Lecture Notes on SQL Constraints

Introduction to SQL Constraints

In the world of databases, constraints are rules that define the permissible values and operations on data stored in tables. They ensure data integrity by enforcing restrictions and conditions on the data, preventing invalid or inconsistent entries. SQL provides several types of constraints, each serving a specific purpose in maintaining the reliability and accuracy of database records.

Common Types of SQL Constraints

1. **PRIMARY KEY**:
   - Defines a column or a combination of columns that uniquely identify each row in a table.
   - Ensures data integrity by preventing duplicate or null values in the primary key column(s).
   - Example:

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE
);
```

2. **UNIQUE**:
   - Ensures that the values in a column or a combination of columns are unique across all rows in the table.
   - Useful for enforcing uniqueness constraints on candidate keys.
   - Example:

```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    name VARCHAR(100) UNIQUE,
    manager_id INT
);
```

3. **NOT NULL**:

- Specifies that a column must contain a value and cannot be null.
- Prevents the insertion of null values into the specified column.
- Example:

```sql
CREATE TABLE projects (
    project_id INT PRIMARY KEY,
    name VARCHAR(100),
    start_date DATE NOT NULL,
    end_date DATE
);
```

4. **CHECK**:
   - Defines a condition that must be satisfied by the values entered into a column.
   - Allows custom data validation rules to be enforced on the column.
   - Example:

```sql
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    salary DECIMAL(10, 2) CHECK (salary > 0)
);
```

```sql
CREATE TABLE Users (
  email VARCHAR(255) NOT NULL,
  CONSTRAINT email_check CHECK (email LIKE '%@%.%')
);
```

5. **DEFAULT**:
   - Specifies a default value for a column if no value is provided during insertion.
   - Automatically assigns the default value when a new row is added.
   - Example:

```sql
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    status VARCHAR(20) DEFAULT 'active'
);
```

6. FOREIGN KEY:
   - Establishes a relationship between two tables by linking the primary key of one table to a column in another table.
   - Ensures referential integrity and maintains consistency between related tables.
   - Example:

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    name VARCHAR(100),
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

Use Cases and Examples

Let's consider a scenario involving the `employees`, `departments`, and `projects` tables:

- Primary Key: The `employee_id` column in the `employees` table uniquely identifies each employee.
- Unique Constraint: Each department name in the `departments` table must be unique to avoid ambiguity.
- Not Null Constraint: The `start_date` column in the `projects` table is mandatory for every project.
- Check Constraint: The `salary` column in the `employees` table must always contain positive values.
- Default Constraint: The `status` column in the `employees` table defaults to 'active' if not specified during insertion.
- Foreign Key Constraint: The `department_id` column in the `employees` table references the `department_id` column in the `departments` table, ensuring that each employee belongs to a valid department.

Conclusion

SQL constraints are vital tools for maintaining data integrity and consistency in relational databases. By enforcing rules and restrictions on table columns, constraints help ensure the accuracy and reliability of database records. Understanding the various types of constraints and their applications is essential for designing robust and efficient database schemas.