Université Med BOUDIAF M'sila

Faculté MI

Département informatique

2ère Année Master (IDO)

Module : Administration des bases de données

TP5

1. Créez un utilisateur kay avec le mot de passe mary.
   Vérifiez qu'aucun objet ni segment de tri n'a été créé dans le tablespace SYSTEM..
2. A partir du dictionnaire de données, affichez les informations sur la quantité d'espace disponible dans les tablespaces de bob..
3. Modifier l'utilisateur kay, pourqu'il peut se connecter dans le tablespace DATA01 dont la taille peut atteindre 1 Mo.
4. En tant que SYSTEM, supprimez le quota de bob sur son tablespace par défaut.
5. Supprimez le compte de kay.
6. bob a oublié sont mot de passe. Allouez-lui le mot de passe olink et demandez lui de changer son mot de passe à sa prochaine connexion.
7. Affichez les informations sur les profils, puis toutes les ressources du profil DEFAULT.
8. Créez un nouveau profil de telle sorte que 2 sessions simultanées par utilisateur soient permises (une minute d'inactivité).
9. Allouez ce profil à bob. A partir du dictionnaire de données, affichez le résultat.
10. En tant que SYSTEM, créez l'utilisateur kay et donnez-lui la possibilité de se connecter à la base de données et de créer des objets dans son schéma.
11. Toujours sous SYSTEM, accordez à bob la possibilité de sélectionner des données dans une table de kay.
12. Activez aussi la capacité pour bob d'accorder aux autres utilisateurs la possibilité de sélection vers EMP..
13. Examinez les vues du dictionnaire de données qui enregistrent ces informations.
14. Créez l'utilisateur todd et donnez-lui la possibilité de se connecter à la base de données.
15. En tant que bob, activez l'accès de todd à la table EMP de kay.
16. En tant que kay, supprimez le privilège de lecture de bob sur ses tables.
17. En tant que todd, interrogez la table EMP de kay. Que se passe-t-il ?
18. Examinez la vue du dictionnaire qui permet d'énumérer les privilèges systèmes du rôle RESOURCE.
19. Créez un rôle DEV permettant de créer une table, une vue et de sélectionner les données dans la table EMP de kay.
20. Allouez les rôles RESOURCE et DEV à bob. Déterminez l'activation automatique du rôle DEV quand bob se connecte.
21. Accordez à bob la possibilité de lire toutes les informations du dictionnaire de données.

22. bob doit vérifier les rollback segments actuellement utilisés par l'instance. Connectez vous en tant que bob et énumérez les rollback segments utilisés.

Oracle / PLSQL: Users

A user is a database account that allows you to log into the Oracle database.

- Create a User
    - CREATE USER statement
- Remove a User
    - DROP USER statement
- Change a User Password
    - ALTER USER statement (Change Password)
- Retrieve User Information
    - Find Users in Oracle
    - Find Users logged into Oracle

-

Oracle / PLSQL: CREATE USER statement

This Oracle tutorial explains how to use the Oracle **CREATE USER statement** with syntax and examples.

Description

The **CREATE USER statement** creates a database account that allows you to log into the Oracle database.

Syntax

The syntax for the **CREATE USER statement** in Oracle/PLSQL is:

```
CREATE USER user_name
  IDENTIFIED { BY password
             | EXTERNALLY [ AS 'certificate_DN' ]
             | GLOBALLY [ AS '[ directory_DN ]' ]
             }
  [ DEFAULT TABLESPACE tablespace
  | TEMPORARY TABLESPACE
       { tablespace | tablespace_group }
  | QUOTA integer [ K | M | G | T | P | E ]
        | UNLIMITED }
        ON tablespace
    [ QUOTA integer [ K | M | G | T | P | E ]
        | UNLIMITED }
            ON tablespace
    ]
  | PROFILE profile_name
  | PASSWORD EXPIRE
  | ACCOUNT { LOCK | UNLOCK }
     [ DEFAULT TABLESPACE tablespace
     | TEMPORARY TABLESPACE
         { tablespace | tablespace_group }
```

```
    | QUOTA integer [ K | M | G | T | P | E ]
          | UNLIMITED }
          ON tablespace
      [ QUOTA integer [ K | M | G | T | P | E ]
          | UNLIMITED }
          ON tablespace
       ]
    | PROFILE profile
    | PASSWORD EXPIRE
    | ACCOUNT { LOCK | UNLOCK } ]
    ] ;
```

Parameters or Arguments

user_name
The name of the database account that you wish to create.
PROFILE profile_name
Optional. It is the name of the profile that you wish to assign to the user account to limit the amount of database resources assigned to the user account. If you omit this option, the DEFAULT profile is assigned to the user.
PASSWORD EXPIRE
Optional. If this option is set, then the password must be reset before the user can log into the Oracle database.
ACCOUNT LOCK
Optional. It disables access to the user account.
ACCOUNT UNLOCK
Optional. It enables access to the user account.

Example

If you wanted to execute a simple CREATE USER statement that creates a new user and assigns a password, you could do the following:

For example:

```
CREATE USER smithj
  IDENTIFIED BY pwd4smithj
  DEFAULT TABLESPACE tbs_perm_01
  TEMPORARY TABLESPACE tbs_temp_01
```

```
  QUOTA 20M on tbs_perm_01;
```

This CREATE USER statement would create a new user called *smithj* in the Oracle database whose password is *pwd4smithj*, the default tablespace would be *tbs_perm_01* with a quota of 20MB, and the temporary tablespace would be *tbs_temp_01*.

If you wanted to make sure that the user changed their password before logging into the database, you could add the PASSWORD EXPIRE option as follows:

```
CREATE USER smithj
  IDENTIFIED BY pwd4smithj
  DEFAULT TABLESPACE tbs_perm_01
  TEMPORARY TABLESPACE tbs_temp_01
  QUOTA 20M on tbs_perm_01
  PASSWORD EXPIRE;
```

External Database User

To create an External Database user, you could execute the following CREATE USER statement:

```
CREATE USER external_user1
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE tbs_perm_01
  QUOTA 5M on tbs_perm_01
  PROFILE external_user_profile;
```

This CREATE USER statement would create an External Database user called *external_user1* that has a default tablespace of *tbs_perm_01* with a quote of 5MB, and is limited by the database resources assigned to *external_user_profile*.

To create an External Database user that is only accessible by an operating system account, you could run the following CREATE USER statement:

```
CREATE USER ops$external_user1
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE tbs_perm_01
  QUOTA 5M on tbs_perm_01
  PROFILE external_user_profile;
```

Note that the only difference between this CREATE USER statement and the previous is the **ops$** in front of the user_name.

Global Database User

To create a Global Database user, you could execute the following CREATE USER statement:

```
CREATE USER global_user1
  IDENTIFIED GLOBALLY AS 'CN=manager, OU=division, O=oracle, C=US'
  DEFAULT TABLESPACE tbs_perm_01
  QUOTA 10M on tbs_perm_01;
```

This CREATE USER statement would create a Global Database user called *global_user1* that has a default tablespace of *tbs_perm_01* with a quote of 10M.

Oracle / PLSQL: Change a user's password in Oracle

Question: How do I change the password for a user in Oracle?

Answer: To change a user's password in Oracle, you need to execute the *alter user* command.

Syntax

The syntax for changing a password in Oracle is:

```
ALTER USER user_name IDENTIFIED BY new_password;
```

Parameters or Arguments

user_name
: The user whose password you wish to change.

new_password
: The new password to assign.

Example

Let's look at an example of how to change a password for a user in Oracle/PLSQL.

For example:

```
ALTER USER smithj IDENTIFIED BY autumn;
```

This example would change the password for the user named *smithj* and set the new password to *autumn*.

Oracle / PLSQL: DROP USER statement

This Oracle tutorial explains how to use the Oracle **DROP USER statement** with syntax and examples.

Description

The **DROP USER statement** is used to remove a user from the Oracle database and remove all objects owned by that user.

Syntax

The syntax for the **DROP USER statement** in Oracle/PLSQL is:

```
DROP USER user_name [ CASCADE ];
```

Parameters or Arguments

user_name
The name of the user to remove from the Oracle database.

CASCADE
Optional. If *user_name* owns any objects (ie: tables or views in its schema), you must specify CASCADE to drop all of these objects.

Example

Let's look at a simple DROP USER statement.

If the user does not own any objects in its schema, you could execute the following DROP USER statement:

```
DROP USER smithj;
```

This would drop the user called *smithj*. This DROP USER statement will only run if *smithj* does not own any objects in its schema.

If *smithj* did own objects in its schema, you would need to run the following DROP USER statement instead:

```
DROP USER smithj CASCADE;
```

This DROP USER statement would remove the user *smithj*, drop all objects (ie: tables and views) owned by *smithj*, and all referential integrity constraints on smithj's objects would also be dropped

Oracle / PLSQL: Find Users in Oracle / PLSQL

This Oracle tutorial explains how to find all users that are created in the Oracle database with syntax and examples.

Description

You can find all users created in Oracle by running a query from a command prompt. The user information is stored in various system tables - *ALL_USERS* and *DBA_USERS*, depending on what user information you wish to retrieve.

ALL_USERS

If you need to find all users that are visible to the current users, you can query the *ALL_USERS* table. The syntax to retrieve user information from the *ALL_USERS* table in Oracle/PLSQL is:

```
SELECT *
FROM ALL_USERS;
```

The *ALL_USERS* table contains the following columns:

Column	Explanation
USERNAME	Name of the user
USER_ID	Numeric ID assigned to the user
CREATED	Date that user was created

DBA_USERS

If you need to find out all users that exist in Oracle or require more information about the user, there is also another system table called *DBA_USERS*.

The syntax to retrieve user information from the *DBA_USERS* table in Oracle/PLSQL is:

```
SELECT *
FROM DBA_USERS;
```

The *DBA_USERS* table contains the following columns:

Column	Explanation
USERNAME	Name of the user
USER_ID	Numeric ID assigned to the user
PASSWORD	Deprecated
ACCOUNT_STATUS	Status of the user such as: • OPEN • EXPIRED • EXPIRED(GRACE) • EXPIRED(TIMED) • LOCKED • EXPIRED & LOCKED(TIMED) • EXPIRED(GRACE) & LOCKED(TIMED) • EXPIRED & LOCKED • EXPIRED(GRACE) & LOCKED
LOCK_DATE	Date that User was locked (if applicable)
EXPIRY_DATE	Date that User was expired
DEFAULT_TABLESPACE	Default tablespace for the user
TEMPORARY_TABLESPACE	Temporary tablespace for the user
CREATED	Date that user was created
PROFILE	User resource profile name
INITIAL_RSRC_CONSUMER_GROUP	Initial resource consumer group for the user
EXTERNAL_NAME	External name for the user
PASSWORD_VERSIONS	List of versions of the password hashes
EDITIONS_ENABLED	Y/N indicating whether editions have been enabled for the user
AUTHENTICATION_TYPE	Authentication method for the user
PROXY_ONLY_CONNECT	Y/N indicating whether a user can connect directly or by proxy only
COMMON	YES/NO indicating whether a user is common

Column	Explanation
LAST_LOGIN	Last login time
ORACLE_MAINTAINED	Y/N indicating whether a user was created and maintained by Oracle-supplied scripts

Oracle / PLSQL: Find Users logged into Oracle / PLSQL

This Oracle tutorial explains how to find all users currently logged into the Oracle database.

Description

You can find all users currently logged into Oracle by running a query from a command prompt. In Oracle/PLSQL, there is a system view called *V$SESSION* which shows the session information for each current session in the database. You can run a query against this system view to return all users that currently have a connection running in the Oracle/PLSQL database.

Syntax

The syntax to retrieve the users logged into Oracle is:

```
SELECT USERNAME FROM V$SESSION;
```

This SELECT statement will return each username that is logged in.

The *V$SESSION* view contains the following columns:

Column	Explanation
SADDR	Address for session
SID	Identifier for session
SERIAL#	Serial number for session
AUDSID	Auditing session ID
PADDR	Address of the process that owns the session
USER#	User identifier

Column	Explanation
USERNAME	User name (ie: root, techonthenet, etc)
COMMAND	Last statement parsed
OWNERID	User identifier who owns the migratable session
TADDR	Address of the transaction state object
LOCKWAIT	Address for lock wait
STATUS	Status of the session. It can be one of the following: ACTIVE, INACTIVE, KILLED, CACHED, or SNIPED.
SERVER	Type of server. It can be one of the following: DEDICATED, SHARED, PSEUDO, or NONE.
SCHEMA#	User identifier for schema
SCHEMANAME	User name for schema
OSUSER	Operation system client user name
PROCESS	Operating system client process ID
MACHINE	Operating system machine name
TERMINAL	Operating system terminal name
PROGRAM	Operating system program name
TYPE	Type of session
SQL_ADDRESS	Identifies the SQL statement currently being executed (used with SQL_HASH_VALUE)
SQL_HASH_VALUE	Identifies the SQL statement currently being executed (used with SQL_ADDRESS)
SQL_ID	SQL identifier for the SQL statement currently being executed

Column	Explanation
SQL_CHILD_NUMBER	Child number for the SQL statement currently being executed
PREV_SQL_ADDR	Identifies the last SQL statement executed (used with PREV_HASH_VALUE)
PREV_HASH_VALUE	Identifies the last SQL statement executed (used with PREV_SQL_ADDR)
PREV_SQL_ID	SQL identifier for the last SQL statement executed
PREV_CHILD_NUMBER	Child number for the last SQL statement executed
MODULE	Name of the currently executing module (as per DBMS_APPLICATION_INFO.SET_MODULE)
MODULE_HASH	Hash value of the currently executing module
ACTION	Name of the currently executing action (as per DBMS_APPLICATION_INFO.SET_ACTION)
ACTION_HASH	Hash value of the currently executing action
CLIENT_INFO	Client information (as per DBMS_APPLICATION_INFO.SET_CLIENT_INFO)
FIXED_TABLE_SEQUENCE	Sequence number incremented each time there has been an intervening select from a dynamic performance table
ROW_WAIT_OBJ#	Object identifier for table specified by ROW_WAIT_ROW#
ROW_WAIT_FILE#	Identifier for datafile specified in ROW_WAIT_ROW#
ROW_WAIT_BLOCK#	Identifier for block specified in ROW_WAIT_ROW#
ROW_WAIT_ROW#	Row that is currently locked
LOGON_TIME	Time that user logged in
LAST_CALL_ET	If STATUS is ACTIVE, LAST_CALL_ET is the elapsed time (in seconds) since the session became active. If STATUS is

Column	Explanation
	INACTIVE, LAST_CALL_ET is the elapsed time (in seconds) since the session became inactive.
PDML_ENABLED	Replaced by PDML_STATUS
FAILOVER_TYPE	What type of transparent application failover is enabled for the session. It can be one of the following: NONE, SESSION, or SELECT.
FAILOVER_METHOD	Method of transparent application failure for the session. It can be one of the following: NONE, BASIC, or PRECONNECT.
FAILED_OVER	YES or NO to indicate whether failover has occurred
RESOURCE_CONSUMER_GROUP	Resource consumer group for the session
PDML_STATUS	ENABLED or DISABLED
PDDL_STATUS	ENABLED or DISABLED
PQ_STATUS	ENABLED or DISABLED
CURRENT_QUEUE_DURATION	Length of time that session has been queued
CLIENT_IDENTIFIER	Client identifier for the session
BLOCKING_SESSION_STATUS	It can be one of the following values: VALID, NO HOLDER, GLOBAL, NOT IN WAIT, or UNKNOWN
BLOCKING_INSTANCE	Instance identifier of blocking session
BLOCK_SESSION	Session identifier of blocking session
SEQ#	Sequence number that is incremented for each wait
EVENT#	Event number
EVENT	Resource that the session is waiting for

Column	Explanation
P1TEXT	Description of the first additional parameter
P1	First additional parameter
P1RAW	First additional parameter
P2TEXT	Description of the second additional parameter
P2	Second additional parameter
P2RAW	Second additional parameter
P3TEXT	Description of the third additional parameter
P3	Third additional parameter
P3RAW	Third additional parameter
WAIT_CLASS_ID	Identifier of the wait class
WAIT_CLASS#	Number of the wait class
WAIT_CLASS	Name of the wait class
WAIT_TIME	Value of session's last wait time. If 0, then the session is currently waiting
SECONDS_IN_WAIT	If WAIT_TIME > 0, then SECOND_IN_WAIT is the number of seconds since the start of the last wait. If WAIT_TIME = 0, then SECONDS_IN_WAIT is the number of seconds elapsed in the current wait.
STATE	0 means WAITING -2 means WAITED UNKNOWN TIME -1 means WAITED SHORT TIME >0 means WAITED KNOWN TIME
SERVICE_NAME	Service name of the session
SQL_TRACE	ENABLED or DISABLED

Column	Explanation
SQL_TRACE_WAITS	TRUE or FALSE
SQL_TRACE_BINDS	TRUE or FALSE

Oracle / PLSQL: Roles

This Oracle tutorial explains how to create roles, grant/revoke privileges to roles, enable/disable roles, set roles as the default, and drop roles in Oracle with syntax and examples.

Description

A **role** is a set or group of privileges that can be granted to users or another role. This is a great way for database administrators to save time and effort.

Create Role

You may wish to create a role so that you can logically group the users' permissions. Please note that to create a role, you must have CREATE ROLE system privileges.

Syntax

The syntax for creating a role in Oracle is:

```
CREATE ROLE role_name
[ NOT IDENTIFIED |
IDENTIFIED {BY password | USING [schema.] package | EXTERNALLY | GLOBALLY } ;
```

role_name

The name of the new role that you are creating. This is how you will refer to the grouping of privileges.

NOT IDENTIFIED

It means that the role is immediately enabled. No password is required to enable the role.

IDENTIFIED

It means that a user must be authorized by a specified method before the role is enabled.

BY password

It means that a user must supply a password to enable the role.

USING package

It means that you are creating an application role - a role that is enabled only by applications using an authorized package.

EXTERNALLY

It means that a user must be authorized by an external service to enable the role. An external service can be an operating system or third-party service.

GLOBALLY

It means that a user must be authorized by the enterprise directory service to enable the role.

Note

- If both *NOT IDENTIFIED* and *IDENTIFIED* are omitted in the CREATE ROLE statement, the role will be created as a NOT IDENTIFIED role.

Example

Let's look at an example of how to create a role in Oracle.

For example:

```
CREATE ROLE test_role;
```

This first example creates a role called *test_role*.

```
CREATE ROLE test_role
IDENTIFIED BY test123;
```

This second example creates the same role called *test_role*, but now it is password protected with the password of test123.

Grant TABLE Privileges to Role

Once you have created the role in Oracle, your next step is to grant privileges to that role.

Just as you granted privileges to users, you can grant privileges to a role. Let's start with granting table privileges to a role. Table privileges can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

The syntax for granting table privileges to a role in Oracle is:

```
GRANT privileges ON object TO role_name
```

privileges

The privileges to assign to the role. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

object

The name of the database object that you are granting privileges for. In the case of granting privileges on a table, this would be the table name.

role_name

The name of the role that will be granted these privileges.

Example

Let's look at some examples of how to grant table privileges to a role in Oracle.

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called *suppliers* to a role named *test_role*, you would run the following GRANT statement:

```
GRANT select, insert, update, delete ON suppliers TO test_role;
```

You can also use the ALL keyword to indicate that you wish all permissions to be granted. For example:

```
GRANT all ON suppliers TO test_role;
```

Revoke Table Privileges from Role

Once you have granted table privileges to a role, you may need to revoke some or all of these privileges. To do this, you can execute a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

The syntax for revoking table privileges from a role in Oracle is:

```
REVOKE privileges ON object FROM role_name;
```

Privileges object

The privileges to revoke from the role. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

The name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

role_name

The name of the role that will have these privileges revoked.

Example

Let's look at some examples of how to revoke table privileges from a role in Oracle.

For example, if you wanted to revoke DELETE privileges on a table called *suppliers* from a role named *test_role*, you would run the following REVOKE statement:

```
REVOKE delete ON suppliers FROM test_role;
```

If you wanted to revoke ALL privileges on the table called *suppliers* from a role named *test_role*, you could use the ALL keyword. For example:

```
REVOKE all ON suppliers FROM test_role;
```

Grant Function/Procedure Privileges to Role

When dealing with functions and procedures, you can grant a role the ability to EXECUTE these functions and procedures.

Syntax

The syntax for granting EXECUTE privileges on a function/procedure to a role in Oracle is:

```
GRANT EXECUTE ON object TO role_name;
```

EXECUTE

The ability to compile the function/procedure and the ability to execute the function/procedure directly.

object

The name of the database object that you are granting privileges for. In the case of granting EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

role_name

The name of the role that will be granted the EXECUTE privileges.

Example

Let's look at an example of how to grant EXECUTE privileges on a function or procedure to a role in Oracle.

For example, if you had a function called *Find_Value* and you wanted to grant EXECUTE access to the role named *test_role*, you would run the following GRANT statement:

```
GRANT execute ON Find_Value TO test_role;
```

Revoke Function/Procedure Privileges from Role

Once you have granted EXECUTE privileges on a function or procedure to a role, you may need to revoke these privileges from that role. To do this, you can execute a REVOKE command.

Syntax

The syntax for the revoking privileges on a function or procedure from a role in Oracle is:

```
REVOKE execute ON object FROM role_name;
```

EXECUTE

Revoking the ability to compile the function/procedure and the ability to execute the function/procedure directly.

object

The name of the database object that you are revoking privileges for. In the case of revoking EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

role_name

The name of the role that will have the EXECUTE privileges revoked.

Example

Let's look at an example of how to grant EXECUTE privileges on a function or procedure to a role in Oracle.

If you wanted to revoke EXECUTE privileges on a function called *Find_Value* from a role named *test_role*, you would run the following REVOKE statement:

```
REVOKE execute ON Find_Value FROM test_role;
```

Grant Role to User

Now, that you've created the role and assigned the privileges to the role, you'll need to grant the role to specific users.

Syntax

The syntax to grant a role to a user in Oracle is:

```
GRANT role_name TO user_name;
```

role_name

The name of the role that you wish to grant.

user_name

The name of the user that will be granted the role.

Example

Let's look at an example of how to grant a role to a user in Oracle:

```
GRANT test_role TO smithj;
```

This example would grant the role called *test_role* to the user named *smithj*.

Enable/Disable Role (Set Role Statement)

To enable or disable a role for a current session, you can use the SET ROLE statement.

When a user logs into Oracle, all *default* roles are enabled, but non-default roles must be enabled with the SET ROLE statement.

Syntax

The syntax for the SET ROLE statement in Oracle is:

```
SET ROLE
( role_name [ IDENTIFIED BY password ] | ALL [EXCEPT role1, role2, ... ] | NONE );
```

role_name

The name of the role that you wish to enable.

IDENTIFIED BY password

The password for the role to enable it. If the role does not have a password, this phrase can be omitted.

ALL

It means that all roles should be enabled for this current session, except those listed in *EXCEPT*.

NONE

Disables all roles for the current session (including all default roles).

Example

Let's look at an example of how to enable a role in Oracle.

For example:

```
SET ROLE test_role IDENTIFIED BY test123;
```

This example would enable the role called test_role with a password of test123.

Set role as DEFAULT Role

A default role means that the role is always enabled for the current session at logon. It is not necessary to issue the SET ROLE statement. To set a role as a DEFAULT ROLE, you need to issue the ALTER USER statement.

Syntax

The syntax for setting a role as a DEFAULT ROLE in Oracle is:

```
ALTER USER user_name
```

```
DEFAULT ROLE
( role_name | ALL [EXCEPT role1, role2, ... ] | NONE );
```

user_name

The name of the user whose role you are setting as DEFAULT.

role_name

The name of the role that you wish to set as DEFAULT.

ALL

It means that all roles should be enabled as DEFAULT, except those listed in *EXCEPT*.

NONE

Disables all roles as DEFAULT.

Example

Let's look at an example of how to set a role as a DEFAULT ROLE in Oracle.

For example:

```
ALTER USER smithj
DEFAULT ROLE
test_role;
```

This example would set the role called *test_role* as a DEFAULT role for the user named *smithj*.

```
ALTER USER smithj
DEFAULT ROLE
ALL;
```

This example would set all roles assigned to *smithj* as DEFAULT.

```
ALTER USER smithj
DEFAULT ROLE
ALL EXCEPT test_role;
```

This example would set all roles assigned to *smithj* as DEFAULT, except for the role called *test_role*.

Drop Role

Once a role has been created in Oracle, you might at some point need to drop the role.

Syntax

The syntax to drop a role in Oracle is:

```
DROP ROLE role_name;
```

role_name

The name of the role that is to be dropped.

Example

Let's look at an example of how to drop a role in Oracle.

For example:

```
DROP ROLE test_role;
```

This DROP statement would drop the role called *test_role* that we defined earlier.

Oracle / PLSQL: Tablespaces
A tablespace is an allocation of space in an Oracle database where schema objects are stored.

The following is a list of topics that explain how to use Tablespaces in Oracle/PLSQL:

- **Create a Tablespace**
  - **CREATE TABLESPACE statement**
- **Modify a Tablespace (or datafiles/tempfiles associated)**
  - **ALTER TABLESPACE statement**
- **Remove a Tablespace**
  - **DROP TABLESPACE statement**
- **Default Tablespaces for Database**
  - **Find Default Tablespaces (both Permanent and Temp)**
  - **Set Default Tablespaces (both Permanent and Temp)**

Oracle / PLSQL: CREATE TABLESPACE statement

This Oracle tutorial explains how to use the Oracle **CREATE TABLESPACE statement** with syntax and examples.

Description

The CREATE TABLESPACE statement is used to allocate space in the Oracle database where schema objects are stored.

The CREATE TABLESPACE statement can be used to create the 3 kinds of tablespaces:

1. Permanent Tablespace
2. Temporary Tablespace
3. Undo Tablespace

We will take a look at all 3 kinds of tablespaces.

#1 - PERMANENT TABLESPACE

A permanent tablespace contains persistent schema objects that are stored in data files.

Syntax

The syntax for the CREATE TABLESPACE statement when creating a permanent tablespace is:

```
CREATE
  [ SMALLFILE | BIGFILE ]
  TABLESPACE tablespace_name
  { DATAFILE { [ 'filename' | 'ASM_filename' ]
               [ SIZE integer [ K | M | G | T | P | E ] ]
               [ REUSE ]
               [ AUTOEXTEND
                   { OFF
                   | ON [ NEXT integer [ K | M | G | T | P | E ] ]
                   [ MAXSIZE { UNLIMITED | integer [ K | M | G | T | P | E ] } ]
                   }
               ]
```

```
          | [ 'filename | ASM_filename'
          | ('filename | ASM_filename'
              [, 'filename | ASM_filename' ] )
          ]
          [ SIZE integer [ K | M | G | T | P | E ] ]
          [ REUSE ]
          }
  { MINIMUM EXTENT integer [ K | M | G | T | P | E ]
  | BLOCKSIZE integer [ K ]
  | { LOGGING | NOLOGGING }
  | FORCE LOGGING
  | DEFAULT [ { COMPRESS | NOCOMPRESS } ]
storage_clause
  | { ONLINE | OFFLINE }
  | EXTENT MANAGEMENT
     { LOCAL
        [ AUTOALLOCATE
        | UNIFORM
           [ SIZE integer [ K | M | G | T | P | E ] ]
        ]
     | DICTIONARY
     }
  | SEGMENT SPACE MANAGEMENT { AUTO | MANUAL }
  | FLASHBACK { ON | OFF }
      [ MINIMUM EXTENT integer [ K | M | G | T | P | E ]
      | BLOCKSIZE integer [ K ]
      | { LOGGING | NOLOGGING }
      | FORCE LOGGING
      | DEFAULT [ { COMPRESS | NOCOMPRESS } ]
      storage_clause
      | { ONLINE | OFFLINE }
      | EXTENT MANAGEMENT
           { LOCAL
```

```
                [ AUTOALLOCATE | UNIFORM [ SIZE integer [ K | M | G | T | P | E ] ]
]
                | DICTIONARY
              }
         | SEGMENT SPACE MANAGEMENT { AUTO | MANUAL }
         | FLASHBACK { ON | OFF }
         ]
     }
```

SMALLFILE

A tablespace that contains 1,022 data or temp files (each file can be up to 4 million blocks in size). This is the most common tablespace size to create.

BIGFILE

A tablespace that contains only one data or temp file (this file can be up to 4 million blocks in size).

TIP: If you omit the SMALLFILE or BIGFILE option, the Oracle database will use the default tablespace type.

tablespace_name

The name of the tablespace to create.

storage_clause

The syntax for the *storage_clause* is:

```
STORAGE
   ({ INITIAL integer [ K | M | G | T | P | E ]
    | NEXT integer [ K | M | G | T | P | E ]
    | MINEXTENTS integer
    | MAXEXTENTS { integer | UNLIMITED }
    | PCTINCREASE integer
    | FREELISTS integer
    | FREELIST GROUPS integer
    | OPTIMAL [ integer [ K | M | G | T | P | E ] | NULL ]
    | BUFFER_POOL { KEEP | RECYCLE | DEFAULT }
    }
       [ INITIAL integer [ K | M | G | T | P | E ]
       | NEXT integer [ K | M | G | T | P | E ]
       | MINEXTENTS integer
```

```
      | MAXEXTENTS { integer | UNLIMITED }
      | PCTINCREASE integer
      | FREELISTS integer
      | FREELIST GROUPS integer
      | OPTIMAL [ integer [ K | M | G | T | P | E ] | NULL ]
      | BUFFER_POOL { KEEP | RECYCLE | DEFAULT }
      ]
  )
```

Example - PERMANENT TABLESPACE

The following is a CREATE TABLESPACE statement that creates a simple permanent tablespace:

```
CREATE TABLESPACE tbs_perm_01
  DATAFILE 'tbs_perm_01.dat'
    SIZE 20M
  ONLINE;
```

This CREATE TABLESPACE statement creates a permanent tablespace called *tbs_perm_01* that has one data file called *tbs_perm_01.dat*.

The following is a CREATE TABLESPACE statement that creates a permanent tablespace that will extend when more space is required:

```
CREATE TABLESPACE tbs_perm_02
  DATAFILE 'tbs_perm_02.dat'
    SIZE 10M
    REUSE
    AUTOEXTEND ON NEXT 10M MAXSIZE 200M;
```

This CREATE TABLESPACE statement creates a permanent tablespace called *tbs_perm_02* that has one data file called *tbs_perm_02.dat*. When more space is required, 10M extents will automatically be added until 200MB is reached.

The following is a CREATE TABLESPACE statement that creates a BIGFILE permanent tablespace that will extend when more space is required:

```
CREATE BIGFILE TABLESPACE tbs_perm_03
  DATAFILE 'tbs_perm_03.dat'
    SIZE 10M
    AUTOEXTEND ON;
```

This CREATE TABLESPACE statement creates a BIGFILE permanent tablespace called *tbs_perm_03* that has one data file called *tbs_perm_03.dat*.

#2 - TEMPORARY TABLESPACE

A temporary tablespace contains schema objects that are stored in temp files that exist during a session.

Syntax

The syntax for the CREATE TABLESPACE statement when creating a temporary tablespace is:

```
CREATE
  [ SMALLFILE | BIGFILE ]
  TEMPORARY TABLESPACE tablespace_name
    [ TEMPFILE { [ 'filename' | 'ASM_filename' ]
                 [ SIZE integer [ K | M | G | T | P | E ] ]
                 [ REUSE ]
                 [ AUTOEXTEND
                     { OFF
                     | ON [ NEXT integer [ K | M | G | T | P | E ] ]
                     [ MAXSIZE { UNLIMITED | integer [ K | M | G | T | P | E ] } ]
                     }
                 ]
               | [ 'filename | ASM_filename'
               | ('filename | ASM_filename'
                   [, 'filename | ASM_filename' ] )
               ]
               [ SIZE integer [ K | M | G | T | P | E ] ]
               [ REUSE ]
               }
    [ TABLESPACE GROUP { tablespace_group_name | '' } ]
    [ EXTENT MANAGEMENT
       { LOCAL
          [ AUTOALLOCATE | UNIFORM [ SIZE integer [ K | M | G | T | P | E ] ] ]
       | DICTIONARY
       } ]
```

SMALLFILE

A tablespace that contains 1,022 data or temp files (each file can be up to 4 million blocks in size). This is the most common tablespace size to create.

BIGFILE

A tablespace that contains only one data or temp file (this file can be up to 4 million blocks in size).

TIP: If you omit the SMALLFILE or BIGFILE option, the Oracle database will use the default tablespace type.

tablespace_name

The name of the tablespace to create.

Example - TEMPORARY TABLESPACE

The following is a CREATE TABLESPACE statement that creates a temporary tablespace:

```
CREATE TEMPORARY TABLESPACE tbs_temp_01
  TEMPFILE 'tbs_temp_01.dbf'
    SIZE 5M
    AUTOEXTEND ON;
```

This CREATE TABLESPACE statement creates a temporary tablespace called *tbs_temp_01* that has one temp file called *tbs_temp_01.dbf*.

#3 - UNDO TABLESPACE

An undo tablespace is created to manage undo data if the Oracle database is being run in automatic undo management mode.

Syntax

The syntax for the CREATE TABLESPACE statement when creating an undo tablespace is:

```
CREATE
  [ SMALLFILE | BIGFILE ]
  UNDO TABLESPACE tablespace_name
    [ DATAFILE { [ 'filename' | 'ASM_filename' ]
                 [ SIZE integer [ K | M | G | T | P | E ] ]
                 [ REUSE ]
                 [ AUTOEXTEND
                     { OFF
                     | ON [ NEXT integer [ K | M | G | T | P | E ] ]
```

```
                    [ MAXSIZE { UNLIMITED | integer [ K | M | G | T | P | E ] } ]
                    }
                ]
              | [ 'filename | ASM_filename'
              | ('filename | ASM_filename'
                  [, 'filename | ASM_filename' ] )
              ]
              [ SIZE integer [ K | M | G | T | P | E ] ]
              [ REUSE ]
              }
    [ EXTENT MANAGEMENT
       { LOCAL
          [ AUTOALLOCATE | UNIFORM [ SIZE integer [ K | M | G | T | P | E ] ] ]
       | DICTIONARY
       } ]
    [ RETENTION { GUARANTEE | NOGUARANTEE } ]
```

SMALLFILE

A tablespace that contains 1,022 data or temp files (each file can be up to 4 million blocks in size). This is the most common tablespace size to create.

BIGFILE

A tablespace that contains only one data or temp file (this file can be up to 4 million blocks in size).

TIP: If you omit the SMALLFILE or BIGFILE option, the Oracle database will use the default tablespace type.

tablespace_name

The name of the tablespace to create.

Example - UNDO TABLESPACE

The following is a CREATE TABLESPACE statement that creates an undo tablespace:

```
CREATE UNDO TABLESPACE tbs_undo_01
  DATAFILE 'tbs_undo_01.f'
    SIZE 5M
    AUTOEXTEND ON
  RETENTION GUARANTEE;
```

This CREATE TABLESPACE statement creates an undo tablespace called *tbs_undo_01* that is 5MB in size and has one data file called *tbs_undo_01.f*.

Oracle / PLSQL: ALTER TABLESPACE statement

This Oracle tutorial explains how to use the Oracle **ALTER TABLESPACE statement** with syntax and examples.

Description

The ALTER TABLESPACE statement is used to modify a tablespace or one of its data files or temp files. A tablespace is used to allocate space in the Oracle database where schema objects are stored.

Syntax

The syntax for the ALTER TABLESPACE statement in Oracle/PLSQL is:

```
ALTER TABLESPACE tablespace_name
  { DEFAULT
     [ { COMPRESS | NOCOMPRESS } ] storage_clause
  | MINIMUM EXTENT integer [ K | M | G | T | P | E ]
  | RESIZE integer [ K | M | G | T | P | E ]
  | COALESCE
  | RENAME TO new_tablespace_name
  | { BEGIN | END } BACKUP
  | { ADD { DATAFILE | TEMPFILE }
       [ file_specification
          [, file_specification ]
       ]
    | DROP {DATAFILE | TEMPFILE } { 'filename' | file_number }
    | RENAME DATAFILE 'filename' [, 'filename' ] TO 'filename' [, 'filename' ]
    | { DATAFILE | TEMPFILE } { ONLINE | OFFLINE }
    }
  | { logging_clause | [ NO ] FORCE LOGGING }
  | TABLESPACE GROUP { tablespace_group_name | '' }
  | { ONLINE
```

```
    | OFFLINE [ NORMAL | TEMPORARY | IMMEDIATE ]
    }
    | READ { ONLY | WRITE }
    | { PERMANENT | TEMPORARY }
  | AUTOEXTEND
     { OFF
     | ON [ NEXT integer [ K | M | G | T | P | E ] ]
        [ MAXSIZE { UNLIMITED | integer [ K | M | G | T | P | E ] } ]
     }
  | FLASHBACK { ON | OFF }
  | RETENTION { GUARANTEE | NOGUARANTEE }
  } ;
```

Parameters or Arguments

tablespace_name
The name of the tablespace to remove from the Oracle database.

storage_clause
The syntax for the the *storage_clause* is:

```
STORAGE
   ({ INITIAL integer [ K | M | G | T | P | E ]
    | NEXT integer [ K | M | G | T | P | E ]
    | MINEXTENTS integer
    | MAXEXTENTS { integer | UNLIMITED }
    | PCTINCREASE integer
    | FREELISTS integer
    | FREELIST GROUPS integer
    | OPTIMAL [ integer [ K | M | G | T | P | E ] | NULL ]
    | BUFFER_POOL { KEEP | RECYCLE | DEFAULT }
    }
       [ INITIAL integer [ K | M | G | T | P | E ]
       | NEXT integer [ K | M | G | T | P | E ]
       | MINEXTENTS integer
       | MAXEXTENTS { integer | UNLIMITED }
       | PCTINCREASE integer
       | FREELISTS integer
```

```
       | FREELIST GROUPS integer
       | OPTIMAL [ integer [ K | M | G | T | P | E ] | NULL ]
       | BUFFER_POOL { KEEP | RECYCLE | DEFAULT }
       ]
   )
```

file_specification

The syntax for the *file_specification* is:

```
{ [ 'filename' | 'ASM_filename' ]
  [ SIZE integer [ K | M | G | T | P | E ] ]
  [ REUSE ]
  [ AUTOEXTEND
      { OFF
      | ON [ NEXT integer [ K | M | G | T | P | E ] ]
      [ MAXSIZE { UNLIMITED | integer [ K | M | G | T | P | E ] } ]
      }
  ]
| [ 'filename | ASM_filename'
| ('filename | ASM_filename'
    [, 'filename | ASM_filename' ] )
]
[ SIZE integer [ K | M | G | T | P | E ] ]
[ REUSE ]
}
```

Example - Rename Datafile

Let's look at an ALTER TABLESPACE statement that renames a datafile associated with a tablespace.

For example:

```
ALTER TABLESPACE tbs_perm_01 OFFLINE NORMAL;


ALTER TABLESPACE tbs_perm_01
  RENAME DATAFILE 'tbs_perm_01.dat'
  TO 'tbs_perm_01_new.dat';
```

```
ALTER TABLESPACE tbs_perm_01 ONLINE;
```

This ALTER TABLESPACE statement would take the tablespace offline, rename the datafile from *tbl_perm_01.dat* to *tbl_perm_01_new.dat*, and then bring the tablespace back online again.

Example - Add Datafile

Let's look at an ALTER TABLESPACE statement that adds a datafile to a tablespace.

For example:

```
ALTER TABLESPACE tbs_perm_02
 ADD DATAFILE 'tbs_perm_02.dat'
   SIZE 20M
   AUTOEXTEND ON;
```

This ALTER TABLESPACE statement add the datafile called *tbs_perm_02.dat* to the *tbs_perm_02* tablespace.

Example - Drop Datafile

Let's look at an ALTER TABLESPACE statement that drops a datafile from a tablespace.

For example:

```
ALTER TABLESPACE tbs_perm_03
 DROP DATAFILE 'tbs_perm_03.dat';
```

This ALTER TABLESPACE statement drops the datafile called *tbs_perm_03.dat* to the *tbs_perm_03* tablespace.

Example - Add Tempfile

Let's look at an ALTER TABLESPACE statement that adds a tempfile to a tablespace.

For example:

```
ALTER TABLESPACE tbs_temp_04
 ADD TEMPFILE 'tbs_temp_04.dat'
   SIZE 10M
   AUTOEXTEND ON;
```

This ALTER TABLESPACE statement add the tempfile called *tbs_temp_04.dat* to the *tbs_temp_04* tablespace.

Example - Drop Tempfile

Let's look at an ALTER TABLESPACE statement that drops a tempfile from a tablespace.

For example:

```
ALTER TABLESPACE tbs_temp_05
 DROP TEMPFILE 'tbs_temp_05.dat';
```

This ALTER TABLESPACE statement drops the tempfile called *tbs_temp_05.dat* to the *tbs_temp_05* tablespace.

Oracle / PLSQL: DROP TABLESPACE statement

This Oracle tutorial explains how to use the Oracle **DROP TABLESPACE statement** with syntax and examples.

Description

The **DROP TABLESPACE statement** is used to remove a tablespace from the Oracle database. A tablespace is used to allocate space in the Oracle database where schema objects are stored.

Syntax

The syntax for the **DROP TABLESPACE statement** is:

```
DROP TABLESPACE tablespace_name
  [ INCLUDING CONTENTS [ {AND DATAFILES | KEEP DATAFILES ]
    [ CASCADE CONSTRAINTS ] ] ;
```

Parameters or Arguments

tablespace_name
: The name of the tablespace to remove from the Oracle database.

INCLUDING CONTENTS
: Optional. If you specify INCLUDING CONTENTS, all contents of the tablespace will be dropped. If there are objects in the tablespace, you must specify INCLUDING CONTENT or you will receive an error.

AND DATAFILES
: Optional. It will delete the associated operating system files. When using Oracle-managed files, you can omit the AND DATAFILES option because Oracle will automatically delete the associated operating system files.

KEEP DATAFILES
Optional. It will NOT delete the associated operating system files. When using Oracle-managed files, if you want to keep the associated operating system files, you must specify the KEEP DATAFILES option.

CASCADE CONSTRAINTS
Optional. If you specify CASCADE CONSTRAINTS, all referential integrity constraints will be dropped that meet the following criteria: A referential integrity constraint from a table **outside** *tablespace_name* that refers to a primary key or unique key on a table that is **inside** *tablespace_name*.

Example

Let's look at a simple DROP TABLESPACE statement.

For example:

```
DROP TABLESPACE tbs_perm_01
  INCLUDING CONTENTS
    CASCADE CONSTRAINTS;
```

This would drop tablespace called *tbs_perm_01*, delete all contents from the *tbs_perm_01* tablespace, and drop all referential integrity constraints (Referential integrity constraints from a table **outside** *tablespace_name* that refers to a primary key or unique key on a table that is **inside** *tablespace_name*.)

Let's look at a another DROP TABLESPACE statement.

For example:

```
DROP TABLESPACE tbs_perm_02
  INCLUDING CONTENTS AND DATAFILES
    CASCADE CONSTRAINTS;
```

This would drop tablespace called *tbs_perm_02*, delete all contents from the *tbs_perm_02* tablespace, remove the associated operating system files, and drop all referential integrity constraints (Referential integrity constraints from a table **outside** *tablespace_name* that refers to a primary key or unique key on a table that is **inside** *tablespace_name*.)

Let's look at a one file DROP TABLESPACE statement.

For example:

```
DROP TABLESPACE tbs_perm_03
  INCLUDING CONTENTS KEEP DATAFILES;
```

This would drop tablespace called *tbs_perm_03*, delete all contents from the *tbs_perm_03* tablespace, but keep the associated operating system files.

Oracle / PLSQL: Find Default Tablespaces (both Permanent and Temp)

This Oracle tutorial explains how to **find default permanent and temporary tablespaces** in Oracle with syntax and examples.

How to Find Out Default Permanent Tablespace

To find the default permanent tablespace in Oracle, you can run the following SELECT statement:

```
SELECT PROPERTY_VALUE

FROM DATABASE_PROPERTIES

WHERE PROPERTY_NAME = 'DEFAULT_PERMANENT_TABLESPACE';
```

This will query the Oracle system tables and return the value of the default **permanent** tablespace.

How to Find Out Default Temporary Tablespace

To find the default temporary tablespace in Oracle, you can run the following query:

```
SELECT PROPERTY_VALUE

FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME = 'DEFAULT_TEMP_TABLESPACE';
```

This will query the Oracle system tables and return the value of the default **temporary** tablespace.

How to Find Out Both Default Tablespaces (Permanent and Temporary)

To find both the default permanent tablespace as well as the default temporary tablespace in Oracle, you can run the following command:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE

FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME IN ('DEFAULT_PERMANENT_TABLESPACE','DEFAULT_TEMP_TABLESPACE');
```

This will query the Oracle system tables and return the **both** the default permanent and default temporary tablespaces.

Oracle / PLSQL: Set Default Tablespaces (both Permanent and Temp)

This Oracle tutorial explains how to **set default permanent and temporary tablespaces** in Oracle with syntax and examples.

How to SET Default Permanent Tablespace

First, make sure that you have created a permanent tablespace.

Next you will need to change the Oracle database to use your permanent tablespace as the default permanent tablespace.

To set the default permanent tablespace in Oracle, you can run the following ALTER DATABASE statement:

```
ALTER DATABASE DEFAULT TABLESPACE tbs_perm_01;
```

This will update the default permanent tablespace to use the tbs_perm_01 tablespace.

You can run the following query to see that the default permanent tablespace has, in fact, been changed:

```
SELECT PROPERTY_VALUE
FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME = 'DEFAULT_PERMANENT_TABLESPACE';
```

This will query the Oracle system tables and return the value of the default **permanent** tablespace.

How to Set Default Temporary Tablespace

First, make sure that you have created a temporary tablespace.

Next you will need to change the Oracle database to use your temporary tablespace as the default temporary tablespace.

To set the default temporary tablespace in Oracle, you can run the following ALTER DATABASE statement:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tbs_temp_01;
```

This will update the default temporary tablespace to use the tbs_temp_01 tablespace.

You can run the following query to see that the default temporary tablespace has, in fact, been changed:

```
SELECT PROPERTY_VALUE
FROM DATABASE_PROPERTIES
WHERE PROPERTY_NAME = 'DEFAULT_TEMP_TABLESPACE';
```

This will query the Oracle system tables and return the value of the default **temporary** tablespace.

Oracle / PLSQL: Grant/Revoke Privileges

This Oracle tutorial explains how to **grant and revoke privileges** in Oracle with syntax and examples.

Description

You can GRANT and REVOKE privileges on various database objects in Oracle. We'll first look at how to grant and revoke privileges on tables and then how to grant and revoke privileges on functions and procedures in Oracle.

Grant Privileges on Table

You can grant users various privileges to tables. These privileges can be any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

The syntax for granting privileges on a table in Oracle is:

```
GRANT privileges ON object TO user;
```

privileges

The privileges to assign. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.

Privilege	Description
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

object
The name of the database object that you are granting privileges for. In the case of granting privileges on a table, this would be the table name.

user
The name of the user that will be granted these privileges.

Example

Let's look at some examples of how to grant privileges on tables in Oracle.

For example, if you wanted to grant SELECT, INSERT, UPDATE, and DELETE privileges on a table called *suppliers* to a user name *smithj*, you would run the following GRANT statement:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON suppliers TO smithj;
```

You can also use the ALL keyword to indicate that you wish ALL permissions to be granted for a user named *smithj*. For example:

```
GRANT ALL ON suppliers TO smithj;
```

If you wanted to grant only SELECT access on your table to all users, you could grant the privileges to the public keyword. For example:

```
GRANT SELECT ON suppliers TO public;
```

Revoke Privileges on Table

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can run a revoke command. You can revoke any combination of SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALTER, INDEX, or ALL.

Syntax

The syntax for revoking privileges on a table in Oracle is:

```
REVOKE privileges ON object FROM user;
```

privileges
The privileges to revoke. It can be any of the following values:

Privilege	Description
SELECT	Ability to perform SELECT statements on the table.
INSERT	Ability to perform INSERT statements on the table.
UPDATE	Ability to perform UPDATE statements on the table.
DELETE	Ability to perform DELETE statements on the table.
REFERENCES	Ability to create a constraint that refers to the table.
ALTER	Ability to perform ALTER TABLE statements to change the table definition.
INDEX	Ability to create an index on the table with the create index statement.
ALL	All privileges on table.

object
The name of the database object that you are revoking privileges for. In the case of revoking privileges on a table, this would be the table name.

user
The name of the user that will have these privileges revoked.

Example

Let's look at some examples of how to revoke privileges on tables in Oracle.

For example, if you wanted to revoke DELETE privileges on a table called *suppliers* from a user named *anderson*, you would run the following REVOKE statement:

```
REVOKE DELETE ON suppliers FROM anderson;
```

If you wanted to revoke ALL privileges on a table for a user named *anderson*, you could use the ALL keyword as follows:

```
REVOKE ALL ON suppliers FROM anderson;
```

If you had granted ALL privileges to public (all users) on the *suppliers* table and you wanted to revoke these privileges, you could run the following REVOKE statement:

```
REVOKE ALL ON suppliers FROM public;
```

Grant Privileges on Functions/Procedures

When dealing with functions and procedures, you can grant users the ability to EXECUTE these functions and procedures.

Syntax

The syntax for granting EXECUTE privileges on a function/procedure in Oracle is:

```
GRANT EXECUTE ON object TO user;
```

EXECUTE
: The ability to compile the function/procedure. The ability to execute the function/procedure directly.

object
: The name of the database object that you are granting privileges for. In the case of granting EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

user
: The name of the user that will be granted the EXECUTE privileges.

Example

Let's look at some examples of how to grant EXECUTE privileges on a function or procedure in Oracle.

For example, if you had a function called *Find_Value* and you wanted to grant EXECUTE access to the user named *smithj*, you would run the following GRANT statement:

```
GRANT EXECUTE ON Find_Value TO smithj;
```

If you wanted to grant ALL users the ability to EXECUTE this function, you would run the following GRANT statement:

```
GRANT EXECUTE ON Find_Value TO public;
```

Revoke Privileges on Functions/Procedures

Once you have granted EXECUTE privileges on a function or procedure, you may need to REVOKE these privileges from a user. To do this, you can execute a REVOKE command.

Syntax

The syntax for the revoking privileges on a function or procedure in Oracle is:

```
REVOKE EXECUTE ON object FROM user;
```

EXECUTE
: The ability to compile the function/procedure. The ability to execute the function/procedure directly.

object
: The name of the database object that you are revoking privileges for. In the case of revoking EXECUTE privileges on a function or procedure, this would be the function name or the procedure name.

user
: The name of the user that will be revoked the EXECUTE privileges.

Example

Let's look at some examples of how to revoke EXECUTE privileges on a function or procedure in Oracle.

If you wanted to revoke EXECUTE privileges on a function called *Find_Value* from a user named *anderson*, you would run the following REVOKE statement:

```
REVOKE execute ON Find_Value FROM anderson;
```

If you had granted EXECUTE privileges to public (all users) on the function called *Find_Value* and you wanted to revoke these EXECUTE privileges, you could run the following REVOKE statement:

```
REVOKE EXECUTE ON Find_Value FROM public;
```