More SQL & Introduction to PL/SQL

TABLESPACE

The CREATE TABLESPACE statement is used to create a tablespace, which is an allocation of space in the database that can contain schema objects.

- A permanent tablespace contains persistent schema objects. Objects in permanent tablespaces are stored in datafiles.
- A temporary tablespace contains schema objects only for the duration of a session. Objects in temporary tablespaces are stored in tempfiles.

When you create a tablespace, it is initially a read/write tablespace. You can subsequently use the ALTER TABLESPACE statement to take the tablespace offline or online, add datafiles or tempfiles to it, or make it a read-only tablespace.

You can also drop a tablespace from the database with the DROP TABLESPACE statement.

E.g.

Table Space Name: CSE781TS
Description: This tablespace is created with the intention of storing the entire CSE781 oracle project separately.

Script:
/* Creating TableSpace */
CREATE TABLESPACE "CSE781TS"
DATAFILE 'C:\ORACLE\PRODUCT\10.1.0\ORADATA\CSE781\CSE781TS.dat'
SIZE 40M;
SEQUENCES

Sequences are useful database object that allows you to create a series of sequence numbers (integers only) automatically. These numbers are oftentimes used to create values for primary keys when you don’t have an alternate for a user entered primary key value.

Syntax:

```
CREATE SEQUENCE sequence_name
  Start with <number>
  Increment by <number>
  Maxvalue <number>
  Nomaxvalue
  Minvalue <number>
  Nominvalue
  Cycle
  NoCycle
  Cache <number>
  NoCache
  Order
  NoOrder;
```

All of the clauses are optional and if excluded will follow the Oracle 10g default settings identified below:

- Start with - identifies the starting value of the sequence. The default is 1.
- Increment by - identifies the value that you want each sequence number to be adjusted by. The default is 1.
- Maxvalue - the terminal value you wish to use to stop your sequence from producing additional sequence numbers. The default is Nomaxvalue which is the largest value allowed, 10 to the 27th power-1, essentially a very big number.
- Minvalue - also a terminal value if you want to assign sequence numbers in descending sequence. The default is Nominvalue.
- Cycle - allows sequence numbers to be used if the sequence reaches its maximum value (should not be used in case of unique constraints). The default is Nocycle. With Nocycle, if you attempt to generate a sequence number after you reach your maximum value you get an error message.
- Cache - specifies to Oracle to generate a number of sequence numbers ahead, stored in the SGA buffer cache area and are to be used at a later time. This is primarily used to improve performance. The default value is 20. If you specify NoCache then no sequence numbers will be generated in advance and stored in cache.
- Order - specifies that sequence numbers are to be allocated in the exact chronological order in which they were requested. NoOrder is the default; but ensures that applications or users get a unique sequence number but necessarily in chronological order.
To use a sequence you need to use one of two pseudo columns: currval and nextval.

- **Currval** - returns the current value of the sequence
- **Nextval** - the first time used Nextval returns the initial value then in all subsequent uses return the sum of the current value of the sequence plus the value as indicated by the Increment By option.

E.g. INSERT INTO OrderInfo VALUES (OrderIDSeq.NEXTVAL);

**ABSTRACT DATA TYPE**

Suppose a company may have various tables that all require attributes that define a person. All persons have attributes in common such as last name, first name, data of birth, and gender. A company would probably have employees, customers and applicants. Each of these entities would require a table and each table would also require similar attributes. Now, it is certainly possible to define each attribute for each table. However, it would be easier to define a datatype called Person and then use Person as the datatype for each attribute that defined a person. This is accomplished by creating an Abstract Data Type (ADT). The advantage here is that you define the Person attributes once and then use it over and over. If modifications are required (perhaps you want to change the number of characters that are define the last name), they are made in one place, in the ADT, and applied to all.

The format of an ADT is:

```
CREATE TYPE MyDataType AS OBJECT
(
    variable1  dataType,
    variable2  dataType
);
```

**E.g.** The Person datatype described above might be:

```
CREATE TYPE PersonType AS OBJECT
(
    lastName           VARCHAR(30),
    firstName          VARCHAR(30),
    MI                     CHAR
);
```
CREATE TABLE Emp
(
    empId         INTEGER,
    empName       PersonType,
    empSalary     DECIMAL(9,2)  NOT NULL,
    empTax        DECIMAL(5,2)  NOT NULL,
    CONSTRAINT empPK PRIMARY KEY (empId)
);
PL/SQL

PL/SQL (Procedural Language/Structured Query Language) is Oracle Corporation's procedural extension language for SQL and the Oracle relational database.

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other. Typically, each block performs a logical action in the program. A block has the following structure:

DECLARE

/* Declarative section: variables, types, and local subprograms. */

BEGIN

/* Executable section: procedural and SQL statements go here. */

/* This is the only section of the block that is required. */

EXCEPTION

/* Exception handling section: error handling statements go here. */

END;

Only the executable section is required. The other sections are optional. The only SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE, DELETE and several other data manipulation statements plus some transaction control. However, the SELECT statement has a special form in which a single tuple is placed in variables; more on this later.

Data definition statements like CREATE, DROP, or ALTER are not allowed. The executable section also contains constructs such as assignments, branches, loops, procedure calls, and triggers.

PL/SQL is not case sensitive. C style comments (/* ... */) may be used.

As with Oracle SQL programs, we can invoke a PL/SQL program either by typing it in sqlplus editor or by putting the code in a file and invoking the file in the various ways to be taught during lab sessions.
FUNCTIONS

A function is a reusable program unit that performs work via executable logic. It can accept multiple input parameters, and must return a single output value.

The syntax of a function consists of a header and a body. The header encompasses everything from the CREATE up to the AS, the body entails everything from the AS to the END.

- CREATE FUNCTION

```
CREATE FUNCTION
CREATE OR REPLACE FUNCTION
Function_name
(Parameter1_name mode datatype,
Parameter2_name mode datatype)
Return datatype
AS
    returnVar datatype;
BEGIN
    /* I need to return a value */
    RETURN returnVar;
END;
```

You may have desired no. of parameters as needed.

Mode is IN for input. Each parameter requires a datatype.

Identify the datatype of the variable being returned.

E.g.

A function to calculate total salary for all of the employees in the EMP table is given as follows:
/* Function to calculate a total salary */

CREATE OR REPLACE FUNCTION CalcTotalSalaryFN

(  
salaryIN IN DECIMAL,
  taxIN IN DECIMAL
)

RETURN FLOAT

IS
  totalSalVar DECIMAL(10,2) :=0;  -- initialize variable to 0
BEGIN
  totalSalVar := salaryIN - salaryIN * taxIN /100;
  RETURN totalSalVar;
END;
/

To execute this function, you can embed it in SQL just as you would an Oracle supplied (predefined) functions.

E.g.

SELECT empId, empName, CalcTotalSalaryFN(empSalary, empTax) FROM Emp;
CONTROL STRUCTURES

Decision Control

The PL/SQL commands used to make decisions, that is, alter the order in which commands are executed are the **IF/THEN, IF/THEN/ELSE and the IF/THEN/ELSIF.**

- **IF/THEN** format

  **IF** condition **THEN**

  Commands that you want to execute if the condition is true;

  **END IF**;

  The IF portion of the command tests the condition. If the condition is true, control is transferred to the command that immediately follows the **THEN**. If the condition is false, control is transferred to the command following the **END IF**.

  [Note that /*.. */ is multiline comment and “—” is single line comment, that they are ignored when the code is compiled or executed]

  Here is an example of the IF/THEN command, which sends 0 or 1 whether a person can vote or not:

  ```plsql
  CREATE OR REPLACE FUNCTION findEligibilityFN
  (  
    ageIN IN INT
  )  
  --return type is INT
  RETURN INT
  IS
      voteVAR  INT := 0;  -- by default a person cannot vote
  BEGIN
      IF ageIN > 18 THEN
          voteVAR := 1;
      END IF;
      RETURN voteVAR;
  END;
  /
  ```
IF/THEN/ELSE format

**IF** condition **THEN**

Commands that you want to execute if the condition is true;

**ELSE**

Commands that you want to execute if the condition is false;

**END IF**;

The IF portion of the command tests the condition. If the condition is true, control is transferred to the command that immediately following the THEN. If the condition is false, control is transferred to the command following the ELSE.

Here is an example of the IF/THEN/ELSE command:

CREATE OR REPLACE FUNCTION votingEligibilityFN
(
    ageIN IN INT
)
--return type is INT
RETURN INT
IS
    voteVAR INT;

```sql
CREATE OR REPLACE FUNCTION findEligibilityFN
(
    ageIN IN INT
)
--return type is INT
RETURN INT
IS
    voteVAR INT;
BEGIN
    IF ageIN > 18 THEN
        voteVAR := 1;
    END IF;
    RETURN voteVAR;
END;
/

Function created.

SQL> select findEligibilityFN(27) from dual;

FINDELIGIBILITYFN(27)
----------------------
   1

SQL> |
```
BEGIN
    IF ageIN > 18 THEN
        voteVAR := 1;
    ELSE
        voteVAR := 0;
    END IF;
    RETURN voteVAR;
END;
/

Oracle SQL*Plus

SQL> CREATE OR REPLACE FUNCTION votingEligibilityFN
        (ageIN IN INT)
        RETURN INT IS
        voteVAR INT;
    BEGIN
        IF ageIN > 18 THEN
            voteVAR := 1;
        ELSE
            voteVAR := 0;
        END IF;
        RETURN voteVAR;
    END;
/

Function created.

SQL> select votingEligibilityFN(3) from dual;

VOTINGELIGIBILITYFN(3)
---------------------
0

SQL>