LECTURE 4

Unit 2

WHAT IS SQL?

- **SQL – Structured Query Language**
- Developed by IBM
- **It is a common database language which has statements for both data definitions, query, and update**
- It is both a DDL (**D**ata **D**efinition **L**anguage) and DML (**D**ata **M**anipulation **L**anguage)
- SQL uses the terms table, row, and column for the formal relational model terms *relation*, *tuple*, and *attribute*, respectively

PROS AND CONS

- Pros
  - Very flexible
  - Universal (Oracle, Access, etc)
  - Few Commands to Learn
- Cons
  - Requires Detailed Knowledge of the Structure of the Database
  - Can Provide Misleading Results

CHARACTERISTICS OF SQL

- SQL allows the user to create, update, delete, and retrieve data from a database.
- SQL is very simple and easy to learn.
- SQL works with database programs like DB2, Oracle, MS Access, MS SQL Sever etc.

GENERAL DATA TYPES IN SQL

- CHARACTER (n) – character string of fixed length n
- VARCHAR (n) – character string of variable length, maximum length is n
- BOOLEAN – stores TRUE or FALSE values
- INTEGER and SMALLINT – integer number
- FLOAT and REAL – floating point number
- BIT (n) – array of n bits
- DATE – stores year, month and day values
- TIME – stores hour, minute and second values
- TIMESTAMP – stores year, month, day, hour, minute and second values

TYPES OF SQL COMMANDS

- SQL commands are instructions used to communicate with the database to perform specific task that work with data.

- Two common SQL commands
  - DDL
  - DML

DATA DEFINITION LANGUAGE (DDL)

- These SQL commands are used for creating, modifying, and dropping the structure of database objects.

- The most basic items of DDL are:
  - **CREATE** creates an object (a table, for example) in the database
  - **DROP** deletes an object in the database
  - **ALTER** modifies the structure an existing object in various ways—for example, adding a column to an existing table.

DATA MANIPULATION LANGUAGE (DML)

- These SQL commands are used for storing, retrieving, modifying, and deleting data.

- These commands are
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

CREATE SCHEMA

- It defines a database schema

- Syntax

  CREATE SCHEMA schema-name AUTHORIZATION user-name

- Example

  CREATE SCHEMA LIBRARY;

  CREATE SCHEMA LIBRARY AUTHORIZATION James;

CREATE TABLE

- Create a table in the database

- Syntax

  CREATE TABLE table-name (col-name1 data-type, co-name2 data-type,…);

- Example

  CREATE TABLE Student (Name VARCHAR(40), RollNo INTEGER , Class INTEGER);

SQL CONSTRAINTS

- Used to limit the type of data that can go in a table
- Important ones
  - PRIMARY KEY
  - NOT NULL
  - UNIQUE
  - DEFAULT
  - FOREIGN KEY

- PRIMARY KEY
  - This constraint uniquely identifies each record (row) in a table

CREATE TABLE STUDENT (Name VARCHAR(40), RollNo INTEGER **PRIMARY KEY**, ClassNo INTEGER);

- FOREIGN KEY
  - It points to a primary key in another table

CREATE TABLE STUDENT (Name VARCHAR(40), RollNo INTEGER, ClassNo INTEGER **FOREIGN KEY REFERENCES CLASS (ClassNo)**);

- NOT NULL
  - Enforces a column to not accept NULL values

CREATE TABLE STUDENT (Name VARCHAR(40) **NOT NULL**, RollNo INTEGER, ClassNo INTEGER);

- UNIQUE
  - Similar to PRIMARY KEY
  - Uniquely identifies each record in a table

```
CREATE TABLE STUDENT (Name VARCHAR(40),
RollNo INTEGER UNIQUE, ClassNo INTEGER);
```

- DEFAULT
  - Used to insert a default value in a column

CREATE TABLE STUDENT (Name VARCHAR(40) **DEFAULT 'James'**, RollNo INTEGER, ClassNo INTEGER);

DROP TABLE

- Remove / delete a table from a database along with its definition

DROP TABLE EMPLOYEE;

ALTER TABLE

- Add, delete or modify columns in an existing table

- To add a column

  ALTER TABLE STUDENT ADD Address VARCHAR(40);

- To delete a column

  ALTER TABLE STUDENT DROP COLUMN Address;

- To change a data type of a column

```
ALTER TABLE STUDENT ALTER COLUMN RollNo
CHAR(10);
```

INSERT INTO

- Insert a new row in a table

- Two forms
  - Do not specify column names

    INSERT INTO STUDENT VALUES (‘John’, 12, 4);

  - Specify column names and values

    INSERT INTO STUDENT (Name, RollNo, ClassNo)
    VALUES (‘John’, 12, 4);

UPDATE

- Update existing records in a table

- Syntax

```
UPDATE table-name
SET col1=value, col2=value,…
WHERE some-col=some-value;
```

- Example

```
UPDATE STUDENT
SET Name='Mary'
WHERE RollNo=3;
```

DELETE

- Delete rows in a table

- Syntax

  DELETE FROM table-name

  WHERE some-col=some-val;

- Example

  DELETE FROM STUDENT

  WHERE ClassNo=4;

SELECT

- Used for retrieving rows from database

- General syntax

  SELECT column-name-list
  FROM table-name-list
  WHERE condition;

- SELECT – attributes to extract
- FROM – tables from where to extract
- WHERE – optional, specifies the condition(s)

- Retrieve names and roll numbers of students

```
SELECT Name, RollNo
FROM STUDENT;
```

- Retrieve all details of students

```
SELECT *
FROM STUDENT;
```

- Retrieve roll numbers of all students belonging to class 6.

```
SELECT RollNo
FROM STUDENT
WHERE ClassNo=6;
```

USING DISTINCT

- Select only distinct values from a table

```
SELECT DISTINCT ClassNo
FROM STUDENT;
```

  - This statement will list only distinct values of Class Number, even though multiple occurrences may exist

ALIAS

- Alias names can also be given to tables

```
SELECT S.Name, S.RollNo
FROM STUDENT AS S;
```

SET OPERATIONS

- There is a union operation (**UNION**), and in *some versions* of SQL there are set difference (**MINUS**) and intersection (**INTERSECT**) operations

- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*

- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

- Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
(SELECT  PNAME
FROM   PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN  AND
                    LNAME='Smith')
UNION
(SELECT  PNAME
 FROM  PROJECT, WORKS_ON, EMPLOYEE
 WHERE PNUMBER=PNO AND ESSN=SSN AND
                    LNAME='Smith');
```

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*

Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT FNAME, LNAME, ADDRESS
FROM   EMPLOYEE
WHERE DNO IN
(SELECT  DNUMBER
FROM   DEPARTMENT
WHERE DNAME='Research' );
```

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*

Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT          E.FNAME, E.LNAME
FROM            EMPLOYEE AS E
WHERE  E.SSN IN (SELECT ESSN
                 FROM    DEPENDENT
                 WHERE  ESSN=E.SSN AND
                  E.FNAME=DEPENDENT_NAME);
```

- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query.

```
SELECT    E.FNAME, E.LNAME
FROM      EMPLOYEE E, DEPENDENT D
WHERE     E.SSN=D.ESSN AND
          E.FNAME=D.DEPENDENT_NAME;
```

EXISTS

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT      FNAME, LNAME
FROM        EMPLOYEE
WHERE       EXISTS
            (SELECT     *
            FROM        DEPENDENT
            WHERE       SSN=ESSN AND
            FNAME=DEPENDENT_NAME);
```

Retrieve the names of employees who have no dependents.

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE
WHERE     NOT EXISTS
          (SELECT    *
          FROM       DEPENDENT
          WHERE SSN=ESSN);
```

ORDER BY

- Used when we want data to appear in a specific order

  - Retrieve all details of students and then arrange the result on the basis of ascending order of class number

```
SELECT *
FROM STUDENT
ORDER BY ClassNo;

SELECT *
FROM STUDENT
ORDER BY ClassNo ASC;
```

- Retrieve all details of students and then arrange the result on the basis of descending order of class number

```
SELECT *
FROM STUDENT
ORDER BY ClassNo DESC;
```

COMMON OPERATORS

=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
AND	Display if both conditions are true
OR	Display if either conditions are true

- Retrieve all names of students from class 4 to class 6

```
SELECT Name
FROM STUDENT
WHERE ClassNo BETWEEN 4 AND 6;
```

JOIN

- Used to query data from two or more tables

Retrieve the student names, roll numbers and class who belong to a class whose class teacher is 'Mary'

```
SELECT S.Name, S.RollNo, C.ClassNo
FROM STUDENT AS S, CLASS AS C
WHERE S.ClassNo = C.ClassNo AND C.Teacher =
'Mary';
```

SAMPLE TABLE

EmployeeStatisticsTable

EmployeeIDNo	Salary	Benefits	Position
010	75000	15000	Manager
105	65000	15000	Manager
152	60000	15000	Manager
215	60000	12500	Manager
244	50000	12000	Staff
300	45000	10000	Staff
335	40000	10000	Staff
400	32000	7500	Entry-Level
441	28000	7500	Entry-Level

- *Retrieve the employee numbers of those employees who have salary more than Rs. 50,000*

```
SELECT EmployeeIDNo
FROM EMPLOYEESTATISTICSTABLE
WHERE Salary > 50000;
```

- Try the queries below
  - Retrieve employee numbers who has position as ‘Manager’
  - Retrieve employee numbers and benefits who has position as ‘Manager’ and whose salary is above Rs. 50,000

TAKE THESE THREE TABLES AS SHOWN

AntiqueOwners

OwnerID	OwnerLastName	OwnerFirstName
01	Jones	Bill
02	Smith	Bob
15	Lawson	Patricia
21	Akins	Jane
50	Fowler	Sam

Orders

OwnerID	ItemDesired
02	Table
02	Desk
21	Chair
15	Mirror

- Here the SellerID and BuyerID also means they are OwnerID

Antiques

SellerID	BuyerID	Item
01	50	Bed
02	15	Table
15	02	Chair
21	50	Mirror
50	01	Desk
01	21	Cabinet
02	21	Coffee Table
15	50	Chair
01	15	Jewelry Box
02	21	Pottery
21	02	Bookcase
50	01	Plant Stand

QUERIES

- List all the details of all the Antique owners

```
SELECT *
FROM ANTIQUEOWNERS;
```

- Try the queries
  - List all details of all the Orders
  - List all details of all Antiques

- Retrieve all items ordered by the owner whose name is ‘Bob Smith’

```
SELECT O.ItemDesired
FROM ORDERS AS O, ANTIQUEOWNERS AS A
WHERE A.OwnerFirstName = ‘Bob’ AND
A.OwnerLastName = “Smith’ AND A.OwnerID
=       O.OwnerID;
```

- Retrieve the names of the buyers who bought the item ‘Table’

```
SELECT A.OwnerFirstName, A.OwnerLastname
FROM ANTIQUEOWNERS AS A, ANTIQUES AS Q
WHERE Q.Item=‘Table’ AND Q.BuyerID=A.OwnerID;
```

- Try these queries
  - Retrieve the names of sellers who sell ‘Mirror’ or ‘Desk’
  - Retrieve the seller IDs of who sell items to buyer ID 50
  - Retrieve the seller names who sell items to ‘Bill Jones’
  - Retrieve the buyer names who buy item from ‘Jane Akins’

AGGREGATE FUNCTIONS

- Some of the common functions are
  - SUM() : total of all the rows satisfying a condition (s) of the given column, where the column is numeric
  - MIN() : gives the smallest value in the given column
  - MAX() : gives the highest value in the given column
  - COUNT(*) : gives the number of rows satisfying the condition
  - AVG() : gives the average of the given column

SAMPLE TABLE

EmployeeStatisticsTable

EmployeeIDNo	Salary	Benefits	Position
010	75000	15000	Manager
105	65000	15000	Manager
152	60000	15000	Manager
215	60000	12500	Manager
244	50000	12000	Staff
300	45000	10000	Staff
335	40000	10000	Staff
400	32000	7500	Entry-Level
441	28000	7500	Entry-Level

- Retrieve the total of all salaries of employees

```
SELECT SUM (Salary)
FROM EMPLOYEESTATISTICSTABLE;
```

- Retrieve the average salary of all employees

```
SELECT AVG (Salary)
FROM EMPLOYEESTATISTICSTABLE;
```

- Retrieve the maximum and the minimum benefits of all employees

```
SELECT MIN (Benefits), MAX (Benefits)
FROM EMPLOYEESTATISTICSTABLE;
```

- List the total number of employees

```
SELECT COUNT (*)
FROM EMPLOYEESTATISTICSTABLE;
```

- List the total number of employees who has ‘Staff’ position

```
SELECT COUNT (*)
FROM EMPLOYEESTATISTICSTABLE
WHERE Position = ‘Staff’;
```

- List the highest salary of the employee who has position ‘Manager’

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT FNAME, LNAME, ADDRESS
     FROM   EMPLOYEE
     WHERE DNO IN
     (SELECT  DNUMBER
     FROM   DEPARTMENT
     WHERE DNAME='Research' );
```

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT       E.FNAME, E.LNAME
     FROM         EMPLOYEE AS E
     WHERE E.SSN IN (SELECT       ESSN
                     FROM   DEPENDENT
                     WHERE ESSN=E.SSN AND
                      E.FNAME=DEPENDENT_NAME);
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not

THE EXISTS FUNCTION (CONT.)

- Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12:    SELECT    FNAME, LNAME
        FROM EMPLOYEE
        WHERE     EXISTS
        (SELECT   *
        FROM DEPENDENT
        WHERE     SSN=ESSN AND
        FNAME=DEPENDENT_NAME);
```

THE EXISTS FUNCTION (CONT.)

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:     SELECT      FNAME, LNAME
        FROM EMPLOYEE
        WHERE NOT EXISTS
        (SELECT     *
        FROM  DEPENDENT
        WHERE SSN=ESSN);
```

  - In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13:    SELECT    DISTINCT ESSN
        FROM WORKS_ON
        WHERE     PNO IN  (1, 2, 3);
```

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:SELECT      FNAME, LNAME
    FROM        EMPLOYEE
    WHERE       SUPERSSN  IS  NULL;
```

- Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING (CONT.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20: SELECT       DNO, COUNT (*), AVG (SALARY)
     FROM   EMPLOYEE
     GROUP BY     DNO;
```

GROUPING (CONT.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21: SELECT PNUMBER, PNAME, COUNT (*)
     FROM   PROJECT, WORKS_ON
     WHERE PNUMBER=PNO
     GROUP BY PNUMBER, PNAME;
```

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE (CONT.)

- Query 22: For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:    SELECT PNUMBER, PNAME, COUNT (*)
        FROM   PROJECT, WORKS_ON
        WHERE PNUMBER=PNO
        GROUP BY PNUMBER, PNAME
        HAVING COUNT (*) > 2;
```

KINDLY GO THROUGH THESE TOPICS

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings

- Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character

SUBSTRING COMPARISON (CONT.)

- Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX'.

```
Q25:    SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     ADDRESS LIKE
                        '%Houston,TX%';
```

SUBSTRING COMPARISON (CONT.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_____5_', with each underscore as a place holder for a single arbitrary character.

```
Q26:SELECT      FNAME, LNAME
            FROM  EMPLOYEE
            WHERE       BDATE LIKE
      '_____5_';
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible; hence, in SQL, character string attribute values are not atomic

ARITHMETIC OPERATIONS (

- The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27: SELECT       FNAME, LNAME, 1.1*SALARY
                  FROM   EMPLOYEE, WORKS_ON,
PROJECT
        WHERE SSN=ESSN AND PNO=PNUMBER AND
                  PNAME='ProductX';
```

SUMMARY OF SQL QUERIES

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

SELECT <attribute list>
FROM <table list>
[WHERE <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING <group condition>]
[ORDER BY <attribute list>]

SUMMARY OF SQL QUERIES (CONT.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause