Assignment 2

Database Design - SQL

Tanishq Prasad
21CS30054

1. Relational Schema
   a. Entities
      i. student (name, roll, dept)
      ii. role (rid, rname, description, *student.roll*)
      iii. event (eid, date, ename, type)
      iv. participant (pid, name, *college.name*)
      v. college (name, location)
      vi. volunteer (roll)
   b. Relationships
      i. manage (*student.roll*, *eid*)
      ii. event_has_volunteer (*volunteer.roll*, *eid*)
      iii. event_has_participant (*pid*, *eid*)

The attributes that are underlined are the primary keys.
The attributes that are *italicised* are the foreign keys.

Table Definitions, Attribute Definitions and Attribute Datatypes:-

-
  ```
  student (
    name varchar(255),
    roll varchar(50) PRIMARY KEY,
    dept varchar(100)
  );
  ```

-
  ```
  role (
    rid varchar(50) PRIMARY KEY,
    rname varchar(255) NOT NULL,
    description varchar(1024),
    student_roll varchar(50) NOT NULL,
    FOREIGN KEY (student_roll) REFERENCES student(roll)
  );
  ```

-
  ```
  event (
    eid varchar(50) PRIMARY KEY,
    date date NOT NULL,
    ename varchar(255) NOT NULL,
    type varchar(100)
  );
  ```

-
  ```
  participant (
    pid varchar(50) PRIMARY KEY,
    name varchar(255) NOT NULL,
    college_name varchar(511) NOT NULL,
    FOREIGN KEY (college_name) REFERENCES college(name)
  );
  ```

-
  ```
  college (
    name varchar(511) PRIMARY KEY,
    location varchar(1023) NOT NULL
  );
  ```

-
  ```
  volunteer (
    roll varchar(50) PRIMARY KEY
  );
  ```

-
  ```
  manage (
    student_roll varchar(50),
    eid varchar(50),
    PRIMARY KEY (student_roll, eid),
    FOREIGN KEY (student_roll) REFERENCES student(roll),
    FOREIGN KEY (eid) REFERENCES event(eid)
  );
  ```

-
  ```
  event_volunteer (
    volunteer_roll varchar(50),
    eid varchar(50),
    PRIMARY KEY (volunteer_roll, eid),
    FOREIGN KEY (volunteer_roll) REFERENCES volunteer(roll),
    FOREIGN KEY (eid) REFERENCES event(eid)
  );
  ```

-
  ```
  event_participant (
    pid varchar(50),
    eid varchar(50),
    PRIMARY KEY (pid, eid),
    FOREIGN KEY (pid) REFERENCES participant(pid),
    FOREIGN KEY (eid) REFERENCES event(eid)
  );
  ```

The primary keys are implicitly NOT NULL.

2. The SQL commands used:-

- The `CREATE TABLE` commands are used to structure and define the database with appropriate tables and relationships.
- The `INSERT INTO` commands populate these tables with actual data.

- The `SELECT` commands, often combined with `JOIN`, `WHERE`, `GROUP BY`, `ORDER BY`, `LIMIT`, and `DISTINCT`, are used to retrieve and organise data from the database according to specific requirements.
- Relationship commands like `FOREIGN KEY` ensure data integrity by defining how tables relate.

3. The records inserted:-

- student

```
  name   | roll | dept
---------+------+-------
 Alice   | 001  | CSE
 Bob     | 002  | ECE
 Charlie | 003  | MECH
  name   | roll | dept
---------+------+-------
 David   | 004  | CSE
 Eve     | 005  | CIVIL
```

- role

```
 rid |     rname      |        description          | student_roll
-----+----------------+-----------------------------+--------------
 R01 | Secretary      | Handles administrative tasks | 001
 R02 | Treasurer      | Manages finances            | 002
 R03 | President      | Leads the student body      | 003
 R04 | Vice President | Assists the President       | 004
 R05 | Member         | Active member               | 005
```

- event

```
eid |    date    |      ename       |  type
-----+------------+------------------+-----------
 E01 | 2024-02-15 | Megaevent        | Cultural
 E02 | 2024-03-20 | Techfest         | Technical
 E03 | 2024-04-25 | Sports Day       | Sports
 E04 | 2024-05-30 | Music Concert    | Cultural
 E05 | 2024-08-15 | Independence Day | National
```

- participant

```
 pid |  name  | college_name
-----+--------+--------------
 P01 | Frank  | IITB
 P02 | Grace  | MIT
 P03 | Hannah | Stanford
 P04 | Ivan   | Cambridge
 P05 | Julia  | Oxford
```

- college

```
  name    |  location
-----------+---------------
 IITB     | Mumbai
 MIT      | Massachusetts
 Stanford  | California
 Cambridge | Cambridge
 Oxford   | Oxford
```

- volunteer

```
 roll
------
 001
 002
 003
004
 005
```

- manage

```
 student_roll | eid
--------------+-----
 001          | E01
 002          | E02
 003          | E03
 004          | E04
 005          | E05
```

- event_volunteer

```
 volunteer_roll | eid
----------------+-----
 001            | E01
 002            | E02
 003            | E03
 004            | E04
 005            | E05
```

- event_participant

```
 pid | eid
-----+-----
 P01 | E01
 P02 | E02
 P03 | E03
 P04 | E04
 P05 | E05
```

4. Output of all the queries:-

- Roll number and name of all the students who are managing the “Megaevent ”
  Relational Algebra query :
  π_Roll, Name (σ_EName='Megaevent' (Student ⋈ MANAGE ⋈ Event))
  Output:
  ```
   roll | name
  ------+-------
   001  | Alice
  ```

- Roll number and name of all the students who are managing “Megevent ” as a “Secretary”.
  Relational Algebra query :
  π_Roll, Name (σ_EName='Megaevent' ∧ Rname='Secretary' (Student ⋈ MANAGE ⋈ Event ⋈ Role))
  Output:
  ```
   roll | name
  ------+-------
   001  | Alice
  ```

- Name of all the participants from the college “IITB” in “Megaevent”
  Relational Algebra query :
  π_name (σ_C.name='IITB' ∧ E.ename='Megaevent' (Participant ⋈ College ⋈ Event_Participant ⋈ Event))
  Output:
  ```
   name
  -------
   Frank
  ```

- Name of all the colleges who have at least one participant in “Megaevent”
  Relational Algebra query :
  π_C.name (σ_E.ename='Megaevent' (College ⋈ Participant ⋈ Event_Participant ⋈ Event))
  Output:
  ```
   name
  ------
   IITB
  ```

- Name of all the events which are managed by a “Secretary”
  Relational Algebra query :
  π_E.ename (σ_R.rname='Secretary' (Event ⋈ Manage ⋈ Student ⋈ Role))
  Output:
  ```
    ename
  -----------
   Megaevent
  ```

- Name of all the “CSE” department student volunteers of “Megaevent ”
  Relational Algebra query :
  π_S.name (σ_S.dept='CSE' ∧ E.ename='Megaevent' (Student ⋈ Volunteer ⋈ Event_Volunteer ⋈ Event))
  Output:
```
 name
-------
 Alice
```

- Name of all the events which have at least one volunteer from “CSE”
  Relational Algebra query :
  σ (σ_dept='CSE'(Student)) ⋈ Volunteer ⋈ Event_Volunteer ⋈ Event
  Output:
```
    ename
---------------
 Megaevent
 Music Concert
```

- Name of the college with the largest number of participants in “Megaevent”
  Relational Algebra query :
  Π_name(σ_ename='Megaevent'(Event) ⋈ Event_Participant ⋈ Participant ⋈ College) ↠ (COUNT(P.pid), DESC) ↠ LIMIT 1
  Output:
```
 name
------
 IITB
```

- Name of the college with the largest number of participants overall
  Relational Algebra query :
  Π_name(σ_(College ⋈ Participant)) ↠ (COUNT(P.pid), DESC) ↠ LIMIT 1
  Output:
```
  name
----------
 Stanford
```

- Name of the department with the largest number of volunteers in all the events which has at least one participant from “IITB”
  Relational Algebra query :
  Π_dept, COUNT(DISTINCT volunteer_roll) (σ_college_name='IITB'(student ⋈ volunteer ⋈ event_volunteer ⋈ event ⋈ event_participant ⋈ participant)) ↠ GROUP BY dept ↠ ORDER BY COUNT(DISTINCT volunteer_roll) DESC ↠ LIMIT 1
  Output:
```
 dept | volunteer_count
------+-----------------
```

Meaning of symbols:-

- `**π**` (Pi): Projection operator. It is used to select specific columns from a relation.
- `**σ**` (Sigma): Selection operator. It is used to filter rows based on a specified condition.
- `=`: Equality comparison.
- `⋈` (Join): Natural join operator. It combines two relations based on common attributes.
- ` ∧ `: Logical AND operator used to combine conditions.
- `↠`: Extended projection operator for complex operations like aggregation or sorting.
- `**COUNT**`: An aggregation function that counts the number of rows.
- `**DESC**`: Descending order sorting.
- `**LIMIT 1**`: Restricts the result to only one row.
- `**DISTINCT**`: The operator is used to select unique values.
- `**GROUP BY**`: Groups results by a specified column.
- `**ORDER BY**`: Specifies the order to return the results.