CSE – 302
DATABASE MANAGEMENT SYSTEMS SESSIONAL

LAB - 12

ABSTRACT DATA TYPE

Abstract Data Type

- Sometimes we may want a type of data which holds all types of data including numbers, chars and special characters something like this.
- We cannot achieve this using pre-defined types.
- In Oracle 7 there was no way to extend the data typing.

Abstract Data Type (Contd.)

- For example, to select all of the address information from a table, we have to specify all of the columns in the group.

```
CREATE TABLE STUDENT
(
ID NUMBER(2),
NAME VARCHAR2(20),
HOUSENO NUMBER(5),
STREETNO NUMBER(5),
CITY VARCHAR(10)
);
```

These data columns can be group together by data type

Abstract Data Type (Contd.)

```
CREATE TYPE ADDR AS OBJECT
(
        HOUSENO NUMBER(3),
        STREETNO NUMBER(5),
        CITY VARCHAR(10)
);
```

The data type ADDR is created

Now we can create table using new abstract data types that we have created:

```
CREATE TABLE STUDENT
(
        ID NUMBER(2),
        NAME VARCHAR2(20),
        ADDRESS ADDR
);
```

Now we can reference ADDRESS in our sql as if it were a primitive data type.

Abstract Data Type (Contd.)

- We can’t insert data into ADDR. The reason is straightforward.
- A data type describes data, it does not store data.
- We cannot store data in a NUMBER data type, so we cannot store data in a data type that we define, either.
- To store data, you have to create a table that uses your data type.

Abstract Data Type (Contd.)

```
INSERT INTO STUDENT
 VALUES
(
        1, 'MASUD', ADDR(111, 43, 'DHAKA')
);
```

Abstract Data Type (Contd.)

```
SELECT S.ADDRESS.HOUSENO, S.ADDRESS.CITY FROM STUDENT S;
```

This SQL would produce a listing like this:

ADDRESS.HOUSENO	ADDRESS.CITY
111	DHAKA
234	KHULNA

```
SELECT S.ADDRESS.HOUSENO FROM STUDENT S WHERE
S.ADDRESS.CITY LIKE 'D%' ;
```

This would produce the following listing:

ADDRESS.HOUSENO
111

Abstract Data Type (Contd.)

- UPDATING ADT TABLES

```
UPDATE STUDENT S SET S.ADDRESS.CITY = 'RAJSHAHI'
WHERE S.ADDRESS.HOUSENO = 111;
```

- DELETE FROM ADT TABLES

```
DELETE STUDENT S
WHERE S.ADDRESS.HOUSENO = 111;
```

- DROPPING ADT

```
DROP TYPE ADDR;
```

Nesting In Abstract Data Type

```
CREATE TYPE ADDRESS_TY AS OBJECT
(
        STREET VARCHAR2(20),
        CITY VARCHAR2(10),
        PIN NUMBER
);

CREATE TYPE PERSON_TY AS OBJECT
(
        NAME VARCHAR2 (20),
        ADDRESS ADDRESS_TY
);
```

- Now PERSON_TY contains Name and address of a person we can use this to create table.

Nesting In Abstract Data Type (Contd.)

```
CREATE TABLE CUSTOMER
(
CUSTOMER_ID NUMBER,
PERSON PERSON_TY
);
```

- To Insert rows into CUSTOMER do following.

```
INSERT INTO CUSTOMER
VALUES (1,
PERSON_TY ('SANAN', ADDRESS_TY ('102 Dhanmondi',
'DHAKA',
10101))
);
```

Nesting In Abstract Data Type (Contd.)

- To select data from customer table:

```
SELECT CUSTOMER_ID,
C.PERSON.ADDRESS.STREET
FROM CUSTOMER C;
```

What is a Schema?

- A ***schema*** is a collection of database objects (tables, views, stored procedure etc.) associated with one particular database username.
- This username is called the ***schema owner,*** or the owner of the related group of objects.
- You may have one or multiple schemas in a database.
- Basically, any user who creates an object has just created his or her own schema.
- So, based on a user's privileges within the database, the user has control over objects that are created, manipulated, and deleted.
- A schema can consist of a single table and has no limits to the number of objects that it may contain, unless restricted by a specific database implementation.

What is a Schema? (Contd.)

- Suppose Your username is **USER1**.
- Now you log on to the database and then create a table called **EMPLOYEE_TBL**.
- According to the database, your table's actual name is **USER1.EMPLOYEE_TBL**.
- The schema name for that table is **USER1**, which is also the owner of that table. You have just created the first table of a schema.
- You do not have to refer to the schema name. For instance, you could refer to your table as either one of the following

  EMPLOYEE_TBL

  USER1.EMPLOYEE_TBL

What is a Schema? (Contd.)

- Two user accounts in the database that own tables:
  - *USER1 and USER2.*
- Each user account has its own schema.
- Some examples for how the two users can access their own tables and tables owned by the other user follow:
- User1 has two table (table1, test)
- User 2 has two tables (table10, test)

USER1 accesses own table1:	TABLE1
USER1 accesses own test:	TEST
USER1 accesses USER2's table10:	USER2.TABLE10
USER1 accesses USER2's test:	USER2.TEST

GRANT AND REVOKE

Data Control Language Statements

- Data Control Language Statements are used to grant privileges on tables, views, sequences, synonyms, procedures to other users or roles.
- **GRANT** :Use to grant privileges to other users or roles.
  REVOKE :Use to take back privileges granted to other users and roles.

- Privileges are of two types :
  - **System Privileges:** System Privileges are normally granted by a DBA to users. Examples of system privileges are CREATE SESSION, CREATE TABLE, CREATE USER etc.
  - **Object privileges:** Object privileges means privileges on objects such as tables, views, synonyms, procedure. These are granted by owner of the object.

GRANT AND REVOKE

- Object privileges are:

ALTER	Change the table definition with the ALTER TABLE statement.
DELETE	Remove rows from the table with the DELETE statement. Note: You must grant the SELECT privilege on the table along with the DELETE privilege.
INDEX	Create an index on the table with the CREATE INDEX statement.
INSERT	Add new rows to the table with the INSERT statement.
REFERENCES	Create a constraint that refers to the table. You cannot grant this privilege to a role.
SELECT	Query the table with the SELECT statement.
UPDATE	Change data in the table with the UPDATE statement.
	Note: You must grant the SELECT privilege on the table along with the UPDATE privilege.

GRANT AND REVOKE (Contd.)

GRANT:

- Grant is use to grant privileges on tables, view, procedure to other users or roles

Examples:

- Suppose user1 owns employee table. Now s/he wants to grant select, update, insert privilege on this table to other user “user2”.

```
GRANT SELECT, UPDATE, INSERT ON EMPLOYEE TO USER2;
```

- Suppose user1 wants to grant all privileges on employee table to user2.

```
GRANT  ALL ON EMPLOYEE TO USER2;
```

- If user1 wants to grant select privilege on employee to all other users of the database.

```
GRANT SELECT ON EMPLOYEE TO PUBLIC;
```

GRANT AND REVOKE (Contd.)

- Suppose user1 wants to grant update and insert privilege on only certain columns not on all the columns then include the column names in grant statement. For example, if s/he wants to grant update privilege on emp_name column only and insert privilege on emp_no and emp_name columns only, then the following statement will do that job:

```
GRANT UPDATE (EMP_NAME), INSERT (EMP_NO, EMP_NAME)
 ON EMPLOYEE TO USER2;
```

- To grant select statement on employee table to user2 and to make user2 be able further pass on this privilege you have to give WITH GRANT OPTION clause in GRANT statement like this.

```
GRANT SELECT ON EMPLOYEE TO USER2 WITH GRANT OPTION;
```

GRANT AND REVOKE (Contd.)

REVOKE:

- Use to revoke privileges already granted to other users.
- For example to revoke select, update, insert privilege user1 has granted to user2:

  REVOKE SELECT, UPDATE, INSERT ON EMPLOYEE FROM USER2;

- To revoke select statement on employee granted to public give the following command.

  REVOKE SELECT ON EMPLYEE FROM PUBLIC;

- To revoke update privilege on emp_name column and insert privilege on emp_no and emp_name columns give the following revoke statement.

  REVOKE UPDATE, INSERT ON EMPLOYEE FROM USER2;

- Note :You cannot take back column level privileges. Suppose you just want to take back insert privilege on emp_name column then you have to take back the whole insert privilege.

GRANT AND REVOKE (Contd.)

ROLES:

- A ***role*** is a group of Privileges. A role is very handy in managing privileges, Particularly in such situation when number of users should have the same set of privileges.
- For example you have four users : **user1, user2, user3, user4** in the database. To these users you want to grant select ,update privilege on employee table, select, delete privilege on department table. To do this first create a role by giving the following statement

```
CREATE ROLE CLERKS
```

- Then grant privileges to this role.

```
GRANT SELECT,UPDATE ON EMP TO CLERKS;
GRANT SELECT,DELETE ON DEPT TO CLERKS;
```

- Now grant this clerks role to users like this

```
GRANT CLERKS TO USER1, USER2, USER3, USER4 ;
```

- Now all 4 users have all the privileges granted on clerks role.

GRANT AND REVOKE (Contd.)

- Suppose after one month you want grant delete on privilege on employee table all these users then just grant this privilege to clerks role and automatically all the users will have the privilege.

```
GRANT DELETE ON EMP TO CLERKS;
```

- If you want to take back update privilege on employee table from these users just take it back from clerks role.

```
REVOKE UPDATE ON EMP FROM CLERKS;
```

- To Drop a role

```
DROP ROLE CLERKS;
```

LISTING INFORMATION ABOUT PRIVILEGES

- To see which table privileges are granted by you to other users.

  SELECT * FROM USER_TAB_PRIVS_MADE

- To see which table privileges are granted to you by other users

  SELECT * FROM USER_TAB_PRIVS_RECD;

- To see which column level privileges are granted by you to other users.

  SELECT * FROM USER_COL_PRIVS_MADE

- To see which column level privileges are granted to you by other users

  SELECT * FROM USER_COL_PRIVS_RECD;

- To see which privileges are granted to roles

  SELECT * FROM USER_ROLE_PRIVS;

ORACLE SYNONYM

Oracle Synonym

- A synonym is an alias for a schema object.
- Synonyms can provide a level of security by masking the name and owner of an object and by providing location transparency for remote objects of a distributed database.
- Also, they are convenient to use and reduce the complexity of SQL statements for database users.

Oracle Synonym (Contd.)

- Synonyms can be created as **PRIVATE** (by default) or **PUBLIC**.
- Public synonyms are available to all users in the database.
- Private synonyms exist only in specific user schema (they are available only to a user and to grantees for the underlying object)
- The syntax for a synonym is as follows:
- To create private synonym in your own schema

```
CREATE [PUBLIC] SYNONYM SYNONYM_NAME FOR OBJECT_NAME
```

Oracle Synonym (Contd.)

- Assume we have a database with a schema called USER1. This schema contains a table called ORDERS.

```
CREATE PUBLIC SYNONYM ORDERS_DATA FOR USER1.ORDERS;
```

- From now on, user1 can query this table using the synonym:

```
SELECT * FROM ORDERS_DATA;
```

Oracle Synonym (Contd.)

- To **create a private synonym in your own schema**, you must have the **CREATE SYNONYM** system privilege.
- To **create a private synonym in another user's schema**, you must have the **CREATE ANY SYNONYM** system privilege.
- To create a PUBLIC synonym, you must have the CREATE PUBLIC SYNONYM system privilege.

```
GRANT CREATE ANY SYNONYM, CREATE SYNOYM CREATE PUBLIC SYNONYM,
DROP PUBLIC SYNONYM TO USERNAME;        // by database administrator
```

Oracle Synonym (Contd.)

- In Oracle you can refer to synonyms in the following statements:
  - select
  - insert
  - update
  - delete
  - flashback table
  - explain plan
  - lock table

When to Use Synonyms

- As database systems grow and applications improve, there is usually a need to change the names of tables and views to better reflect their new functionality.
- For example, a stored procedure named ADD_CLIENT might be used to verify credit limits as well as to add a new client.
- Therefore, it might be natural to change the name of the procedure to something like VERIFY_NEW_CLIENT.
- The problem comes in changing the name in all places before placing the revised application into production.
- Thanks to synonyms, both names might be used in the application and database.

Synonyms for Synonyms

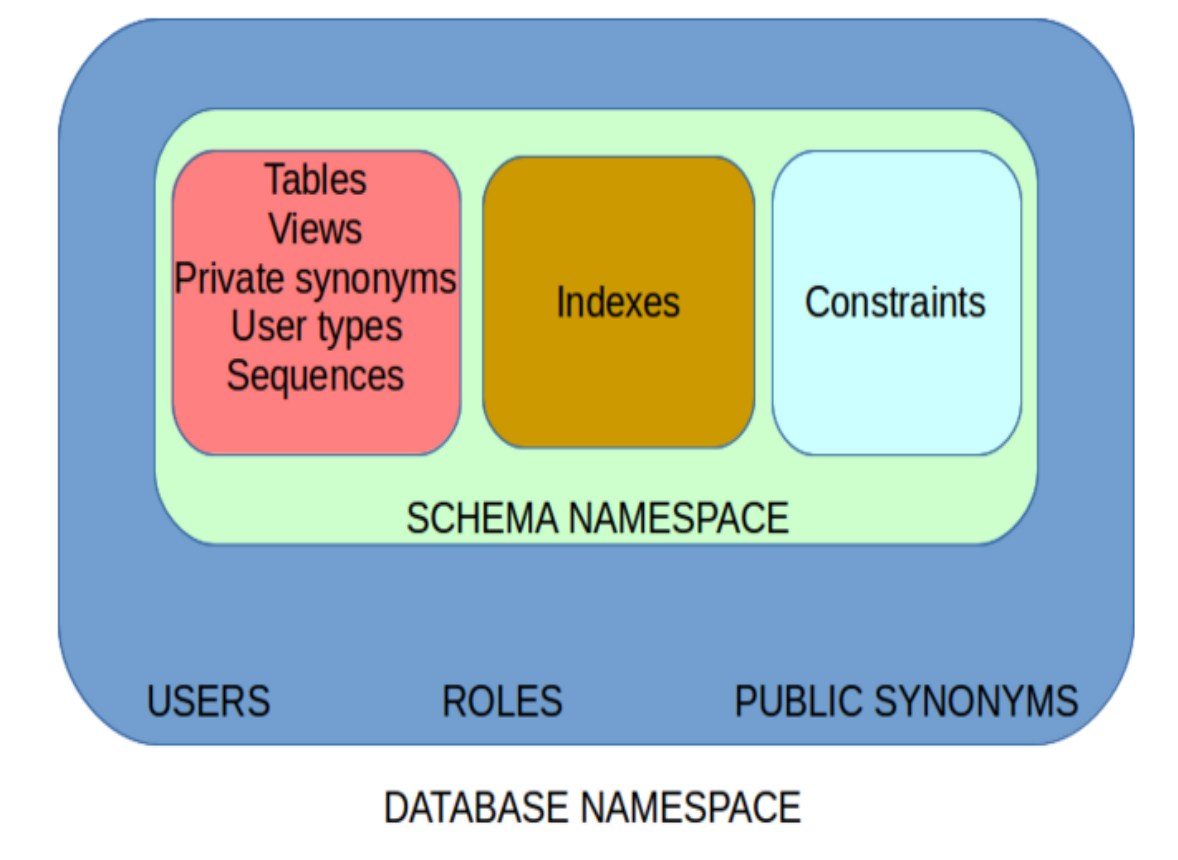
- A curious feature of a synonym is that each may have its own synonym or many synonyms.
- Let's assume that we have a table with a formal name such as **USER1_PURCHASE_TABLE**, a simple name to allow easier querying **PURCHASE_TABLE**, and a lazy name, **PT**, for people who prefer to type fewer characters.
- PURCHASE_TABLE and PT could be defined as synonyms for the base table:

```
CREATE SYNONYM PURCHASE_TABLE FOR TABLE REL001_PURCHASE_TABLE;
CREATE SYNONYM PT FOR TABLE REL001_PURCHASE_TABLE;
```

OR

```
CREATE SYNONYM PURCHASE_TABLE FOR REL001_PURCHASE_TABLE;
CREATE SYNONYM PT FOR PURCHASE_TABLE;
```

Naming Synonyms



- According to the picture: Public synonyms are non-schema objects, when private synonyms as tables are schema objects.
- USER, ROLE and PUBLIC SYNONYM are in their own collective namespace.
- TABLE, VIEW, SEQUENCE, PRIVATE SYNONYM have their own unique namespace.
- INDEXes have their own unique namespace.
- CONSTRAINTs objects have their own unique namespace within a given schema

Naming Synonyms (Contd.)

- While the objects **don't share the same namespace**, you can give them the **same names**.
- Remember, that you can have:
  - a table and public synonym with the same name
  - a public synonym and a private synonym with same name
- You cannot have:
  - a table and a private synonym with the same name inside the same schema.

VIRTUAL COLUMN

Virtual Column

- A **virtual** column is a column whose value is computed as a reaction for a defined operation.
- This means that they are computed “on the fly” – only when needed.
- It’s accessible like any other column except that there is no physical space associated with it.
- If a SQL query doesn’t reference a virtual column, the value is not calculated.
- Of course, it’s impossible to insert a value into a virtual column. The attempt will cause a SQL error.
- It is a feature of Oracle 11g.

);

Virtual Column (Contd.)

Syntax:

```
CREATE TABLE TABLE_NAME
( ...
  ...
COLUMN_NAME [DATATYPE] [GENERATED ALWAYS] AS (EXPRESSION) [VIRTUAL]
);
```

- Keyword AS is sufficient to create a virtual column.
- Others: GENERATED ALWAYS and VIRTUAL are optional as well as datatype. If datatype is omitted, the virtual column **datatype** is based on the **result of the expression**.

Virtual Column (Contd.)

Example:

```
CREATE TABLE PRODUCT(
ID NUMBER(5),
NAME VARCHAR2(30),
PRICE NUMBER (5,2),
VAT NUMBER(5,2),
PRICE_INCLUDING_VAT AS (PRICE+PRICE*VAT)
);
```

```
INSERT INTO PRODUCT VALUES (1, 'BOOK', 20, 0.07);
```

The corresponding row will look as follows:

id	name	price	vat	price_including_vat
1	book	20	0.07	21.4

Virtual Column (Contd.)

- To get information about an expression that produces the value of a virtual column, check the data_default column in view USER_TAB_COLS:

```
SELECT column_name , data_type, data_default
FROM   USER_TAB_COLS
WHERE  table_name = 'PRODUCT';
```

	COLUMN NAME	DATA_TYPE	DATA_DEFAULT
1	ID	NUMBER	(NULL)
2	NAME	VARCHAR2	(NULL)
3	PRICE	NUMBER	(NULL)
4	VAT	NUMBER	(NULL)
5	PRICE_INCLUDING_VAT	NUMBER	PRICE + PRICE * VAT

Truncate Table Command

- The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.
- You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.

Syntax:

```
TRUNCATE TABLE TABLE_NAME;
```

THANK YOU