lab1

Sunday, April 22, 2018 2:27 AM

lab1

Lab1

Objectives:

At the end of this lab, you should be able to

- Specify fields and columns of a table.
- To write conditional SQL query statement.

Data represents known facts that can be recorded and that have implicit meaning.
For example:

_____

_____

_____

Database is a collection of related data. Example of database

_____

_____

_____

Database management system (DBMS) is a collection of programs that enables users to create and maintain a databases. Example of DBMS

_____

_____

_____

Manipulating a database includes :
querying the database to retrieve specific data, adding, deleting, or updating.

People who works with database systems:
1-Database administrator (DBA).
2-Database designers
3-Application programmer.
4-End users.

Tables: every database composed of one or more tables, which store the database's data/information. Each table has its own unique name and consists of columns and rows. The columns called fields and the rows called records. A table has a specified number of columns, but can have any number of rows.

Record

A record is the collection of values for all the fields pertaining to one entity: i.e. a person, product, company, transaction, etc.

Field :

A field is an area (within a record) reserved for a specific piece of data. Examples: customer number, customer name, street address,
Fields are defined by:

- Field name
- Data type:Character, Numeric,Date, Logical, ..etc
- Field size

Example: assume you want to save data about students in school, specify some of possible fields that can be in this table, data type, size of each field.
Give two examples of records for that table.

Oracle: is an relational database management system (RDBMS) that provides database tools for storing and managing data. It also provides advanced tools to manage all types of data in web sites.

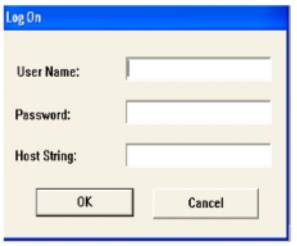
SQL is a command language for communication with the Oracle Server from any tool or application. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new statement.

SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle Server for execution and contains its own command language. Statements can be executed from the SQL prompt or from a script file.

Logging into SQL*Plus:

To log in through a Windows environment:

1. Click Start—>Programs—>Oracle home92—>Application Development-- > SQL Plus.
2. Fill in username**(scott)**, password(**tiger**) , and database.



Prepared by : Dr. Saleh Alhazbi

Writing SQL Statements

- Within SQL*Plus, a SQL statement is entered at the SQL prompt, and the subsequent lines are numbered. This is called the *SQL buffer*. Only one statement can be current at any time within the buffer.
- SQL statements are not case sensitive, unless indicated.
- SQL statements can be entered on one or many lines.

Tables used in this course:

1- **EMP** table: which gives details of all the employees.
2- **DEPT** table: which gives details of all the departments.
3- **SALGRADE**: which gives details of salaries for various grades.

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	1400		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1800	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1450	500	30
7566	JONES	MANAGER	7839	02-APR-81	3175		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1450	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	3050		30
7782	CLARK	MANAGER	7566	09-JUN-81	2650		10
7788	SCOTT	ANALYST		19-APR-87	3200		20
7839	KING	PRESIDENT	7788	17-NOV-81	5200		10
7876	ADAMS	CLERK	7698	23-MAY-87	1300		20
7900	JAMES	CLERK	7566	03-DEC-81	1150	300	30
7902	FORD	ANALYST	7566	03-DEC-81	3200		20
7934	MILLER	CLERK	7782	23-JAN-82	1500		10

Displaying Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table.

DESC[RIBE] ***tablename***

Example :

```
SQL> DESCRIBE dept;

Name           Null?    Type
-------------- -------- ------------
DEPTNO         NOT NULL NUMBER(2)
DNAME                   VARCHAR2(14)
LOC                     VARCHAR2(13)
```

NULL : indicates whether a column *must* contain data;
NOT NULL: indicates that a column must contain data

The data types are described in the following table:

Datatype	Description
NUMBER(*p*,*s*)	Number value having a maximum number of digits *p*, the number of digits to the right of the decimal point *s*
VARCHAR2(*s*)	Variable-length character value of maximum size *s*
DATE	Date and time value between January 1, 4712 B.C. and December 31, 9999 A.D.
CHAR(*s*)	Fixed-length character value of size *s*

Basic SELECT Statement

```
SELECT [DISTINCT]{*, column] alias[,...}
FROM   table;
```

Examples:

```
SQL> SELECT *
  2  FROM   dept;

SQL> SELECT deptno, loc
  2  FROM   dept;
```

Using Arithmetic Operators

```
SQL> SELECT ename, sal, sal+300
  2  FROM emp;
```

Exercise: Write SQL statement to display employee name and total of his income(salary +commission)

_____

_____

_____

What do you notice ??!!!

_____

Defining a Null Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A null is not the same as zero or a blank space.

```
SQL> SELECT ename, job, sal, comm
  2  FROM   emp;
```

Note: *some records do not have values in comm field.*

Duplicate Rows

```
SQL> SELECT deptno  FROM   emp;
```

Eliminating Duplicate Rows

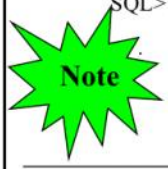
```
SQL> SELECT DISTINCT deptno  FROM   emp;
```

Limiting Rows Selected

SELECT	**[DISTINCT]** {*| *column* [*alias*], ...}
FROM	*table*
[WHERE	*condition(s)*];

Example:

```
SQL> SELECT        ename, job, deptno  FROM    emp  WHERE        ename ='JAMES' ;
```



- Character strings and date values are enclosed in single quotation marks.
- Character values are case sensitive and date values are format sensitive.
- The default date format is DD-MON-YY.

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Examples:

```
SQL> SELECT        ename, sal FROM      emp
  WHERE       sal BETWEEN 1000 AND 1500;

SQL> SELECT        empno, ename, sal, mgr  FROM       emp
  WHERE       mgr IN (7902, 7566, 7788);

SQL> SELECT        ename  FROM          emp
  WHERE       ename LIKE 'S%';

SQL> SELECT  ename, mgr  FROM    emp
  WHERE   mgr IS NULL;
```

Logical Operators

Operator	Meaning
AND	Returns TRUE if *both* component conditions are TRUE
OR	
NOT	Returns TRUE if *either* component condition is TRUE
Operator	Returns TRUE if the following condition is FALSE

Examples:

```
SQL> SELECT empno, ename, job, sal
  2 FROM   emp
  3 WHERE  sal>=1100
  4 AND    job='CLERK';

SQL> SELECT empno, ename, job, sal
  2 FROM   emp
  3 WHERE  sal>=1100
  4 OR     job='CLERK';

SQL> SELECT ename, job
  2 FROM   emp
  3 WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

ORDER BY Clause

- Sort rows with the ORDER BY clause
- ASC: ascending order, default
- DESC: descending order
- The ORDER BY clause comes last in the SELECT statement.

Examples:

```
SQL> SELECT          ename, job, deptno, hiredate
  2 FROM       emp
  3 ORDER BY hiredate;

SQL> SELECT   ename, job, deptno, hiredate
  2 FROM       emp
  3 ORDER BY hiredate DESC;
```

Sorting by Multiple Columns

```
SQL> SELECT          ename, deptno, sal  FROM     emp
 ORDER BY    deptno, sal DESC;
```

Practice

1. Show the structure of the EMP table. Create a query to display the name, job, hire date, and employee number for each employee.
2. Create a query to display the name and salary of employees earning more than $2850.
3. Create a query to display unique jobs from the Emp table;
4. Create a query to display the name and salary for all employees whose salary is not in the range of $1500 and $2850.
5. Display the name and job title of all employees who do not have a manager.

6. Display the name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

7. Display the names of all employees where the third letter of their name is an *A*.

lab2

Sunday, April 22, 2018 2:28 AM

lab2

Lab2

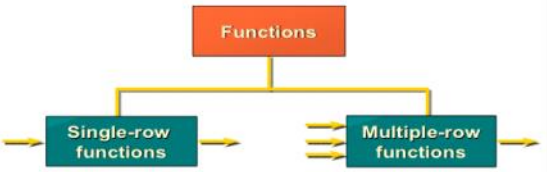
Objectives:

At the end of this lab, you should be able to

- Use some character functions.
- Use some number functions
- Converts date to char and vice versa.
- Use multiple row function :avg,sum,count,max,min

There are two distinct types of functions:

- Single-row functions
- Multiple-row functions



1. Single-row functions

a. **Character functions**: Accept character input and can return both character and number values
b. **Number functions**: Accept numeric input and return numeric values
c. **Date functions:** Operate on values of the date data type
d. **Conversion functions:** Convert a value from one data type to another
e. **General functions:** NVL function

a. Character functions:

- **Lower(*column|expression*) :** converts alpha character values to lowercase.
- **Upper ((*column|expression*) :** converts alpha character values to uppercase.

Example : *select upper(ename),lower(ename) from emp;*

- **Length(*column|expression*):** returns number of characters in value.

Example : *select ename, length(ename) from emp;*

- **Substr(*column|expression*,f,n)**: returns characters from values starting at character position m, n characters long.

Example : *select ename, substr(ename,2,3) from emp;*

B) Number Functions

Note: *The DUMMY is a dummy table which is generally used for SELECT clause syntax completeness, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from actual tables.*

i. **ROUND(*column*|*expression*, *n*): r**ounds the column, expression, or value to n decimal places or if n is omitted, no decimal places (If n is negative, numbers to left of the decimal point are rounded.)

Example : *SELECT ROUND(45.923,2), ROUND(45.923,0),ROUND(45.923,-1) FROM DUMMY;*

ii. **TRUNC(column|expression,n):** Truncates the column, expression, or value to n decimal places or if n is omitted, no decimal places (If n is negative, numbers left of the decimal point are truncated to zero.)

Example : *SELECT TRUNC(45.923,2), TRUNC(45.923),TRUNC(45.923,-1) FROM DUMMY;*

C) Date Functions

a. Oracle stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
b. The default date format is DD-MON-YY.
c. Arithmetic with Dates
   Date+ number =Date (adds a number of days to a date)
   Date- number =Date (subtracts a number of days to a date)
   Date-date= number of days.

Note All date functions return a value of DATE data type except MONTHS_BETWEEN, which returns a numeric value.

i. **MONTHS_BETWEEN(date1, date2)**: Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result represents a portion of the month.

Example : *Select ename,months_between(sysdate,hiredate) from emp;*

ii. **ADD_MONTHS(date, n):** Adds *n* number of calendar months to *date*. The value of *n* must be an integer and can be negative.

Example : *Select ename ADD_MONTHS(hiredate, 6) REVIEW from emp;*

Practice:

1) For each employee display the employee name and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

2) Display the name, hire date, and day of the week on which the employee started. Label the column DAY.

D) Data type Conversion

SQL provides three functions to convert a value from one datatype to another:

i. **TO_CHAR(*date, 'fmt'*)**

Example : *SELECT ename, TO_CHAR(hiredate, 'DD Month YYYY') FROM emp;*

ii. **TO_NUMBER(*char*[, '*fmt*'])**

Example : SELECT TO_NUMBER('123')+4 FROM DUMMY;

iii. **TO_DATE(*char*, '*fmt*')**

Example : SELECT ename, hiredate FROM emp WHERE hiredate = TO_DATE('February 22, 1981', 'Month dd, YYYY');

E) NVL Function

Converts null to an actual value

- NVL(comm,0)
- NVL(hiredate,'01-JAN-97')
- NVL(job,'No Job Yet')

Example: SELECT ename, sal, comm, (sal*12)+NVL(comm,0) FROM emp;

Practice:

Create a query that will display the employee name and commission amount. If the employee does not earn commission, put "No Commission." Label the column COMM.

2- Multiple-row functions:

Note:

- You can use MIN and MAX for any datatype.
- Group functions ignore null values in the column.

example:

```
SQL> SELECT   AVG(sal), MAX(sal),MIN(sal), SUM(sal)   FROM emp
2   WHERE     job LIKE 'SALES%';

SQL> SELECT   COUNT(*) FROM   emp  WHERE deptno = 30;

SQL> SELECT   COUNT(comm) FROM      emp  WHERE deptno = 30;

SQL> SELECT AVG(comm) FROM  emp;

SQL> SELECT AVG(NVL(comm,0)) FROM  emp;
```

Note the difference

Note the difference

Creating Groups of Data

Divide rows in a table into smaller groups by using the GROUP BY clause

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[ORDER BY   column];
```

Example:

```
SQL> SELECT   deptno, AVG(sal)  FROM     emp  GROUP BY deptno;
```

The HAVING Clause

- **You cannot use the WHERE clause to restrict groups.**
- The Oracle Server performs the following steps when you use the HAVING clause:
  - Rows are grouped.
  - The group function is applied to the group.
  - The groups that match the criteria in the HAVING clause are displayed.

Example:

```
SQL>SELECT    job, SUM(sal) from emp group by job
 2      having sum(sal)<5000 ;

SQL> SELECT    job, SUM(sal) PAYROLL FROM  emp  WHERE  job NOT LIKE 'SALES%'
2    GROUP BY  job  ORDER BY  SUM(sal);
```

Practice

1. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number.

2. Display the minimum, maximum, sum, and average salary for each job type

3. Write a query to display the number of people with the same job.

4. Write a query to display dept. numbers that number of employee in that dept. is less than 5.

lab3

Sunday, April 22, 2018 2:28 AM

lab3

Lab3

Objectives:

At the end of this lab, you should be able to

- Write select statement to retrieve data from more one table.
- Use subquires

Cartesian Product

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table

Example:

```
SELECT ename, dname FROM emp, dept;
```

You can explicitly create Cartesian product using "cross join:
For example

```
Select ename, dname from emp cross join dept;
```

To avoid a Cartesian product, always include a valid join condition.

Question:
Create a query to display employee name, his department name for all employees.

To get such a report, the data is in separate tables.

- Ename exists in the EMP table.
- Dname exists in DEPT table.

To produce the report, you need to link EMP and DEPT tables and access data from both of them, write a simple join condition in the WHERE clause.

```
select emp.ename,dept.dname from emp,dept where emp.deptno=dept.deptno;
```

Notes:

- When writing a SELECT statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.

Types of Joins

The types of join conditions:

1. **Equijoins**
2. **Non-equijoins**
3. **Outer joins**
4. **Self joins**

1) Equijoins:

When the query is relating two tables using an equality operator(=) it is an equijoin or (inner join).

Example: **select emp.ename,dept.dname from emp,dept where emp.deptno=dept.deptno;**

Note: You can join tables automatically based on the columns in the two tables that have matching data types and names using NATURAL JOIN keywords.

Selet ename, dname from emp natural join dept;

Note:

- If the columns have the same name but different types then it causes syntax error.
- If the two tables have multiple similar columns, and you want to join based only one table, then you should use clause ON

```
Select ename, dept from emp
join  dept on (emp.deptno=dept.deptno);
```

Applying additional conditions with Equijoin

Select ename, dname from emp natural join dept where mgr=7839;

Which is equivalent to

```
Select  emp.ename,dept.dname from emp,dept where emp.deptno=dept.deptno
and
mgr=7839;
```

---

2) Self joins:

How to retrieve the name of each employee with name of his manager? Managers also are employees and their data is in the table emp.

```
SELECT worker.ename,manager.ename
FROM       emp worker, emp manager
WHERE      worker.mgr = manager.empno;
```

With self-join, you need to use alias to refer to the same table twice.

3) Non-Equijoins:

Example :

```
SELECT    e.ename, e.sal, s.grade FROM emp e,  salgrade s
WHERE     e.sal
BETWEEN   s.losal AND s.hisal;
```

Notes : In the above example, we use table aliases, in this case they are not optional.

---

4) Outer joins:

Note: Joining two tables using equijoin will not display unmatched rows in both tables, this is called "Inner join". To return the unmatched rows, we use Outer join.

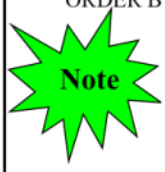
Example :

If we want to display all departments including those that do not have employees.

```
Select ename, dname from emp
right outer join dept on (emp.deptno=dept.deptno);
```

Note: in Oracle, you can use + operator for outer join

```
SELECT      e.ename, d.deptno, d.dname  FROM  emp e,   dept d
WHERE       e.deptno(+) = d.deptno
ORDER BY          e.deptno;
```



The outer join operator (+)can appear on only one side of the expression—the side that has information missing. It returns those rows from one table that have no direct match in the other table.
The (+) after deptno in emp table (left side) indicates that employees will be displyay as null for the departments that do not have employees.

Practice:

Write a query to display employee name and manger name including the employees who have no managers.

Note: To include data from both sides, even if they have no match on the other side, use "Full outer"

```
Select e.name, m.ename from emp e
Full outer join emp m on (e.mgr=m.empno);
```

Practice

1. Write a query to display the employee name, department name, and location of all employees who earn a commission.
2. Show the structure of the SALGRADE table. Create a query that will display the name, job, department name, salary, and grade for employees who are in grade 3.
3. Create a query to display the name and hire date of any employee hired after employee Blake.
4. Display all employees' names and hire dates along with their manager's name and hire date for all employees who were hired before their managers. Label the columns Employee, Emp, Hiredate, Manager, and Mgr Hiredate, respectively.

Subqueries

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT      select_list
                    FROM         table);
```

Notes:

- The subquery (inner query) executes before the outer query.
- The result of the subquery is used by the main query.
- The subquery can be used in SQL clauses including: where clause, having clause, from clause.
- *Operator* includes comparison conditions (>,>=,<,<=,=,<>) for single row subquery and (IN, ANY, ALL) for multiple-row subquery.

Single Row subquery:

The subquery returns only one row.

Example: Assume we want to display names of all employees whose salaries are greater than the salary of employee no. 7566

```
SELECT ename  FROM   emp
WHERE  sal >   (SELECT sal      FROM   emp
                WHERE  empno=7566);
```

In the example, the inner query determines the salary of employee 7566. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

Practice:

Write a query to display name of the employee who was hired in the same day as "JAMES', but his salary is greater than "JAMES" salary.

Using Group Functions in a Subquery

To display name, job and salary of employees whose salary is the minimum , we use min group function in the subquery as follows :

```
SELECT      ename, job, sal  FROM emp
  WHERE     sal =
      (SELECT     MIN(sal)
              FROM        emp);
```

Using HAVING clause in the subquery

```
SELECT DEPTNO,MIN(SAL) FROM EMP
GROUP BY DEPTNO
HAVING MIN(SAL)>(SELECT MIN(SAL) FROM EMP WHERE DEPTNO=20);
```

Multiple-Row Subqueries

Subqueries that return more than one row are called *multiple-row subqueries*. You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values.

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

Examples

Display employees numbers ,names and jobs for all employees whose salaries are less than any clerk

```
SELECT  empno, ename, job
 FROM    emp
 WHERE   sal < ANY
              (SELECT      sal
                      FROM emp
              WHERE        job = 'CLERK')
```

Display employees numbers ,names and jobs for all employees whose salaries are greater than all departments' averages .

```
SELECT  empno, ename, job
  FROM    emp
  WHERE   sal > ALL
               (SELECT      avg(sal)
                       FROM         emp
                       GROUP BY    deptno);
```

Practice:

1. Write a query to display the employee name and hire date for all employees in the same department as Blake. Exclude Blake.
2. Create a query to display the employee number and name for all employees who earn more than the average salary. Sort the results in descending order of salary.
3. Write a query that will display the employee number and name for all employees who work in a department with any employee whose name contains a *T*.
4. Display the employee name, department number, and job title for all employees whose department location is Dallas.
5. Display the department number, name, and job for all employees in the Sales department.

lab4

Sunday, April 22, 2018 2:29 AM

lab4

Lab4

Objectives:

At the end of this lab, you should be able to

- Use DML: Insert, update, delete.
- Database Transactions: commit, rollback.

Data Manipulation Language (DML)

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement.

The INSERT Statement

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

Example

```
INSERT INTO        dept (deptno, dname, loc)
 VALUES     (50, 'DEVELOPMENT', 'DETROIT');
```

Note: the column list is not required in the INSERT clause if it contains values for each column in the table.

```
INSERT INTO        dept   VALUES        (60, 'HR', 'DOHA');
```

Inserting Rows with Null Values:

Implicit method:

```
INSERT INTO     dept (deptno, dname )  VALUES(60, 'MIS');
```

Explicit method

```
INSERT INTO dept  VALUES (70, 'FINANCE', NULL);
```

Practice:

- Add the following employee to EMP table
  No : 200 Name: AHMED Salary: 4000 Hiredate: 1/1/2008
- Add the following department to DEPT table
  No : 80 Name: IT Location: None.
- Add a new record to EMP table
  No: 123 Name: Sara hiredate: today date.

Note:

1- Try to insert the following record to the dept table:
   Deptno: 10
   Dname: SUPPORT

Loc :null

What is the reason for such problem?_____

2- Add the following record to table emp

Empno: 123

Ename : Ahmed

Deptno: 99

What is the reason for such problem?_____

The UPDATE Statement

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

Example

UPDATE emp SET deptno = 20 WHERE empno = 7782;

- If you do not put a condition, update will be applied to all records in the table.
- If you attempt to update a record with a value that is tied to an integrity constraint, you will experience an error, for example try to update deptno 20 into 55 in the table dept

Practice:

- Write a SQL statement to change the location column of department table with 'Doha' for department number 80.
- Write a SQL statement to change salary of the employee whose ID is 7499 to 8000, if the existing salary is less than 5000.

Note:

1- Try to update the deptno 10 to be 20 .

What is the reason for such problem?_____

2- Try to update deptno of employee 7900 to be 45.

What is the reason for such problem?_____

Using subqueries in UPDATE statement:

- Example: increase the salary of all employees who work in RESEARCH dept by 1000.

UPDATE EMP SET SAL=SAL+1000

WHERE DEPTNO=(SELECT DEPTNO FROM DEPT WHERE
DNAME='RESEARCH');

- Example: make the salary of employee whose id is 7900 as the salary of the employee whose id 7369

  UPDATE EMP SET SAL=(SELECT SAL FROM EMP WHERE EMPNO=7369)
  WHERE EMPNO=7900

The DELETE Statement

```
DELETE [FROM]   table
[WHERE          condition];
```

Example

```
DELETE FROM     dept  WHERE dname = 'DEVELOPMENT';

DELETE FROM     emp  WHERE          deptno =
      (SELECT  deptno  FROM    dept WHERE   dname ='SALES');
```

Integrity Constraint Error:

```
SQL> DELETE FROM dept  WHERE deptno = 10;
```

Practice:

- Write SQL statement to delete from the table EMP all employees in dept. 30.

Note:

1- Try to delete deptno 10 .
What is the reason for such problem?_____

Using subqueries in DELETE statement:

Example: Delete all employees who work in 'NEW YORK'.

```
DELETE FROM EMP WHERE DEPTNO=
(SELECT DEPTNO FROM DEPT WHERE
LOC='NEW YORK');
```

Database Transactions

Try to insert a new record to table EMP. Then open another copy of SQL Developer, now in the new one write a query to display the data in EMP table. What do you find?

When Does a Transaction Start and End?

A transaction begins when the first executable SQL statement is encountered and terminates when one of the following occurs:

- A COMMIT or ROLLBACK statement is issued
- A DDL statement, such as CREATE, is issued
- A DCL statement is issued
- The user exits SQL*Plus
- A machine fails or the system crashes

Explicit Transaction Control Statements

You can control the logic of transactions by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

Statement	Description
COMMIT	Ends the current transaction by making all pending data changes permanent
ROLLBACK [TO *SAVEPOINT name*]	ROLLBACK ends the current transaction by discarding all pending data changes; ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding the savepoint and any subsequent changes. If you omit this clause, the ROLLBACK statement rolls back the entire transaction.

Examples:

```
SQL> DELETE FROM        employee;
SQL>ROLLBACK;
SQL>UPDATE emp  SET  deptno = 10  WHERE empno = 7782;
SQL> SELECT ename,deptno from emp where empno=7782;
SQL>Rollback;
SQL>UPDATE emp  SET  deptno = 10  WHERE empno = 7782;
SQL>COMMIT;
SQL> SELECT ename,deptno from emp where empno=7782;
```

Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement is issued Normal exit from SQL*Plus, without explicitly issuing COMMIT or ROLLBACK
Automatic rollback	Abnormal termination of SQL*Plus or system failure

lab5

Sunday, April 22, 2018 2:29 AM

lab5

Lab5: Creating and Managing Tables

Objectives:

At the end of this lab, you should be able to:

- Use DDL: create table, alter table, drop table, truncate table.
- Creating and using Views.

Database Objects

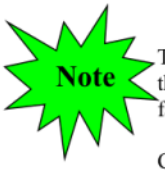
- **Table:** Stores data
- **View:** Subset of data from one or more tables
- **Sequence:** Generates primary key values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives alternative names to objects

Creating Tables

```
CREATE TABLE dept2     (deptno          NUMBER(2), dname VARCHAR2(14),
loc    VARCHAR2(13));
```

Creating a Table by Using a Subquery

```
CREATE TABLE   dept30 AS
        SELECT          empno, ename, sal*12 ANNSAL, hiredate
        FROM    emp  WHERE          deptno = 30;
```



To create a table with the same structure as an existing table, but without the data from the existing table, use a subquery with a WHERE clause, that will always evaluate as false. For example:

```
CREATE TABLE TEST AS
 (SELECT * FROM emp      WHERE 1 = 2);
```

Copying Rows from another Table

Create a new table called managers that has the same structure as EMP

Copy data of managers to the new table as following:

```
INSERT INTO managers(id, name, salary, hiredate)
SELECT       empno, ename, sal, hiredate  FROM   emp
WHERE        job = 'MANAGER';
```

Exercise: Create table Tax which has the following structure

Tax	
Empno	Number(5)
Tax	Number(10,2)

Fill the table with employees' data and the tax of their salaries, where tax is 5%.

Solution:

```
CREATE TABLE TAX ( EMPNO NUMBER(5),
TAX NUMBER (10,2));

INSERT INTO TAX SELECT EMPNO, SAL*0.05 FROM EMP;
```

It can be done in one command as follows :

```
CREATE TABLE TAX AS SELECT EMPNO, SAL *0.05
AS TAX FROM EMP ;
```

RENAME A TABLE:

-You can rename any database object using the command rename:

Example :

```
RENAME DEPT2 TO DEPARTMENT2.
```

The ALTER TABLE Statement

1) Adding a Column

```
ALTER TABLE dept30
ADD          (PHONE NUMBER (6));
```

You can add more than one column in a single ALTER command:

```
ALTER TABLE DEPT30 ADD (ADDRESS VARCHAR2(20), EMAIL VARCHAR2(10));
```

2) Modifying a Column

```
ALTER TABLE     dept30
MODIFY          (ename VARCHAR2(15));
```

Try to modify the size of column DNAME in DEPT30 table to 5 characters.

To change the name of a specific field (column):
ALTER TABLE DEPT30 RENAME COLUMN PHONE TO MOBILE;

3) Dropping a Column

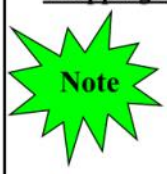
```
ALTER TABLE     dept30
DROP  COLUMN   hiredate ;
```

Try to drop the field *deptno* in table dept.

To drop more than one column :

```
ALTER TABLE     dept30
DROP    (address, email);
```

Dropping a Table



- **All data and structure in the table is deleted.**
- **You *cannot* roll back this statement.**

```
SQL> DROP TABLE dept30;
```

Try to drop the table dept.

TRUNCATE a TABLE

TRUNCATE TABLE command is used to delete complete data from an existing table.

- It removes the data in the table
- It is considered a DDL command.
- It cannot be rolled back

Example:

```
Truncate Table Tax;
```

Difference between TRUNCATE, DELETE:

- Truncate will delete all data, while delete will delete records based on a condition.
- Delete is DML, truncate is DDL.
- Delete needs comment, truncate does not.
- Delete can be rolled back and restore the data, while truncate cannot.

Practice:

1. Create the EMPLOYEE2 table based on the structure of the EMP table. Include only the EMPNO, ENAME, and DEPTNO columns. Name the columns in your new table ID,LAST_NAME, and DEPT_ID, respectively.

2. Modify the EMPLOYEE2 table to allow for longer employee last names. Confirm your modification.

3. Drop the LAST_NAME column from the EMPLOYEE2 table. Confirm your modification by checking the description of the table.

4. Drop the EMPLOYEE2 table.

Views

A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed.

Simple Views

Example

```
CREATE VIEW     v1
      AS SELECT empno, ename, job
      FROM      emp
      WHERE     deptno = 10;

SQL> DESCRIBE v1
SQL>  SELECT *
      FROM v1;
```

Exercise:

1. Insert a new employee to emp table :
   Empno: 456, name : Khaled, deptno: 20.
2. Query the view v1
3. Create a view contains empno, ename, dname based on the tables emp and dept.

Complex View

Example

```
SQL> CREATE VIEW      dept_sum (name, minsal, maxsal, avgsal)
  AS SELECT       d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)
  FROM            emp e, dept d
  WHERE           e.deptno = d.deptno
  GROUP BY        d.dname;
```

Exercise:

1. Increase the salaries of employees in dept 10 by 1000.
2. Display the data in dept_sum.

Dropping a VIEW

To remove a view

```
DROP VIEW V1
```

Practice

1. Create a view v.
2. Create a view called EMP_VU based on the employee number, employee name, and department number from the EMP table. Change the heading for the employee name to EMPLOYEE
3. Using your view EMP_VU, enter a query to display all employee names and department numbers.

lab6

Sunday, April 22, 2018 2:29 AM

lab6

Lab 6

Objectives:

At the end of this lab, you should be able to:

- Use Oracle constraints.
- Use set operators to combine two or more queries.
- Use oracle data dictionary.

Constraints

The Oracle Server uses *constraints* to prevent invalid data entry into tables.
You can use constraints to do the following:

- Enforce rules at the table level whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables.

Constraint	Description
NOT NULL	Specifies that this column may not contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a foreign key relationship between the column and a column of the referenced table
CHECK	Specifies a condition that must be true

Create a constraint:

- At the same time as the table is created
- After the table has been created

1- The NOT NULL Constraint

Example

```
CREATE TABLE emp5(
        empno NUMBER(4),
        ename  VARCHAR2(10) NOT NULL,
        deptno NUMBER(2) NOT NULL);
```

Try to insert a new record to the table emp5 and make valued of *ename* null.
NOTE: because no constraint name is provided, the constraint is named automatically by Oracle server.

Prepared by Dr. Saleh Alhazbi

2- The UNIQUE Key Constraint
Example

```
CREATE TABLE   dept5(
        deptno   NUMBER(2),
        dname    VARCHAR2(14),
        loc      VARCHAR2(13),
        CONSTRAINT dept5_dname_uk UNIQUE(dname));
```

Try to insert two records to the table dept5 with the same values for dname field

---

3- The PRIMARY KEY Constraint
Example

```
CREATE TABLE   dept6(
        deptno   NUMBER(2),
        dname    VARCHAR2(14),
        loc      VARCHAR2(13),
        CONSTRAINT dept6_dname_uk2 UNIQUE (dname),
        CONSTRAINT dept6_deptno_pk PRIMARY KEY(deptno));
```

Try to insert two records with the same primary key.

Difference between primary key and unique constraints:

1- Primary key field cannot have null values, while unique fileds can have null.
2- Only one primary key constraint in a table, but you can have multiple unique constraints for a table.

Composite primary key:

It is a combination of two or more columns

Example:

```
CREATE TABLE CUSTOMER( YEAR NUMBER(4),
ID NUMBER (4),
NAME VARCHAR2(20),
PHONE NUMBER (7),
CONSTRAINT CUSTOMER_YEAR_ID_PK PRIMARY KEY (YEAR,ID) );
```

4- The FOREIGN KEY Constraint
Example

```
CREATE TABLE emp6(
 empno          NUMBER(4),
 ename VARCHAR2(10) NOT NULL,
deptno number(4),
 CONSTRAINT emp6_deptno_fk FOREIGN KEY (deptno)
REFERENCES dept (deptno));
```

Prepared by Dr. Saleh Alhazbi

The foreign key is defined in the child table, and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- FOREIGN KEY is used to define the column in the child table at the table constraint level.
- REFERENCES identifies the table and column in the parent table.
- ON DELETE CASCADE indicates that when the row in the parent table is deleted, the dependent rows in the child table will also be deleted.
- Without the ON DELETE CASCADE option, the row in the parent table cannot be deleted if it is referenced in the child table.

Example

```
CREATE TABLE CUSTOMER2 (ID NUMBER(4) PRIMARY KEY,
NAME VARCHAR2(20),
DEPTNO NUMBER (3),
CONSTRAINT CUSTOMER2_DEPTNO_FK FOREIGN KEY(DEPTNO)
REFERENCES DEPT(DEPTNO) ON DELETE CASCADE );
```

- Insert a new record to dept table as follows
  Deptno: 88 , Dname: HR, Loc: Doha.
- Insert a new record to table customer2 as follows
  Id: 123, name : Ahmed , deptno :88.
- Now delete the dept 88.
- Check if the customer 123 was deleted.

5- The CHECK Constraint

Example

```
CREATE TABLE emp7(
        empno NUMBER(4),
        ename  VARCHAR2(10) NOT NULL,
        deptno NUMBER(2) NOT NULL)
        gender CHAR(1),
        CONSTRAINT emp7_gender_ck  CHECK (gender='M' or gender='F'));
```

Try to insert a new record with the gender 'm'.

Note: This type of constraint can be used to add null constraint with a name.

Example

```
CREATE TABLE CUSTOMER3 (
ID NUMBER (3),
NAME VARCHAR2(20),
CONSTRAINT NAME_NN CHECK (NAME IS NOT NULL));
```

Prepared by Dr. Saleh Alhazbi

Adding a Constraint after creating the table:

Example

```
CREATE TABLE EMP8 AS SELECT * FROM EMP;

ALTER TABLE     emp8
ADD CONSTRAINT  EMP8_EMPNO_PK primary key (empno);
```

Note : a constraint cannot be added if the data already in that table violates that constraint.

Dropping a Constraint

Example

```
ALTER TABLE       emp6
DROP CONSTRAINT  emp_mgr_fk;
```

Disabling and Enabling Constraint

You can disable a constraint without dropping it ore recreating it.

Example

```
ALTER TABLE EMP8 DISABLE CONSTRAINT EMP8_EMPNO_PK;

ALTER TABLE EMP8 ENABLE CONSTRAINT EMP8_EMPNO_PK;
```

Set Operators:

Create a new table called job_history to keep track of all different positions held by an employee since his/her hire date, the table job_history contains the following fields

Name	Not null?	Type
Emp_id	Not null	Number (4)
Job	Not null	Varchar2(10)
Start_date	Not null	date
End_date	Not null	Date
Deptno		Number (2)

Insert the following records to that table

Emp_id	Job	Start_date	End-date	Deptno
7369	SALESMAN	17-DEC-80	25-JAN-82	30
7369	CLERK	26-JAN-82	30-MAY-85	10
7839	ANALYST	17-NOV-81	12-JUN-83	20
7839	PRESIDENT	13-JUN-83	01-MAR-84	30
7654	CLERK	28-SEP-81	02-FEB-84	10

Prepared by Dr. Saleh Alhazbi

To display the current and previous job for the employees :

```
Select empno, job from emp Union
Select emp_id, job from job_history;
```

To display all data (include duplicated ones)

```
Select empno, job from emp Union  all
Select emp_id, job from job_history;
```

To display data for those who currently hold same position as before

```
Select empno, job from emp intersect
Select emp_id, job from job_history;
```

To display employee ID's for those who never change their jobs

```
Select empno  from emp minus
Select emp_id  from job_history;
```

Oracle Data Dictionary :

The *data dictionary* is a **read-only** set of tables that provides information about its associated database. For example the names of Oracle users, privileges and roles each user has been granted …etc.

The data dictionary has two primary uses:

- Oracle accesses the data dictionary every time that a DDL statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

Oracle divides data dictionary views into the three families, as indicated by the following prefixes:

- **USER**:

USER views return information about objects owned by the currently-logged-on database user. Here some of those views:

a. **USER_TABLES** all tables with their name, number of columns, storage
b. **USER_CONSTRAINTS** constraint definitions for tables
c. **USER_INDEXES** all information about indexes created for tables (IND)
d. **USER_OBJECTS** all database objects owned by the user (OBJ)
e. **USER_TRIGGERS** triggers defined by the user
f. **USER_USERS** information about the current user
g. **USER_VIEWS** views defined by the user

Prepared by Dr. Saleh Alhazbi

For example, a query to USER_TABLES returns a list of all of the relational tables that you own.

```
select table_name, tablespace_name from user_tables
```

```
SELECT constraint_name, constraint_type, status, table_name  FROM user_constraints
```

- **ALL:**

_ALL views return information about all objects to which you have access, regardless of who owns them. For example, a query to ALL_TABLES returns a list not only of all of the relational tables that you own, but also of all relational tables to which their owners have specifically granted you access (using the GRANT command).

```
select table_name, tablespace_name from all_tables
```

- **DBA:**

DBA views are generally accessible only to database administrators, and return information about all objects in the database, regardless of ownership or access privileges. For example, a query to DBA_TABLES will return a list of all relational tables in the database, whether or not you own them or have been granted access to them.

Prepared by Dr. Saleh Alhazbi

lab7

Sunday, April 22, 2018 2:25 AM

lab7

Lab 7

Objectives:

At the end of this lab, you should be able to:
Write simple Procedural Language/SQL (PL/SQL) program.

Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for relational databases.
PL/SQL Block Structure

PL/SQL is a block-structured language, meaning that programs can be divided into logical blocks. A PL/SQL block consists of up to three sections:

- **Declarative (optional):** contains all variables, constants, coursers that are used in the executable section.
- **Executable (required):** contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.
- **Exception handling (optional):** specifies the section to perform when errors and abnormal conditions arise in the executable section.

Declarative
Executable
Exception handling

```
DECLARE
 V      VARCHAR2(5);

BEGIN
 SELECT     column_name
  INTO      v
  FROM      table_name;

EXCEPTION
 WHEN exception_name THEN
 ...
END;
```

Declaring PL/SQL Variables

```
Declare
        v_hiredate          DATE;
        v_deptno            NUMBER(2) NOT NULL := 10;
        v_location          VARCHAR2(13) := 'Atlanta';
        c_comm              CONSTANT NUMBER := 1400;
```

Initialize the variable to an expression with the assignment operator (:=) or, equivalently, with the DEFAULT reserved word. If you do not assign an initial value, the new variable contains NULL by default until you assign it later.

The %TYPE Attribute

You can use the %TYPE attribute to declare a variable according to another previously declared variable or database column

```
...
  v_ename                emp.ename%TYPE;
  v_balance              NUMBER(7,2);
  v_min_balance          v_balance%TYPE := 10;
...
```

The %ROWTYPE Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table.
- Fields in the record take their names and datatypes from the columns of the table or view.

For example:

dept_record dept%ROWTYPE;

To access a field of the variable of dept_record, you use dot.

dept_record.depno=50;

Boolean Variables

- Only the values TRUE, FALSE, and NULL can be assigned to a Boolean variable.
- The variables are connected by the logical operators **AND**, **OR**, and **NOT.**

V_sal Boolean :=true;

Writing Executable Statements

Because PL/SQL is an extension of SQL, the general syntax rules that apply to SQL also apply to the PL/SQL language.

1. Commenting Code

Comment the PL/SQL code with two dashes (--) if the comment is on a single line, or enclose the comment between the symbols /* and */ if the comment spans several lines

2. Retrieving Data Using PL/SQL

- Use the SELECT statement to retrieve data from the database with into clause. The INTO clause is mandatory and occurs between the SELECT and FROM clauses.
- You must give one variable for each item selected, and their order must correspond to the items selected.
- Queries Must Return One and Only One Row.

```
DECLARE
  v_deptno    NUMBER(2);
  v_loc       VARCHAR2(15);
BEGIN
  SELECT      deptno, loc
  INTO        v_deptno, v_loc
  FROM        dept
  WHERE       dname = 'SALES';
...
END;
```

To display information from a PL/SQL block you can use *DBMS_OUTPUT.PUT_LINE*. DBMS_OUTPUT is an Oracle-supplied package, and PUT_LINE is a procedure within that package. Within a PL/SQL block, reference DBMS_OUTPUT.PUT_LINE and, in parentheses, the information you want to print to the screen. The package must first be enabled in your SQL*Plus session. To do this, execute the SQL*Plus command *SET SERVEROUTPUT ON.*

Example:

```
SET SERVEROUTPUT ON
DECLARE
  v  NUMBER(6,2) ;
BEGIN
SELECT AVG(SAL) INTO v FROM EMP;

  DBMS_OUTPUT.PUT_LINE ('The average salary is ' || TO_CHAR(v));
END;
```

Manipulating Data Using PL/SQL

You can issue the DML commands INSERT, UPDATE, and DELETE without restriction in PL/SQL. Including COMMIT or ROLLBACK statements

```
DECLARE
  v_sal_increase   emp.sal%TYPE := 2000;
BEGIN
  UPDATE      emp
  SET         sal = sal + v_sal_increase
  WHERE       job = 'ANALYST';
END;
```

Note: PL/SQL variable assignments always use := and SQL column assignments always use =.

Writing Control Structures

1. IF Statements

Syntax :

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

For example:
write a program that displays number of employees whose salaries is less than 5000 or display a message " No employee found"

```
Declare
        x  number(3):=0;
Begin
        Select count(*) into x  from emp where sal<5000;
        If x=0 then
                Dbms_output.put_line('No employee found');
        Else
                Dbms_output.put_line(x);
        End if;
End;
```

Exception

```
Declare
        x  number(3):=0;
Begin
        Select deptno into x from dept where dname='IT';
Exception
When no_data_found then
        Dbms_output.put_line(' No such department');
End;
```

PL/SQL - Procedures

A **subprogram** is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts:

- **Declarative (optional):** contains all variables, constants, coursers that are used in the executable section.
- **Executable (required):** contains SQL statements to manipulate data in the database and PL/SQL statements to manipulate data in the block.
- **Exception handling (optional):** specifies the sction to perform when errors and abnormal conditions arise in the executable section.

```
CREATE [OR REPLACE] PROCEDURE
procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
< procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

No Parameters

```
CREATE OR REPLACE PROCEDURE pr_class IS
    var_code VARCHAR2 (30):= 'CMPS352'
    var_name VARCHAR2 (50) := 'Fundamentals of DB Systems LAB;'
BEGIN
    DBMS_OUTPUT.PUT_LINE('This course is '||var_name||' which is  '||var_code);
END pr_class;
```

--Stored Procedure for Department Wide Salary Raise

```
CREATE OR REPLACE PROCEDURE emp_sal( dep_id NUMBER, sal_raise
NUMBER)
IS
BEGIN
  UPDATE EMP SET SAL = SAL * sal_raise WHERE DEPTNO = dep_id;
  DBMS_OUTPUT.PUT_LINE ('salary updated successfully');
END;
```

Testing

```
EXEC emp_sal;(20,0.5)
SELECT * FROM EMP;
```

-- Adjusting Salaries Employees Table

```
--This procedure might need you to disable a trigger in order to work.
CREATE OR REPLACE PROCEDURE adjust_salary(
    in_employee_id IN EMP.EMPNO%TYPE,
    in_percent IN NUMBER
)IS
BEGIN
 --   update employee's salary
   UPDATE emp
   SET sal = sal + sal * in_percent / 100
   WHERE empno = in_employee_id;
END;

--Testing
 --before adjustment
SELECT sal FROM emp WHERE empno= 7788;
 --call procedure
exec adjust_salary(7788,5)
 --after adjustment
SELECT sal FROM emp WHERE empno= 7788;
```

PL/SQL - Functions

A function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows –

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Where,

- *function-name* specifies the name of the function.
- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

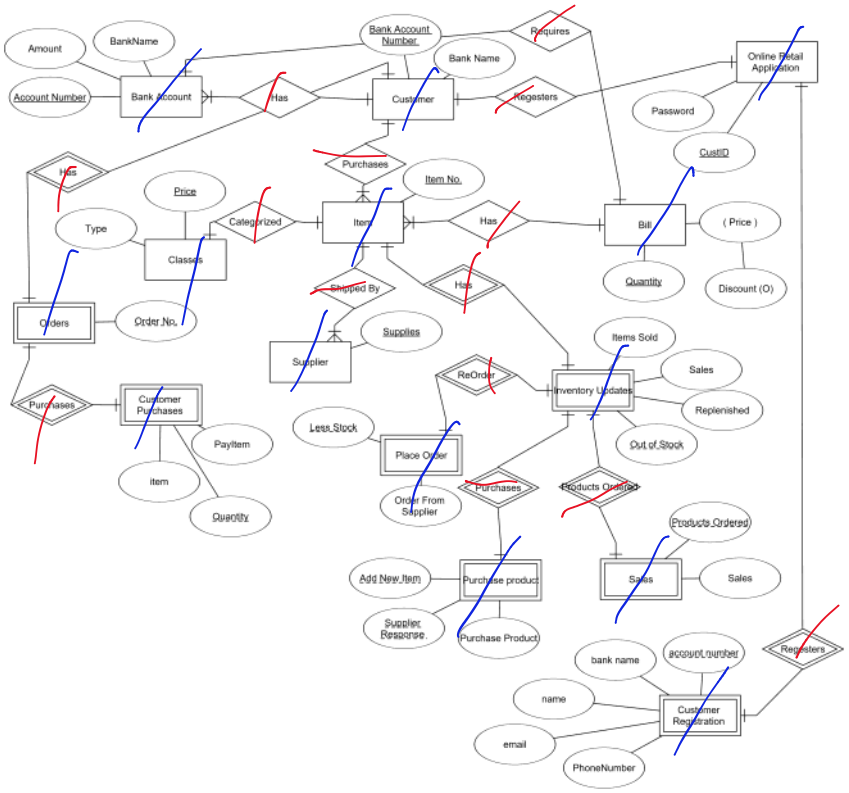
Example

The following example illustrates how to create and call a standalone function. This function returns the total number of EMPLOYEES in the emp table.

```
CREATE OR REPLACE FUNCTION totalEmployees
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM emp;
  RETURN total;
END;
```

Calling a Function

```
DECLARE
  c number(2);
BEGIN
  c := totalEmployees ();
  dbms_output.put_line('Total no. of Employees: ' || c);
END;
```


Amount
BankName
Bank Account Number
Requires
Account Number
Bank Account
Has
Customer
Bank Name
Registers
Online Retail Application
Password
CustID
Purchases
Item No.
Has
Price
Type
Categorized
Item
Has
Bill
(Price)
Classes
Quantity
Discount (O)
Shipped By
Has
Orders
Order No.
Supplies
Supplier
Items Sold
ReOrder
Inventory Updates
Sales
Replenished
Purchases
Customer Purchases
Less Stock
Out of Stock
PayItem
Place Order
item
Quantity
Order From Supplier
Purchases
Products Ordered
Products Ordered
Add New Item
Purchase product
Sales
Sales
Supplier Response
Purchase Product
Registers
bank name
account number
name
Customer Registration
email
PhoneNumber