SIT103 Database and Information Retrieval

Practical 4: More on Table Manipulations

Objectives:

- To learn more about creating, populating, altering, and deleting tables
- To learn how to add constraints to tables

---

1. Introduction

Up to now we have learnt some commonly used SQL commands from the previous practicals. Since creating proper tables in a database has a great impact on the performance of data management and information retrieval, it is necessary to take a more professional way to create tables in a database, and manipulate tables and data more effectively. On the other hand, more practice on skilfully using SQL commands for a range of scenarios will be helpful to better understanding the importance of database design for real applications.

In a database, data is stored in tables (relations). In previous practical sessions, we have learnt how to write queries to retrieve data from *existing* tables by using the SELECT statement. In this session, we will systematically learn how to create tables and insert values into them, how to alter, update and delete tables using SQL. In addition, we will learn how to add constraints to tables to increase the data integrity.

We start this session with the introduction of data types. You need to know different data types before you can use the CREATE TABLE command to create tables.

2. Commonly Used Data Types in Oracle

The commonly used data types in SQL are listed below for your reference. For more information about data types, please refer to our textbook.

Data Type	Data Type Name	Description
Numerical (Note: *precision* is the total number of digits stored)	NUMBER (*p*,*s*)	Store numbers up to 38 decimal places. *p* is the precision, *s* is the number of digits to the right of the decimal point. The decimal point is not counted in the precision.
	NUMBER	Decimal number that has a variable number of decimal places. The decimal point may appear after any number of digits, and it may not appear at all.
	INTEGER (INT)	Whole number without any decimal part. Store numbers from -2^{31} to $2^{31} - 1$ (use 4 bytes).
	REAL	Has a precision of 7 digits (use 4 bytes).
	FLOAT	Has a precision of 15 digits (use 8 bytes).

	DECIMAL	Storage size varies based on the specified precision (use 2-17 bytes).
	DOUBLE PRECISION	Floating-point number.
Character	CHAR	CHAR(n) is a character data type to store fixed-length (n) alphanumerical data. The maximum allowable size is 2,000 characters.
	VARCHAR2	VARCHAR2(n) is a character data type to store variable-length alphanumerical data. The maximum allowable size is 4,000 characters.
	NCHAR	Similar to CHAR but uses 2-byte binary encoding for each character.
	LONG	The LONG type is used for variable-length character data up to 2 gigabytes. There can be only one LONG-type column in a table.
Date and Time	DATE	Store date and time values. The default date format is DD-MON-YYYY, where DD indicates the day of the month, MON represents the month's first three letters (capitalized), and YYYY represents the year.

Some general rules that you can follow to determine which data type to use to define a table column:

- Use the smallest possible column sizes.
- Use the smallest possible data type for a column that will hold your data. For example, if you are going to be storing numbers from 1 to 99 in a column, it would be better off selecting NUMBER(2) instead of the INT data type.
- For numerical data, it is better to use a numerical data type such as INT, in stead of using VARCHAR2 or CHAR.
- FLOATs or REALs should not be used to define primary keys. Integer data type (INT) can be used for primary key.
- Avoid selecting the fixed-length type CHAR, if your column will have a lot of nulls. VARCHAR2 is better.
- If you are going to be using a column for frequent sorts, consider an integer-based column rather than a character-based column.

3. Creating a Table

In SQL, the CREATE TABLE command is used to create a table. The general syntax of the CREATE TABLE statement is:

CREATE TABLE *Tablename*
(*column_name1 type1*, *column_name2 type2*, …);

Connect to and open your Oracle database, type and execute the following commands at the SQL> prompt:

```
DROP TABLE Employee;

CREATE TABLE Employee (names     VARCHAR2(20),
                       address   VARCHAR2(20),
                       employee_number INT,
                       salary    INT);
```

This CREATE TABLE query created a table called `Employee` with four columns: `names`, `address`, `employee_number`, and `salary`. The data type of `names` and `address` is VARCHAR2 (variable-length character) with a maximum length of 20 characters. The data type of `employee_number` and `salary` is INT (INTEGER).

Use the same way, type and execute the following to create another table called `Names` in your database:

```
CREATE TABLE Names (fullname     VARCHAR2(20))
```

This table has only one column `fullname`.

To view the tables `Employee` and `Names` you just created, you can execute the following command at the SQL> prompt:

```
SELECT table_name FROM user_tables;
```

To view the table structure of these two tables you just created, you can execute the following commands at the SQL> prompt:

```
DESCRIBE Employee
DESCRIBE Name
```

4. Inserting Values into a Table

There are several ways to insert values into a table using SQL in Oracle. We will illustrate the two most commonly used ways: using INSERT INTO … VALUES and using INSERT INTO … SELECT.

Using INSERT INTO … VALUES

The general syntax of this command is:

```
INSERT INTO Tablename
VALUES ('character_attribute_value', numerical_attribute_value, …);
```

Try to execute the following commands:

```
INSERT INTO Names
VALUES ('Joe Smith');

INSERT INTO Names
VALUES ('David Jones');
```

Then execute the following and check the results:

```
SELECT * FROM Names;
```

You can see two names are stored in the table `Names`.

If you created a table with *n* columns, you usually would have *n* values in the INSERT INTO … VALUES statement, in the **order** of the definition of the columns in the table with the **matched data types**.

Try to execute the following commands and check the results:

```
INSERT INTO Employee
VALUES (‘Joe Smith’, ‘123 4th St.’, 101, 2500);

SELECT * FROM Employee;
```

Try to execute the followings one by one to see what happens:

```
INSERT INTO Employee
VALUES (‘Joe Smith’, ‘123 4th St.’);

INSERT INTO Employee
VALUES (2500,‘Joe Smith’, 101,‘123 4th St.’);
```

You may INSERT a row with less than all the columns by naming the columns you want to insert into. You can also define the column order by which the data is inserted into. Try the followings:

```
INSERT INTO Employee (names, address)
VALUES (‘Joe Smith’, ‘123 4th St.’);

INSERT INTO Employee (salary,names,employee_number,address)
VALUES (2500,‘Joe Smith’, 101,‘123 4th St.’);

SELECT * FROM Employee;
```

You may actually include the keyword, null, if some values for some columns were unknown (Note: these columns must allow null values in the table definition). Try the following:

```
INSERT INTO Employee
VALUES (‘Ben Howard’, null, 102, null);

SELECT * FROM Employee;
```

To delete all the rows in a table, the command syntax is:

`DELETE FROM` ***Tablename;***

Try the followings and check the results:

```
DELETE FROM Employee;

DELETE FROM Names;

SELECT * FROM Employee;

SELECT * FROM Names;
```

Now insert the following data into the table *Employee* using the INSERT INTO … VALUES command.

names	address	employee_number	salary
Joe Smith	123 4th St.	101	2500.00
David Jones	27 Shillingford Rd	103	3300.00
Sumit White	95 Oxford Rd	105	1200.00
Joya Das	23 Springfield Cr	114	2290.00
Terry Livingstone	465 Easter Ave	95	3309.00

Using INSERT INTO … SELECT

With the INSERT INTO … VALUES option, you insert only one row at a time into a table. With the INSERT INTO … SELECT option, you may (and usually do) insert many rows into a table at one time.

The general syntax is:

```
INSERT INTO target_table (column1, column2, …)
"SELECT clause";
```

Try the followings and check the results:

```
SELECT * FROM Names;

INSERT INTO Names(fullname)
SELECT names
FROM Employee;

SELECT * FROM Names;
```

You can see all the names in the `Employee` table are copied into the `Names` table.

You could restrict the INSERT INTO … SELECT with a WHERE clause. Try the followings and check the results:

```
DELETE FROM Names;

SELECT * FROM Names;

INSERT INTO Names(fullname)
SELECT names
FROM Employee
WHERE salary > 2600;

SELECT * FROM Names;
```

As with the INSERT INTO … VALUES option, if you create a table with *n* columns, you usually would have *n* values in the INSERT INTO … SELECT option in the order of the table definition, or you would have to name the columns you are inserting.

Now create another table called `Emp1` with three columns

```
Emp1(addr, sal, empno)
```

The columns, addr, sal, empno, stand for address, salary and employee number respectively (which SQL command should we use?).

After the table Emp1 is created, try the followings and check the results:

```
SELECT * FROM Emp1;

INSERT INTO Emp1(addr, sal, empno)
SELECT address, salary, employee_number
FROM Employee;

SELECT * FROM Emp1;
```

If we created a table called Emp2 with the identical columns as Emp1, we can use the following command to create a backup table Emp2 of the table Emp1:

```
INSERT INTO Emp2
SELECT *
FROM Emp1;
```

Caution: An erroneous INSERT INTO … SELECT could succeed if the data types of the SELECT match the data types of the columns in the table to which you are inserting.

Try the followings and check the results:

```
DELETE FROM Emp1;

SELECT * FROM Emp1;

INSERT INTO Emp1(addr, sal, empno)
SELECT address, employee_number, salary
FROM Employee;

SELECT * FROM Emp1;
```

You will see the wrong information in the table Emp1.

As you might have already guessed from the INSERT INTO … VALUES command, you do not have to insert the whole row with an INSERT INTO … SELECT. You may load fewer columns. Try the followings and check the results:

```
DELETE FROM Emp1;

SELECT * FROM Emp1;

INSERT INTO Emp1(addr, sal,)
SELECT address, salary
FROM Employee;

SELECT * FROM Emp1;
```

In conclusion, you must be careful with the INSERT INTO … SELECT command, because you almost always insert multiple rows, and if types match, the insert will take place whether it makes sense or not.

5. The UPDATE Command

This command is used for setting/changing data values in a table. The general syntax of this command is:

```
UPDATE Tablename
SET fieldname …
[WHERE conditions];
```

Try the followings and see what happens in the `salary` column:

```
SELECT * FROM Employee;

UPDATE Employee
SET salary = 0
WHERE employee_number = 101;

SELECT * FROM Employee;
```

If you want to set all salaries in the table `Emp1` to zero, you may do so with the following UPDATE command:

```
UPDATE Emp1
SET sal = 0;
```

Check the data in the table `Emp1`:

```
SELECT * FROM Emp1;
```

6. The ALTER TABLE Command

ALTER TABLE commands are known as data definition (DDL) commands, because they change the definition of a table.

Adding a Column to a Table
You may add columns to a table with little difficulty. The general syntax is:

```
ALTER TABLE Tablename
ADD column_name type;
```

Try to execute the following command, and then check the table definition on your database screen to see if a new column `bonus` is added to the table `Employee`:

```
ALTER TABLE Employee
ADD bonus INT;
```

Changing a Column's Data Type in a Table
You can change a column's data type with existing data in it, provided that the new column data type will accommodate the exiting data. The syntax is:

```
ALTER TABLE Tablename
MODIFY column_name new_type;
```

Try the following and then check the definition of table Employee, especially the definition of the column bonus:

```
ALTER TABLE Employee
MODIFY bonus FLOAT;
```

Changing a Column's Length in a Table
You may want to change the size of a column in a table. You typically make a column larger, and the Oracle will not have a problem with that, because larger columns will accommodate existing data. But, if you want to make a column smaller, sometimes Oracle will let you do it and other times it will not.

The Oracle allows you to reduce the length of your column without any problems when you do not have any data in that column yet (it's all NULL).
If you try to reduce the column size to a size where you would be cutting off some of the data, the Oracle will give you an error and will not let you do it.

Try the followings, see what happens and check the definition of the table Employee:

```
ALTER TABLE Employee
MODIFY names VARCHAR2(5);

ALTER TABLE Employee
MODIFY names VARCHAR(19);
```

Deleting a Column from a Table
The syntax is as follow:

```
ALTER TABLE Tablename
DROP COLUMN column_name;
```

The DROP COLUMN command will also delete a column even if there is data in it. Try the following command and check the definition of the table Employee:

```
ALTER TABLE Employee
DROP COLUMN bonus;
```

7. The DELETE Command

In previous sections, we saw that the DELETE command can be used to remove all rows of a table. Actually, the syntax of the DELETE command is:

```
DELETE FROM Tablename
WHERE condition(s);
```

The condition(s) determines which rows of the table will be deleted. As you saw earlier, if no WHERE condition is used, all the rows of the table will be deleted.

Try the followings and see what happened:

```
DELETE FROM Employee
WHERE salary < 1500;

SELECT * FROM Employee;
```

Deleting a Table
The general syntax to delete an entire table, its definition and contents is:

```
DROP TABLE Tablename
```

Try the following and check the tables in your database:

```
DROP TABLE Names
```

8. Constraints on Tables

Constraints are added to give tables more integrity. This section will introduce some of the constraints available in Oracle.

The NOT NULL Constraint
The NOT NULL constraint is an integrity constraint that allows the database creator to deny the creation of a row where a column would have a null value. To deny nulls, we can create a table with the NOT NULL constraint on a column(s) after the data type.

Try the following command:

```
CREATE TABLE Test2
(name VARCHAR2(20),
 ssn CHAR(9),
 dept_number INT NOT NULL,
 acct_balance INT);
```

In this newly created table `Test2`, the `dept_number` column now has a NOT NULL constraint included. You can see this constraint in the table definition if you execute the command

```
DESCRIBE Test2
```

The NOT NULL constraint can also be added to the column after the table has been created. You can use SQL to do this.

Suppose we created the `Test2` table as follows (without NOT NULL constraint):

```
DROP TABLE Test2;

CREATE TABLE Test2
(name VARCHAR2(20),
 ssn CHAR(9),
 dept_number INT,
 acct_balance INT);
```

To add NOT NULL constraint to the column `dept_number`, we use the MODIFY option within the ALTER TABLE statement. Try the following:

```
ALTER TABLE Test2
MODIFY dept_number INT NOT NULL;

DESCRIBE Test2
```

It can be seen that we get the same table definition as we got before.

NOTE: You cannot put a NOT NULL constraint on a column that already contains nulls.

The PRIMARY KEY Constraint
When creating a table, a PRIMARY KEY constraint will prevent duplicate values for the column(s) defined as a primary key.

Designation of a primary key will be necessary for the referential integrity constraints that follow. The designation of a primary key also automatically puts the NOT NULL constraint in the definition of the column(s). A fundamental rule of relational database is that primary keys cannot be null.

One of the following three options can be used to set the primary key.

Option 1 The first option is to declare the primary key while creating the table in the CRETAE TABLE statement. Example:

```
CREATE TABLE Test2a
(name VARCHAR2(20),
 ssn CHAR(9) CONSTRAINT ssn_pk PRIMARY KEY,
 dept_number INT,
 acct_balance INT);
```

ssn_pk is the name of the PRIMARY KEY constraint for the ssn column. You can give another name as you like. It is conventional to name all CONSTRIANTs.

Option 2 The second option set the primary key constraint(s) after the columns are declared. It provides greater flexibility. Example:

```
CREATE TABLE Test2b
(name VARCHAR2(20),
 ssn CHAR(9),
 dept_number INT,
 acct_balance INT,
 CONSTRAINT ssn_pk1 PRIMARY KEY (ssn));
```

Option 3 The third option is to set the primary key constraint(s) using the ALTER TABLE command. The syntax is

```
ALTER TABLE Tablename
ADD CONSTRAINT constraint_name PRIMARY KEY (column_name(s));
```

Try the following commands and see what happens:

```
CREATE TABLE Test2c
(name VARCHAR2(20),
 ssn CHAR(9),
 dept_number INT,
 acct_balance INT);

DESCRIBE Test2c;

ALTER TABLE Test2c
```

```
ADD CONSTRAINT ssn_pk2 PRIMARY KEY (ssn);
DESCRIBE Test2c;
```

Concatenated Primary Keys

In relational databases, it is sometimes necessary to define more than one column as the primary key. When more than one column makes up a primary key, it is called a *concatenated primary key* or a *combination primary key*.

In Oracle, you can define the concatenated primary key in the following way:

```
CREATE TABLE Test2d
(name VARCHAR2(20),
 ssn CHAR(9),
 dept_number INT,
 acct_balance INT,
 CONSTRAINT ssn_name_pk PRIMARY KEY (ssn, name));
```

Similar to the previous example, you can also use ALTER TABLE command to define the concatenated primary key.

Referential Integrity Constraints

A relational database consists of relations (tables) and relationships between tables. To define a relationship between two tables, we create a referential integrity constraint. A referential integrity constraint is one in which a row is one table (with a foreign key) cannot exist unless a value (column) in that row refers to a primary key value (column) in another table. This is a primary key - foreign key relationship between two tables.

We will use two tables, `Department` and `Employee1`, to demonstrate how to add referential integrity constraints. The structures of these two tables are as follows:

```
Department (Deptno, Deptname)
Employee1 (Empno, Empname, Dept)
```

Where the underlined columns are primary keys in the tables (e.g. `Deptno` is the primary key of the table `Department`). The column `Dept` in the table `Employee1` is the foreign key that refers to the primary key `Deptno` in the table `Department`.

To enable a referential integrity constraint, it is necessary for the column that is being referenced to be firstly defined as a primary key. In our Employee1-Department example, we have to firstly create the Department table with a primary, then the Employee1 table. Try the followings and notice how the referential integrity constraint is defined:

```
CREATE TABLE Department
(deptno INT,
 deptname VARCHAR2(20),
 CONSTRAINT deptno_pk PRIMARY KEY (deptno));


CREATE TABLE Employee1
```

```
(empno INT CONSTRAINT empno_pk PRIMARY KEY,
 empname VARCHAR2(20),
 dept INT CONSTRAINT dept_fk REFERENCES Department(deptno));
```

You can add options to a referential integrity constraint to indicate what actions the Oracle should take when deleting or updating data in the referred tables. Details are as follows.

ON DELETE SET NULL If this option is added, and we try to delete a row from the parent table (in this case, the Department table) that has a referencing row in the dependent table (in this case, the Employee table), then Oracle will set the corresponding values to NULL in the dependent table. This rule is defined using the SQL command as follows:

```
CREATE TABLE Employee1
(empno INT CONSTRAINT empno_pk PRIMARY KEY,
 empname VARCHAR2(20),
 dept INT CONSTRAINT dept_fk REFERENCES Department(deptno)
      ON DELETE SET NULL);
```

ON DELETE CASCADE This option will allow the deletion in the dependent table (in this case, the Employee table) that are affected by the deletions of the rows in the references table (in this table, the Department table). The rule can be defined using the SQL command as follows:

```
CREATE TABLE Employee1
(empno INT CONSTRAINT empno_pk PRIMARY KEY,
 empname VARCHAR2(20),
 dept INT CONSTRAINT dept_fk REFERENCES Department(deptno)
      ON DELETE CASCADE);
```

Viewing Constraint Information

You can view all constraints of a table by using the following command syntax:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'Tablename';
```

Try the following command to view all constraints on table Employee1:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'Employee1';
```

Deleting a Constraint

The syntax is as follow:

```
ALTER TABLE Tablename
DROP CONSTRAINT constraint_name;
```

An example:

```
ALTER TABLE Test2d
DROP CONSTRAINT ssn_name_pk;
```

For other constraints, such as UNIQUE and CHECK constraints, please refer to the textbook and other SQL reference books.

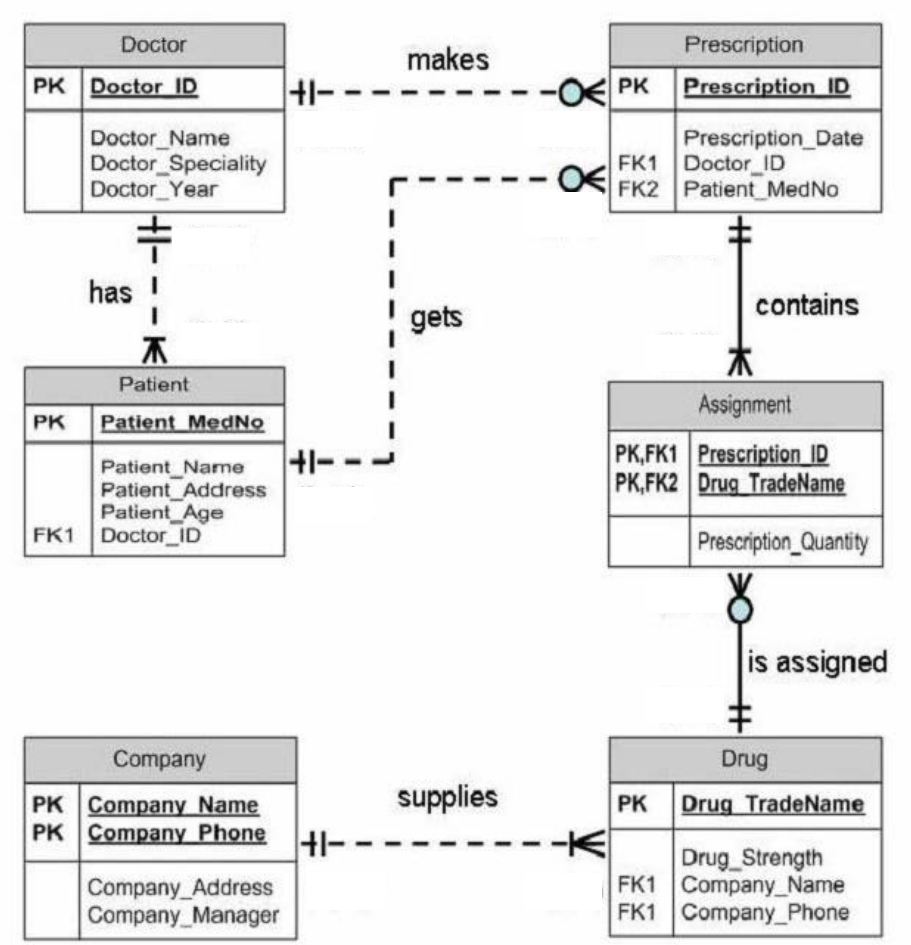
9. Practice

You are given the following table schemes:

DOCTOR (DoctorID, Name, Specialty, Year)
PATIENT (PatientMedNo, Name, Address, Age, DoctorID)
PRESCRIPTION (PrescriptionID, Date, DoctorID, PatientMedNo)
ASSIGNMENT (PrescriptionID, DrugTradeName, PrescriptionQuantity)
DRUG (DrugTradeName, Strength, CompanyName, CompanyPhone)
COMPANY (CompanyName, CompanyPhone, Address, CompanyMgr)

The underlined column(s) is the primary key of the table. These table structures also define primary-foreign referential integrity constraints for the tables.

The relationships between these tables via PKs and FKs are described in the following Entity-Relationship (ER) diagram (we are going to learn ER diagram later, but I am sure now you can understand it without big problems).



Things To Do:

You are required to create the above tables with primary and foreign key constraints in your database, and populate these tables with sample data. During the population, please pay attention to how the constraints work to guarantee the data integrity in the database.
