Constraint Statements

Introduction

In the last unit you learned how to create tables with the CREATE TABLE statement, and how to modify those tables using the ALTER TABLE command. We will now discuss how to use these statements to set constraints at the column and table levels. As you recall, constraints are rules that the data in a table must satisfy, and are broken down into five types: primary key, foreign key, unique, check, and not null.

Creating Constraints

Regardless of the type, Oracle requires that each constraint have a name. You can choose to name the constraint yourself, or let Oracle generate the name for you. It is considered good practice to name the constraint yourself, following industry standards, or standards set by the organization. Industry standard is tablename_columnname_constrainttype, where constrainttype is: pk (primary key), fk (foreign key), uk (unique), ck (check), and nn (not null). As an example, to name a primary key constraint for the customer number on the customer table, we would use: customer_cust#_pk.

Setting a constraint at the column level or the table level has the same effect; the only difference is where the constraint is coded on the CREATE TABLE statement. To explicitly name the constraint, the keyword CONSTRAINT must be used; otherwise Oracle will name the constraint for you. The only constraint that must be coded at the column level is the NOT NULL constraint.

To illustrate the above point, let's create a table containing a student number and student name, with the student number being the primary key. The first example will be at the column level, and the second example at the table level. Again, it makes no difference to the database which method you use. Also, more than one constraint can be coded on a column during the create process.

Column level:

```
CREATE TABLE Student
(StudentNum NUMBER (5) CONSTRAINT Student_StudentNum_pk PRIMARY KEY,
 StudentName CHAR (40)
StudentMajCod CHAR (2)
StudentGpa NUMBER (3, 2));
```

Table level:

```
CREATE TABLE Student
(StudentNum NUMBER (5),
 StudentName CHAR (40),
StudentMajCod CHAR (2),
StudentGpa NUMBER (3, 2),
CONSTRAINT Student_StudentNum_pk PRIMARY KEY (StudentNum));
```

To add a constraint to a table after it has been created, we use the ALTER TABLE command. If we've determined that the student name cannot be null, we would code:

```
ALTER TABLE Student
ADD CONSTRAINT Student_StudentName_nn NOT NULL (StudentName);
```

Besides the PRIMARY KEY and NOT NULL constraints, we can add FOREIGN KEY, UNIQUE and CHECK constraints to a table. Recall that a foreign key is a column on a table whose data is required to match the primary key of the table it is related to, also referred to referential integrity. The UNIQUE constraint is used to ensure that a value in a column of a specific row is not repeated for any other row. Keep in mind that UNIQUE allows null values in a column, but enforces uniqueness if a value exists. The CHECK constraint forces the data in a column to meet a condition, such as a certain value or range.

The following examples illustrate the use of the FOREIGN KEY, UNIQUE and CHECK constraints:

Assuming the students major code has to match a major code on the major table:

```
ALTER TABLE Student
ADD CONSTRAINT Student_StudentMajCod_fk FOREIGN KEY (StudentMajCod)
REFERENCES Major (MajCod);
```

Assume no student can have the same exact name (for an example, not practical in real life):

```
ALTER TABLE Student
ADD CONSTRAINT Student_StudentName_uk UNIQUE (StudentName);
```

The student's grade point average must be greater than zero and less than or equal to 4.00:

```
ALTER TABLE Student
ADD CONSTRAINT Student_StudentGpa_ck CHECK (StudentGpa BETWEEN 0.00 AND 4.00);
```

An important point in creating constraints is to keep in mind that if any of the constraints being created results in a potential integrity issue, the DBMS will reject the attempt right away. For instance, trying to create a foreign key constraint but the table being referenced isn't created yet. Or adding a NOT NULL constraint but the table contains a row with a null in that column already.

Viewing, Disabling and Dropping Constraints

Once constraints have been added to a table, it may become necessary to view, disable or drop them altogether. The data dictionary that the DBMS maintains for the database has a view called user_constraints that contains all the constraints for that database. This view can be queried similar to any other table in order to list the constraints for a table. To disable/enable or drop a constraint, we again use the ALTER TABLE command with the DISABLE, ENABLE or DROP keywords. Keep in mind these commands may be rejected if they would result in compromising the data in the database.