Birla Institute of Technology and Science, Pilani.
Database Systems
Lab No #2

More on Creating, Deleting, and Altering Tables: (DDL)

- In the previous section, creation tables with defaults values, NOT NULL constraint, UNIQUE constraint, Primary key, foreign key and check constraints have been discussed.

Creating a table from another table:

- SQL allows you to create and populate a new table from existing tables with one statement. Execute the following query.

```
CREATE TABLE students_copy(idno, name, dob, cga, age) AS
SELECT idno, name, dob, cgpa, age
FROM students
```

Create Type

- Oracle SQL allows the specification of user-defined *types* based on existing data types. These *types* work just like any other data type.

```
CREATE OR REPLACE TYPE <type_name>
AUTHID <CURRENT USER | DEFINER>
AS OBJECT (
<attribute> <attribute data_type>,
<inheritance clause>,
<subprogram spec>)
<FINAL | NOT FINAL> <INSTANTIABLE | NOT INSTANTIABLE>;
```

```
CREATE OR REPLACE TYPE phone_t AS OBJECT (
a_code   CHAR(3),
p_number CHAR(8));
```

```
CREATE TABLE phone (
per_id    NUMBER(10),
per_phone phone_t);
```

```
INSERT INTO phone
(per_id, per_phone)
VALUES
(1, phone_t('206', '555-1212'));
```

```
SELECT p.per_phone.p_number
FROM phone p
WHERE p.per_phone.a_code = '206';
```

Alter Table

- You can modify the columns and constraints of a table using the following:

```
ALTER TABLE <table name> <action>;
```

There are several types of actions:

- **Rename table:**

```
ALTER TABLE phone RENAME TO phone1;
```

- **ADD [COLUMN] *<column definition>***

Adds a new column to *<table name>*. The *<column definition>* may contain any of the elements of a column definition in CREATE TABLE, such as types and constraints.

Rules for adding a Column

1) You may add a column at any time if NOT NULL isn't specified.
2) You may add a NOT NULL column in three steps
   - 2.1 Add the column without NOT NULL specified.
   - 2.2 Fill every row in that column with data.
   - 2.3 Modify the column to be NOT NULL.

```
SQL> alter table staff add email varchar(20);
SQL> alter table students add hostel varchar(20);
```

- **MODIFY *<column name>* [SET DEFAULT *<value expression>* | DROP DEFAULT]**

Change or drop the specified default value from *<column name>*. Your DBMS will likely let you alter much more than just the default values.

Rules for modifying a Column

1) You can increase a character column's width at any time.
2) You can increase the number of digits in a NUMBER Column at any time.
3) You can increase or decrease the number of decimal places in a NUMBER Column at any time.

In addition, if a column is NULL for every row of the table, you can make any of these changes.

1) You can change its datatype.
2) You can decrease a character Column's width.
3) You can decrease the number of digits in a NUMBER column.

```
SQL> alter table course modify  compcode      numeric(5,0);
```

- **DROP COLUMN *<column name>* [CASCADE | RESTRICT]**

Delete *<column name>* from *<table name>*. If there are any external dependencies on this column, the drop will be rejected. If CASCADE is specified, the DBMS will attempt to eliminate all external dependencies before deleting the column. If RESTRICT is specified, the DBMS will not allow the DROP if any external dependencies would be violated. This is the default behavior.

```
SQL> alter table staff drop column email;
```

- **ADD *<table constraint>***

Add a new constraint to *<table name>*. *<table constraint>* uses the same syntax as table constraints in CREATE TABLE.

```
SQL> alter table students add constraint agecheck
     check(age>15);
```

- **Add Multiple things: constraints/columns**

```
SQL> alter table students add (
testcol number(9) NOT NULL, --add a new column
constraint agecheck    check(age>15),
constraint studnetpk primary key(idno),
constraint cgcheck check(cgpa>=2.0)
);
```

- **DROP CONSTRAINT *<constraint name>*** [**CASCADE | RESTRICT**]

Delete *<constraint name>* from *<table name>*. If there are any external entities that depend on this constraint, the drop could invalidate the entity. If RESTRICT is specified and an entity could be invalidated, the DBMS will not drop the constraint. This is the default behavior. If CASCADE is specified, the DBMS will attempt to eliminate all potentially violated entities before deleting the constraint

```
SQL> alter table course drop constraint un_course;
```

Generated Values

- SQL provides several mechanisms for auto generation of data.
- The following query will not work in Oracle although supported by SQL standard.

```
CREATE TABLE purchases(
orderno INTEGER GENERATED BY DEFAULT AS IDENTITY,
vendorid CHAR(5),
ordertime DATETIME
);
```

- This IDENTITY is not supported by Oracle 11g. A cross comparison of identity implementation in different DBMS is here. (http://troels.arvin.dk/db/rdbms/)

Sequences

- Identity is one way of automatically generating values in SQL, but a more general approach is a sequence. A sequence is a database object that generates values. The values are numeric, but the sequence can be ascending or descending, can skip values, and can start at any valid value.

- The 2003 standard defines a sequence as follows:

```
CREATE SEQUENCE <sequence name> [AS <data type>]
[START WITH <signed numeric literal>]
[INCREMENT BY <signed numeric literal>]
[MAXVALUE <signed numeric literal> | NO MAXVALUE]
[MINVALUE <signed numeric literal> | NO MINVALUE]
[CYCLE | NO CYCLE]
```

- Effectively, each time a new value is generated by a sequence, the increment value is added to the current value. If the increment value is negative, then the current value decreases and the sequence is *descending.* If the increment value is positive, then the current value increases and the sequence is *ascending.* It is illegal for the increment value to be 0.

- The following statement creates the sequence customers_seq. This sequence could be used to provide customer ID numbers when rows are added to the customers table.

```
CREATE SEQUENCE customers_seq
 START WITH     1000
 INCREMENT BY   1
 NOCACHE
 NOCYCLE;
```

- CACHE 30 refers to pre-computing 30 sequence values and keeping in memory for faster processing. NOCACHE refers to no pre-computing.
- The first reference to customers_seq.nextval returns 1000. The second returns 1001. Each subsequent reference will return a value 1 greater than the previous reference.

```
SELECT customers_seq.CURRVAL FROM DUAL;
SELECT customers_seq.NEXTVAL FROM DUAL;


create table customers(
    customer_id number(9) primary key,
    customer_name char(50)
   );


INSERT INTO CUSTOMERS(CUSTOMER_ID, CUSTOMER_NAME) VALUES
(customers_seq.nextval,'Jiva');
```

dropping a table

```
DROP TABLE <tablename> [CASCADE | RESTRICT];
```

```
Sql> drop table student;
```

Modifying Data: (DDL)

- SQL provides three statements for modifying data: INSERT, UPDATE, and DELETE.

INSERT

- The INSERT statement adds new rows to a specified table. There are two variants of the INSERT statement. One inserts a single row of values into the database, whereas the other inserts multiple rows returned from a SELECT.
- The most basic form of INSERT creates a single, new row with either user-specified or default values. This is covered in the previous lab.

- The INSERT statement allows you to create new rows from existing data. It begins just like the INSERT statements we've already seen; however, instead of a VALUES list, the data are generated by a nested SELECT statement. The syntax is as follows:

```
INSERT INTO <table name>
[(<attribute>, : : :, <attribute>)]
<SELECT statement>;
```

- SQL then attempts to insert a new row in *<table name>* for each row in the SELECT result table. The attribute list of the SELECT result table must match the attribute list of the INSERT statement.

```
SQL> create table customers_copy(
    customer_id number(9) primary key,
    customer_name char(50)
   );

SQL> insert into customers_copy(customer_id, customer_name) select *
from customers;
```

UPDATE

- You can change the values in existing rows using UPDATE.

```
UPDATE <table-name> [[AS] <alias>]
SET <column>=<expression>, : : :, <attribute>=<expression>
[WHERE <condition>];
```

- UPDATE changes all rows in *<table name>* where *<condition>* evaluates to true. For each row, the SET clause dictates which attributes change and how to compute the new value. All other attribute values do not change. The WHERE is optional, but beware! If there is no WHERE clause, UPDATE changes *all* rows. UPDATE can only change rows from a single table.

```
Sql> update  student  set name='test', email='test@yahoo.com' where
idno='2000A7PS001';

Sql> update student set email='f2000001@bits-pilani.ac.in' where
idno='2000A7PS001';

Sql> update student set  cgpa=10; (check out what this command does)
```

DELETE

- You can remove rows from a table using DELETE.

```
DELETE FROM <table name> [[AS] <alias>]

[WHERE <condition>];
```

- DELETE removes all rows from *<table name>* where *<condition>* evaluates to true. The WHERE is optional, but beware! If there is no WHERE clause, DELETE removes *all* rows. DELETE can only remove rows from a single table.

```
Sql> delete from student where idno='2000A7PS001'; (delete records
where idno = '2001A7PS001')

Sql> Delete from student; (Deletes all the records from the student
table)

Sql> truncate table student; (see what it does to your table)
```

- TRUNCATE removes all rows without making entries into recovery log but it can't be rolled back. TRUNCATE is faster than DELETE. TRUNCATE is DDL statement and DELETE is a DML statement.

Exercise:

- Create the following tables (Don't specify any constraints):

Category_details (category_id numeric (2), category_name varchar (30))

Sub_category_details (sub_category_id numeric(2), category_id numeric(2),sub_category_name varchar(30))

Product_details (Product_id numeric (6), category_id numeric(2),sub_category_id numeric(2), product_name varchar(30))

Now perform the following operations:

1) Add a primary key constraint (without any constraint name) on column category_id of category_details table.
2) Add a primary key constraint with a constraint name on column sub_category_id of sub_category_details table.
3) Add a foreign key constraint with constraint name on column category_id of sub_category_details table referencing category_id of category_details table.

4) For product_details table add primary key constraint on product_id. Also add foreign key constraint on category_id and sub_category_id columns referencing category_details(category_id) and sub_category_details (sub_category_id). Give appropriate names for all constraints.
5) Add a new column (price numeric(2)) to product_details table
6) Modify the data type of price to numeric(6,2)
7) Insert four tuples in the table. (With valid data)
8) Drop the price column
9) Using the rules defined in the beginning, add a new column BRANDNAME varchar(20) NOT NULL
10) Rename Category_details table to Cat_dt

--&--