MODULE 2

FEATURES OF SQL:

- IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s.
- The SQL language has several parts:
  - **Data-definition language (DDL)**: The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas

The SQL language has several parts:

  - **Data-manipulation language (DML):** The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in thedatabase.

- **Integrity**: The SQL DDL includes commands for specifying integrity constraints that the data stored in the database mustsatisfy.

- The SQL language has several parts:
  - **View definition:** The SQL DDL includes commands fordefining views.
  - **Transaction control:** SQL includes commands for specifyingthe beginning and end points of transactions.
- The SQL language has several parts:
  - **Embedded SQL and dynamic SQL:** Embedded and dynamicSQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.
  - **Authorization:** The SQL DDL includes commands for specifying access rights to relations and views.

Data Types

- **char(n)**: A fixed-length character string with user-specified length n.
- **varchar(n)**: A variable-length character string with user-specifiedmaximum length n. (**character varying**)
- **int**: An integer (a finite subset of the integers that is machinedependent).
- **numeric(p, d)**:
  - A fixed-point number with user-specified precision.
  - The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point.
  - Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 nor 0.32 can be stored exactly in a field of this type.

- **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- **float(n)**: A floating-point number with precision of at least n digits.
- Each type may include a special value called the **null value**.
- A **null value** indicates an absent value that may exist but beunknown or that may not exist at all.

Integrity Constraints

- **primary key (Aj_1 , Aj_2 ,..., Aj_m):** The primary-key specification says that attributes Aj_1 , Aj_2 ,..., Aj_m form the primary key for the relation.
- **foreign key (Ak_1, Ak_2,..., Ak_m) references s**: The foreign key specification says that the values of attributes (Ak_1, Ak_2,..., Ak_m) for any tuple in the relation must correspond to values of the

primary key attributes of some tuple in relation s.

- **not null:** The not null constraint on an attribute specifies that the null value is not allowed for that attribute; in other words, the constraint excludes the null value from the domain of that attribute.
- **unique (Aj_1 , Aj_2 ,..., Aj_m):** The unique specification says that attributes Aj_1 , Aj_2 ,..., Aj_m form a superkey; that is, no two tuples in the relation can be equal on all the listed attributes

.

- **Check clause:** The clause check(P) specifies a predicate P thatmust be satisfied by every tuple in a relation. A common use ofthe check clause is to ensure that attribute values satisfy specified conditions, in effect creating a powerful type system.

Create Statement

The general form of the create table command is:

- r is the name of the relation
- each Ai is the name of an attribute inthe schema of relation r
- Di is the domain of attribute Ai;

```
create table r(A1 D1,
        A2 D2,
        . . . ,
        An Dn,
        ⟨integrity-constraint1⟩,
        ...,
        ⟨integrity-constraintk⟩
        );
```

DEPARTMENTS(department_id,department_name)

```
CREATE TABLE departments( department_id int NOT
        NULL, department_name  char(50)  NOT  NULL,
        CONSTRAINT  departments_pk  PRIMARY  KEY  (department_id)
);
```

```
CREATE TABLE employees ( employee_id
        int NOT NULL,
        employee_name char(50) NOT NULL, department_id int,age int,CONSTRAINT
        employees_pk PRIMARY KEY (employee_id), CONSTRAINT fk_departments FOREIGN
        KEY (department_id)
         REFERENCES departments(department_id) ,
        CONSTRAINT age_chk CHECK (age > 0 and age < 120)
        );
```

Drop Statement

- To remove a relation from an SQL database, we use the drop tablecommand.
- The drop table command deletes all information about the droppedrelation from the database.
- The command "**DROP TABLE TABLE_NAME**"
- **DROP TABLE** employees
- After r is dropped, no tuples can be inserted into r unless it is re-createdwith the create table command

Alter Statement

- **alter table** command to add attributes to an existing relation.
- All tuples in the relation are assigned null as the value for the newattribute.
- Command: "**alter table r add A D;**"
- We can drop attributes from a relation by the command "**alter table rdrop A;**"

- To modify a column in an existing table, the SQL ALTER TABLE syntaxis:
  - **ALTER TABLE** table_name **MODIFY** column_name column_type;
  - **ALTER TABLE** supplier **MODIFY** (supplier_name char(100) **NOTNULL**, city char(75));

- To rename a column in an existing table, the SQL ALTER TABLE syntaxis:
  - **ALTER TABLE** table_name **RENAME COLUMN** old_name **TO** new_name;
  - **ALTER TABLE** supplier **RENAME COLUMN** supplier_name **TO** sname;
- To rename a table, the SQL ALTER TABLE syntax is:
  - **ALTER TABLE** table_name **RENAME** TO new_table_name;
  - **ALTER TABLE** supplier **RENAME** TO vendor;

- To add and drop constraints to a table, the SQL ALTER TABLE syntax is:
  - **ALTER TABLE** table_name **ADD CONSTRAINT** constraint_nameconstraints ;
  - **ALTER TABLE** table_name **DROP CONSTRAINT** constraint_name;
- The syntax for the SELECT statement in SQL is:

SELECT expressions**FROM**

tables [**WHERE** conditions]

[**GROUP BY** attributes]

[**HAVING** conditions]

[**ORDER BY** expression [**ASC** | **DESC**]];

- **Expressions**: The columns or calculations that you wish to retrieve. Use ***** if you wish to select all columns.
- **Tables**: The tables that you wish to retrieve records from. There must beat least one table listed in the FROM clause.
- **WHERE** conditions: Optional. The conditions that must be met for the records to be selected. If no conditions are provided, then all records willbe selected.
- **ORDER BY** expression: Optional. The expression used to sort the records in the result set. If more than one expression is provided, the values should be comma separated.

- The **select clause** is used to list the attributes desired in the result of aquery.
- The **from clause** is a list of the relations to be accessed in the evaluationof the query.
- • The **where clause** is a predicate involving attributes of the relation inthe from clause.
- Operational order:
    - **FROM**
    - **WHERE**
    - **GROUP BY**
    - **HAVING**
    - **SELECT**

- **ORDER BY**

- **Find the names of all instructors.**
    - select name from instructor;
- **Find the department names of all instructors**
    - select dept_name from instructor;
    - select **distinct** dept_name from instructor;
    - select **all** dept_name from instructor;

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

- **select** ID, name, dept_name, salary * 1.1 **from** instructor;
- This shows what would result if we gave a 10% raise to eachInstructor
- It does not result in any change to the instructor relation.

Find the names of all instructors in the Computer Science department who havesalary greater than $70,000.Instructor(name,dept_name,salary).

select name

from instructor

where dept name = 'Computer Science'and salary >

70000;

- SQL allows the use of the logical connectives **and**, **or**, and **not** in thewhere clause.
- The operands of the logical connectives can be expressions involvingthe comparison operators **<**, **<=**, **>**, **>=**, **=**, and **<>**.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

The *instructor* relation.

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

The *department* relation.

- Retrieve the names of all instructors, along with their department namesand department building name
- Retrieve the names of all instructors, along with their department names and departmentbuilding name

select name, instructor.dept name, building from instructor, department

where instructor.dept name= department.dept name;

name	*dept_name*	*building*
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

Rename Operator

The RENAME operation is used to rename the output of a relation.

```
select
name as instructor_name,
Instructor.dept_name  instructor_department_name,building
from instructor, department
where instructor.dept name= department.dept name;
```

```
select name,I.dept_name, building from
instructor as I, department Dwhere I.dept name=
D.dept name;
```

STRINGS:

SQL specifies strings by enclosing them in single quotes.

- SQL also permits a variety of functions on character strings, suchas
    - Concatenating - using “||”
- SQL specifies strings by enclosing them in single quotes.
- SQL also permits a variety of functions on character strings, suchas
    - Concatenating - using “||”
    - extracting substrings – SUBSTRING(string, start, length)

- SQL specifies strings by enclosing them in single quotes.
- SQL also permits a variety of functions on character strings, suchas
    - Concatenating - using “||”
    - extracting substrings – SUBSTRING(string, start, length)
    - finding the length of strings – LENGTH(string)
- SQL also permits a variety of functions on character strings, suchas
    - converting strings to uppercase (upper(s) where s is astring) and lowercase (lower(s)),

SQL also permits a variety of functions on character strings, suchas

- converting strings to uppercase (upper(s) where s is astring) and lowercase (lower(s)),

Pattern matching can be performed on strings using the operator:

like.

- We describe patterns by using two special characters:
    - Percent (%): The % character matches any substring.
    - Underscore ( ): The character matches any character.
- Patterns are case sensitive

Find name of employees start with ‘San’

SELECT name

FROM employees

WHERE name **like** ‘San%’

Find name of employees end with ‘th’

SELECT name

FROM employees

```
    WHERE name like '%th'
```

Find name of employees start with 'San' and end with 'th'

```
    SELECT name

    FROM employees

    WHERE name like 'San%th'
```

Find name of employees having third letter n.

```
    SELECT name

    FROM employees

    WHERE name like '__n%'
```

Find name of employees having third last letter e.

```
    SELECT name
```

FROM employees

WHERE name like '%e_ _'

Asterisk Symbol

- The asterisk symbol " * " can be used in the select clause to denote"all attributes."
- select instructor.*

  from instructor, teaches

  where instructor.ID= teaches.ID;

- A select clause of the form **select *** indicates that all attributes ofthe result relation of the from clause are selected.

ORDER BY

- By default, the order by clause lists items in ascending order.

- To specify the sort order, we may specify desc for descendingorder or asc for ascending order
- SELECT name FROM

  employees

  ORDER BY name desc

BETWEEN

- SQL includes a between comparison operator to simplify where clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.
- **SELECT** name

  FROM instructor

  WHERE salary **between** 90000 **and** 100000;

COMMIT AND ROLLBACK

Commit work commits the current transaction; that is, it makesthe updates performed by the transaction become permanent in the database.

- After the transaction is committed, a new transaction is automatically started.

Rollback work causes the current transaction to be rolled back;that is, it undoes all the updates performed by the SQL statements in the Transaction.

- Thus, the database state is restored to what it was before thefirst

statement of the transaction was executed.

VIEWS:

- Consider a clerk who needs to know an instructor’s ID, name, and department name, but does not have authorization to see the instructor’s salary amount.
- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.

- A view can either have specific rows based on certain condition or all the rows of a table.

VIEWS

- A view can be created using the **CREATE VIEW** statement.

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

- create view faculty as

  select ID, name, dept name from instructor

A good database should contain views due to the given reasons

1. **Restricting data access :** Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity :** A view can hide the complexity that exists in a multiple table join.

3. **Simplify commands for the user:** Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries** : Views can be used to store complex queries.

3. **Rename Columns:** Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to to hide the names of the columns of the base tables.
4. **Multiple view facility:** Different views can be created on the same table for different users.

TRIGGER

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.
- A trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

create trigger [trigger_name]

[before | after]

{insert | update | delete}on

[table_name]

[for each row]

[trigger_body]

- BEFORE triggers run the trigger action before the triggering statement is run.
- AFTER triggers run the trigger action after the triggering statement is run.
- [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

- Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL.
- Triggers are also useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

EMBEDDED AND DYNAMIC SQL

- Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.
- **Embedded** SQL are SQL statements in an application that do not change at runtime and, therefore, can be hard-coded into the application.
- **Dynamic** SQL is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own queries.

- In embedded SQL, SQL statements are compiled at compile time.
- In Dynamic SQL, SQL statements are compiled at runtime.