**1. Primary Keys**

A **Primary Key** is a column or a set of columns that uniquely identifies each row in a table.

* **Uniqueness**: No two rows can have the same primary key value.
* **Non-null**: Primary keys cannot have NULL values.

**Key Points:**

* Ensures that every row in a table is unique.
* Can consist of a single column or multiple columns (called a **composite primary key**).

**Syntax:**

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...,

PRIMARY KEY (column1, column2)

);

**Example:**

Let's create a table for **Employees** where each employee has a unique ID.

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Department VARCHAR(50),

PRIMARY KEY (EmployeeID)

);

Here, EmployeeID is the **primary key** for the table.

**2. Foreign Keys**

A **Foreign Key** is a column (or a set of columns) in a table that establishes a link between data in two tables. It points to a **primary key** in another table, thus creating a relationship between the two tables.

**Key Points:**

* Helps enforce referential integrity.
* Prevents invalid data from being inserted into the foreign key column.
* Can accept **NULL** values unless explicitly stated otherwise.

**Syntax:**

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

...,

FOREIGN KEY (column1) REFERENCES another\_table (column)

);

**Example:**

Let's create a second table, **Departments**, where each department has a unique ID, and link it to the **Employees** table using a foreign key.

CREATE TABLE Departments (

DepartmentID INT NOT NULL,

DepartmentName VARCHAR(50),

PRIMARY KEY (DepartmentID)

);

-- Add a foreign key in the Employees table to refer to the Departments table

ALTER TABLE Employees

ADD FOREIGN KEY (Department) REFERENCES Departments(DepartmentID);

Here, the Department column in the **Employees** table becomes a **foreign key** referencing the DepartmentID in the **Departments** table.

**3. Types of Relationships Between Tables**

1. **One-to-One (1:1) Relationship**  
   In this relationship, each record in one table is linked to one, and only one, record in another table.
   * Example: A table of **EmployeeDetails** linked to the **Employees** table, where each employee has only one detailed record.

CREATE TABLE EmployeeDetails (

EmployeeID INT,

Address VARCHAR(100),

Phone VARCHAR(20),

PRIMARY KEY (EmployeeID),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

1. **One-to-Many (1**

**) Relationship**  
In this relationship, a record in one table can be associated with multiple records in another table.

* + Example: A department can have multiple employees.

CREATE TABLE Employees (

EmployeeID INT NOT NULL,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT,

PRIMARY KEY (EmployeeID),

FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)

);

1. **Many-to-Many (M**

**) Relationship**  
This relationship requires a **junction (or associative)** table to manage the relationship between two tables.

* + Example: If an employee can work on multiple projects and each project can have multiple employees.

CREATE TABLE Projects (

ProjectID INT,

ProjectName VARCHAR(50),

PRIMARY KEY (ProjectID)

);

CREATE TABLE EmployeeProjects (

EmployeeID INT,

ProjectID INT,

PRIMARY KEY (EmployeeID, ProjectID),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID),

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID)

);

Here, the **EmployeeProjects** table is the junction table managing the many-to-many relationship between **Employees** and **Projects**.

**4. Practical Examples: Queries**

**Example 1: Insert Data into Employees and Departments**

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID) VALUES

(101, 'John', 'Doe', 1),

(102, 'Jane', 'Smith', 2),

(103, 'Mike', 'Johnson', 2),

(104, 'Sara', 'Williams', 3);

**Example 2: Query to Show Employees and Their Departments**

SELECT e.EmployeeID, e.FirstName, e.LastName, d.DepartmentName

FROM Employees e

JOIN Departments d ON e.DepartmentID = d.DepartmentID;

**Example 3: Insert Data into Projects and Assign Employees to Projects**

INSERT INTO Projects (ProjectID, ProjectName) VALUES

(1001, 'Website Development'),

(1002, 'App Design');

INSERT INTO EmployeeProjects (EmployeeID, ProjectID) VALUES

(101, 1001),

(102, 1001),

(102, 1002),

(103, 1002);

**Example 4: Query to Show Which Employees Are Working on Which Projects**

SELECT e.FirstName, e.LastName, p.ProjectName

FROM Employees e

JOIN EmployeeProjects ep ON e.EmployeeID = ep.EmployeeID

JOIN Projects p ON ep.ProjectID = p.ProjectID;

**5. Cascading Actions (ON DELETE/UPDATE CASCADE)**

Sometimes, you may want changes in the parent table to automatically propagate to the child table. This is where cascading actions come into play.

**Example: Set Cascading Deletes**

If you delete a department, you might want to automatically delete all employees associated with it.

ALTER TABLE Employees

ADD CONSTRAINT fk\_department

FOREIGN KEY (DepartmentID)

REFERENCES Departments(DepartmentID)

ON DELETE CASCADE;

Now, if you delete a row in the **Departments** table, all related employees will be deleted.

**6. Exercises for Practice**

1. Create tables for a **School Database** with entities like Students, Classes, and Teachers. Define the relationships between these tables (e.g., a student can take multiple classes, a class can have multiple students, but only one teacher).
2. Write SQL queries to:
   * Insert data into the Students and Classes tables.
   * Assign students to classes.
   * Retrieve a list of students along with their assigned classes.
3. Modify your table design to include **cascading updates** or **deletes** when necessary.