Lab Experiment | 4

Implement Various SQL Constraints to Enforce Data Integrity.

Objectives

- Understand and implement various SQL constraints to maintain data integrity.
- Learn how constraints help enforce rules at the database level.
- Practice using different types of constraints such as PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, and DEFAULT.

Introduction

Constraints in SQL are rules enforced on data to maintain accuracy and reliability. Constraints can be applied at the column level or table level. The following constraints will be covered:
- PRIMARY KEY: Ensures a unique identifier for each record.
- FOREIGN KEY: Maintains referential integrity between tables.
- UNIQUE: Ensures all values in a column are distinct.
- NOT NULL: Prevents NULL values in a column.
- CHECK: Restricts the values that can be inserted.
- DEFAULT: Assigns a default value if no value is provided.

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARYKEY	Uniquely identifies each row of the table
FOREIGNKEY	Establishes and enforces a referential integrity between the column and a column of the referenced table such that values in one table match values in another table.
CHECK	Specifies a condition that must be true

Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules, except that the name cannot be the same as another object owned by the same user. If you do not name your constraint, the Oracle server generates a name with the format SYS_Cn, where n is an integer so that the constraint name is unique.

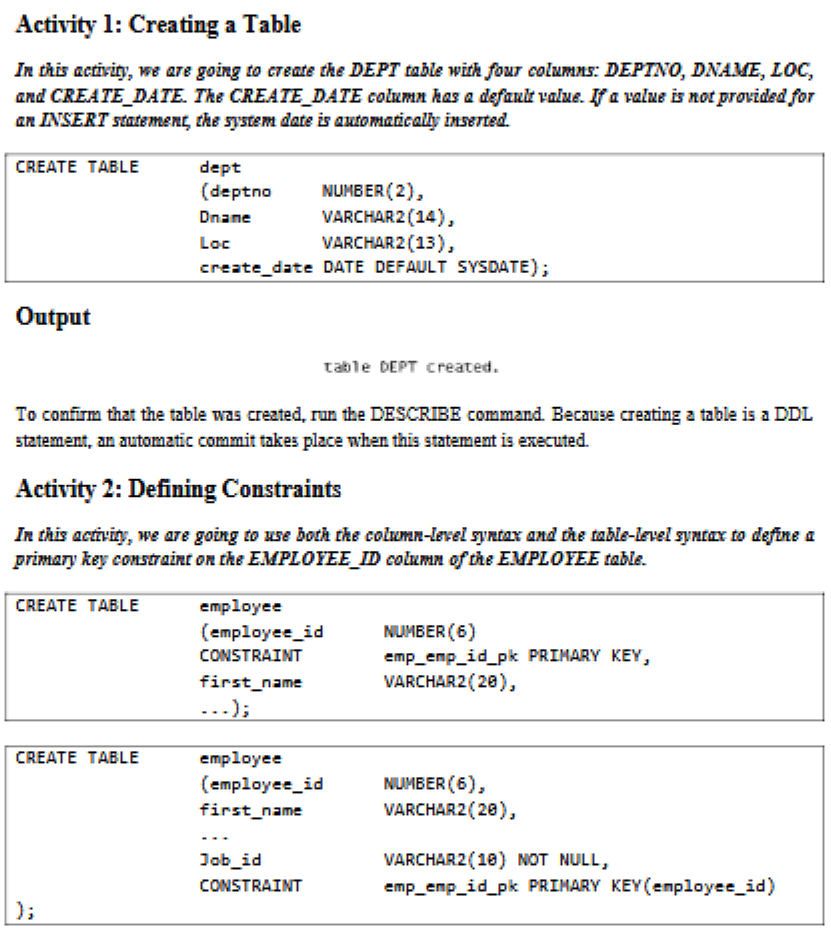
Constraints can be defined at the time of table creation or after the creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

```
CREATE TABLE [schema.]table
             (column datatype [DEFAULT expr] [column_constraint],
             ...
             [table_constraint][,...]);
```

Defining Constraints

You can create constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition, and must refer to the column or columns on which the constraint pertains in a set of parentheses. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint.
NOT NULL constraints must be defined at the column level.
Constraints that apply to more than one column must be defined at the table level.
In the above syntax:

- *schema* is the same as the owner's name
- *table* is the name of the table
- *DEFAULT expr* specifies a default value to be used if a value is omitted in the INSERT statement
- *column* is the name of the column
- *datatype* is the column's data type and length
- *column_constraint* is an integrity constraint as part of the column definition



Activity 1: Creating a Table

In this activity, we are going to create the DEPT table with four columns: DEPTNO, DNAME, LOC, and CREATE_DATE. The CREATE_DATE column has a default value. If a value is not provided for an INSERT statement, the system date is automatically inserted.

```
CREATE TABLE        dept
                    (deptno      NUMBER(2),
                    Dname        VARCHAR2(14),
                    Loc          VARCHAR2(13),
                    create_date  DATE DEFAULT SYSDATE);
```

Output

```
table DEPT created.
```

To confirm that the table was created, run the DESCRIBE command. Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

Activity 2: Defining Constraints

In this activity, we are going to use both the column-level syntax and the table-level syntax to define a primary key constraint on the EMPLOYEE_ID column of the EMPLOYEE table.

```
CREATE TABLE        employee
                    (employee_id      NUMBER(6)
                    CONSTRAINT        emp_emp_id_pk PRIMARY KEY,
                    first_name        VARCHAR2(20),
                    ...);
```

```
CREATE TABLE        employee
                    (employee_id      NUMBER(6),
                    first_name        VARCHAR2(20),
                    ...
                    Job_id            VARCHAR2(10) NOT NULL,
                    CONSTRAINT        emp_emp_id_pk PRIMARY KEY(employee_id)
);
```

Example Table

Consider the following table for storing employee records:

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name VARCHAR(50) NOT NULL,
   Age INT CHECK (Age >= 18),
   Email VARCHAR(100) UNIQUE,
   DepartmentID INT,
   Salary DECIMAL(10,2) DEFAULT 3000.00,
   FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Explanation:

- PRIMARY KEY (EmployeeID): Ensures each employee has a unique ID.
- NOT NULL (Name): Ensures the name is mandatory.
- CHECK (Age >= 18): Ensures only employees aged 18 or older are added.
- UNIQUE (Email): Prevents duplicate emails.
- DEFAULT (Salary = 3000.00): Assigns a default salary if none is provided.
- FOREIGN KEY (DepartmentID): Links the employee to a department.

Lab Tasks

Task 1: Creating Tables with Constraints

1. Create a Departments table with the following constraints:
   - DepartmentID as PRIMARY KEY
   - DepartmentName as UNIQUE and NOT NULL

2. Create an Employees table with all the constraints from the example above.

Task 2: Testing Constraints

1. Attempt to insert an employee without providing a name (should fail due to NOT NULL constraint).
2. Try inserting two employees with the same email (should fail due to UNIQUE constraint).
3. Insert an employee under the age of 18 (should fail due to CHECK constraint).
4. Add an employee without specifying a salary (should default to 3000.00).

Task 3: Foreign Key Integrity Check

1. Try inserting an employee with a DepartmentID that does not exist in the Departments table (should fail due to FOREIGN KEY constraint).

Rubric for Lab Assessment

Criteria	Description	Points
Excellent	Successfully implements all constraints and executes test cases without errors.	4
Good	Implements most constraints correctly with minor issues.	3
Average	Implements some constraints but fails multiple test cases.	2
Needs Improvement	Fails to implement constraints correctly.	1

Instructor Signature: ______________________ Date: ___________