**DBMS PROJECT**

PROJECT NAME: PlsLetMeGo

PROJECT TOPIC: Leave Management System

STUDENT NAME: Nakul Krishnakumar

ROLL NO: 2023BCS0010

GROUP NO: 3

Entities and Attributes

|  |
| --- |
| **User** |
| * User\_id * Password * User\_type |

|  |
| --- |
| **Student** |
| * Student\_ID * Student\_Name * Email\_ID * Address * Guardian\_Phone\_No |

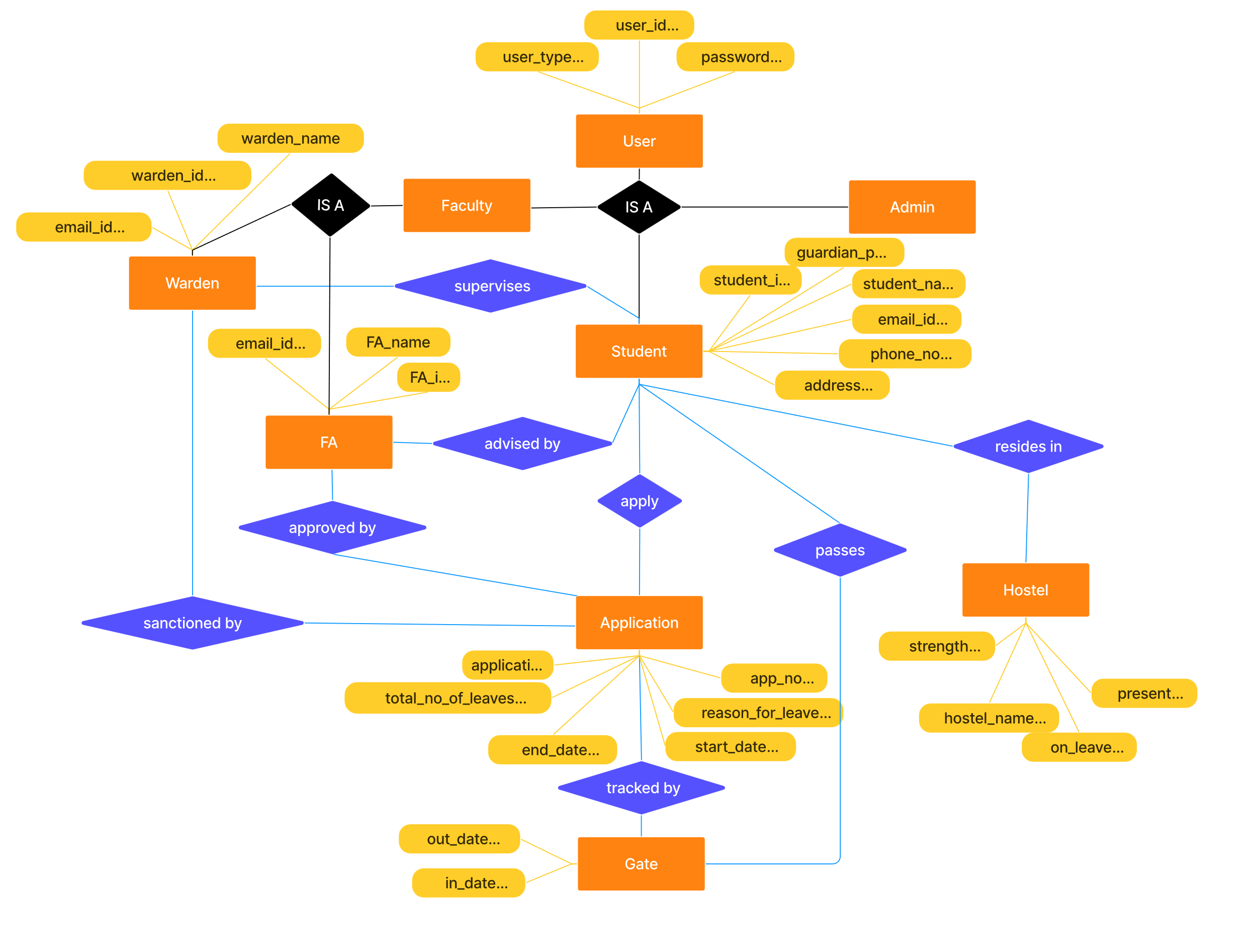
|  |
| --- |
| **Warden** |
| * Warden\_ID * Warden\_Name * Email\_ID |

|  |
| --- |
| **Faculty advisor** |
| * FA\_ID * FA\_Name * Email\_ID |

|  |
| --- |
| **Application** |
| * App\_no * Application\_status * Reason\_for\_leave * Start\_date * End\_date * Total\_no\_of\_working\_days |

|  |
| --- |
| **Hostel** |
| * Hostel\_name * Strength * On\_leave * Present |

|  |
| --- |
| **Gate** |
| * Out\_date * In\_date |

ER Diagram

**Relations and their Cardinality and Participation**

**STUDENT apply APPLICATION**

* **Cardinality (One-to-many):** Each student can submit multiple applications, but each application is linked to only one student.
* **Participation (Partial-partial):** Not every student submits an application, and not all applications are submitted.

**STUDENT advised by FA**

* **Cardinality (Many-to-one):** Multiple students may share the same faculty advisor, but each student has only one assigned advisor.
* **Participation (Total-total):** Every student must have an advisor, and each advisor is responsible for students.

**APPLICATIONS approved by FA**

* **Cardinality (Many-to-one):** Each application is approved by only one faculty advisor, but an advisor can approve multiple applications.
* **Participation (Total-partial):** All applications require approval, but not every advisor needs to approve applications.

**APPLICATIONS sanctioned by WARDEN**

* **Cardinality (Many-to-many):** Multiple wardens can sanction the same application, and a warden can sanction multiple applications.
* **Participation (Total-partial):** All applications are eligible for sanction, but not every application will require it.

**APPLICATIONS tracked by GATE**

* **Cardinality (One-to-one):** Each application is uniquely tracked by a specific gate, and each gate entry tracks a specific application.
* **Participation (Total-total):** Every application has a corresponding gate entry, and all gate entries are linked to applications.

**STUDENT passes GATE**

* **Cardinality (One-to-many):** Each student can pass through the gate multiple times, but each gate entry is linked to only one student.
* **Participation (Total-partial):** All students have the opportunity to pass through, though some may not.

**STUDENT resides in HOSTEL**

* **Cardinality (Many-to-one):** Multiple students may stay in the same hostel, but each student resides in only one hostel at a time.
* **Participation (Partial-partial):** Not all students are required to stay in a hostel, and not all hostels may be occupied by students.

**USER represent STUDENT/FACULTY/ADMIN**

* **Cardinality (One-to-one):** Each user account uniquely represents one entity (student, faculty, or admin), with no overlap.
* **Participation (Total-total):** Every user must represent exactly one role, and each role requires a corresponding user.

**WARDEN supervises STUDENT**

* **Cardinality (Many-to-many):** Each warden can supervise multiple students, and each student can be supervised by multiple wardens.
* **Participation (Total-total):** Every student must have a supervising warden, and each warden must oversee students in the hostel system.

**Relational Schema**

# **USER** ( User\_id, Password, User\_type)

# **STUDENT** ( Student\_id, Student\_name, Email\_ID, Phone\_No, Guardian\_Phone\_No, FA\_id)

# **FA**( FA\_id, FA\_Name, Email\_ID)

# **WARDEN**( Warden\_id, Warden \_Name, Email\_ID, Hostel\_Name)

# **APPLICATION**(app\_no, reason, start\_date, end\_date, no\_working\_days, application\_status, student\_id)

# **GateEntry**(entry\_id, app\_no, out\_date, in\_date)

**Hostel**( hostel\_name strength on\_leave )

**Application\_Warden**( app\_no warden\_id )

**Student\_Warden**( student\_id warden\_id )

**Functional Dependencies**

|  |  |
| --- | --- |
| **USER** | user\_id → password  user\_id → user\_type |
| **STUDENT** | student\_id → (student\_name, email\_id, phone\_no, address, guardian\_phone\_no, FA\_id, hostel\_name)  email\_id → (student\_id, student\_name, phone\_no, address, guardian\_phone\_no, FA\_id, hostel\_name)  phone\_no → (student\_id, student\_name, address, email\_id, guardian\_phone\_no, FA\_id, hostel\_name) |
| **FACULTY\_ADVISOR** | fa\_id →( fa\_name, email\_id)  email\_id → (fa\_id, fa\_name) |
| **WARDEN** | warden\_id → (wd\_name, email\_id, hostel\_name) |
| **APPLICATION** | app\_no → (reason, start\_date, end\_date, no\_working\_days, application\_status, student\_id, fa\_id)  student\_id → fa\_id |
| **GATE\_ENTRY** | entry\_id → (app\_no, out\_date, in\_date)  app\_no → (out\_date, in\_date) |
| **HOSTEL** | hostel\_name → (strength, on\_leave) |
| **APPLICATION\_WARDEN** | (app\_no, warden\_id) → (app\_no, warden\_id) |
| **STUDENT\_WARDEN** | (student\_id, warden\_id) → (student\_id, warden\_id) |

**NORMALIZATION APPLIED**

**USER :**

* Table is already in 1NF as no multivalued attribute exists.
* Both user\_type and password are fully dependent on candidate key (user\_id), hence the table is in 2NF.
* In all functional dependency, the LHS is candidate (which is subset of super key), hence table is in 3NF.

**STUDENT :**

* Table is already in 1NF as no multivalued attribute exists.
* Since each candidate key is a single attribute, there is no possibility of partial dependency. Each non-prime attribute depends on a whole candidate key.
* In each case, non-prime attributes (student\_name, address, guardian\_phone, FA\_id, hostel\_name) are directly dependent on the candidate keys and not on any other non-prime attribute.Thus, there are no transitive dependencies, and the table is in 3NF.

**FACULTY\_ADVISOR:**

* Table is already in 1NF as no multivalued attribute exists.
* Since both candidate keys (fa\_id and email\_id) are single attributes, there can’t be any partial dependency (as partial dependency only occurs with composite keys). fa\_name is fully functionally dependent on each candidate key (fa\_id and email\_id), so there are no partial dependencies. Conclusion: The table is in 2NF.
* Here, we have two FDs: fa\_id → fa\_name, email\_id and email\_id → fa\_id, fa\_name. Both fa\_id and email\_id are superkeys (candidate keys) for this table. The non-prime attribute fa\_name is directly dependent on candidate keys without any intermediary dependencies. Conclusion: The table is in 3NF.
* Both FDs (fa\_id → fa\_name, email\_id and email\_id → fa\_id, fa\_name) have superkeys on the left side (fa\_id and email\_id are candidate keys), which satisfies BCNF.

**WARDEN:**

* Table is already in 1NF as no multivalued attribute exists.
* Since there is only one candidate key (warden\_id), and all non-prime attributes are fully dependent on warden\_id, this table is in 2NF.
* In this case, there are no transitive dependencies. All non-prime attributes depend directly on the candidate key (warden\_id), so the table is in 3NF.
* Here, the only functional dependency is warden\_id → (wd\_name, email\_id, hostel\_name). Since warden\_id is a candidate key (and hence a super key), the table is also in BCNF.

**APPLICATION** :

* Table is already in 1NF as no multivalued attribute exists.
* Here, fa\_id is only dependent on student\_id and not directly on app\_no, creating a partial dependency. To bring the table into 2NF, we need to remove this partial dependency by separating fa\_id into a different table. Since student table contains fa\_id we remove it from the application table
* The table is in 3NF form as there are no transitive dependency present.

**GATE\_ENTRY:**

* Table is already in 1NF as no multivalued attribute exists.
* In this case, all non-prime attributes (app\_no, out\_date, in\_date) are fully dependent on entry\_id, the candidate key.Therefore, the GateEntry table is already in 2NF.

**HOSTEL:**

* Table is already in 1NF as no multivalued attribute exists.
* Here, strength and on\_leave are fully dependent on hostel\_name, which is the only candidate key. Therefore, the Hostel table is already in 2NF.
* In this table, there are no transitive dependencies. strength and on\_leave are directly dependent on hostel\_name, the candidate key. Therefore, the Hostel table is already in 3NF.
* Since hostel\_name is the only candidate key and all functional dependencies have hostel\_name as their determinant, the table is also in BCNF.

**APPLICATION\_WARDEN:**

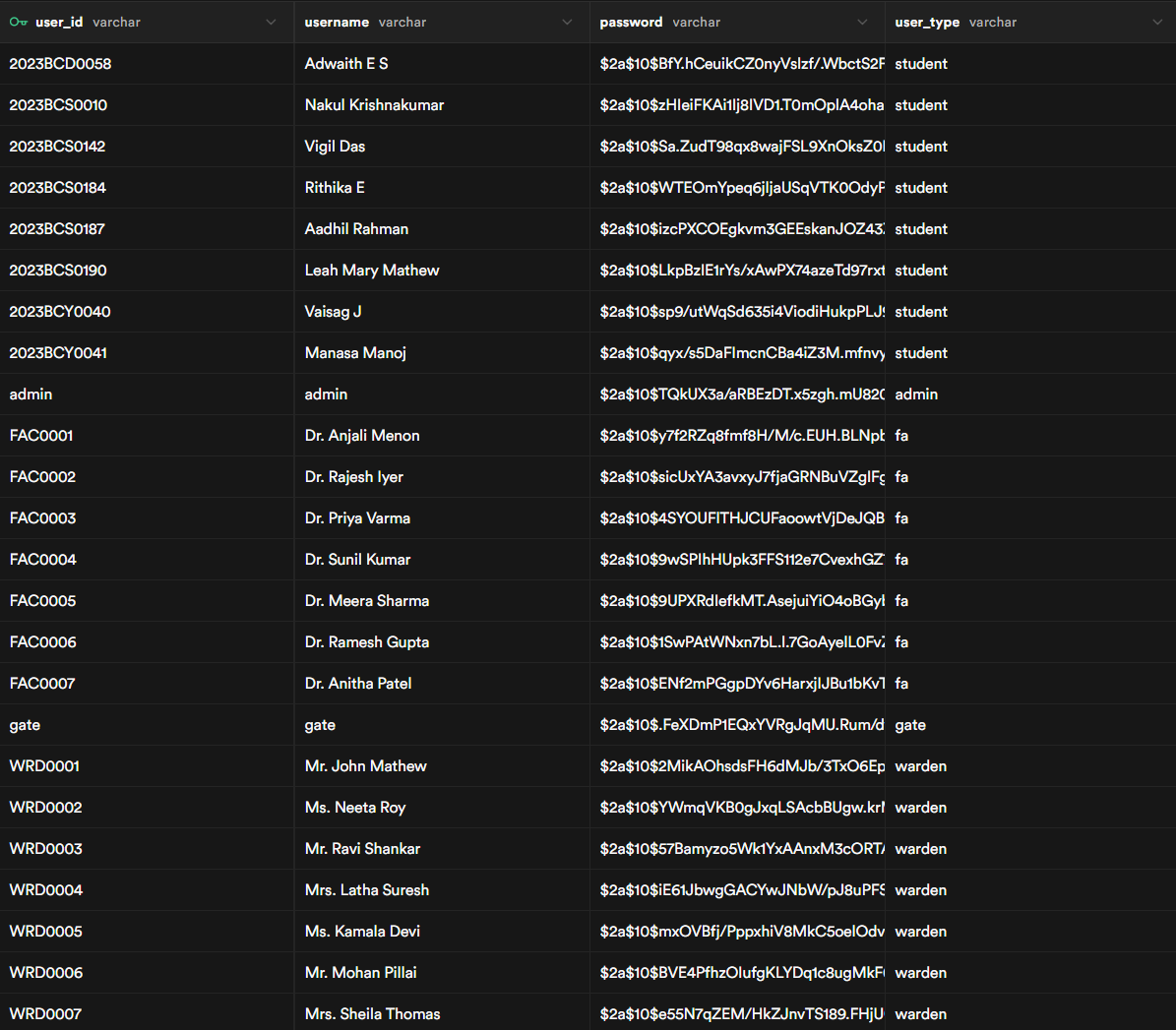
* Table is already in 1NF as no multivalued attribute exists.
* All attributes (in this case, app\_no and warden\_id as the composite key) are fully functionally dependent on the entire candidate key.
* There are no transitive dependencies, as there are no non-key attributes.
* The table is also in BCNF because: For every functional dependency, the determinant is a superkey. Here, the only functional dependency is the primary key itself, which is also a superkey.

**STUDENT\_WARDEN:**

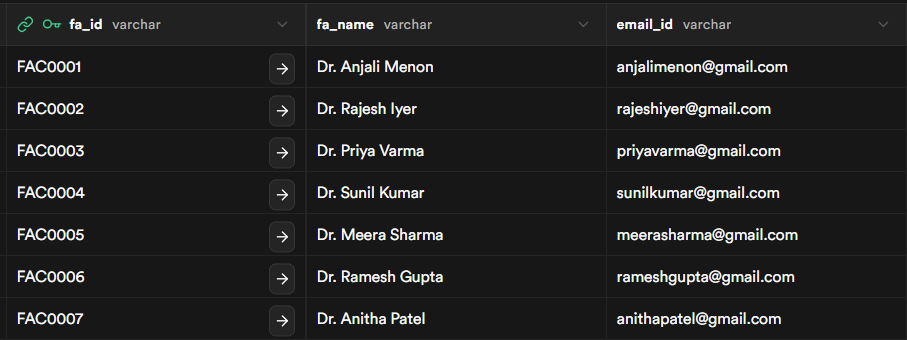
* Table is already in 1NF as no multivalued attribute exists.
* All attributes (in this case, student\_id and warden\_id as the composite key) are fully functionally dependent on the entire candidate key.
* There are no transitive dependencies, as there are no non-key attributes.
* The table is also in BCNF because: For every functional dependency, the determinant is a superkey. Here, the only functional dependency is the primary key itself, which is also a superkey.

**VALUES ENTERED INTO TABLES**

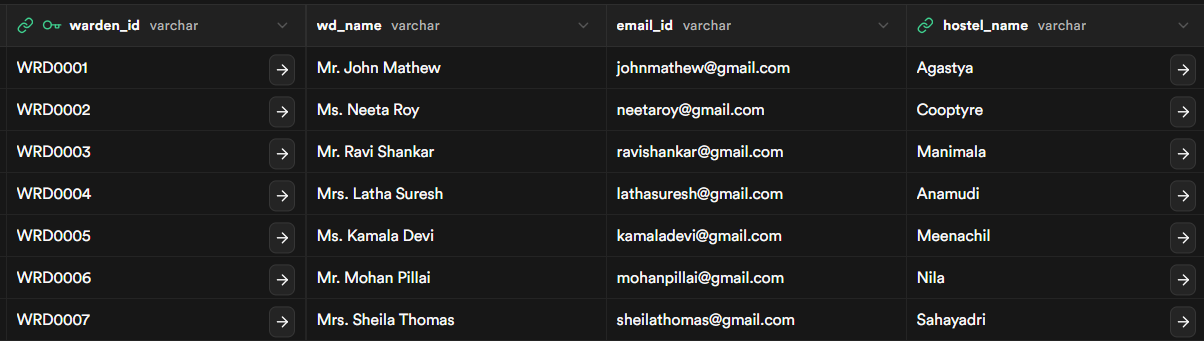
**TABLE -Users**

****

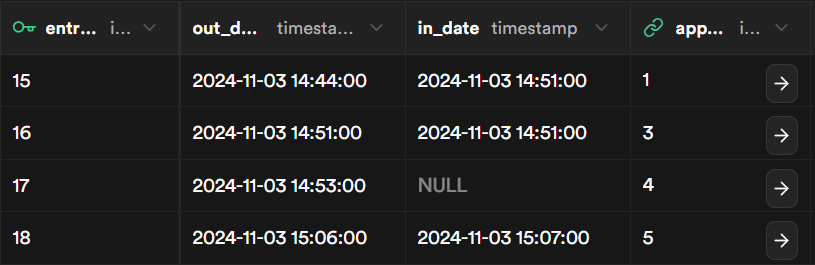
**TABLE-FacultyAdvisor**

****

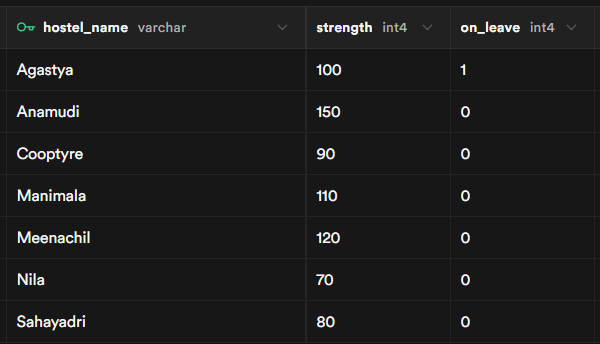
**TABLE-Warden**

****

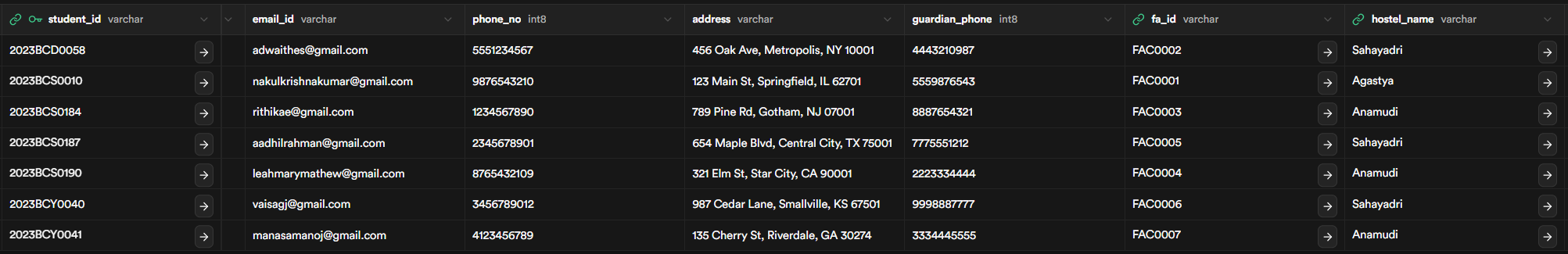
**TABLE – GateEntry**

****

**TABLE-Hostel**

****

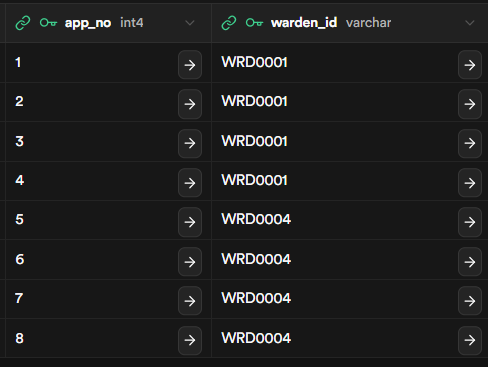
**Table-Student**

****

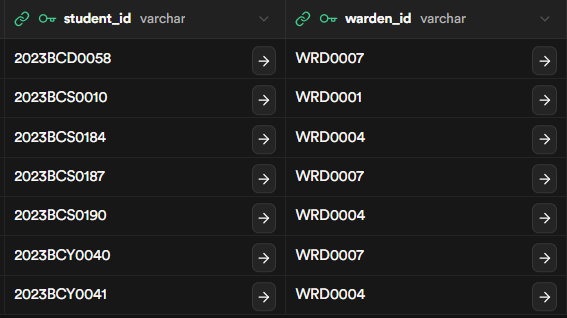
**Table – Application**

****

**Table – Application\_warden**

****

**Table – Student\_warden**

****

**TRIGGERS USED**

|  |  |
| --- | --- |
| **TRIGGER 1** | Trigger to Reset Application Number to last tuple's app\_no – 1 or set it to 0 if application table is empty  CREATE OR REPLACE FUNCTION reset\_app\_no()  RETURNS TRIGGER AS $$  DECLARE  max\_app\_no INT;  BEGIN  -- Check if the table is empty  SELECT COUNT(\*) INTO max\_app\_no FROM application;  IF max\_app\_no = 0 THEN  -- If empty, set app\_no to 1 for the new row  NEW.app\_no := 1;  ELSE  -- Otherwise, find the next available app\_no  SELECT COALESCE(MAX(app\_no), 0) + 1 INTO NEW.app\_no FROM application;  END IF;  RETURN NEW; -- Return the modified new row with updated app\_no  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER reset\_app\_no\_trigger  BEFORE INSERT ON application  FOR EACH ROW  EXECUTE FUNCTION reset\_app\_no(); |
| **TRIGGER 2** | Trigger to add entries into application\_warden table when a new application is made  CREATE OR REPLACE FUNCTION add\_application\_warden()  RETURNS TRIGGER AS $$  BEGIN  INSERT INTO application\_warden (app\_no, warden\_id)  SELECT NEW.app\_no, warden.warden\_id  FROM warden  JOIN student\_warden ON student\_warden.warden\_id = warden.warden\_id  WHERE student\_warden.student\_id = NEW.student\_id;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_add\_application\_warden  AFTER INSERT ON application  FOR EACH ROW  EXECUTE FUNCTION add\_application\_warden(); |
| **TRIGGER 3** | :Increment On\_leave attribute in hostel table when an application is checked out of gate  CREATE OR REPLACE FUNCTION increment\_on\_leave()  RETURNS TRIGGER AS $$  BEGIN  -- Check if the application status has changed to "On Leave"  IF NEW.app\_status = 'On Leave' AND OLD.app\_status IS DISTINCT FROM 'On Leave' THEN  UPDATE Hostel  SET on\_leave = on\_leave + 1  WHERE hostel\_name = (  SELECT hostel\_name FROM Student WHERE student\_id = NEW.student\_id  );  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_increment\_on\_leave  AFTER UPDATE OF app\_status ON Application  FOR EACH ROW  WHEN (NEW.app\_status = 'On Leave')  EXECUTE FUNCTION increment\_on\_leave();  DROP TRIGGER trigger\_increment\_on\_leave ON application; |
| **TRIGGER 4** | Decrement On\_leave attribute when an application is checked in at the gate  CREATE OR REPLACE FUNCTION decrement\_on\_leave()  RETURNS TRIGGER AS $$  BEGIN  -- Check if the application status has changed to "Late" or "Expired"  IF (NEW.app\_status = 'Late' OR NEW.app\_status = 'Expired')  AND OLD.app\_status IS DISTINCT FROM NEW.app\_status THEN  UPDATE Hostel  SET on\_leave = on\_leave - 1  WHERE hostel\_name = (  SELECT hostel\_name FROM Student WHERE student\_id = NEW.student\_id  );  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER trigger\_decrement\_on\_leave  AFTER UPDATE OF app\_status ON Application  FOR EACH ROW  WHEN (NEW.app\_status = 'Late' OR NEW.app\_status = 'Expired')  EXECUTE FUNCTION decrement\_on\_leave(); |
| **TRIGGER 5** | :Add entry into student\_warden table when a new student is registered (according to their hostel\_name )  CREATE OR REPLACE FUNCTION add\_student\_warden\_entry()  RETURNS TRIGGER AS $$  DECLARE  wardenID varchar(50);  BEGIN  -- Retrieve the warden\_id associated with the new student's hostel  SELECT warden\_id INTO wardenID  FROM Warden  WHERE hostel\_name = NEW.hostel\_name;  -- Insert into student\_warden if a matching warden\_id is found  IF wardenID IS NOT NULL THEN  INSERT INTO student\_warden (student\_id, warden\_id)  VALUES (NEW.student\_id, wardenID);  END IF;  RETURN NEW;  END;  $$ LANGUAGE plpgsql;  CREATE TRIGGER after\_student\_insert  AFTER INSERT ON Student  FOR EACH ROW  EXECUTE FUNCTION add\_student\_warden\_entry(); |

**QUESTIONS**

***-- 1. Find all applications that were approved for medical reasons***

SELECT a.app\_no, a.student\_id, a.reason, a.start\_date, a.end\_date

FROM APPLICATION a

WHERE a.reason LIKE '%Medical%'

AND a.app\_status = 'Approved';

***-- 2. Count the number of applications handled by each warden***

SELECT w.warden\_id, w.wd\_name, COUNT(aw.app\_no) as applications\_handled

FROM WARDEN w

LEFT JOIN Application\_Warden aw ON w.warden\_id = aw.warden\_id

GROUP BY w.warden\_id, w.wd\_name;

***-- 3. List students who have more than 2 rejected applications***

SELECT s.student\_id, s.email\_id, COUNT(\*) as rejected\_count

FROM STUDENT s

JOIN APPLICATION a ON s.student\_id = a.student\_id

WHERE a.app\_status = 'Rejected'

GROUP BY s.student\_id, s.email\_id

HAVING COUNT(\*) > 2;

***-- 4. Find the warden who has approved the most leave applications***

SELECT w.warden\_id, w.wd\_name, COUNT(\*) as approved\_count

FROM WARDEN w

JOIN Application\_Warden aw ON w.warden\_id = aw.warden\_id

JOIN APPLICATION a ON aw.app\_no = a.app\_no

WHERE a.app\_status = 'Approved'

GROUP BY w.warden\_id, w.wd\_name

ORDER BY approved\_count DESC

LIMIT 1;

***-- 5. Calculate the average duration of approved leaves for each hostel***

SELECT h.hostel\_name,

AVG(DATEDIFF(a.end\_date, a.start\_date)) as avg\_leave\_duration

FROM Hostel h

JOIN STUDENT s ON h.hostel\_name = s.hostel\_name

JOIN APPLICATION a ON s.student\_id = a.student\_id

WHERE a.app\_status = 'Approved'

GROUP BY h.hostel\_name;

***-- 6. List all hostels that have strength greater than 100***

SELECT hostel\_name, strength

FROM Hostel

WHERE strength > 100;

***-- 7. List wardens and their students who have pending applications***

SELECT w.warden\_id, w.wd\_name, s.student\_id, a.app\_no, a.reason

FROM WARDEN w

JOIN Student\_Warden sw ON w.warden\_id = sw.warden\_id

JOIN STUDENT s ON sw.student\_id = s.student\_id

JOIN APPLICATION a ON s.student\_id = a.student\_id

WHERE a.app\_status = 'Pending';

***-- 8. Display the name and email of warden who manages Nila hostel***

SELECT wd\_name, email\_id

FROM WARDEN

WHERE hostel\_name = 'Nila';

***-- 9. Insert a new warden record***

*INSERT INTO WARDEN (warden\_id, wd\_name, email\_id, hostel\_name)*

*VALUES ('WRD0008', 'Mr. James Wilson', 'jameswilson@gmail.com', 'Nila');*

***-- 10. Update the email address of a student***

UPDATE STUDENT

SET email\_id = 'newemail@gmail.com'

WHERE student\_id = '2023BCS0010';

***-- 11. Update application status to 'Expired' for all past leaves***

UPDATE APPLICATION

SET app\_status = 'Expired'

WHERE end\_date < CURDATE()

AND app\_status = 'Approved';

***-- 12. Find hostels with no students on leave***

*SELECT hostel\_name*

*FROM Hostel*

*WHERE on\_leave = 0;*

***-- 13. Find students who have never applied for leave but are assigned to a warden***

SELECT s.student\_id, s.email\_id, w.warden\_id, w.wd\_name

FROM STUDENT s

JOIN Student\_Warden sw ON s.student\_id = sw.student\_id

JOIN WARDEN w ON sw.warden\_id = w.warden\_id

LEFT JOIN APPLICATION a ON s.student\_id = a.student\_id

WHERE a.app\_no IS NULL;

***-- 14. List applications that were approved on the same day they were submitted***

SELECT app\_no, student\_id, start\_date, reason

FROM APPLICATION

WHERE DATE(start\_date) = DATE(end\_date)

AND app\_status = 'Approved';

***-- 15. Transfer all students from one warden to another***

BEGIN TRANSACTION;

UPDATE Student\_Warden

SET warden\_id = 'WRD0002'

WHERE warden\_id = 'WRD0001';

UPDATE Application\_Warden

SET warden\_id = 'WRD0002'

WHERE warden\_id = 'WRD0001'

AND app\_no IN (

SELECT app\_no

FROM APPLICATION

WHERE app\_status = 'Pending'

);

COMMIT;