LECTURE # 3 & 4

Oracle Database Objects

TABLES, VIEWS

PROCEDURES, FUNCTIONS, PACKAGES

RECAP

- Introduction to DBMS
- Industry software used for DBMS
- Most popular software [Top 10]
- Database architecture
  - 1 Tier
  - 2 Tier
  - 3 Tier
- Introduction to SQL and PLSQL
- Database languages
  - DDL, DML, DCL, TCS, SCS
  - 4th Generation languages
- Normalization Techniques

LEARNING OBJECTIVES

- What are database schema objects?
- When & How we need to use it?
- Types of database schema objects

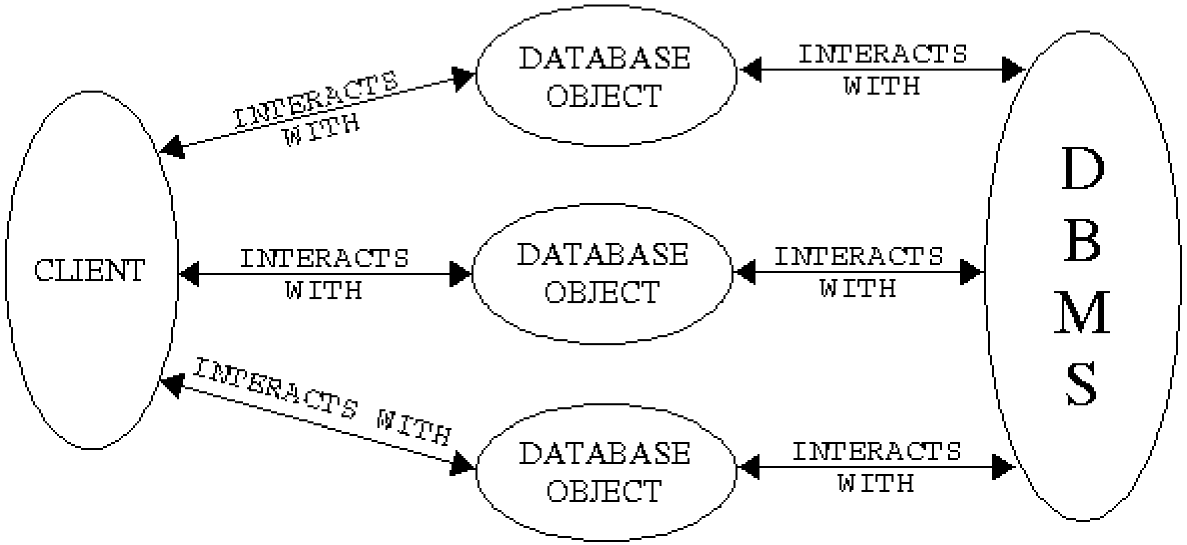
TABLES	Views	Sequences
Synonyms	Indexes	Clusters
Database Links	Snapshots	Procedures
Functions	Packages	

ORACLE SCHEMA

- A schema is a collection of logical structures of data, or schema objects
- A schema is owned by a database user and has the same name as that user
- Each user owns a single schema
- Schema objects can be created and manipulated with SQL and PLSQL
- Schema objects are logical data storage structures
- Oracle stores a schema object logically within a tablespace of the database.
- The data of each object is physically contained in one or more of the tablespace's data files
- There is no relationship between schemas and tablespaces: a tablespace can contain objects from different schemas, and the objects for a schema can be contained in different tablespaces

ORACLE SCHEMA OBJECTS

- A database object is any defined object in a database that is used to store or reference data.
- Some examples of database objects include tables, views, clusters, sequences, indexes, and synonyms.
- The table is the primary and simplest form of data storage in a relational database



ORACLE TABLES

- One of the first steps in creating a database is to create tables stores organization's data
- In order to create a table , four pieces of information must be determined

1. The table name
2. The column (field) names
3. Column data types and
4. Column sizes

- Table and Column names should be meaningful and reflect the nature of the data that is to be stored
- If the data stored is about the products that a firm sells, then the table should probably be named *product.*
- If products are identified by a string of characters, then the column that stores the product information data should be named *product_number*, or *product_code*

ORACLE TABLES

- The data type chosen for a column determines the nature of the data that can be stored in the column.
- This is termed the *Domain* of valid column values.
- Oracle provides 14 pre-determined data types as well as the ability to declare user defined data types.
- Creating a simple table that stores five items of information about employees for an organization.
- The table is named "EMPLOYEE" and stores information about each employee's
  - Social security number
  - Last name
  - First name
  - Date hired
  - Annual salary

ORACLE TABLES

```
CREATE TABLE EMPLOYEE(
    emp_ssn                CHAR(9),
    emp_last_name          VARCHAR2(25),
    emp_first_name         VARCHAR2(25),
    emp_date_of_birth      DATE,
    emp_salary             NUMBER(7,2)
);
```

- The table name “EMPLOYEE”, is specified along with five data columns.

- Each column has a name that is unique within the table and is specified to store a specific type of data.

ORACLE TABLES

DATA INTEGRITY

- The term data integrity simply means that the data stored in the table is valid
- There are different types of data integrity, often referred to as constraints
- The specifications of different data types aids in maintaining certain aspects of the data stored for employees

VIEWING A TABLE DESCRIPTION

- The SQL*PLUS DESCRIBE (DESC) command can display the column names and data types for any table.

- This command can be used when exact data types and column sizes for a table are unknown

ORACLE TABLES

```
DESC employee;

Name                Null?    Type
------------------- -------- ------------
EMP_SSN             NOT NULL CHAR(9)
EMP_LAST_NAME       NOT NULL VARCHAR2(25)
EMP_FIRST_NAME      NOT NULL VARCHAR2(25)
EMP_DATE_OF_BIRTH            DATE
EMP_SALARY          NOT NULL NUMBER(7,2)
EMP_PARKING_SPACE            NUMBER(4)
```

- Note that while *emp_ssn* column was specified to have a PRIMARY KEY constraint, the *Null?* Column displayed in the table description indicates whether or not a column is constrained as NOT NULL.

ORACLE TABLES

- Employee table can be deleted with the DROP TABLE command
- This command deletes both the table structure, its data, related constraints, and indexes

```
DROP TABLE employee;
```

- A table can be renamed with the RENAME command
- This command does not affect table structure or data, it simply gives the current table a new name

```
RENAME employee TO worker;
```

ORACLE TABLES

- Modifying existing tables to either add new columns or alter existing columns can be accomplished with the ALTER TABLE MODIFY and ALTER TABLE ADD commands.
- The current data type of the *emp_parking_space* column is NUMBER(4). A very large organization may have in excess of 9,999 employees.
- The ALTER TABLE command can be used to modify the *emp_parking_space* column to enable the allocation of upto 99,999 parking spaces.

```
ALTER TABLE employee MODIFY (emp_parking_space NUMBER(5));
```

- If a column modification will result in a column that is smaller than was originally specified, Oracle will return an error message if data rows exist such that their data will not fit into the new specified column size.

ORACLE TABLES

- The INSERT command is used to store data in tables
- The INSERT command is often embedded in higher-level programming language applications as an embedded SQL command.
- There are two different forms of the INSERT command.
- The first form is used if a new row will have a value inserted into each column of the row.
- The general form of the INSERT command is

```
 INSERT INTO table
VALUES (column1 value, column2 value, …);
```

- The second form of the INSERT command is used to insert rows where some of the column data is unknown (NULL).

ORACLE TABLES

- This form of the INSERT command requires that you specify the names of the columns for which data are being stored.

```
INSERT INTO employee (emp_ssn, emp_last_name,
                      emp_first_name)
VALUES ('999111111', 'Bock', 'Douglas');
```

- The DELETE command is perhaps the simplest of the SQL statements
- It removes one or more rows from a table. Multiple table delete operations are not allowed in SQL
- The syntax of the DELETE command is:

```
DELETE FROM table_name
WHERE condition;
```

ORACLE TABLES

- Since the WHERE clause is optional, you can easily delete all rows from a table by omitting a WHERE clause. WHERE clause limits the scope of the DELETE operation
- For example, the DELETE FROM command shown here removes all rows in the *assignment* table.

```
DELETE FROM assignment;
```

- Values stored in individual columns of selected rows can be modified (updated) with the UPDATE command.
- The ALTER command changes the table structure, but leaves the table data unaffected
- The UPDATE command changes data in the table, not the table structure
- The general syntax of the UPDATE command is:

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

ORACLE TABLES

- INSERT, UPDATE, and DELETE commands are not committed to the database until the COMMIT statement is executed

- SQL command ROLLBACK can be issued immediately to cancel any database operations since the most recent COMMIT

- Like COMMIT, ROLLBACK is also a transaction managing command, however, it cancels operations instead of confirming them

LEARNING OBJECTIVES

- What are database schema objects?
- When & How we need to use it?
- Types of database schema objects

Tables	**VIEWS**	Sequences
Synonyms	Indexes	Clusters
Database Links	Snapshots	Procedures
Functions	Packages	

ORACLE VIEWS

- A database view is a *logical* or *virtual table* based on a query.
- It is useful to think of a *view* as a stored query.
- Views are created through use of a CREATE VIEW command that incorporates use of the SELECT statement.
- Views are queried just like tables.
- For presenting different information to different users.

ORACLE VIEWS

```
CREATE VIEW employee_parking (parking_space, last_name,
    first_name, ssn) AS
SELECT emp_parking_space, emp_last_name,
emp_first_name, emp_ssn
FROM employee
ORDER BY emp_parking_space;
```

View Created.

ORACLE VIEWS

```
SELECT *
FROM employee_parking;

PARKING_SPACE LAST_NAME  FIRST_NAME   SSN
------------- ---------- -----------  --------
            1 Bordoloi   Bijoy        999666666
            3 Joyner     Suzanne      999555555
           32 Zhu        Waiman       999444444
```

Notice that the only columns in the query are those defined as part of the view.

ORACLE VIEWS

Virtual views:

- Used in databases
- Computed only on-demand – slower at runtime
- Always up to date

Materialized views

- Used in data warehouses
- Precomputed offline – faster at runtime
- May have stale data

- Additionally, we have renamed the columns in view so that they are slightly different than the column names in the underlying employee table

- Further, the rows are sorted by *parking_space* column even though there is no ORDER BY in the SELECT command used to access the view

ORACLE VIEWS

CREATE VIEW

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW <view name>
[(column alias name….)] AS <query> [WITH [CHECK OPTION]
[READ ONLY] [CONSTRAINT]];
```

- The OR REPLACE option is used to create a view that already exists. This option is useful for modifying an existing view without having to drop or grant the privileges that system users have acquired with respect to the view

- If you attempt to create a view that already exists without using the OR REPLACE option, Oracle will return the ORA-00955: *name is already used by an existing object* error message and the CREATE VIEW command will fail

ORACLE VIEWS

- The FORCE option allows a view to be created even if a base table that the view references does not already exist

- This option is used to create a view prior to the actual creation of the base tables and accompanying data. Before such a view can be queried, the base tables must be created and data must be loaded into the tables. This option can also be used if a system user does not currently have the privilege to create a view

- The NOFORCE option is the opposite of FORCE and allows a system user to create a view if they have the required permissions to create a view, and if the tables from which the view is created already exist. This is the default option

ORACLE VIEWS

```
CREATE VIEW empview7 AS
SELECT emp_ssn, emp_first_name, emp_last_name
FROM employee
WHERE emp_dpt_number=7;
```

View created.

- A simple query of the empview7 shows the following data.

```
SELECT *
FROM empview7;

EMP_SSN   EMP_FIRST_NAME            EMP_LAST_NAME
--------- ------------------------- -------------------------
999444444 Waiman                    Zhu
999111111 Douglas                   Bock
999333333 Dinesh                    Joshi
999888888 Sherri                    Prescott
```

ORACLE VIEWS

- It is also possible to create a view that has exactly the same structure as an existing database table.

- The view named *dept_view* shown next has exactly the same structure as *department* table.

```
CREATE VIEW dept_view AS
SELECT *
FROM department;
```

View created.

ORACLE VIEWS

- In addition to specifying columns from existing tables, you can use single row functions consisting of number, character, date, and group functions as well as expressions to create additional columns in views.

- This can be extremely useful because the system user will have access to data without having to understand how to use the underlying functions

- MIN
- MAX
- AVG
- COUNT
- SUM

ORACLE VIEWS

```
CREATE OR REPLACE VIEW dept_salary
    (name, min_salary, max_salary, avg_salary) AS
SELECT d.dpt_name, MIN(e.emp_salary),
    MAX(e.emp_salary), AVG(e.emp_salary)
FROM employee e, department d
WHERE e.emp_dpt_number=d.dpt_no
GROUP BY d.dpt_name;
```

View created.

```
SELECT *
FROM dept_salary;
NAME                         MIN_SALARY          MAX_SALARY     AVG_SALARY
------------------------- -------------  ------------------    -----------

Admin and Records                 25000               43000          31000
Headquarters                      55000               55000          55000
Production                        25000               43000          34000
```

ORACLE VIEWS

- A view does not actually store any data. The data needed to support queries of a view are retrieved from the underlying database tables and displayed to a result table whenever a view is queried. The result table is only stored temporarily.
- If a table that underlies a view is dropped, then the view is no longer valid. Attempting to query an invalid view will produce an ORA-04063: view "VIEW_NAME" has errors error message.

- You can insert a row if the view in use is one that is updateable (not read only).
- A view is updateable if the INSERT command does not violate any constraints on the underlying tables.
- This rule concerning constraint violations also applies to UPDATE and DELETE commands

ORACLE VIEWS

```
CREATE OR REPLACE VIEW dept_view AS
SELECT dpt_no, dpt_name
FROM department;

INSERT INTO dept_view VALUES (18, 'Department 18');
INSERT INTO dept_view VALUES (19, 'Department 20');

SELECT *
FROM dept_view;

DPT_NO        DPT_NAME
------------  --------------------
           7  Production
           3  Admin and Records
           1  Headquarters
          18  Department 18
          19  Department 20
```

ORACLE VIEWS

```
UPDATE dept_view SET dpt_name = 'Department 19‘
 WHERE dpt_no = 19;
```

1 row updated.

```
SELECT *
FROM department
WHERE dpt_no >= 5;

 DPT_NO      DPT_NAME         DPT_MGRSS          DPT_MGR_S
----------  ---------------  ----------------  -----------
         7  Production              999444444    22-MAY-98
        18  Department 18
        19  Department 19
```

ORACLE VIEWS

```
DELETE dept_view
 WHERE dpt_no = 18 OR dpt_no = 19;
```

2 rows deleted.

```
SELECT *
FROM department;
```

```
DPT_NO   DPT_NAME                     DPT_MGRSS         DPT_MGR_S
-------- ------------------------    ---------------  ----------
      7  Production                   999444444         22-MAY-98
      3  Admin and Records            999555555         01-JAN-01
      1  Headquarters                 999666666         19-JUN-81
```

ORACLE VIEWS

- If there are no syntax errors in a CREATE VIEW statement, Oracle will create a view even if the view-defining query refers to a non-existent table or an invalid column of an existing table.

- The view will also be created even if the system user does not have privileges to access the tables which a view references.

- The new view will be unusable and is categorized as “created with errors.”

- In order to create such a view, the system user must use the FORCE option of the CREATE VIEW command.

ORACLE VIEWS

- In the CREATE VIEW command shown below, the table named *divisions* does not exist and the view is created with errors. Oracle returns an appropriate warning message.

```
CREATE FORCE VIEW div_view AS
SELECT *
FROM divisions;

Warning: View created with compilation errors.
```

- If we now create a table named divisions, a query of the invalid div_view view will execute, and the view is automatically recompiled and becomes valid.

ORACLE VIEWS

- A DBA or view owner can drop a view with the DROP VIEW command. The following command drops a view named *dept_view*.

```
DROP VIEW dept_view;
```

```
View dropped
```

LEARNING OBJECTIVES

- What are database schema objects?
- When & How we need to use it?
- Types of database schema objects

Tables	Views	Sequences
Synonyms	Indexes	Clusters
Database Links	Snapshots	**PROCEDURES**
FUNCTIONS	**PACKAGES**	

PROGRAM UNITS

- Program unit

  Self-contained group of program statements that can be used within larger program

- Anonymous PL/SQL programs

  Programs that do not interact with other program units

- Stored PL/SQL program units

  Programs that other programs can reference

  Programs that other DB users can execute

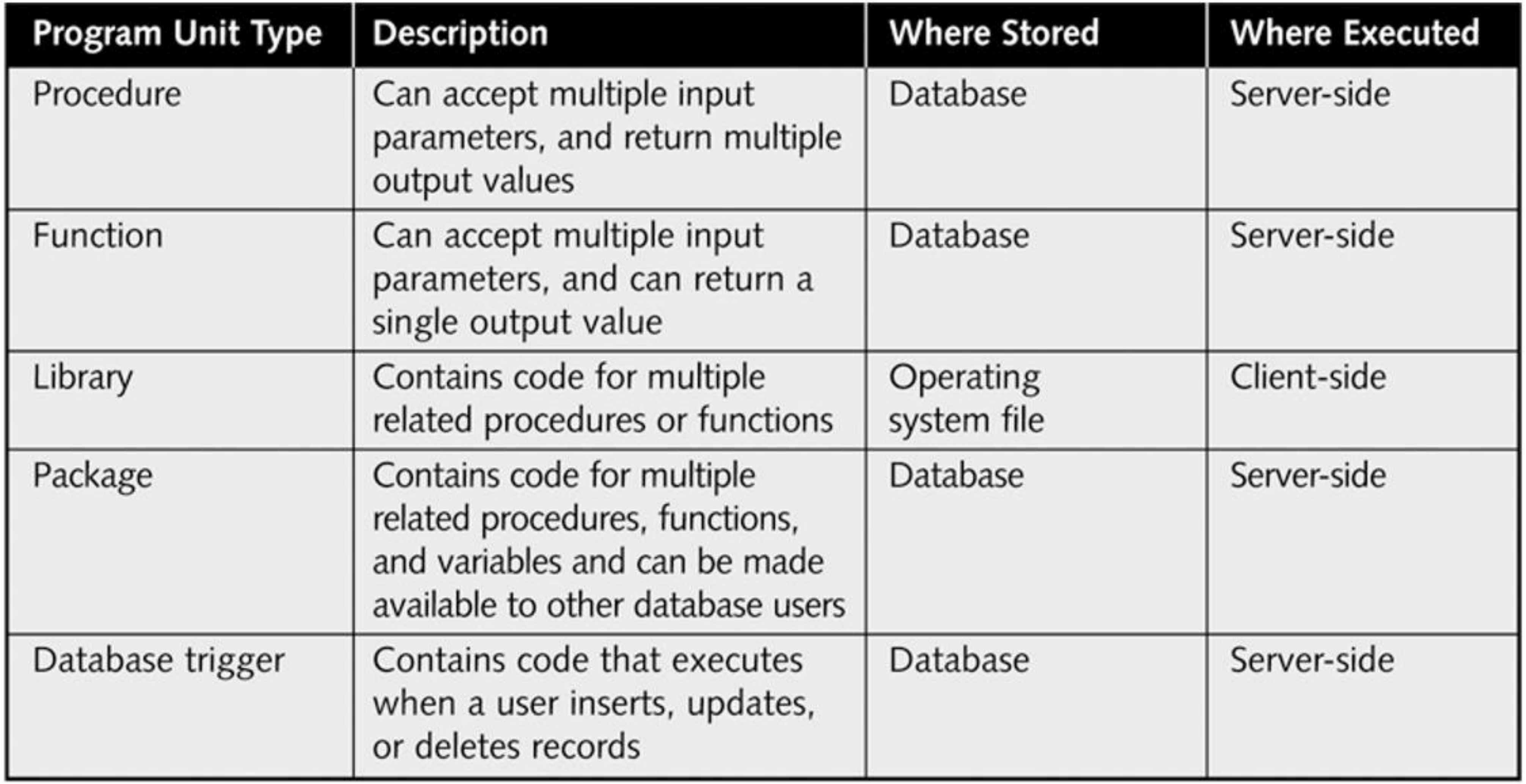
- Server-side program units

  Stored as DB objects and execute on the DB server

- Client-side program units

  Stored in the workstation's file system & execute on the client

PROGRAM UNITS



Program Unit Type	Description	Where Stored	Where Executed
Procedure	Can accept multiple input parameters, and return multiple output values	Database	Server-side
Function	Can accept multiple input parameters, and can return a single output value	Database	Server-side
Library	Contains code for multiple related procedures or functions	Operating system file	Client-side
Package	Contains code for multiple related procedures, functions, and variables and can be made available to other database users	Database	Server-side
Database trigger	Contains code that executes when a user inserts, updates, or deletes records	Database	Server-side

PROCEDURES

- Oracle subprograms - includes both procedures and functions.

- Both procedures and functions:
  - Can be programmed to perform a data processing task.
  - Are named PL/SQL blocks, and both can be coded to take parameters to generalize the code.
  - Can be written with declarative, executable, and exception sections.

- Functions are typically coded to perform some type of calculation.

- Primary difference - procedures are called with PL/SQL statements while functions are called as part of an expression.

PROCEDURES

- Normally stored in the database within package specifications - a package is a sort of wrapper for a group of named blocks.
- Can be stored as individual database objects.
- Are parsed and compiled at the time they are stored.
- Compiled objects execute faster than nonprocedural SQL scripts because nonprocedural scripts require extra time for compilation.
- Can be invoked from most Oracle tools like SQL*Plus, and from other programming languages like C++ and JAVA.
- Procedures are named PL/SQL blocks.
- Created/owned by a particular schema
- Privilege to execute a specific procedure can be granted to or revoked from application users in order to control data access.
- Requires CREATE PROCEDURE (to create in your schema) or CREATE ANY PROCEDURE privilege (to create in other schemas).

PROCEDURES

Improved data security – controls access to database objects while enabling non-privileged application users to access just the data needed.

Improved data integrity – related actions on database tables are performed as a unit enforcing transaction integrity – all updates are executed or none are executed.

Improved application performance – avoids reparsing objects used by multiple users through the use of shared SQL for Oracle – reduces number of database calls thus reducing network traffic.

Improved maintenance – procedures and functions that perform common tasks can be modified without having to directly work on multiple applications that may call these common procedures and functions – this approach eliminates duplicate testing

PROCEDURES

```
CREATE [OR REPLACE] PROCEDURE <procedure_name> (<parameter1_name>
<mode> <data type>, <parameter2_name> <mode> <data type>, ...)
{AS|IS}
    <Variable declarations>
BEGIN
    Executable statements
[EXCEPTION
    Exception handlers]
END <optional procedure name>;
```

- Unique procedure name is required.
- OR REPLACE clause facilitates testing.
- Parameters are optional – enclosed in parentheses when used.
- AS or IS keyword is used – both work identically.
- Procedure variables are declared prior to the BEGIN keyword.

PROCEDURES

- To Compile/Load a procedure use either the “@” symbol or the START SQL command to compile the file. The *<SQL filename>* parameter is the .sql file that contains the procedure to be compiled.

```
SQL>@<SQL filename>
SQL>start <SQL filename>
```

- Filename does not need to be the same as the procedure name. The .sql file only contains the procedure code.
- Compiled procedure is stored in the database, not the .sql file.
- Use SHOW ERRORS command if the procedure does not compile without errors. Use EXECUTE to run procedure.

```
SQL> show errors;
SQL> EXECUTE Insert_Employee
```

PROCEDURES

- Both procedures and functions can take parameters.
- Values passed as parameters to a procedure as arguments in a calling statement are termed *actual parameters*.
- The parameters in a procedure declaration are called *formal parameters*.
- The values stored in actual parameters are values passed to the formal parameters – the formal parameters are like placeholders to store the incoming values.
- When a procedure completes, the actual parameters are assigned the values of the formal parameters.
- A formal parameter can have one of three possible modes:

  IN

  OUT

  IN OUT

PROCEDURES

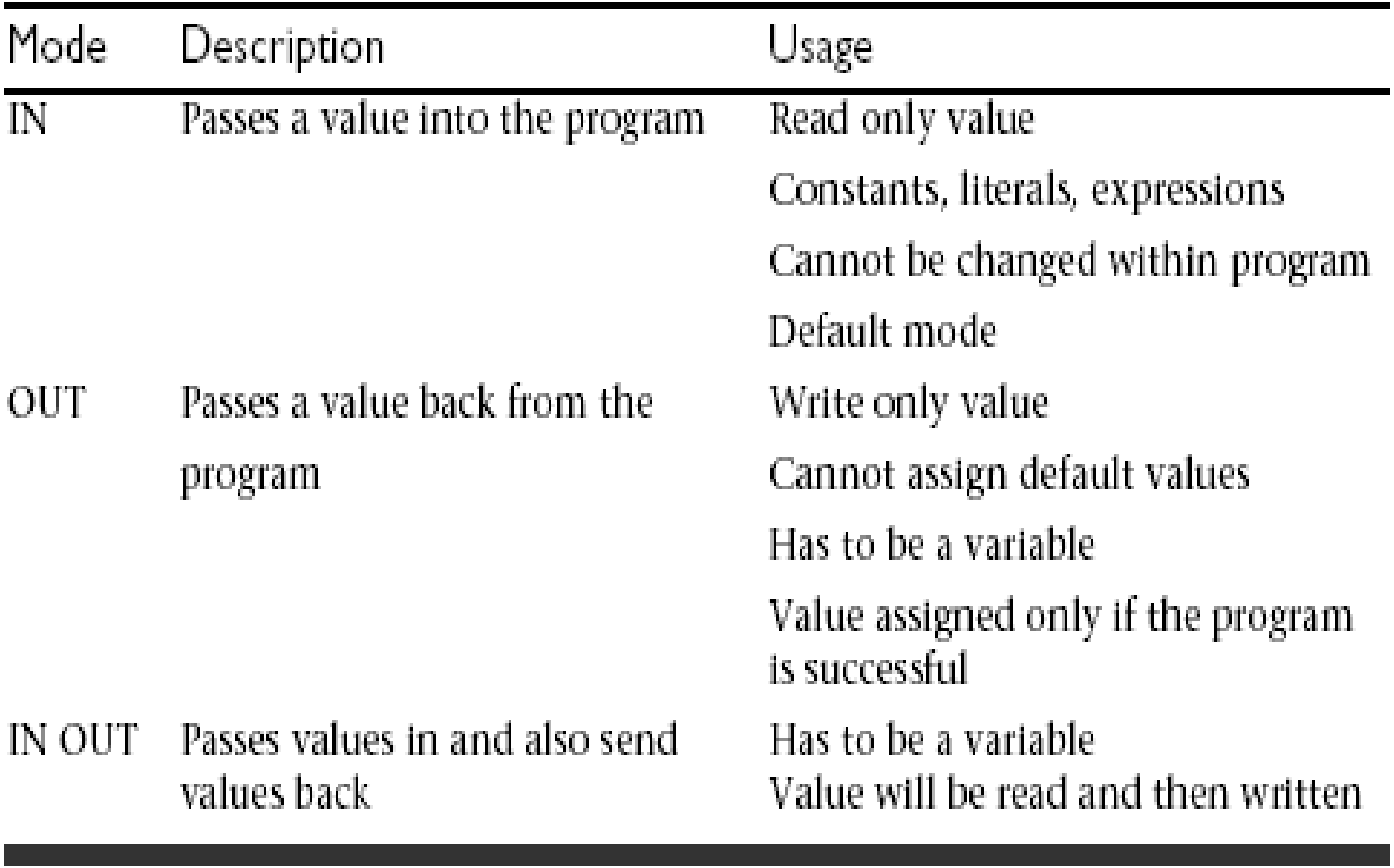
IN – This parameter type is passed to a procedure as a read-only value that cannot be changed within the procedure – this is the default mode.

OUT – This parameter type is write-only, and can only appear on the left side of an assignment statement in the procedure – it is assigned an initial value of NULL.

IN OUT – This parameter type combines both IN and OUT; a parameter of this mode is passed to a procedure, and its value can be changed within the procedure.

If a procedure raises an exception, the formal parameter values are not copied back to their corresponding actual parameters.

PROCEDURES



Mode	Description	Usage
IN	Passes a value into the program	Read only value Constants, literals, expressions Cannot be changed within program Default mode
OUT	Passes a value back from the program	Write only value Cannot assign default values Has to be a variable Value assigned only if the program is successful
IN OUT	Passes values in and also send values back	Has to be a variable Value will be read and then written

PROCEDURES

- Procedures do not allow specifying a constraint on the parameter data type.
- Example: the following CREATE PROCEDURE statement is not allowed because of the specification that constrains the *v_Variable* parameter to NUMBER(2). Instead use the general data type of NUMBER.

```
/* Invalid constraint on parameter. */
CREATE OR REPLACE PROCEDURE proSample (v_Variable
NUMBER(2), ...)

/* Valid parameter. */
CREATE OR REPLACE PROCEDURE proSample (v_Variable
NUMBER, ...)
```

PROCEDURES

```
CREATE OR REPLACE PROCEDURE UpdateEquipment (
    p_EquipmentNumber IN Equipment.EquipmentNumber%TYPE,
    p_Description IN Equipment.Description%TYPE,
    p_Cost IN Equipment.OriginalCost%TYPE,
    p_Quantity IN Equipment.QuantityAvailable%TYPE,
    p_Project IN Equipment.ProjectNumber%TYPE )
AS
    e_EquipmentNotFound EXCEPTION;
    v_ErrorTEXT VARCHAR2(512);
```

PROCEDURES

```
BEGIN
    UPDATE Equipment SET Description = p_Description,
        OriginalCost = p_Cost, QuantityAvailable =
        p_Quantity, ProjectNumber = p_Project
        WHERE EquipmentNumber = p_EquipmentNumber;
    IF SQL%ROWCOUNT = 0 THEN
        Raise e_EquipmentNotFound;
    END IF;
EXCEPTION
    WHEN e_EquipmentNotFound THEN
        DBMS_OUTPUT.PUT_LINE ('Invalid Equipment Number: '
            ||p_EquipmentNumber);
    WHEN OTHERS THEN
        v_ErrorText := SQLERRM;
        DBMS_OUTPUT.PUT_LINE ('Unexpected error'
            ||v_ErrorText);
END UpdateEquipment;
/
```

PROCEDURES

```
DECLARE
    v_EquipmentNumber Equipment.EquipmentNumber%TYPE:='5000';
    v_Description Equipment.Description%TYPE := 'Printer';
    v_Cost Equipment.OriginalCost%TYPE := 172.00;
    v_Quantity Equipment.QuantityAvailable%TYPE := 2;
    v_Project Equipment.ProjectNumber%TYPE := 5;
BEGIN
    UpdateEquipment(v_EquipmentNumber, v_Description,
        v_Cost, v_Quantity, v_Project);
END;
/
```

PROCEDURES

- There are several points that you need to understand about calling a procedure and the use of parameters for this example.
- The *UpdateEquipment* procedure is first created, compiled, and stored in the database as a compiled object.
- The actual parameters are declared within PL/SQL Example 13.2 and assigned values – the assigned values here merely illustrate that the parameters would have values that are passed to the *UpdateEquipment* procedure.
- The calling statement is a PL/SQL statement by itself and is not part of an expression – control will pass from the calling statement to the first statement inside the procedure.
- Because the formal parameters in *UpdateEquipment* are all declared as mode IN, the values of these parameters cannot be changed within the procedure.

PROCEDURES

```
CREATE OR REPLACE PROCEDURE DisplaySalary IS
    -- create local variable with required constraint
    temp_Salary NUMBER(10,2);
BEGIN
     SELECT Salary INTO temp_Salary FROM Employee
     WHERE EmployeeID = '01885';
     IF temp_Salary > 15000 THEN
         DBMS_OUTPUT.PUT_LINE ('Salary > 15,000.');
     ELSE
         DBMS_OUTPUT.PUT_LINE ('Salary < 15,000.');
     END IF;
EXCEPTION
     WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE ('Employee not found.');
END DisplaySalary;
/
```

PROCEDURES

```
SQL> @ DisplaySalary.sql
Procedure created.

SQL> exec DisplaySalary
Salary > 15,000.
PL/SQL procedure successfully completed.
```

PROCEDURES

```
CREATE OR REPLACE PROCEDURE DisplaySalary2(p_EmployeeID
    IN CHAR, p_Salary OUT NUMBER) IS
    v_Salary NUMBER(10,2);
BEGIN
    SELECT Salary INTO v_Salary FROM Employee
    WHERE EmployeeID = p_EmployeeID;
    IF v_Salary > 15000 THEN
        DBMS_OUTPUT.PUT_LINE ('Salary > 15,000.');
    ELSE
        DBMS_OUTPUT.PUT_LINE ('Salary <= 15,000.');
    END IF;
    p_Salary := v_Salary;
EXCEPTION
     WHEN NO_DATA_FOUND THEN
          DBMS_OUTPUT.PUT_LINE ('Employee not found.');
END DisplaySalary2;
```

PROCEDURES

```
DECLARE
    v_SalaryOutput NUMBER := 0;
BEGIN
    -- call the procedure
    DisplaySalary2('01885', v_SalaryOutput);
    -- display value of salary after the call
    DBMS_OUTPUT.PUT_LINE ('Actual salary: '
        ||TO_CHAR(v_SalaryOutput));
END;
/

Salary > 15,000.
Actual salary: 16250
PL/SQL procedure successfully completed.
```

PROCEDURES

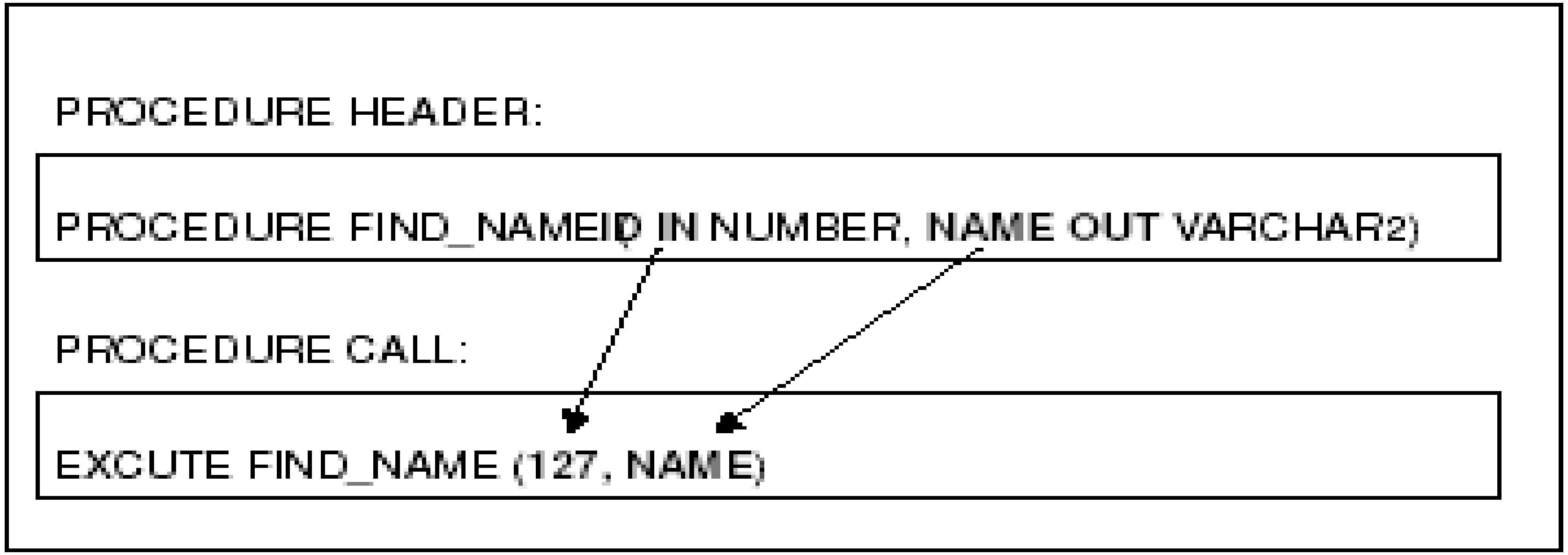
- Another approach to test a procedure. This approach uses a *bind variable* in Oracle.
- A bind variable is a variable created at the SQL*Plus prompt that is used to reference variables in PL/SQL subprograms.
- A bind variable used in this fashion must be prefixed with a colon “:” – this syntax is required.

```
SQL> var v_SalaryOutput NUMBER;
SQL> EXEC DisplaySalary2('01885', :v_SalaryOutput);
Salary > 15,000.
PL/SQL procedure successfully completed.

SQL> PRINT v_SalaryOutput;

V_SALARYOUTPUT
--------------
         16250
```

PROCEDURES



PROCEDURE HEADER:

PROCEDURE FIND_NAMEID IN NUMBER, NAME OUT VARCHAR2)

PROCEDURE CALL:

EXCUTE FIND_NAME (127, NAME)

PROCEDURES

- The SQL statement to drop a procedure is the straight-forward DROP PROCEDURE <procedureName> command.
- This is a data definition language (DDL) command, and so an implicit commit executes prior to and immediately after the command.

```
SQL> DROP PROCEDURE DisplaySalary2;
Procedure dropped.
```

FUNCTIONS

- Like a procedure, a function can accept multiple parameters, and the data type of the return value must be declared in the header of the function.

```
CREATE [OR REPLACE] FUNCTION <function_name> (<parameter1_name>
<mode> <data type>,
   <parameter2_name> <mode> <data type>, ...)
RETURN <function return value data type> {AS|IS}
    <Variable declarations>
BEGIN
    Executable Commands
    RETURN (return_value);
    . . .
[EXCEPTION
    Exception handlers]
END;
```

- The general syntax of the RETURN statement is:

```
RETURN <expression>;
```

FUNCTIONS

```
CREATE OR REPLACE FUNCTION RetrieveSalary
     RETURN NUMBER
IS
     v_Salary NUMBER(10,2);
BEGIN
     SELECT Salary INTO v_Salary
     FROM Employee
     WHERE EmployeeID = '01885';
     RETURN v_Salary;
END RetrieveSalary;
/
```

FUNCTIONS

```
SQL> @RetrieveSalary
Function created.

SQL> var v_SalaryOutput NUMBER;
SQL> EXEC :v_SalaryOutput := RetrieveSalary;

PL/SQL procedure successfully completed.

SQL> print v_SalaryOutput;

V_SALARYOUTPUT
--------------
         16250
```

FUNCTIONS

PL/SQL Example illustrates a function that has a single IN parameter and that returns a VARCHAR2 data type.

```
CREATE OR REPLACE FUNCTION FullName (p_EmployeeID IN
        employee.EmployeeID%TYPE)
    RETURN VARCHAR2 IS
    v_FullName VARCHAR2(100);
    v_FirstName employee.FirstName%TYPE;
    v_MiddleName employee.MiddleName%TYPE;
    v_LastName employee.LastName%TYPE;
BEGIN
    SELECT FirstName, MiddleName, LastName INTO
        v_FirstName, v_MiddleName, v_LastName
        FROM Employee
        WHERE EmployeeID = p_EmployeeID;
```

FUNCTIONS

```
-- Store last name, comma and blank and first name to variable
    v_FullName := v_LastName||', '||v_FirstName;

-- Check for existence of a middle name
    IF LENGTH(v_MiddleName) > 0 THEN
        v_FullName := v_FullName|| ' '
            ||SUBSTR(v_MiddleName,1,1)||'.';
    END IF;
    RETURN v_FullName;
END FullName;
/
```

FUNCTIONS

A simple SELECT statement executed within SQL*Plus can return the full name for any employee identifier value as shown in PL/SQL Example 13.10.

```
SQL> SELECT FullName('01885')
     FROM Employee
     WHERE EmployeeID = '01885';

FULLNAME('01885')
-----------------------------------
Bock, Douglas B.
```

FUNCTIONS

```
SQL> SELECT FullName(EmployeeID)
     FROM Employee
     ORDER BY FullName(EmployeeID);


FULLNAME(EMPLOYEEID)
-------------------------------------
Adams, Adam A.
Barlow, William A.
Becker, Robert B.
Becker, Roberta G.
Bock, Douglas B.
... more rows will display
```

FUNCTIONS

- As with the DROP PROCEDURE statement, the DROP FUNCTION <functionName> is also straight-forward.

- As with DROP PROCEDURE, the DROP FUNCTION statement is a DDL command that causes execution of an implicit commit prior to and immediately after the command.

```
SQL> DROP FUNCTION FullName;
Function dropped.
```

PACKAGES

- A *package* is a collection of PL/SQL objects grouped together under one package name.
- Packages provide a means to collect related procedures, functions, cursors, declarations, types, and variables into a single, named database object that is more flexible than the related database objects are by themselves.
- Package variables – can be referenced in any procedure, function, (other object) defined within a package.

PACKAGES

- A package consists of a package specification and a package body.
- The *package specification*, also called the package header.
- Declares global variables, cursors, exceptions, procedures, and functions that can be called or accessed by other program units.
- A package specification must be a uniquely named database object.
- Elements of a package can be declared in any order. If element “A” is referenced by another element, then element “A” must be declared before it is referenced by another element. For example, a variable referenced by a cursor must be declared before it is used by the cursor.
- This means the declaration only includes the subprogram name and arguments, but does not include the actual program code.

PACKAGES

- Basically, a package is a named declaration section.
- Any object that can be declared in a PL/SQL block can be declared in a package.
- Use the CREATE OR REPLACE PACKAGE clause.
- Include the specification of each named PL/SQL block header that will be public within the package.
- Procedures, functions, cursors, and variables that are declared in the package specification are *global*.
- The basic syntax for a package is:

```
CREATE [OR REPLACE PACKAGE[ <package name> {AS|IS}
    <variable declarations>;
    <cursor declarations>;
    <procedure and function declarations>;
END <package name>;
```

PACKAGES

- To declare a procedure in a package - specify the procedure name, followed by the parameters and variable types:

```
PROCEDURE <procedure_name> (param1 datatype, param2 datatype, ...);
```

- To declare a function in a package, you must specify the function name, parameters and return variable type:

```
FUNCTION <function_name> (param1 datatype, param2 datatype, ...)
RETURN <return data type>;
```

PACKAGES

- Contains the code for the subprograms and other constructs, such as exceptions, declared in the package specification.

- Is optional – a package that contains only variable declarations, cursors, and the like, but no procedure or function declarations does not require a package body.

- Any subprograms declared in a package must be coded completely in the package body. The procedure and function specifications of the package body must match the package declarations including subprogram names, parameter names, and parameter modes.

PACKAGES

Use the CREATE OR REPLACE PACKAGE BODY clause to create a package body. The basic syntax is:

```
CREATE [OR REPLACE] PACKAGE BODY <package name> AS
    <cursor specifications>
    <subprogram specifications and code>
END <package name>;
```

PACKAGES

```
CREATE OR REPLACE PACKAGE ManageEmployee AS
    -- Global variable declarations go here
    -- Procedure to find employees
    PROCEDURE FindEmployee(
        emp_ID        IN employee.EmployeeID%TYPE,
        emp_FirstName OUT employee.FirstName%TYPE,
        emp_LastName  OUT employee.LastName%TYPE);
    -- Exception raised by FindEmployee
    e_EmployeeIDNotFound EXCEPTION;
    -- Function to determine if employee identifier is valid
    FUNCTION GoodIdentifier(
        emp_ID        IN employee.EmployeeID%TYPE)
        RETURN BOOLEAN;
END ManageEmployee;
/
```

PACKAGES

```
CREATE OR REPLACE PACKAGE BODY ManageEmployee AS
    -- Procedure to find employees
    PROCEDURE FindEmployee(
        emp_ID        IN employee.EmployeeID%TYPE,
        emp_FirstName OUT employee.FirstName%TYPE,
        emp_LastName  OUT employee.LastName%TYPE ) AS
    BEGIN
        SELECT FirstName, LastName
        INTO emp_FirstName, emp_LastName
        FROM Employee
        WHERE EmployeeID = emp_ID;

        -- Check for existence of employee
        IF SQL%ROWCOUNT = 0 THEN
            RAISE e_EmployeeIDNotFound;
        END IF;
    END FindEmployee;
```

PACKAGES

```
-- Function to determine if employee identifier is valid
    FUNCTION GoodIdentifier(
        emp_ID        IN employee.EmployeeID%TYPE)
        RETURN BOOLEAN
    IS
        v_ID_Count NUMBER;
    BEGIN
        SELECT COUNT(*) INTO v_ID_Count
        FROM Employee
        WHERE EmployeeID = emp_ID;

        -- return TRUE if v_ID_COUNT is 1
        RETURN (1 = v_ID_Count);
    EXCEPTION
        WHEN OTHERS THEN
            RETURN FALSE;
    END GoodIdentifier;
END ManageEmployee;
```

PACKAGES

```
DECLARE
    v_FirstName   employee.FirstName%TYPE;
    v_LastName    employee.LastName%TYPE;
    search_ID     employee.EmployeeID%TYPE;
BEGIN
    ManageEmployee.FindEmployee (&search_ID, v_FirstName,
        v_LastName);
    DBMS_OUTPUT.PUT_LINE ('The employee name is: ' ||
        v_LastName || ', ' || v_FirstName);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('Cannot find an employee with that
ID.');
END;
/
```

PACKAGES

- When the employee identifier is valid, the code displays the employee name as shown here.

```
Enter value for search_id: '01885'
The employee name is: Bock, Douglas
PL/SQL procedure successfully completed.
```

- When the identifier is not valid, the exception raised within the called procedure is propagated back to the calling procedure and is trapped by the EXCEPTION section’s WHEN OTHERS clause and an appropriate message is displayed as shown here.

```
Enter value for search_id: '99999'
Cannot find an employee with that ID.
PL/SQL procedure successfully completed.
```

RECAP

- Oracle TABLES are used to save or retain data in database
- There are four pieces of information must be determined

1. The table name
2. The column (field) names
3. Column data types and
4. Column sizes

- Create, Drop, Alter table structures
- Insert into, Delete from, Update table data
- Commit or Rollback table data

RECAP

- A view does not store data, but a view does display data through a SELECT query as if the data were stored in the view.
- A view definition as provided by the CREATE VIEW statement is stored in the database. Further, Oracle develops what is termed an "execution plan" that is used to "gather up" the data that needs to be displayed by a view. This execution plan is also stored in the database.
- A view can simplify data presentation as well as provide a kind of data security by limiting access to data based on a "need to know."
- A view can display data from more than one table.
- Views can be used to update the underlying tables. Views can also be limited to read-only access.