CENTER OF INFORMATION TECHNOLOGY AND SCIENTIFIC COMPUTING

Department of Software Engineering

Database Administration (DBA) *Assignment One*

Submitted by Henok Addis

Id: ATR/6271/09

Submitted to Mr. Natnael A

Submitted Date: 07/10/2019

Database Objects

Oracle Database recognizes objects that are associated with a particular schema and objects that are not associated with any particular schema, as described in the sections that follow.

Schema Objects

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 include the following types of objects:

Clusters
Constraints
Database links
Database triggers
Dimensions
Index-organized tables
Indexes
Object types
Object views
Operators
Packages
Sequences
Synonyms
Tables
Views

No schema Objects

Other types of objects are also stored in the database and can be created and manipulated with SQL but are not contained in a schema:

Contexts
Directories
Editions
Restore points
Roles
Rollback segments
Tablespaces Users

Implementation of database objects

1. Clusters

Database Clustering is the process of combining more than one servers or instances connecting a single database. Sometimes one server may not be adequate to manage the amount of data or the number of requests, that is when a Data Cluster is needed. Database clustering, SQL server clustering, and SQL clustering are closely associated with SQL is the language used to manage the database information.

The main reasons for database clustering are its advantages a server receives; Data redundancy, Load balancing, High availability, and lastly, Monitoring and automation

To create a cluster in another user's schema you must have the CREATE ANY CLUSTER system privilege, and the owner must have a quota for the tablespace intended to contain the cluster or the UNLIMITED TABLESPACE system privilege.

You create a cluster using the CREATE CLUSTER statement. The following statement creates a cluster named emp_dept, which stores the emp and dept tables, clustered by the deptno column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
  SIZE 600
  TABLESPACE users
  STORAGE (INITIAL 200K
    NEXT 300K
    MINEXTENTS 2
    PCTINCREASE 33);
```

Creating Clustered Tables

To create a table in a cluster, you must have either the CREATE TABLE or CREATE ANY TABLE system privilege. You do not need a tablespace quota or the UNLIMITED TABLESPACE system privilege to create a table in a cluster.

You create a table in a cluster using the CREATE TABLE statement with the CLUSTER clause. The emp and dept tables can be created in the emp_dept cluster using the following statements:

```
CREATE TABLE emp (    empno
NUMBER(5) PRIMARY KEY,    ename
VARCHAR2(15) NOT NULL,
  . . .
  deptno NUMBER(3) REFERENCES dept)
  CLUSTER emp_dept (deptno);


CREATE TABLE dept (    deptno
NUMBER(3) PRIMARY KEY, . . . )
  CLUSTER emp_dept (deptno);
```

2. Constraints

⇨ They are constants and indicators that restricts the database rules and writes and access rights.

1.Not Null – each value in each column must not be null

```
Create table table_name
(
  column1 datatype [ NOT NULL ],
  column2 datatype [ NOT NULL ],
  ...
  column_n datatype [NOT NULL ]

);
```

When we insert the data in the table the above constraint show the data must be specified in each columns.

2.uniques – values specified in columns must be unique for each rows.

```
Create table table_name
(
  column1 datatype [ NOT NULL],
  column2 datatype [ NOT NULL ],
  ...
  column_n datatype [NOT NULL ]
CONSTRAINT constraint_name UNIQUE(column_name condition) [DISABLE]
);
```

From the above column one its primary key for each rows. And its unique

3.primary key – values specified in columns must be unique for each row and not null.

4.foreign key – values specified in column that reference values in another tables.

```
CREATE TABLE table_name
(
 column1 datatype null/not null,
 column2 datatype null/not null,
 ?
 CONSTRAINT constraint_name FOREIGN KEY (reference_key) references
other_table_name
);
```

5. check - an expression is specified , for constraints that satisfied true.

```
CREATE TABLE table_name
(
 column1 datatype null/not null,
 column2 datatype null/not null,
 ?
 CONSTRAINT constraint_name CHECK (column_name condition) [DISABLE]
);
```

This show the database checks the checks constraint if its met before it runs.

3. **A database trigger is** fired when a DML operation **is** fired on some **database** table.

Triggers

- is procedural code that automatically executed in response of certain events happens on particular table or views.

Triggers Fires When the following events Occurs.

- Data Manipulation Language statements occurs on certain table or row.
    - This means inserting, updating or Deleting the data from the databases.

    - Data Definition Language statements occurs on certain table or Row or databases.
        - Creating, updating the table of database or table and row that means we define the data from start.
    - Database Event – such as start or login to database and logout from database

      Advantages of Triggers
    - Prevents from invalid transactions
    - Provide auditing and event logging
    - Recording the Information about the table Access.

      Timing For Triggers
      Triggers fires before and after the action occurs.
    - Before the Firing Statement
      Create TRIGGER trigger_name
      BEFORE insert update or delete on table_name
      DECLARE
      Varname1 datatype(),
      Varname2 datatype
      BEGIN
      ---Trigger code
      When person performs insert, update or delete into **table_name**
      End;

        - We can create trigger using create trigger command with trigger_name that show the actions that we perform.
        - DECLARE
          ➔ This are variable or attributes that we perform operation on.

➔ There Trigger code the type of action performed with specified database operations.

Example
If we have Student table
Create table Student (
Name varchar (20),
Lastname varchar (20),
Age number (3));

We can make trigger when we make operation on student table
Create trigger student_trigger
Before
Insert into student (name, Lastname, age) values ('Yonas', 'Alem',21)
Begin

SQL Server triggers are special stored procedures that are executed automatically in response to the database object, database, and server events. SQL Server provides three type of triggers:

- Data manipulation language (DML) triggers which are invoked automatically in response to INSERT, UPDATE, and DELETE events against tables.
- Data definition language (DDL) triggers which fire in response to CREATE, ALTER, and DROP statements. DDL triggers also fire in response to some system stored procedures that perform DDL-like operations.
- Logon triggers which fire in response to LOGON events

The following illustrates the syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER [schema_name.]trigger_name
 ON table_name

AFTER  {[INSERT],[UPDATE],[DELETE]}
[NOT FOR REPLICATION] AS {sql_statements}
```

To list all triggers in a SQL Server, you query data from the sys.triggers view:

```
SELECT
   name,
```

```
    is_instead_of_trigger
FROM
    sys.triggers
WHERE
    type = 'TR';
```

The following picture shows the output:

name	is_instead_of_trigger
trg_product_audit	0
trg_members_insert	0
trg_members_delete	0
trg_vw_brands	1
trg_index_changes	0

To remove one or more DDL triggers, you use the following form of the `DROP TRIGGER` statement:

```
    DROP TRIGGER [ IF EXISTS ] trigger_name [ ,...n ]
ON { DATABASE | ALL SERVER };
```

4. **A database link** is a connection from the Oracle database to another remote database. The remote database can be an Oracle Database or any ODBC compliant database such as SQL Server or MySQL.

Database Link

- ⇨ This is schema object in one database that enables access to the other databases.
- ⇨ This allows the sharing of information or distribution
- ⇨ They acts a pointer that defines communication from one database server to another one.
- ⇨ They can be either private or public.
    - ➔ If its private the only user that make connection can access it.
    - ➔ If its public all database user can have access to it.

➔ CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';

This create the database link using some parameter called sales.

What is an Oracle database link

A database link is a connection from the Oracle database to another remote database. The remote database can be an Oracle Database or any ODBC compliant database such as SQL Server or MySQL.

Why do you need a database link

A database link allows a user or program to access database objects such as tables and views from another database.

Once you create a database link, you can access the tables or views from the remote database using the following pattern:

```
table_name@database_link
```

For example, you can query data from a table in the remote database as if it was in the local server:

```
SELECT * FROM remote_table@database_link;
```

The following statement shows how to create the private database link to a user in a remote database with a full connection string.

```
CREATE DATABASE LINK dblink
  CONNECT TO remote_user IDENTIFIED BY password
    USING '(DESCRIPTION=
      (ADDRESS=(PROTOCOL=TCP)(HOST=oracledb.example.com)(PORT=1521))
      (CONNECT_DATA=(SERVICE_NAME=service_name))
    )';
```

To create a public database link, just add the PUBLIC keyword:

```
CREATE PUBLIC DATABASE LINK dblink
```

```
    CONNECT TO remote_user IDENTIFIED BY password
USING 'remote_database';
```

Next, use the CREATE DATABASE LINK statement to create a new private database link that connects to the SALES database via bob's account:

```
CREATE DATABASE LINK sales
  CONNECT TO bob IDENTIFIED BY Abcd1234
   USING 'SALES';
```

Then, issue the SELECT statement to query data from the customers table on the SALES database:

```
SELECT * FROM customers@sales;
```

Here is the output:

CUSTOMER_ID	NAME	EMAIL
1	ABC Corp	contact@abccorp.com

After that, insert a new row into the customers table:

```
INSERT INTO customers@sales(customer_id, name, email) VALUES(2,'XYZ
Inc','contact@xyzinc.com');
```

Here are some best practices using the database links:

- Naming convention: the name of the database links should reflect the nature of data, not the database server. For example, instead of naming a database link SALES_PRD, you name it as SALES.
- Remote database users: you should create a user dedicated for a database link. In addition, you should not give this user to anyone else. If you don't follow this, the database will not work when someone changes the password of the user or even delete it.
- Use a service-specific entry in the tnsnames.ora instead of the databasespecific alias so that you copy between product, test, and development environments, you don't have to recreate the database link.

5. **View –** This database object is used to create a view in database.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. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Generally views are:

- are searchable objects that are defined in query.
- define what data the user sees and how the users see the data.
- they are virtual tables because they cannot stored actual data but they can show many tables as Imaginary big tables and we can query and flood over it.
- Views contains row and columns just like tables.
- The field for views are one or more fields from tables.

benefits of Database View

- Enforces Business Rules
- they show items and data that are dummy into logical views.
- Consistency - hides the complicated logic and show the views to the users.
- simplicity - hides the complicated tables joint and keys but the join as one large flattened data.
- restricts the access to tables
- they use little space because they only used storage for the query commands.

- but they have some issues in performance and modifications because they are viewed as one large big cluster of data so they need more computation power.
- they allow full Query operations.

Syntax :

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
        [(alias[, alias]...)]
        AS subquery
        [WITH CHECK OPTION [CONSTRAINT constraint]]
        [WITH READ ONLY [CONSTRAINT constraint]];
```

Example :

```
CREATE VIEW salvu50
        AS SELECT employee_id ID_NUMBER, last_name NAME,
        salary*12 ANN_SALARY
        FROM employees
        WHERE department_id = 50;
```

Output :

```
SELECT *
FROM salvu50;
```

6. Dimensions

A dimension is a schema object that defines hierarchical relationships between pairs of columns or column sets. A hierarchical relationship is a functional dependency from one level of a hierarchy to the next level in the hierarchy. A dimension is a container of logical relationships between columns and does not have any data storage assigned to it.

The CREATE DIMENSION statement specifies:

- multiple LEVEL clauses, each of which identifies a column or column set in the dimension
- one or more HIERARCHY clauses that specify the parent/child relationships between adjacent levels

- optional ATTRIBUTE clauses, each of which identifies an additional column or column set associated with an individual level

The columns in a dimension can come either from the same table *(denormalized)* or from multiple tables *(fully or partially normalized)*. To define a dimension over

columns from multiple tables, you connect the tables using the JOIN KEY option of the HIERARCHY clause.

For example, a normalized time dimension might include a date table, a month table, and a year table, with join conditions that connect each date row to a month row, and each month row to a year row. In a fully denormalized time dimension, the date, month, and year columns would all be in the same table. Whether normalized or denormalized, the hierarchical relationships among the columns need to be specified in the CREATE DIMENSION statement.

7. **Index –** This database object is used to create a indexes in database.An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer.Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. Once an index is created, no direct activity is required by the user.Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

Syntax :

```
CREATE INDEX index
        ON table (column[, column]...);
```

Example :

```
CREATE INDEX emp_last_name_idx
          ON  employees(last_name);
```

8. **Synonym –** This database object is used to create a indexes in database.It simplify access to objects by creating a synonym(another name for an object). With synonyms, you can Ease referring to a table owned by another user and shorten lengthy object names.To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence,procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:
PUBLIC : creates a synonym accessible to all users synonym
: is the name of the synonym to be created object : identifies
the object for which the synonym is created

Syntax :

```
CREATE [PUBLIC] SYNONYM synonym FOR  object;
```

Example :

```
CREATE SYNONYM d_sum FOR dept_sum_vu;
```

9. Indextypes

The *indextype* schema object encapsulates the set of routines that manage and access a domain index. The purpose of an indextype is to enable efficient search and retrieval functions for complex domains such as text, spatial, image, and OLAP data using external application software.

10. **Sequence –** This database object is used to create a sequence in database.A sequence is a user created database object that can be shared by multiple users to generate unique integers. A typical usage for sequences is to create a primary key value, which must be unique for each row.The sequence is generated and incremented (or decremented) by an internal Oracle routine. **Syntax :**

```
CREATE SEQUENCE sequence
        [INCREMENT BY n]
        [START WITH n]
        [{MAXVALUE n | NOMAXVALUE}]
        [{MINVALUE n | NOMINVALUE}]
        [{CYCLE | NOCYCLE}]
        [{CACHE n | NOCACHE}];
```

Example :

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```

Check if sequence is created by :

```
SELECT sequence_name, min_value, max_value,
                increment_by, last_number
                FROM   user_sequences;
```

Sequence

- ⇨ Allow to populate the primary with defined key.
- ⇨ We can use to generate the primary key automatically.
- ⇨ When the sequence number is generated they are incremented independent of the role-backs.
- ⇨ Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables.
  We can create sequence using create sequence command.

```
CREATE SEQUENCE [schema_name . ] sequence_name
   [ AS [ built_in_integer_type | user-defined_integer_type ] ]
   [ START WITH <constant> ]
   [ INCREMENT BY <constant> ]
   [ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
   [ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
   [ CYCLE | { NO CYCLE } ]
   [ { CACHE [ <constant> ] } | { NO CACHE } ]
        [ ; ]
```

This above syntax create sequence with name, defined datatype and validations.

Properties

- ⇨ Start with – kinds of data that sequence start with number, or characters , or other datatypes.
- ⇨ Increment by – this show by what value we increment on the sequences
- ⇨ Max and min value also specified.
- ⇨ Cycle – whether we it come back and assign the already generated number.

⇨ Cache – storage access.

11. Object type

Oracle object types are user-defined **types** that make it possible to model realworld entities such as customers and purchase orders as **objects** in the database.

```
create or replace type t_address as
object
(
   street char(20),
city char(20),
state char(2),
   zip char(5)
);
```

Object views provide the ability to offer specialized or restricted access to the data and objects in a database. For example, you might use an object view to provide a version of an employee object table that doesn't have attributes containing sensitive data and doesn't have a deletion method.

Object views allow the use of relational data in object-oriented applications. They let users

- Try object-oriented programming techniques without converting existing tables.
- Convert data gradually and transparently from relational tables to objectrelational tables.
- Use legacy RDBMS data with existing object-oriented applications.

Advantages of Object Views

Using object views can lead to better performance. Relational data that make up a row of an object view traverse the network as a unit, potentially saving many round trips.

You can fetch relational data into the client-side object cache and map it into *C or C++* structures so 3GL applications can manipulate it just like native structures.

Object views provide a gradual migration path for legacy data.

Object views provide for co-existence of relational and object-oriented applications. They make it easier to introduce object-oriented applications to existing relational data without having to make a drastic change from one paradigm to another.

Object views provide the flexibility of looking at the same relational or object data in more than one way. Thus you can use different in-memory object representations for different applications without changing the way you store the data in the database.

For example, the following SQL statements define an object view:

```
CREATE TABLE emp_table  (
empnum   NUMBER (5),
ename    VARCHAR2 (20),
salary   NUMBER (9, 2),     job
VARCHAR2 (20) );

CREATE TYPE employee_t  (
empno    NUMBER (5),     ename
VARCHAR2 (20),     salary
NUMBER (9, 2),
   job      VARCHAR2 (20) );

CREATE VIEW emp_view1 OF employee_t
   WITH OBJECT OID (empno) AS
      SELECT   e.empnum, e.ename, e.salary, e.job
      FROM     emp_table e
      WHERE    job = 'Developer';
```

12. Operator:

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators

- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators

Operator	Description	Example

=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators

Sr.No.	Operator & Description
1	**ALL** The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.

8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.
11	**UNIQUE** The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

13. **Packages :** are schema objects that groups logically related PL/SQL types, variables, and subprograms.

Package

- ⇨ They encapsulate related procedures, functions, associated cursors and variables as a unit in database.
- ⇨ They usually two parts
  1. Specification – is the interface with applications that access database
     - ➔ It declares variables, constants, exceptions, cursors and sub programs available for use.
  2. Body – defined the cursor and sub programs and have actual data with specific implementation.

Advantage

- ⇨ Encapsulation of related procedures, value and cursors.
- ⇨ Separation of private and public data.

⇨ Improve performance since the entire package is loaded when the object from the package called.

A package will have two mandatory parts –

- Package specification
- Package body or definition

Package Specification

The specification is the interface to the package. It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a **private** object.

The following code snippet shows a package specification having a single procedure. You can have many global variables defined and multiple procedures or functions inside a package.

```
CREATE PACKAGE cust_sal AS
  PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;
/
```

When the above code is executed at the SQL prompt, it produces the following result – Package created.

14. Contexts:

An alternative to the previous method is to use contexts to store the global data. A context is a set of application-defined attributes associated with a namespace that is linked to a managing package. A context is created using the CREATE CONTEXT statement in which the namespace equates to the context name.

CREATE CONTEXT namespace USING package-name;

The managing package does not have to be present at the point when the context is created, but must be present before it is referenced. The dbms_session.set_context procedure is used to associate name-value pairs with the context, but for security purposes this procedure can only be used from within the package associated with the context.

Context attributes can be read using the sys_context function available from SQL and PL/SQL. The following example uses this function to retrieve the current session identifier from the default userenv context.

```
SQL> SELECT SYS_CONTEXT('USERENV', 'SESSIONID') FROM dual;

SYS_CONTEXT('USERENV','SESSION
------------------------------------------------------------
328385

1 row selected.
```

To show how contexts can be used to cache global session data, the examples from the previous section using contexts instead of package variables will be recreated. In order to do this, the CREATE ANY CONTEXT privilege must be granted to the owner of the context, and just for the purpose of this example, the CREATE TRIGGER privilege is also necessary.

```
GRANT CREATE ANY CONTEXT TO username; GRANT
CREATE TRIGGER TO username;
```

15. Directory:

Creates a directory object that specifies an operating system directory for storing BFILE objects.

CREATE [OR REPLACE] DIRECTORY directory_name AS 'path_name'

Removes a directory object from the database.

DROP DIRECTORY *directory_name;*

16. Editions:

An edition is effectively a version label that can be assigned to all editionable objects in a schema. When a new edition is used by a schema, all editionable objects are inherited by the new edition from the previous edition. These objects can subsequently then be altered or dropped as desired, but doing so will stop inheritance of that object.

From Oracle database 11g release 2 onwards, each database has at least one edition, the default being ORA$BASE. The default edition can be displayed using the DATABASE_PROPERTIES view.

```
CONN / AS SYSDBA

SELECT property_value
FROM   database_properties
WHERE  property_name = 'DEFAULT_EDITION';

PROPERTY_VALUE
------------------------------------------------------------------
ORA$BASE

SQL>
```

The CREATE ANY EDITION and DROP ANY EDITION privileges are required to create and drop editions, so edition management is best done from privileged users.

Editions are created using the CREATE EDITION command with an optional AS CHILD OF clause.

```
CREATE EDITION edition-name;
CREATE EDITION edition-name AS CHILD OF parent-edition;
```

17. Roles

administer database privileges. You can add privileges to a role and then grant the role to a user. The user can then enable the role and exercise the privileges granted by the role.

A role contains all privileges granted to the role and all privileges of other roles granted to it. A new role is initially empty. You add privileges to a role with the GRANT statement.

If you create a role that is NOT IDENTIFIED or IDENTIFIED EXTERNALLY or BY *password*, then Oracle Database grants you the role with ADMIN OPTION. However, if you create a role IDENTIFIED GLOBALLY, then the database does not grant you the role.

Creating a Role: Example The following statement creates the role dw_manager:

```
CREATE ROLE dw_manager;
```

Users who are subsequently granted the dw_manager role will inherit all of the privileges that have been granted to this role.

You can add a layer of security to roles by specifying a password, as in the following example:

```
CREATE ROLE dw_manager
  IDENTIFIED BY warehouse;
```

Users who are subsequently granted the dw_manager role must specify the password warehouse to enable the role with the SET ROLE statement.

The following statement creates global role warehouse_user:

```
CREATE ROLE warehouse_user IDENTIFIED GLOBALLY;
```

The following statement creates the same role as an external role:

```
CREATE ROLE warehouse_user IDENTIFIED EXTERNALLY;
```

18. Rollback segment:

Purpose

Use the CREATE ROLLBACK SEGMENT statement to create a **rollback segment**, which is an object that Oracle Database uses to store data necessary to reverse, or undo, changes made by transactions.

The information in this section assumes that your database is not running in automatic undo mode (the UNDO_MANAGEMENT initialization parameter is set to MANUAL or not set at all). If your database is running in automatic undo mode (the UNDO_MANAGEMENT initialization parameter is set to AUTO), then user-created rollback segments are irrelevant.

Further, if your database has a locally managed SYSTEM tablespace, then you cannot create rollback segments in any dictionary-managed tablespace. Instead, you must either use the automatic undo management feature or create locally managed tablespaces to hold the rollback segments.

Creating a Rollback Segment: Example The following statement creates a rollback segment with default storage values in an appropriately configured tablespace:

```
CREATE TABLESPACE rbs_ts
  DATAFILE 'rbs01.dbf' SIZE 10M
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 100K;

/* This example and the next will fail if your database is in     automatic undo mode.
*/
CREATE ROLLBACK SEGMENT rbs_one
  TABLESPACE rbs_ts;
```

19. Tablespace:

Purpose

Use the CREATE TABLESPACE statement 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**.
- An **undo tablespace** is a type of permanent tablespace used by Oracle Database to manage undo data if you are running your database in automatic undo management mode. Oracle strongly recommends that you use automatic undo management mode rather than using rollback segments for undo.
- 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.

Examples

Creating a Bigfile Tablespace: Example The following example creates a bigfile tablespace bigtbs_01 with a datafile bigtbs_f1.dat of 10 MB:

```
CREATE BIGFILE TABLESPACE bigtbs_01
 DATAFILE 'bigtbs_f1.dat'
 SIZE 20M AUTOEXTEND ON;
```

Creating an Undo Tablespace: Example The following example creates a 10 MB undo tablespace undots1:

```
CREATE UNDO TABLESPACE undots1
 DATAFILE 'undotbs_1a.f'
```

```
  SIZE 10M AUTOEXTEND ON
  RETENTION GUARANTEE;
```

Creating a Temporary Tablespace: Example This statement shows how the temporary tablespace that serves as the default temporary tablespace for database users in the sample database was created:

```
CREATE TEMPORARY TABLESPACE temp_demo
  TEMPFILE 'temp01.dbf' SIZE 5M AUTOEXTEND ON;
```

20. USER

Purpose

Use the CREATE USER statement to create and configure a database **user**, which is an account through which you can log in to the database, and to establish the means by which Oracle Database permits access by the user.

You can enable a user to connect to the database through a proxy application or application server. For syntax and discussion, refer to ALTER USER.

Prerequisites

You must have the CREATE USER system privilege. When you create a user with the CREATE USER statement, the user's privilege domain is empty. To log on to Oracle Database, a user must have the CREATE SESSION system privilege. Therefore, after creating a user, you should grant the user at least the CREATE SESSION system privilege

Creating a Database User: Example If you create a new user with PASSWORD EXPIRE, then the user's password must be changed before the user attempts to log in to the database. You can create the user sidney by issuing the following statement:

```
CREATE USER sidney
```

```
IDENTIFIED BY out_standing1
DEFAULT TABLESPACE example     QUOTA 10M ON example
TEMPORARY TABLESPACE temp
QUOTA 5M ON system
PROFILE app_user
PASSWORD EXPIRE;
```

The user sidney has the following characteristics:

- The password out_standing1
- Default tablespace example, with a quota of 10 megabytes
- Temporary tablespace temp
- Access to the tablespace SYSTEM, with a quota of 5 megabytes
- Limits on database resources defined by the profile app_user (which was created in "Creating a Profile: Example")
- An expired password, which must be changed before sidney can log in to the database

Creating External Database Users: Examples The following example creates an external user, who must be identified by an external source before accessing the database:

```
CREATE USER app_user1
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE example
  QUOTA 5M ON example
  PROFILE app_user;
```

The user app_user1 has the following additional characteristics:

- Default tablespace example

- Default temporary tablespace exampl
- space on the tablespace teme
  tablespace of the database and unlimited quota on the example

- Limits on database resources defined by the app_user profile

To create another user accessible only by an operating system account, prefix the user name with the value of the initialization parameter OS_AUTHENT_PREFIX. For example, if this value is "ops$", you can create the externally identified user external_user with the following statement:

```
CREATE USER ops$external_user
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE example
  QUOTA 5M ON example
  PROFILE app_user;
```

Creating a Global Database User: Example The following example creates a global user. When you create a global user, you can specify the X.509 name that identifies this user at the enterprise directory server:

```
CREATE USER global_user
  IDENTIFIED GLOBALLY AS 'CN=analyst, OU=division1, O=oracle, C=US'
  DEFAULT TABLESPACE example
  QUOTA 5M ON example;
```

21,Object

- ⇨ Database is itself an object that contains another objects such as row, columns, schemas and other user defined objects.
- ⇨ They provides mapping relational database to some other application access.
  Advantage of Database Object

➔ Database independence – That means they can be ported from platform to another they are not dependent on some implementation.
➔ Security and validation – they structure as row and column they its easy to map because they are organized.

Suitable for Scaling

22. Record

- ⇨ Set of data stored in a table
- ⇨ They are objects that contains more than one values.
- ⇨ We may think them as the groups or rows and columns.
- ⇨ They provide the practical way to store and retrieve the data from databases.

This is data we enter or retrieve from database so
Select * from student;

This shows we retrieve all the records from student database

Procedures

- Programs that lives in database that is maintained, administered, and executed though database commands.
- they are used to access and modify data in tables.
- They are subroutine that available for the applications that access the database.

We can create procedures using
Create procedure procedure_name as database_statement go;

We have procedure procedure_name the we can execute by calling execution queries.

Example
Create procedure selectSoftwareStudents as select from student where department_id =5 go;
We can execute using exec **selectSoftwareStudents;**

Functions

- provides the results based on the set of input values.
- The are the same as procedure except they return values.
- They generally can not take output arguments (placeholders) that are then passed back out with changed values.

We can create functions using

- **Create function function_name return return_data as select from table_name;**

Cursors

- They are pointer to databases that store information about data manipulation language.
- They show the place from where take the using queries.

```
SELECT department_name, CURSOR(SELECT salary, commission_pct
  FROM employees e
  WHERE e.department_id = d.department_id)
  FROM departments d
  ORDER BY department_name;
```

We access the department_name from department that satisfies a given condition but we define the explicit cursor for data access that show as name with salary and specified pictures.

Data Types

- **CHAR / VARCHAR**
- **CLOB**
- **DBCLOB**
- **BLOB**
- **GRAPHIC / VARGRAPHIC**
- **DATE**
- **TIME**
- **DATETIME / TIMESTAMP**
- **XML**

I.CLOB
The CLOB data type stores single-byte and multibyte character data. Both fixed-width and variable-width character sets are supported, and both use the database character set. CLOB objects can store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character data. If the tablespaces in your database are of standard block size, and if you have used the default value of the CHUNK parameter of LOB storage when creating a LOB column, then this is equivalent to (4 gigabytes - 1) * (database block size).

CLOB objects have full transactional support. Changes made through SQL, the DBMS_LOB package, or Oracle Call Interface (OCI) participate fully in the transaction. CLOB value manipulations can be committed and rolled back. However, you cannot save a CLOB locator in a PL/SQL or OCI variable in one transaction and then use it in another transaction or session.

II.BLOB
The BLOB data type stores unstructured binary large objects. BLOB objects can be thought of as bitstreams with no character set semantics. BLOB objects can store binary data up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage). If the tablespaces in your database are of standard

block size, and if you have used the default value of the CHUNK parameter of LOB storage when creating a LOB column, then this is equivalent to (4 gigabytes - 1) * (database block size).

BLOB objects have full transactional support. Changes made through SQL, the DBMS_LOB package, or Oracle Call Interface (OCI) participate fully in the transaction. BLOB value manipulations can be committed and rolled back. However, you cannot save a BLOB locator in a PL/SQL or OCI variable in one transaction and then use it in another transaction or session.

III.DBCLOB

A DBCLOB (double-byte character large object) is a variable-length graphic string of double-byte characters that can be up to 1 073 741 823 characters long.

If only n by itself is specified, the value of n is the maximum length. If K, M or G is specified the maximum length is 1 024, 1 048 576 or 1 073 741 824 times n, and the maximum value for n is 1 048 576, 1 024 or 1, respectively. Double-byte character large objects are used to store large DBCS's (doublebyte character sets) character-based data.

IV.VARGRAPHIC

The data is a varying-length, double-byte character set (DBCS). It consists of a length subfield followed by a string of double-byte characters. The Oracle database does not support double-byte character sets; however, SQL*Loader reads them as single bytes and loads them as RAW data. Like RAW data, VARGRAPHIC fields are stored without modification in whichever column you specify. The size of the length subfield is the size of the SQL*Loader SMALLINT data type on your system (C type SHORT INT).

VARGRAPHIC data can be loaded with correct results only between systems where a SHORT INT has the same length in bytes. If the byte order is different between the systems, then use the appropriate technique to indicate the byte order of the length subfield.

V.XML

This Oracle-supplied type can be used to store and query XML data in the database. XML Type has member functions you can use to access, extract, and query the XML data using XPath expressions. XPath is another standard developed by the W3C committee to traverse XML documents. Oracle XML Type functions support many W3C XPath expressions. Oracle also provides a

set of SQL functions and PL/SQL packages to create XML Type values from existing relational or object-relational data.

XML Type is a system-defined type, so you can use it as an argument of a function or as the data type of a table or view column. You can also create tables and views of XML Type. When you create an XML Type column in a table, you can choose to store the XML data in a CLOB column, as binary XML (stored internally as a CLOB), or object relationally.