Birla Institute of Technology and Science, Pilani.
CS F212 Database Systems
Lab No #2

Data Manipulation Language(DML)

In today's lab, DML part of SQL will be discussed.

1. Modifying Data: (DML)

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

a. 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. For example -

```
SQL>  CREATE TABLE CUSTOMER(
      CUSTOMER_ID VARCHAR(20) PRIMARY KEY,
      CUSTOMER_NAME CHAR(50)
      );
SQL> INSERT INTO CUSTOMER VALUES('2016A7PS0339P','ALEX');
SQL> INSERT INTO CUSTOMER VALUES(‘2016A7PS0018P’,’GIRINATH’);
      SQL> CREATE TABLE CUSTOMERS_COPY(
      CUSTOMER_ID VARCHAR(20) PRIMARY KEY,
      CUSTOMER_NAME CHAR(50)
   );

SQL>  INSERT INTO CUSTOMERS_COPY(CUSTOMER_ID, CUSTOMER_NAME) SELECT *
      FROM  CUSTOMER;
```

b. 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>  CREATE TABLE STUDENTS (
      IDNO CHAR(50),
      NAME VARCHAR(100),
      EMAIL VARCHAR(100),
      CGPA NUMERIC(4,2));

SQL>  INSERT INTO STUDENTS(NAME,EMAIL,IDNO,CGPA)
      VALUES('ALEX','123@@gmail.com','2000A7PS001',7.34);

SQL>  INSERT INTO STUDENTS(NAME,EMAIL,IDNO,CGPA)
      VALUES(‘GIRI’,'123@@gmail.com','2000A7PS001',7.34);

SQL>  UPDATE STUDENTS SET NAME='TEST', EMAIL='test@yahoo.com' WHERE
      IDNO='2000A7PS001';

SQL>  UPDATE STUDENTS SET EMAIL='f20170011@pilani.bits-pilani.ac.in'
      WHERE IDNO= '2000A7PS001';

SQL>  UPDATE STUDENTS SET CGPA=10;
```

c. 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 STUDENTS WHERE IDNO=’2000A7PS001’; -- (DELETE RECORDS
      WHERE IDNO = ‘2001A7PS001’)

SQL>  DELETE FROM STUDENTS; -- (DELETES ALL THE RECORDS FROM THE STUDENT
      TABLE)

SQL>  TRUNCATE TABLE STUDENT; --(see what it does to your table)
```

TRUNCATE removes all rows but it can't be rolled back (undone). TRUNCATE is faster than DELETE. TRUNCATE is DDL statement and DELETE is a DML statement.

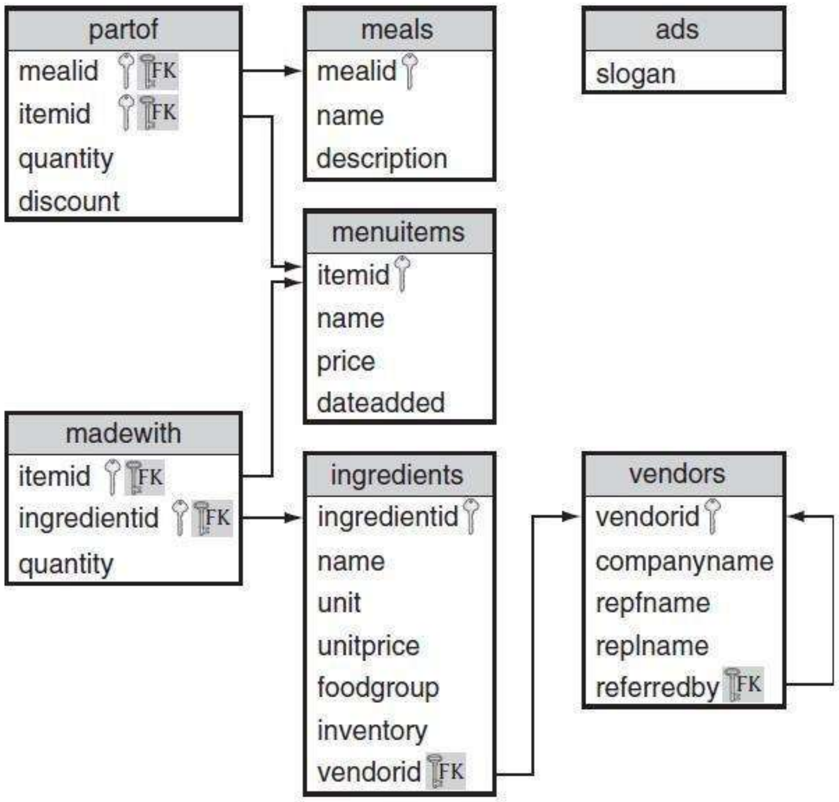
2. **Data Retrieval**

For retrieval, SELECT, WHERE, FROM, GROUP BY, HAVING clauses are used extensively. We will practice some SQL queries on a schema of a restaurant chain. The schema has eight tables and one view.

For Running the Script in Microsoft SQL Server

- **In** Microsoft SQL Server Management Studio**, on the menu, select** File > Open > File**.**
- **In the** Open File **dialog box, browse for the script file, and then click** OK**.**
- On the SQL Editor toolbar, select the appropriate database, and then click **Execute** to run the script.

Download the script 'restaurant.sql'. The table schema is given below:



Identify the referential integrity constraints from the schema. **SELECT, FROM, WHERE clauses:**

❖ The SELECT clause is used to select data from a database. The SELECT clause is a query expression that begins with the SELECT keyword and includes a number of elements that form the expression. WHERE clause is used to specify the Search Conditions. The operators commonly used for comparison are =, > , < , >=, <= , <>.

Create a value menu of all items costing $0.99 or less.

```
SELECT name
       FROM items
       WHERE PRICE <= 0.99;
```

The following operators are also used in WHERE clause.

BETWEEN	Between two values(inclusive)	Vendorid BETWEEN 2 AND 10
IS NULL	Value is Null	Referredby IS NULL
LIKE	Equal Sting using wilds cards(e.g. '%','_')	Name LIKE 'pri%'
IN	Equal to any element in list	Name IN ('soda',' water')
NOT	Negates a condition	NOT item IN ('GDNSD','CHKSD')

Find the food items added after 1999

```
SQL> SELECT *
     FROM items
     WHERE dateadded > '1999-12-31';
```

Find all items with a name less than or equal to 'garden'

```
SQL>  SELECT name
     FROM items
     WHERE name <= 'garden';
```

Find the list of vendor representative first names that begin with 's'

```
SQL>  SELECT repfname
       FROM vendors
      WHERE repfname LIKE 's%';
```

Find all vendor names containing an '_'.

```
SQL>  SELECT companyname
      FROM vendors
       WHERE companyname LIKE '%#_%' ESCAPE '#';
```

- ❖ Note: Here _ is special wildcard character. To escape it # (or any character) is used.

```
SQL>  SELECT companyname
      FROM vendors
      WHERE companyname LIKE '%\_%' ESCAPE '\';
```

- ❖ Note: To escape ' we need to use one more ' before it

```
SQL> SELECT companyname
     FROM vendors
     WHERE companyname LIKE '%'''  ;
```

Using Logical operators-AND, OR, NOT

Find the name of all of the food items other than salads.

```
SQL> SELECT name
     FROM items
     WHERE NOT name LIKE '%Salad';
```

Find all of the ingredients from the fruit food group with an inventory greater than 100

```
SQL> SELECT ingredientid,name
     FROM ingredients
     WHERE foodgroup = 'Fruit' AND inventory >100;
```

Find the food items that have a name beginning with either F or S that cost less than $3.50

```
SQL> SELECT NAME, PRICE FROM ITEMS WHERE NAME LIKE 'F%' OR NAME
     LIKE 'S%' AND PRICE < 3.50
```

❖See the result of the above query. Is it correct? Why did it happen? Modify the query to get the correct result.

NOTE: The precedence order is NOT, AND, OR from highest to lowest.

BETWEEN (inclusive)

Find the food items costing between $2.50 and $3.50.

```
SQL> SELECT *
     FROM items
     WHERE price BETWEEN 2.50 AND 3.50;
```

Selecting a set of values using IN, NOT IN

Find the ingredient ID, name, and unit of items not sold in pieces or strips.

```
SQL> SELECT ingredientid,name,unit
     FROM ingredients
     WHERE unit NOT IN ('piece','stripe');
```

IS NULL

SQL interprets NULL as unknown. So comparing any value with NULL returns unknown result. *Find the details of all vendors not referred by anyone.*

```
SQL> SELECT *
     FROM vendors
     WHERE referredby = NULL;
```

Check the output and try the following.

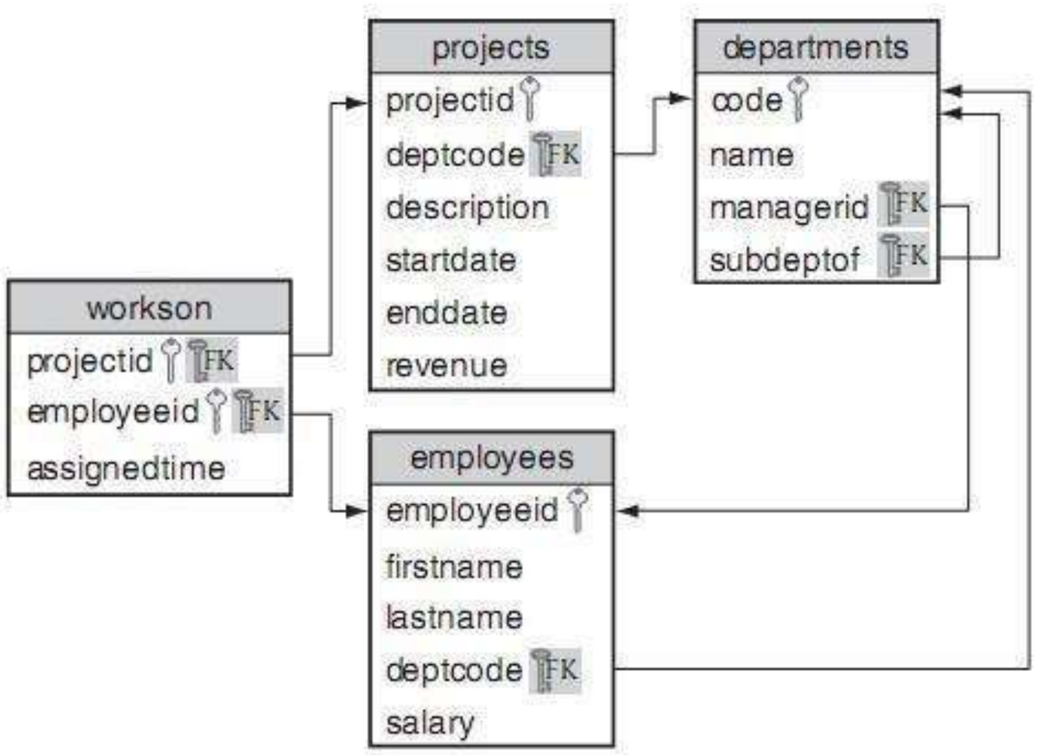
```
SQL> SELECT *
     FROM vendors
     WHERE referredby IS NULL;
```

Do you see the difference? Comparing NULL with any attribute gives an unknown result.

EXERCISES

Write a single SQL query for each of the following based on "employees" database. If you have not created the employee database in the last lab, you can create and insert data using the following employee_new.sql file. Kindly download it from website.

The schema for the employee database is as shown below.



1. List the first and last names of all employees.
2. List all attributes of the projects with revenue greater than $40,000.
3. List the department codes of the projects with revenue between $100,000 and $150,000.
4. List the project IDs for the projects that started on or before July 1, 2004.
5. List the names of the departments that are top level (i.e., not a sub department).
6. List the ID and descriptions of the projects under the departments with code ACCNT, CNSLT, or HDWRE.
7. List all of the information about employees with last names that have exactly 8 characters and end in 'ware'.
8. List the ID and last name of all employees who work for department ACTNG and make less than $30,000.
9. List the "magical" projects that have not started (indicated by a start date in the future or *NULL)* but are generating revenue.
10. List the IDs of the projects either from the ACTNG department or that are ongoing (i.e., *NULL* end date). Exclude any projects that have revenue of $50,000 or less.

Reshaping Results

❖ As we know that every query result is a relation. SQL allows us to specify how this relation should be shown in a desired way. The resulting relation columns can be renamed, duplicates can be eliminated, derived attributes can be added, the rows can be sorted by some criteria etc.

- ❖ Result table columns have the same name as attributes in the original table; however, you can change the result table column using a column alias using ‘AS’

```
SELECT COMPANYNAME AS “COMPANY”, REPFNAME AS “FIRST NAME” FROM VENDORS;
```

- ❖ DISTINCT keyword when used in SELECT clause removes the duplicates. It can be used only once in the SELECT clause. It treats NULL as a distinct value.

Find the distinct list of food groups provided by each vendor

```
SELECT DISTINCT FOODGROUP, VENDORID FROM INGREDIENTS;
```

- ❖ ALL is used to specify that the result table should include duplicates also. Since by default duplicate elimination is not done, it is unnecessary.
- ❖ Data in the original table can be somewhat raw. SQL provides the ability to create derived attributes in our result table that are derived using operations and functions over existing attributes and literals.

Find the value of your pickle inventory if you double your stock of pickles.

```
SELECT   ingredientid,   inventory*2*unitprice    as    “Inventory   Value”   from
ingredients where name=’Pickle’
```

- ❖ It is also possible to concatenate two or more columns in original table as one single column in result table.

Create a mailing label for each store

```
SELECT manager, CONVERT(VARCHAR(10),GETDATE(),105) as "As on", address+'
'+city+' '+state+'  '+zip+ 'USA' as "mail" from stores;
```

- ❖ Note that system date can be printed using the function sysdatetime().

ORDER BY: The ORDER BY keyword is used to sort the result-set by a specified column in ASCending or DESCending order.

- ❖ The ORDER BY clause takes the output from the SELECT clause and orders the query results according to the specifications within the ORDER BY clause
- ❖ The ORDER BY keyword sort the records in ascending order by default.

Find all items from most to least expensive

```
SQL>  SELECT name,price
      FROM items
      ORDER BY price ASC;
```

Find the name and inventory value of all ingredients ordered by inventory value

```
SQL>  SELECT name,inventory*unitprice AS value
      FROM ingredients
      ORDER BY value DESC;
```

CASE, COALESCE, and NULLIF: Conditional Expressions

- ❖SQL also provides basic conditional constructs to determine the correct result. CASE provides a general mechanism for specifying conditional results. SQL also provides the COALESCE and NULLIF statements to deal with NULL values, but these have equivalent CASE statements.

CASE:ValueList

- ❖ The simplest form of the CASE statement determines if a value matches any values from a list and returns the corresponding result.

```
CASE <targetexpression>

WHEN <candidateexpression> THEN <resultexpression>

WHEN <candidateexpression> THEN <resultexpression> ...

WHEN <candidateexpression> THEN <resultexpression>

[ELSE <resultexpression>]  END
```

- ❖ CASE finds the first WHEN clause where <candidateexpression> = <targetexpression> and returns the value of the corresponding <resultexpression>. If no matches are found, the value of the <resultexpression> for the ELSE clause is returned. If the ELSE clause is not specified, an implicit ELSE NULL is added to the CASE statement

Assigning goodness values for the food groups.

```
SELECT name,
CASE foodgroup
WHEN 'Vegetable' THEN 'Good'
WHEN 'Fruit' THEN 'Good'
WHEN 'Milk' THEN 'Acceptable'
WHEN 'Bread' THEN 'Acceptable'
WHEN 'Meat' THEN 'Bad'
END AS quality
FROM ingredients;
```

CASE: Conditional List

- ❖CASE also provides another powerful format where one can specify the conditional expressions in WHEN expressions.

```
CASE
WHEN Boolean_expression1 THEN expression1

[[WHEN Boolean_expression2 THEN expression2] [...]]

[ELSE expression]  END
```

- ❖To show the power of the CASE statement, let's put together an order for ingredients. The amount that we want to order is based on the current inventory. If that inventory is below a threshold, then we want to place an order to raise it to the threshold; otherwise, we want to order a percentage of our inventory. The exact amount is based on the type of the food item because some will spoil more quickly than others.

```
SELECT name,
FLOOR (CASE
            WHEN inventory < 20 THEN 20 - inventory
            WHEN foodgroup = 'Milk' THEN inventory * 0.05
            WHEN foodgroup IN ('Meat', 'Bread') THEN inventory * 0.10
            WHEN foodgroup = 'Vegetable' AND unitprice <= 0.03 THEN
            inventory * 0.10
```

```
                    WHEN foodgroup = 'Vegetable' THEN inventory * 0.03
                    WHEN foodgroup = 'Fruit' THEN inventory * 0.04
                    WHEN foodgroup IS NULL THEN inventory * 0.07 ELSE 0   END)
              AS size, vendorid
        FROM ingredients
        WHERE inventory < 1000 AND vendorid IS NOT NULL
        ORDER BY vendorid, size;
```

NULLIF

❖ NULLIF takes two values and returns NULL if they are equal or the first value if the two values are not equal. You can think of it as "NULL IF equal."

```
NULLIF(<value1>,<value2>)
```

❖ The World Health Organization (WHO) has declared that Meat is no longer a food group.

```
SELECT ingredientid, name, unit, unitprice,
NULLIF(foodgroup, 'Meat') AS foodgroup, inventory, vendorid  FROM
ingredients;
```

COALESCE

❖ COALESCE takes a list of values and returns the first non-NULL value.

```
COALESCE(<value1>, <value2>, : : :, <valueN>)
```

❖ One practical use for COALESCE is providing a substitute for NULL values in the results. For example, if we want to display all of our items with a price, we would have to handle the ones with a NULL price.

```
SELECT name, price, COALESCE(price, 0.00) AS "no nulls"  FROM items;
```

EXERCISES

Write a single SQL query for each of the following, based on employees database.

1. List all employee names as one field called name.
2. List all the department codes assigned to a project. Remove all duplicates.
3. Find the project ID and duration of each project.
4. Find the project ID and duration of each project. If the project has not finished, report its execution time as of now. [Hint: Getdate() gives current date]
5. For each completed project, find the project ID and average revenue per day.
6. Find the years a project started. Remove duplicates.
7. Find the IDs of employees assigned to a project that is more than 20 hours per week. Write three queries using 20, 40, and 60 hour work weeks.
8. For each employee assigned to a task, output the employee ID with the following:
   - 'part time' if assigned time is < 0.33
   - 'split time' if assigned time is >= 0.33 and < 0.67
   - 'full time' if assigned time is >= 0.67
9. We need to create a list of abbreviated project names. Each abbreviated name concatenates the first three characters of the project description, a hyphen, and the department code. All characters must be uppercase (e.g., EMP-ADMIN).

10. For each project, list the ID and year the project started. Order the results in ascending order by year.
11. If every employee is given a 5% raise, find the last name and new salary of the employees who will make more than $50,000.
12. For all the employees in the HDWRE department, list their ID, first name, last name, and salary after a 10% raise. The salary column in the result should be named Next Year.
13. Create a neatly formatted directory of all employees, including their department code and name. The list should be sorted first by department code, then by last name, then by first name.