customers` and `orders`, and explain the key concepts in detail.

# Example Tables

## Table 1: `customers`

This table stores information about customers.

| customerNumber | customerName | contactLastName | contactFirstName | phone | city | c |
|---|---|---|---|---|---|---|
| 101 | ABC Corp | Smith | John | 555-1234 | New York | U |
| 102 | XYZ Inc | Doe | Jane | 555-5678 | Los Angeles | U |
| 103 | Acme Ltd | Johnson | Peter | 555-9012 | Chicago | U |

## Table 2: `orders`

This table stores information about customer orders.

| orderNumber | orderDate | customerNumber | status |
|---|---|---|---|
| 2001 | 2024-08-01 | 101 | Shipped |
| 2002 | 2024-08-02 | 102 | Processing |
| 2003 | 2024-08-03 | 103 | Shipped |
| 2004 | 2024-08-04 | 101 | Pending |

# Key Concepts Explained

## 1. Tables and Columns

- **Tables**: In an RDBMS, data is stored in tables. Each table represents an entity (e.g., customers, orders) and is made up of rows and columns.
- **Columns**: Columns in a table define the attributes of the entity. For example, the `customers` table has columns like `customerNumber`, `customerName`, and `city`, representing different attributes of a customer.

### 2. Rows

- **Rows**: A row (also known as a record) in a table represents a single instance of the entity. For example, each row in the `customers` table represents a single customer. In the `orders` table, each row represents a single order.

### 3. Primary Key

- **Primary Key**: A primary key is a unique identifier for each row in a table. In the `customers` table, the `customerNumber` is the primary key. This means that each customer has a unique `customerNumber`, and no two customers can have the same `customerNumber`.
- **Uniqueness and Not Null**: The primary key ensures that each value is unique and cannot be NULL. This ensures that every row is uniquely identifiable.

### 4. Foreign Key

- **Foreign Key**: A foreign key is a column or set of columns in one table that refers to the primary key in another table. In the `orders` table, the `customerNumber` column is a foreign key that links each order to a customer in the `customers` table.
- **Referential Integrity**: The foreign key ensures that the value in the `customerNumber` column of the `orders` table must match a value in the `customerNumber` column of the `customers` table. This enforces a relationship between the two tables.

### 5. Relationships

- **One-to-Many Relationship**: The relationship between `customers` and `orders` is an example of a one-to-many relationship. Each customer can have multiple orders, but each order is linked to only one customer. This is achieved through the foreign key `customerNumber` in the `orders` table.

### 6. Data Integrity

- **Data Integrity**: Data integrity refers to the accuracy and consistency of data in the database. Primary keys and foreign keys help enforce data integrity by ensuring that each record is unique (primary key) and that relationships between tables are valid (foreign key).

### 7. Normalization

- **Normalization**: Normalization is the process of organizing data to reduce redundancy and improve data integrity. In this example, the `customers` and `orders` tables are normalized, with customer data stored separately from order data. This avoids duplication of customer information across multiple orders.

## SQL to Create the Tables

Here's how you would create these tables in SQL:

### `customers` Table

```
CREATE TABLE customers (
    customerNumber int PRIMARY KEY,
    customerName varchar(50) NOT NULL,
    contactLastName varchar(50) NOT NULL,
    contactFirstName varchar(50) NOT NULL,
    phone varchar(15) NOT NULL,
    city varchar(50) NOT NULL,
    country varchar(50) NOT NULL
);
```

### `orders` Table

```
CREATE TABLE orders (
    orderNumber int PRIMARY KEY,
    orderDate date NOT NULL,
    customerNumber int,
    status varchar(20) NOT NULL,
    FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber)
);
```

## How These Tables Work Together

- **Inserting Data**: When inserting a new order, the `customerNumber` in the `orders` table must match an existing `customerNumber` in the `customers` table. This ensures that every order is associated with a valid customer.

- **Querying Data**: You can join these tables in a query to get a list of customers and their orders. For example:

```
SELECT customers.customerName, orders.orderNumber, orders.orderDate,
orders.status
FROM customers
JOIN orders ON customers.customerNumber = orders.customerNumber;
```

This query returns the customer name along with the corresponding order details.

## Summary

In an RDBMS, tables store data in rows and columns, with primary keys ensuring each row is unique. Foreign keys establish relationships between tables, maintaining referential integrity. The structure of relational tables and the relationships between them help organize data efficiently, avoiding redundancy and ensuring data consistency.

In [ ]:
```
-- Active: 1723560744721@@127.0.0.1@3306@sakila
Create DATABASE myDB;
```

In [ ]:
```
Use myDB;
```

In [ ]:
```
-- DROP DATABASE myDB;
```

In [ ]:
```
-- Active: 1723560744721@@127.0.0.1@3306@mydb
CREATE TABLE customers (
    customerNumber int AUTO_INCREMENT PRIMARY KEY,
    customerName varchar(50) NOT NULL,
    contactLastName varchar(50) NOT NULL,
    contactFirstName varchar(50) NOT NULL,
    phone varchar(15) NOT NULL,
    city varchar(50) NOT NULL,
    country varchar(50) NOT NULL
);


CREATE TABLE orders (
    orderNumber int PRIMARY KEY,
    orderDate date NOT NULL,
    customerNumber int,
    status varchar(20) NOT NULL,
    FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber)
);
```

In [ ]:
```
INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstN
VALUES (101, 'ABC Corp', 'Smith', 'John', '555-1234', 'New York', 'USA');

INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstN
VALUES (102, 'XYZ Inc', 'Doe', 'Jane', '555-5678', 'Los Angeles', 'USA');

INSERT INTO customers (customerNumber, customerName, contactLastName, contactFirstN
VALUES (103, 'Acme Ltd', 'Johnson', 'Peter', '555-9012', 'Chicago', 'USA');
```

In [ ]:
```
INSERT INTO orders (orderNumber, orderDate, customerNumber, status)
VALUES
    (2001, '2024-08-01', 101, 'Shipped'),
    (2002, '2024-08-02', 102, 'Processing'),
    (2003, '2024-08-03', 103, 'Shipped'),
    (2004, '2024-08-04', 101, 'Pending');
```

In [ ]:
```
select * from customers where `contactFirstName`='Jane';
```

You can use the `ALTER TABLE` and `DROP` statements to modify the structure of your tables or remove specific columns or constraints. Here's how you can use them with the `customers` and `orders` tables:

## Using `ALTER TABLE`

### 1. Adding a New Column

Suppose you want to add a new column called `email` to the `customers` table to store customer email addresses.

```
ALTER TABLE customers
ADD email varchar(100) DEFAULT NULL;
```

- **Explanation**: This statement adds a new column `email` of type `varchar(100)` to the `customers` table. The `DEFAULT NULL` ensures that if no value is provided for this column, it will be set to `NULL`.

### 2. Modifying an Existing Column

If you want to change the data type of the `phone` column from `varchar(15)` to `varchar(20)` in the `customers` table:

```
ALTER TABLE customers
MODIFY phone varchar(20) NOT NULL;
```

- **Explanation**: This statement changes the data type of the `phone` column to `varchar(20)` and retains the `NOT NULL` constraint.

### 3. Renaming a Column

Suppose you want to rename the `status` column in the `orders` table to `orderStatus`:

```
ALTER TABLE orders
CHANGE status orderStatus varchar(20) NOT NULL;
```

- **Explanation**: This statement renames the `status` column to `orderStatus` while keeping its data type and `NOT NULL` constraint.

### 4. Dropping a Column

If you no longer need the `addressLine2` column in the `customers` table:

```
ALTER TABLE customers
DROP COLUMN addressLine2;
```

- **Explanation**: This statement removes the `addressLine2` column from the `customers` table.

### 5. Adding a Foreign Key Constraint

If the `orders` table was initially created without a foreign key constraint on `customerNumber`, you can add it later:

```
ALTER TABLE orders
ADD CONSTRAINT fk_customer
FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);
```

- **Explanation**: This statement adds a foreign key constraint to the `orders` table, ensuring that the `customerNumber` in the `orders` table corresponds to a valid `customerNumber` in the `customers` table.

## Using `DROP`

### 1. Dropping a Column

If you decide that the `postalCode` column in the `customers` table is no longer necessary:

```
ALTER TABLE customers
DROP COLUMN postalCode;
```

- **Explanation**: This command removes the `postalCode` column from the `customers` table.

### 2. Dropping a Foreign Key Constraint

If you need to remove the foreign key constraint linking `customerNumber` in the `orders` table to `customers`:

```
ALTER TABLE orders
DROP FOREIGN KEY fk_customer;
```

- **Explanation**: This statement drops the foreign key constraint named `fk_customer` from the `orders` table.

### 3. Dropping an Entire Table

If you want to completely remove the `orders` table from the database:

```
DROP TABLE orders;
```

- **Explanation**: This statement permanently deletes the `orders` table and all the data it contains from the database. This action is irreversible, so it should be used with caution.

## Summary

- `ALTER TABLE`: Used to make changes to an existing table, such as adding, modifying, or dropping columns and constraints.
- `DROP`: Used to remove columns, constraints, or even entire tables from the database.

These commands provide powerful ways to manage and update the structure of your database tables as your requirements evolve.

In [ ]:
```
ALTER TABLE customers
  ADD email varchar(100) DEFAULT NULL;
```

In [ ]:
```
SELECT * FROM customers;
```

In [ ]:
```
ALTER TABLE customers
   MODIFY phone varchar(20) NOT NULL;
```

In [ ]:
```
 ALTER TABLE orders
   CHANGE status orderStatus varchar(20) NOT NULL;
```

In [ ]:
```
ALTER TABLE customers
   DROP COLUMN email;
```

In [ ]:
```
ALTER TABLE orders
   ADD CONSTRAINT fk_customer
   FOREIGN KEY (customerNumber) REFERENCES customers(customerNumber);
```

In [ ]:
```
SELECT * from orders;
```

In [ ]:
```
Drop Table orders;
```

In [ ]:
```
DROP DATABASE myDB;
```