SQL Course

DDL : Create Statement

Objectives

- To introduce why and where the SQL create statement is used.
- Describe the syntax of the **CREATE** statement and illustrate its use, using examples.

- Tables are created using the **CREATE TABLE** command.
- Each table consists of at least one column.
- Each column has a:
  - Name.
  - Type.
  - Size (depending on the type).
  - Optional integrity constraints.

A **CREATE TABLE** command creates a table in the sense that it makes the table known to the system; it gives the new table a name and defines its columns. A column is defined by:

- A name
- A data type
- A flag which indicates whether or not the column can accommodate **NULL** values

The table name must be system wide unique. In a system environment which supports many users and many applications this constraint may cause problems.
Some SQL systems solve this problem by allowing a user to choose names that are unique only among the set of tables that they created. The system wide unique name is then obtained by concatenating the name of the table with the name of the user who created the table and separating them with a dot. This is convention is known as **qualified table names**.
Generally the system knows which user is submitting a particular request to the system. Therefore the short name, without the qualifier, is often sufficient, since the system itself can add the qualifier when necessary.
Suppose we wish to specify a table representing an *Airport* with five attributes, namely : an *airport code* (maximum of three characters uniquely identifying the airport e.g. DUB, LHR), an *airport name*, *location*, *country* (each of up to 20 character in length), and *time difference* (number representing difference from GMT, with a default value of 0).

An appropriate table specification in SQL would be:

> **CREATE TABLE** *airport* (*id* CHAR(3) **NOT NULL,** *location* VARCHAR2(20) **NOT NULL,** *country* VARCHAR2(20) **NOT NULL,** *time_difference* NUMBER(2) **DEFAULT 0 NOT NULL, PRIMARY KEY**(*id*))**;**

The general syntax of the **CREATE TABLE** statement is:

> **CREATE TABLE** *tablename* (*column_name* DATA_TYPE [{NULL|NOT NULL}] [**DEFAULT** <*literal*>] [,*column_name* DATA_TYPE[{NULL|NOT NULL}] [**DEFAULT** <*literal*>]] [, ...] [**PRIMARY KEY** (*column_name* [, *column_name*] [, ...])] [**FOREIGN KEY**

(*column_name* [, *column_name*] [, ...])] **REFERENCES***table_name*] [**UNIQUE** (*column_name*[, *column_name*] [, ...])] [**CHECK** (search_condition)])**;**

The SQL2 standard allows several different parts of a column definition, which require the column to contain unique values, to specify that the column is a primary key or a foreign key, or to restrict the data values that a column can contain. The **DEFAULT** value is a value which will automatically be used by the database when a value is not specified by the **INSERT** statement into that particular column.

Note: More complicated examples of table creations which use all of the above features are explained in the **Table Constraints** module.

SQL Course

Table Constraints

- To demonstrate the control that the SQL standard allows over the data that can be entered into the database.

- The tables in a database determine its structure.
- The definition of a database table also includes the relationships among the tables in a database and includes restrictions on the data that can be entered into the database.
- The SQL2 standard allows all types of constraints to be explicitly named, so that they can be manipulated by name after they have been initially defined. The standard also provides explicit control over when constraints are checked, and what happens when an attempt is made to modify the database contents in a way that would violate the constraints.

The database constraints can basically be seen as the control the database has over data added to tables within the database.

A table will **always** include the **Primary Key** constraint.
This constraint enforces that a column or combination of columns be unique, and also that the chosen column or combination of columns be the key for the table.
For example, to ensure that the values of *customer_number* in the *booking* table match one of the values of the *customer_number* column in the *customer* table, the following table specification for *booking* would include:

> **CREATE TABLE** Booking(booking_reference VARCHAR(20) NOT NULL, booking_date DATE NOT NULL, customer_number INTEGER NOT NULL, flight_no VARCHAR(20) NOT NULL, departure_date DATE NOT NULL, seat_class VARCHAR(20) NOT NULL, **PRIMARY KEY**(booking_reference), **FOREIGN KEY**(customer_number) **REFERENCES** customer(customer_number), **FOREIGN KEY**(flight_no, departure_date, seat_class) **REFERENCES** fare(flight_no, departure_date, seat_class)**;**

A table can contain constraints which restrict the allowed values of an attribute in the table.
For example, to ensure that the values in the *movie_type* column are 'Horror', 'Action' or 'Other', the following table specification for *Movie_Titles* would be:

> **CREATE TABLE** Movie_Titles(title CHAR(30) NOT NULL, movie_type CHAR(10), **PRIMARY KEY**(title), CONSTRAINT check_movie_type **CHECK**(movie_type IN ('Horror','Action','Other')))**;**

Uniqueness
: This constraint forces the data in some column or combination of columns to be unique for every row in a table.

Primary Key
: This constraint enforces the same uniqueness constraint but also designates a column or combination of columns as the key for the table.

Foreign Key
: This constraint forces the value of a column or combination of columns to match the value of a primary key in some other table.

When specifying a foreign key we must also specify the table(s) in which that foreign key can be found.

Check

A check constraint is a constraint that restricts the contents of a particular table. In the SQL2 standard, a check constraint is specified as a search condition and it appears as part of the table definition.

The check constraint is not limited to using the **in** keyword. Other keywords include **like**, **between** etc... See SQL Implementation Manual for more details.

Assertions

An assertion is a database constraint that restricts the contents of the database as a whole. Like a check constraint, an assertion is specified as a search condition. But unlike a check constraint, the search condition in an assertion can restrict the contents of multiple tables and the data relationships among them. For that reason, an assertion is specified as part of the overall database definition, via a SQL2 **CREATE ASSERTION** statement.

Domain Definition

The contents of a particular column in a relational database table are restricted to the data type specified in the **CREATE TABLE** statement when the table is first created. In a real world situation the values which would appear in a column of a table would be even more restricted. The SQL2 standard implements the formal concept of a domain as a part of a database definition. Under SQL, a domain is a named collection of data values that effectively function as an additional data type, for use in database definition. A domain is created with a **CREATE DOMAIN** statement. Once created, the domain can be referenced as if it were a data type within a table definition.

The following example illustrates the use of the **Primary Key** and **Foreign Key** constraints in the **CREATE TABLE** statement.

The combination of attributes *call_sign*, *flight_no* and *departure_date* are chosen to be unique and the key for the *Aircraft_Flight* table.

For the *Aircraft_Flight* table, attribute *call_sign* will match a value from the *call_sign* column in the *Aircraft* table.

Also, the combination of attributes *flight_no* and *departure_date* will match a value from the combination of *flight_no* and *departure_date* columns in the *Flight* table.

CREATE TABLE Aircraft_Flight(call_sign VARCHAR(20) NOT NULL, flight_no VARCHAR(20) NOT NULL, departure_time DATE NOT NULL, departure_date DATE NOT NULL, **PRIMARY KEY**(call_sign, flight_no, departure_date), **FOREIGN KEY**(call_sign) **REFERENCES** aircraft(call_sign)**;**

The following example also illustrates the use of the **Primary Key** and **Check** constraints in the **CREATE TABLE** statement.

Attribute *call_sign* is chosen to be unique and the key for the *Aircraft* table.

Attribute *no_club_seats* will have to be in the inclusive range **5** - **50** for the *Aircraft* table.

CREATE TABLE Aircraft(call_sign VARCHAR(20) NOT NULL, aircraft_name VARCHAR(20), model VARCHAR(20) NOT NULL, no_club_seats INTEGER NOT NULL, CONSTRAINT check_no_club_seats **CHECK** (no_club_seats between 5 and 50), no_economy_seats INTEGER NOT NULL, **PRIMARY KEY**(call_sign))**;**

SQL Course

DDL : Alter and Drop Statements

- To demonstrate how to modify a table structure.
- To demonstrate how to remove a table.
- To advice on good database development practice.

It is possible to change the format of a table (enlarge a column or add a column) using the **ALTER TABLE** statement.
Suppose we have a table S, which has S# defined as 4 characters long, and part_description defined as 20 characters long.
S(S#,part_description)
To increase the maximum width of a CHAR or NUMBER field, use the **ALTER TABLE** statement with a **MODIFY** clause:

ALTER TABLE *S* **MODIFY** *S#*(6)**;**

To also permit null values in this column add NULL to the end of the column specification:

MODIFY *S#*(6) NULL**;**

To add a new column to a table, use the **ALTER TABLE** statement with an **ADD** clause:

ALTER TABLE *S* **ADD** *(supplier DATE)***;**

The **DROP TABLE** statement is used to remove tables from the database which are no longer useful.
For example, to remove the table called *Airport*, the SQL command would be:

DROP TABLE *Airport***;**