Oracle Objects – SEQUENCES, VIEWS, SYNONYMS, INDEXES, ROLES

Use the COURSE and PARTICIPANT tables created in earlier tutorial.

Run appropriate DMLs to ensure the COURSE and PARTICIPANT to contain tuples as shown below -

SELECT * FROM COURSE;

CID	CNAME	CREDIT
103	Computer Networks Management	3
104	Algorithms and Programming	4
101	Database Management Systems	5
102	Object-Oriented Systems	4

SELECT * FROM PARTICIPANT;

PID	PNAME	G	CID
1007	Rafael Nadal	M	103
1008	Roger Federer	M	104
1006	Serena Williams	F	101
1009	Gabriela Sabatini	F	102
1011	Maria Sharapova	F	104
1010	Sindhu P V	F	103
1012	Gopichand Pullela	M	104
1001	Albert DCosta	M	101
1002	Hitman Rohit	M	102
1003	Maria Anderson	F	102
1004	Pamela Smith	F	101
1005	Indiana Jones	M	

SEQUENCES

Information systems often use monotonically increasing sequence numbers for primary key columns (e.g., orders, shipments, registrations, or invoices). Creating a small secondary table to maintain the last/current value for each primary key leads to creating performance problems in a multiuser environment. It is much better to use sequences in such cases. Oracle cannot guarantee sequences without gaps.

CREATE SEQUENCE sequence_name
[START WITH start_num] [INCREMENT BY increment_num]
[MAXVALUE max_num | NOMAXVALUE]
[MINVALUE min_num | NOMINVALUE] [CYCLE | NOCYCLE]
[CACHE cache_num | NOCACHE] [ORDER | NOORDER];

Each sequence has two pseudo columns: NEXTVAL and CURRVAL. The sequence is defined but has to be initialized using **sequence_name.NEXTVAL** before using the CURRVAL. The NEXTVAL fetches the next value in the sequence.

To use the same number more than once CURRVAL is used instead of NEXTVAL, after the first use. Using NEXTVAL ensures that the sequence table gets incremented and that you get a unique number, so you have to use NEXTVAL first. Once NEXTVAL has been used, that number is stored in CURRVAL for use anywhere, until a NEXTVAL is use d again. If both NEXTVAL and CURRVAL are used in a single SQL statement, both will contain the value retrieved by NEXTVAL.

Default Values for parameters -

- start_num = increement_num = min_num = 1
- start_num & min_num < max_num
- cache_num = 20 (minimum number to which cache_num can be set = 2)
- max_num = NOMAXVALUE (i.e., 10^{27})
- min_num = NOMINVALUE (i.e., 1 and -10^{26} for asc & desc sequence)

```
CREATE SEQUENCE COURSE_SQ
      INCREMENT BY 1
      START WITH 106
      MINVALUE 101
      MAXVALUE 107;

SET NUMWIDTH 7
SELECT * FROM USER_SEQUENCES;
```

```
SEQUENCE_NAME      MIN_VALUE MAX_VALUE INCREMENT_BY C O CACHE_SIZE LAST_NUMBER
------------------ --------- --------- ------------ - - ---------- -----------
COURSE_SQ                101       107            1 N N         20         106
```

To find sequences in the current schema, use CAT view in place of TAB view.

```
SELECT * FROM CAT;
INSERT INTO COURSE
      VALUES (COURSE_SQ.NEXTVAL, 'Cloud Computing', 3);
SELECT COURSE_SQ.CURRVAL FROM DUAL;
SELECT * FROM USER_SEQUENCES;
INSERT INTO COURSE
      VALUES (COURSE_SQ.NEXTVAL, 'Big Data Analytics', 5);
INSERT INTO COURSE
      VALUES (COURSE_SQ.NEXTVAL, 'Oracle Certified Analyst', 3);
```

```
ERROR at line 1:
ORA-08004: sequence COURSE_SQ.NEXTVAL exceeds MAXVALUE and cannot be instantiated
```

VIEWS

A view is a predefined query on one or more tables (known as base tables). Retrieving information from a view is done in the same manner as retrieving from a table: you simply include the view in the FROM clause of a query. DML operations can also be performed on some views affecting the base tables. Views don't store rows. Rows are always stored in tables.

Benefits of Views

- A complex query can be put into a view and grant users access to the view. This allows to hide complexity from users.
- Users can be stopped from directly querying the base tables by granting them access only to the view.
- It can be allowed to a view to access only certain rows in the base tables. This allows to hide rows from an end user.

CREATE [OR REPLACE]

[FORCE | NOFORCE] **VIEW** view_name
[(alias_name [, alias_name ...])] AS subquery
[WITH [CHECK OPTION | READ ONLY] CONSTRAINT constraint_name];

- FORCE allow creating a view even if base tables don't exist. NOFORCE is default.
- WITH READ ONLY means the rows may only read from the base tables .
- WITH CHECK OPTION means that only the rows that would be retrieved by the subquery can be inserted, updated, or deleted. By default, rows are not checked.
- Two basic types of views -
  - Simple views, which contain a subquery that retrieves from one base table
  - Complex views, which contain a subquery that -- retrieves from multiple base tables, groups rows using a GROUP BY or DISTINCT clause, and contains a function call.

Use CAT database view to find views in the schema or USER_VIEWS view to find particulars about the views. USER_CONSTRAINTS view will provide information about constraints on views defined through READ ONLY & CHECK OPTION clauses.

```
SELECT VIEW_NAME, VIEW_TYPE, READ_ONLY
    FROM USER_VIEWS;

CREATE OR REPLACE VIEW TEST_VW AS
    SELECT * FROM COURSE;

DELETE FROM TEST_VW WHERE CNAME LIKE '%omput%';

SELECT * FROM COURSE;

ROLLBACK;
```

```
INSERT INTO COURSE
      VALUES (105,'Oracle Certified Analyst', 4) ;

COMMIT;

CREATE OR REPLACE VIEW  TEST_VW_RO AS
      SELECT * FROM COURSE
      WITH READ ONLY;

DELETE FROM TEST_VW_RO
      WHERE CNAME LIKE '%omput%';
ERROR at line 1:
ORA-42399: cannot perform a DML operation on a read-only view

DELETE FROM TEST_VW_RO
      WHERE CID = 104;
ERROR at line 1:
ORA-42399: cannot perform a DML operation on a read-only view


COLUMN READ_ONLY HEADING "RO" FORMAT A2

SELECT VIEW_NAME, VIEW_TYPE, READ_ONLY, EDITIONING_VIEW
      FROM USER_VIEWS
      WHERE VIEW_NAME LIKE 'TEST_VW%';

CREATE OR REPLACE VIEW  TEST_VW_CK AS
      SELECT * FROM COURSE
            WHERE CID IN (101, 105, 107)
            WITH CHECK OPTION CONSTRAINT TEST_CK_CID_157;


DELETE FROM TEST_VW_CK
      WHERE CID = 101;

SELECT * FROM TEST_VW_CK;

ROLLBACK;

INSERT INTO TEST_VW_CK
      VALUES (108,'Yet to Offer', 3)
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

INSERT INTO TEST_VW_CK
      VALUES (101,'No One Offered', 3)
ERROR at line 1:
ORA-00001: unique constraint (ABC.COURSE_PK_CID) violated

SELECT CONSTRAINT_NAME, TABLE_NAME
      FROM USER_CONSTRAINTS
      WHERE UPPER(TABLE_NAME) LIKE 'TEST_VW%';
```

SYNONYMS

A synonym allows a user to reference a table in another schema. A view is a predef. Synonyms are especially useful when accessing tables from different schemas, not owned by the user. Without synonyms, one must explicitly prefix those object names with the schema name and a period.
Oracle supports public and private synonyms . Synonyms are a "convenience" feature and don't provide any additional privileges, or create security risks. They allow to make the applications schema-independent.

Use CAT database view to find synonyms in the schema or USER_SYNONYMS view to find particulars about the synonyms.

CREATE [PUBLIC] **SYNONYM** synonym_name
FOR [schema.] object_name [@dblink];

```
CREATE OR REPLACE SYNONYM VALID_STUDENTS_SN
    FOR TEST_VW;

SELECT *
    FROM VALID_STUDENTS_SN;

COLUMN DB_LINK FORMAT A40
COLUMN SYNONYM_NAME FORMAT A20
COLUMN TABLE_NAME FORMAT A10

SELECT SYNONYM_NAME, TABLE_NAME, DB_LINK
    FROM USER_SYNONYMS;
```

Creating aPublic Synonym

To create a public synonym for a table, the user (say USR01) must have CREATE PUBLIC SYNONYM system previlege.
Assuming that USR01 holds the said privilge, let us create a public synonym named EMP_SN_01 for USR01.EMP table.

```
CREATE PUBLIC SYNONYM EMP_SN_01
    FOR ABC01.EMP;
```

Now, for another user(say USR09) having a SELECT previlege on USR01.EMP can retrieve rows from USR01.EMP through the EMP_SN_01 public synonym:

```
SELECT *
    FROM EMP_SN_01;
```

INDEXES

An index for a database table is similar in concept to a book index, except that database indexes are used to find specific rows in a table. Every index also must have a new entry made in it when an insert is done.
Generally, one should create an index on a column when retrieving a small number of rows from a table containing many rows. A good rule of thumb is --

> Create an index when a query retrieves <= 10 percent of the total rows in a table.

This means the column for the index should contain a wide range of values. These types of indexes are called " B-tree" indexes . A good candidate for B-tree indexing would be a column containing a unique value for each row (say SSN, Enroll_No).
An Oracle database automatically creates a B-tree index for the primary key of a table and for columns included in a unique constraint. For columns that contain a small range of values, a "bitmap" index can be used.

Use CAT database view to find indexes in the schema or USER_INDEXES view to find particulars about the indexes.

CREATE [UNIQUE | BITMAP **] INDEX** index_name
table_name (column_name [, column_name ...])
TABLESPACE tab_space;

Creating a B-Tree Index

UNIQUE requires that the indexed column must be unique. The **column_name** is the indexed column. An index on multiple columns (a composite index) can be created. The tablespace when not provided, the index is stored in the default userspace [usually USERS tablespace].

An attempt to create a unique index on a table that already has data in it, the command will fail if any duplicates exist. If the CREATE UNIQUE INDEX statement succeeds, any future attempt to insert (or update) a row that would create a duplicate key will fail and result in an error.

```
CREATE UNIQUE INDEX PARTICIPANT_NDX_PNAME_UQ
    ON PARTICIPANT (PNAME);
```

Assume that a table **CUSTOMER**(C_CODE, FNAME, LNAME, PHONE, AREA) exists with ample (say 100) tuples. C_CODE and LNAME are candidate keys of which C_CODE is a primary key. PHONE values are not unique, whereas AREA may take values among ['E', 'W', 'N', 'S'].

```
CREATE INDEX PARTICIPANT_NDX_PHONE
    ON PARTICIPANT(PHONE);

CREATE UNIQUE INDEX PARTICIPANT_NDX_LNAME
    ON PARTICIPANT(LNAME);
```

```
COLUMN INDEX_NAME FORMAT A25
COLUMN TABLE_NAME FORMAT A12
COLUMN UNIQUENESS FORMAT A10
COLUMN COLUMN_NAME FORMAT A35

SELECT INDEX_NAME, TABLE_NAME, UNIQUENESS, STATUS
    FROM USER_INDEXES
    WHERE TABLE_NAME IN ('CUSTOMER', 'PARTICIPANT')
    ORDER BY INDEX_NAME;

SELECT INDEX_NAME, TABLE_NAME, COLUMN_NAME
    FROM USER_IND_COLUMNS
    WHERE TABLE_NAME IN ('CUSTOMER', 'PARTICIPANT')
    ORDER BY INDEX_NAME;
```

Creating a Function-based Index

When the index definition uses a function on the column on which an unique index has already been constructed, the constructed unique index is not used by DBMS. Any query that performs a function on a column generally does not use that column's index. For the query listed below, the **PARTICIPANT_NDX_LNAME** index could not be used.

```
SELECT *
    FROM CUSTOMER
    WHERE LNAME = 'SAMANTA MCWRIGHT';
```

To allow for such queries to use indexes, a function-based index should be constructed on the column(s) appropriately incorporation the functions as below -

```
CREATE INDEX PARTICIPANT_NDX_LNAME_FN
    ON PARTICIPANT( UPPER(LNAME) );
```

Function-based indexes are useful, but should be created sparingly. Constructing more indexex on a table, will take longer to complete all insert, update, and delete operations.

The DBA must set the initialization parameter QUERY_REWRITE_ENABLED to true in order to take advantage of function-based indexes.

```
CONNECT SYSTEM/system
ALTER SYSTEM SET QUERY_REWRITE_ENABLED = TRUE;
```

Creating a BitMap Index

Bitmap indexes are typically used in data warehouses, which are databases containing very large amounts of data. The data in a data warehouse is typically read using many queries, but the data is not modified by many concurrent transactions. Data warehouses are typically used by organizations for business intelligence analysis, like monitoring sales trends.

A candidate for a bitmap index is a column that is referenced in many queries, but that contains only a small range of values.

```
CREATE BITMAP INDEX CUSTOMER_NDX_AREA_BM
    ON CUSTOMER(AREA);
```

In a bitmap index, however, a bitmap is used for each key value; the bitmap enables the database to locate a row. Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then it means that the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid.

Why BitMap Index

An index basically contains a pointer to a row in a table that contains a given index key value; the key value is used to get the rowid for the row in the table. In a B-tree index, a list of rowids is stored for each key corresponding to the rows with that key value. In a B-tree index, the database stores a list of key values with each rowid, which enables the database to locate an actual row in a table.

Bitmap indexes are typically used in tables containing large amounts of data and whose contents are not modified very often. Also, a bitmap index should only be created on columns that contain a small number of distinct values. If the number of distinct values of a column is less than 1 percent of the number of rows in the table, or if the values in a column are repeated more than 100 times, then the column is a candidate for a bitmap index.

To help tune queries that use nonselective columns in their limiting conditions, bitmap indexes can be used. Bitmap indexes should only be used if the data is infrequently updated, because they add to the cost of all data-manipulation transactions against the tables they index.

Bitmap indexes should not be used for tables involved in OLTP applications due to the internal mechanisms Oracle uses to maintain them. Restrict their usage to tables involved in batch transactions.

USERS, ROLES & PRIVILEGES

An role is a set of privileges (or type of access that each user needs, depending on his/her status and responsibilities). A user can grant or bestow specific privileges to roles and then assign roles to the appropriate users. A user can also grant privileges directly to other users.

Creating a User

The Oracle system comes with many users already created, including SYSTEM and SYS. The SYS user owns the core internal tables Oracle uses to manage database. SYSTEM owns additional tables and views. One must log onto the SYSTEM user to create other users.

```
CREATE USER username IDENTIFIED
    { BY password | EXTERNALLY | GLOBALLY AS 'external_name'}
```

To change the user password, user ALTER USER statement.

```
ALTER USER username IDENTIFIED BY new_password;
```

DBAs can change any user's password. Other users can change only their own password via the PASSWORD command.

Enforcing Password Expiration

DBA can use profiles to manage the expiration, reuse, and complexity of passwords. DBA can limit the lifetime of a password, lock an account whose password is too old, force a password to be at least moderately complex, and lock any account that has repeated failed login attempts.

1. Locking the user account on repeated failed connection attempts.

```
CREATE PROFILE DB_USER_PROFILE
    LIMIT FAILED_LOGIN_ATTEMPTS 3;

CREATE USER AAHIKA IDENTIFIED BY ANGEL
    PROFILE DB_USER_PROFILE;

GRANT CREATE SESSION TO AAHIKA;
```

If there are **three** consecutive failed connection attempts to AAHIKA account, the account will be automatically locked by Oracle. When the user then use the correct password for AAHIKA account, an error will appear.

```
CONNECT AAHIKA/ANGEL
ERROR:
ORA-28000: the account is locked
```

2. Unlocking the user account

```
ALTER USER AAHIKA ACCOUNT UNLOCK;
```

3. Establishing a maximum lifetime for a password

```
ALTER PROFILE DB_USER_PROFILE
    LIMIT PASSWORD_LIFE_TIME 90;
```

The PASSWORD_LIFE_TIME value is set to 90, so each account that uses that profile will have its password expire after 90 days. If the password has expired, the user must change it the next time it logs in.

4. Enforcing password reuse limitations

To prevent a password from being reused, use one of two profile parameters: PASSWORD_REUSE_MAX or PASSWORD_REUSE_TIME. These two parameters are mutually exclusive; if a value is set for one of them, the other must be set to **UNLIMITED**.

The PASSWORD_REUSE_TIME parameter specifies the number of days that must pass before a password can be reused. For example, if you set PASSWORD_REUSE_TIME to 30, the same password cannot be reused within 30 days.

The PASSWORD_REUSE_MAX parameter specifies the number of password changes that must occur before a password can be reused. For any attempt to reuse the password before the limit is reached, Oracle will reject the password change.

```
ALTER PROFILE DB_USER_PROFILE
    LIMIT PASSWORD_REUSE_MAX 5
          PASSWORD_REUSE_TIME UNLIMITED ;
```

Standard Roles

The most important standard roles created during database creation are -

ROLE	DESCRIPTION
CONNECT, RESOURCE, DBA	Provided for compatibility with previous versions.
DELETE_CATALOG_ROLE EXECUTE_CATALOG_ROLE SELECT_CATALOG_ROLE	These roles are provided for accessing data dictionary views and packages.
EXP_FULL_DATABASE IMP_FULL_DATABASE	These roles are provided for convenience in using the Import and Export utilities.

CONNECT, RESOURCE, and DBA are provided for backward compatibility. CONNECT gives users an ability to log in and perform basic functions. In Oracle 11g, users with CONNECT may no longer create tables, views, sequences, clusters, synonyms, and database links.

Users with RESOURCE can create their own tables, sequences, procedures, triggers, datatypes, operators, indextypes, indexes, and clusters. It will receive UNLIMITED TABLESPACE system privilege, allowing them to bypass quotas on all tablespaces.

Users with the DBA role can perform database administration functions, including creating and altering users, tablespaces, and objects.

The GRANT & REVOKE commands

Any system privilege or role can be granted to another user, to another role, or to **public**. The WITH ADMIN OPTION clause permits the grantee to bestow the privilege or role on other users or roles. The ALL clause grants the user or role all privileges except the SELECT ANY DICTIONARY system privilege.

GRANT { system_privilege | ROLE | ALL [privileges] }
[, { system_privilege | ROLE | ALL [privileges] } ...]
TO { USER | ROLE } [, { USER | ROLE } ...]
[IDENTIFIED BY password]
[WITH ADMIN OPTION];

The grantor can revoke a role from a user. Privileges grated can be taken away with REVOKE command. A user with DBA role can revoke CONNECT, RESOURCE, DBA, or any other privilege or role from anyone, including another DBA.

REVOKE { system_privilege | ROLE | ALL [privileges] }
[, { system_privilege | ROLE | ALL [privileges] } ...]
FROM { USER | ROLE } [, { USER | ROLE }] ... ;

Revoking everything from a given user does not eliminate that user from Oracle, nor does it destroy any tables that user had created; it simply prohibits that user's access to them. Other users with access to the tables will still have exactly the same access they've always had.

Removing a User

Use DROP USER command to remove a user and all resources owned by that user.

DROP USER username [CASCADE];

The CASCADE option drops the user along with all the objects owned by the user, including referential integrity constraints. The CASCADE option invalidates views, synonyms, stored procedures, functions, or packages that refer to objects in the dropped user's schema. If you don't use the CASCADE option and there are still objects owned by the user, Oracle does not drop the user and instead returns an error message.

Creating a Role

In addition to default roles, user may create own roles. These roles acn comprise table or system privileges or a combination of both.

Any system privilege or role can be granted to another user, to another role, or to **public**. The WITH ADMIN OPTION clause permits the grantee to bestow the privilege or role on other users or roles. The ALL clause grants the user or role all privileges except the SELECT ANY DICTIONARY system privilege.

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

When a role is first created, it has no privileges associated with it.

Adding and Removing Password from a Role

ALTER ROLE role_name IDENTIFIED BY password;

ALTER ROLE role_name NOT IDENTIFIED;

Default Role & Enabling/Desabling Roles

When a user's account is altered, a list of default roles for that user can be created via the DEFAULT ROLE clause of the alter user command. The default action of this command sets all of a user's roles as default roles, enabling all of them every time the user logs in.

ALTER USER username
DEFAULT ROLE { [role1, role2]
[ALL | ALL EXCEPT role1, role2] [NONE] };

For example,

ALTER USER AAHIKA DEFAULT ROLE MANAGER;

To enable a nondefault role, use the SET ROLE command,

SET ROLE SUPERVIZOR;
SET ROLE ALL EXCEPT CLERK;

To disable a role, use the command,

SET ROLE **NONE**;

To know what roles are enabled within the current session, use the SESSION_ROLES view. Use SESSION_PRIVS to know the currently enabled system privileges.

Revoking Privileges from a Role

REVOKE object_privilege **ON** object **FROM** role;

REVOKE SELECT ON CUSTOMER FROM AAHIKA;