DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES1201802828 HIMANSHU JAIN

The Placement Management System offers an interface for interaction between the students and the companies for on-campus placement activities. This system ensures reliable, secure, transparent, and efficient conduction of interviews by the companies registered with the University to fill their job requirements. Initial process of developing this system is the construction of ER Model and Relational Model. Data Models ensure that the system is built efficiently and is reliable for usage in the real-world applications. Various CRUD operations are performed to build and test the system. SQL queries are executed to meet the general requirements of the companies and the students. Essential triggers are implemented to keep track of redundant data in the database and clear them whenever required. The system is built to serve the real-world requirements and to be used by the companies during their interview process at the University. Job vacancies and the job roles are open to students who wish to apply for the interviews, and companies regularly update them to maintain consistency and transparency. Efficient operation and conduction of the process is observed as a result of building this ingenious system.

Introduction

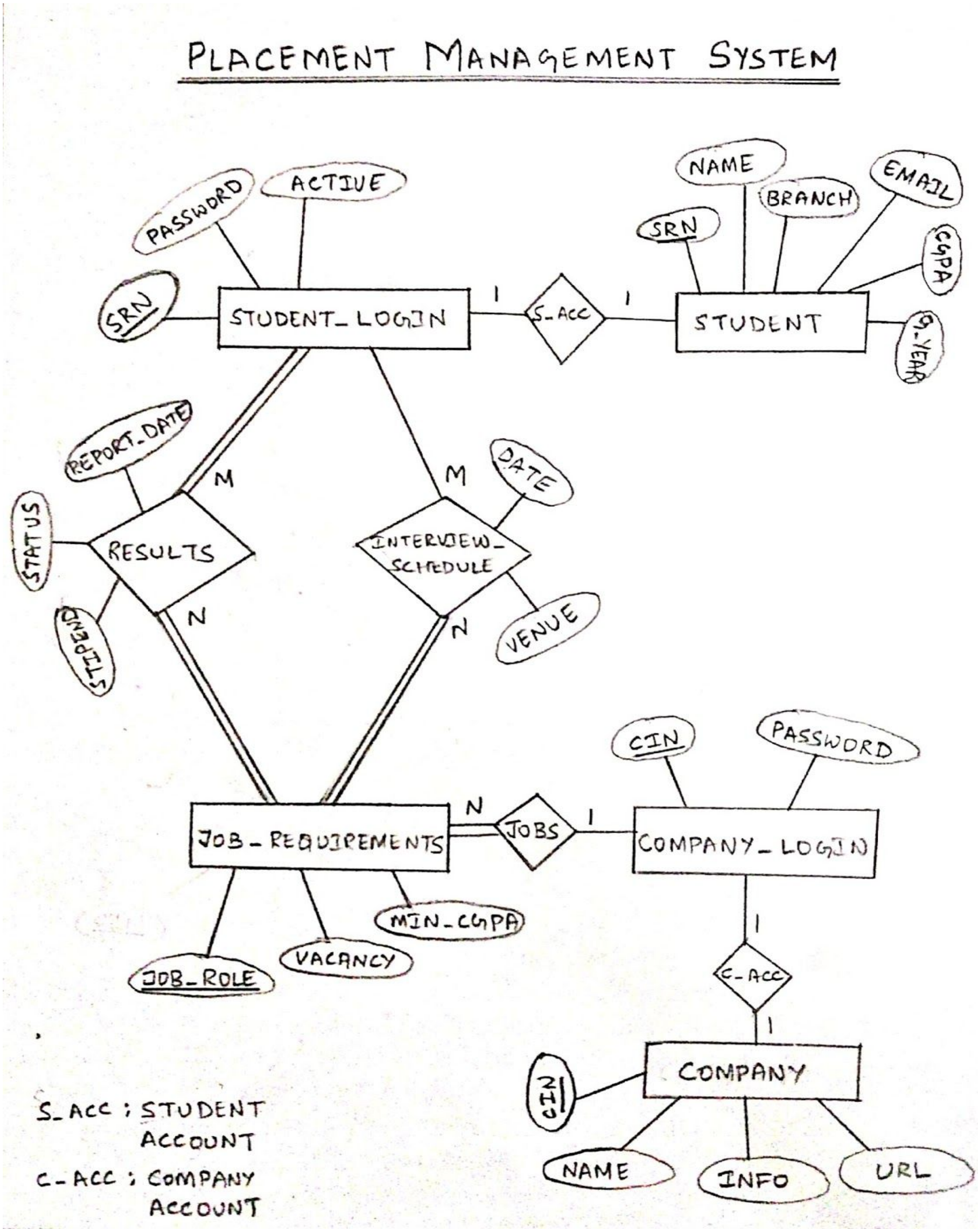
The Placement Management System creates student and company databases. Initially, the students and the companies have to register with the University to be listed in this process. This system maintains a large database of students wherein all the information of students including personal records and academic performance in terms of CGPA is stored. Company information including the available vacancies and job roles they are willing to offer is maintained in this database. It allows companies to post their job requirements. Interviews will be scheduled based on the students application, vacancies available, and their minimum cut-off in CGPA. Once the interview is scheduled and successfully completed, the results are declared by the company and are made available for students to confirm the jobs. Various attributes are used to depict this system as a real-world scenario.

Data Model

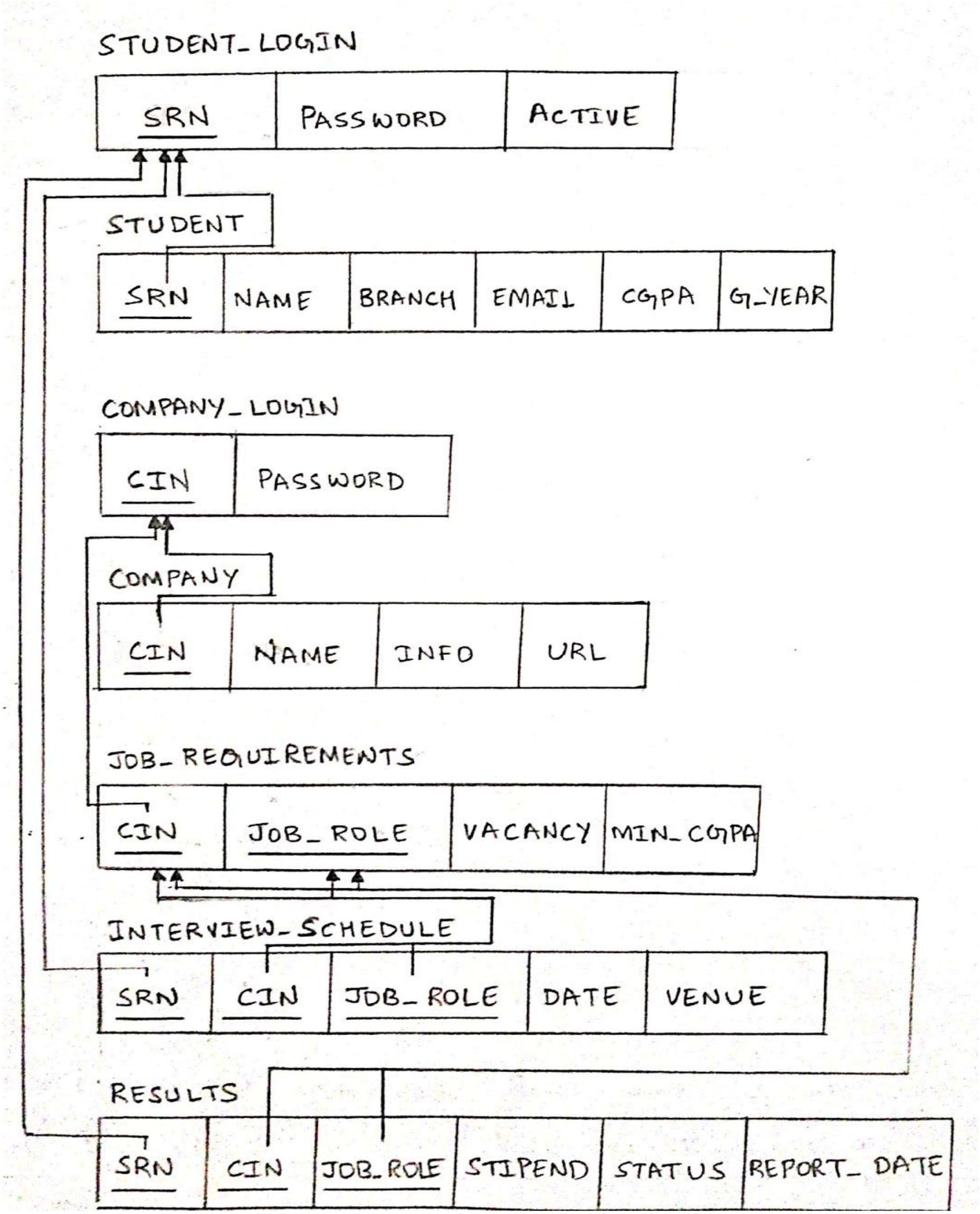
Data models define how the logical structure of a database is modeled. They are the fundamental entities to introduce abstraction in DBMS. The following subsections deal with the following data models :

- Entity-Relationship (ER) Model
- Relational Model

ER DIAGRAM


PLACEMENT MANAGEMENT SYSTEM
PASSWORD
ACTIVE
SRN
STUDENT_LOGIN
1
S_ACC
1
STUDENT
NAME
BRANCH
EMAIL
SRN
CGPA
G_YEAR
REPORT_DATE
STATUS
RESULTS
STIPEND
M
N
M
N
DATE
INTERVIEW_SCHEDULE
VENUE
CIN
PASSWORD
JOB_REQUIREMENTS
N
JOBS
1
COMPANY_LOGIN
MIN_CGPA
VACANCY
JOB_ROLE
1
C_ACC
1
COMPANY
CIN
NAME
INFO
URL
S_ACC : STUDENT ACCOUNT
C_ACC : COMPANY ACCOUNT

SCHEMA DIAGRAM



CHOICE OF KEYS

- Students *SRN* is considered as the primary key in STUDENT_LOGIN relation
- Companies have their unique *Corporate Identification Number (CIN)* which is used as the primary key in COMAPANY_LOGIN relation
- Companies post their job requirements. Hence the combination of *CIN* and *JOB_ROLE* is considered as the primary key in JOB_REQUIREMENTS table
- Students apply for interviews with companies. A student cannot apply for an interview on the same date. Thus, *SRN* and interview *DATE* is taken as the primary key for INTERVIEW_SCHEDULE relation
- On successful completion of the interview, companies post their results on the job portal. Students applying for a job position in a company are unique. Hence, the combination of *SRN, CIN* and *JOB_ROLE* is the primary key in the RESULTS relation

CHOICE OF DATA TYPES

- The general data types such as *VARCHAR, CHAR, INTEGER* and *DECIMAL* have been used.
- CGPA is a decimal number with 2 digits of precision
    - CGPA DECIMAL(4,2) NOT NULL
- DATE is also used as a data type in INTERVIEW_SCHEDULE relation
    - DATE DATE NOT NULL

FD and Normalization

FUNCTIONAL DEPENDENCY

A functional dependency is a constraint between two sets of attributes from the database.
Some of the functional dependencies are as follows:

- ***SRN → {CGPA, G_YEAR}***
- ***SRN → {NAME,BRANCH}***
- ***{CIN, JOB_ROLE} → VACANCY***
- ***{SRN, CIN, JOB_ROLE} → {STIPEND, REPORT_DATE}***
- ***{SRN,DATE} → {VENUE}***

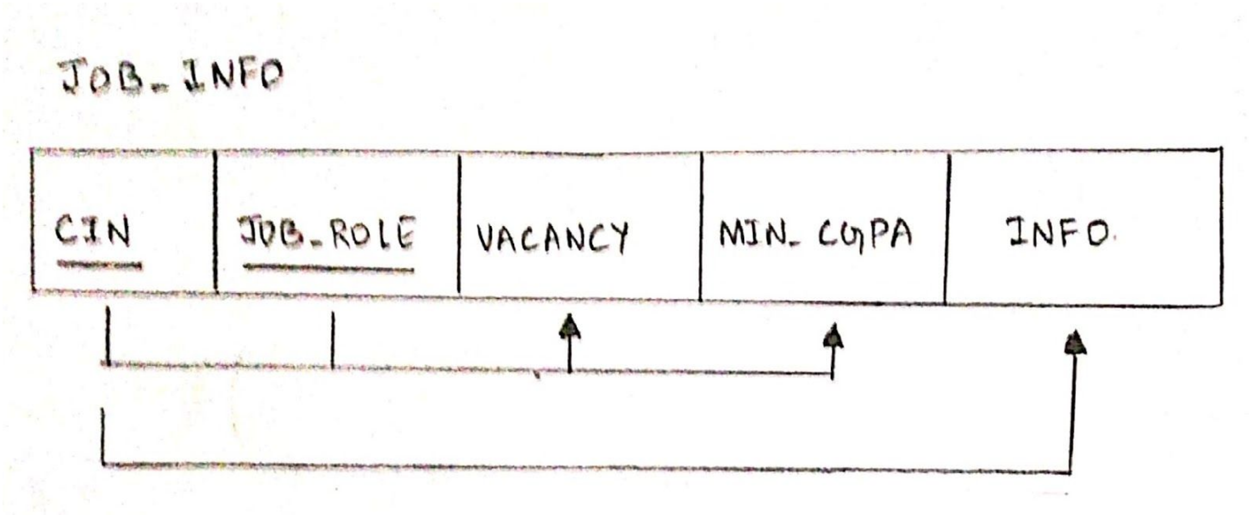
NORMALIZATION

The normalization process takes a relation schema through a series of tests to certify whether it satisfies a certain normal form. The relations obtained using ER diagrams are in the normal form. But these can be violated on addition of redundant attributes.

The attributes of the relations are all *atomic*. Hence they are in their ***First Normal Form (NF)***.

VIOLATION OF SECOND NORMAL FORM

Consider the *INFO* attribute of the COMPANY database. If this attribute is added to the JOB_REQUIREMENTS relation, then the second normal form will be violated.

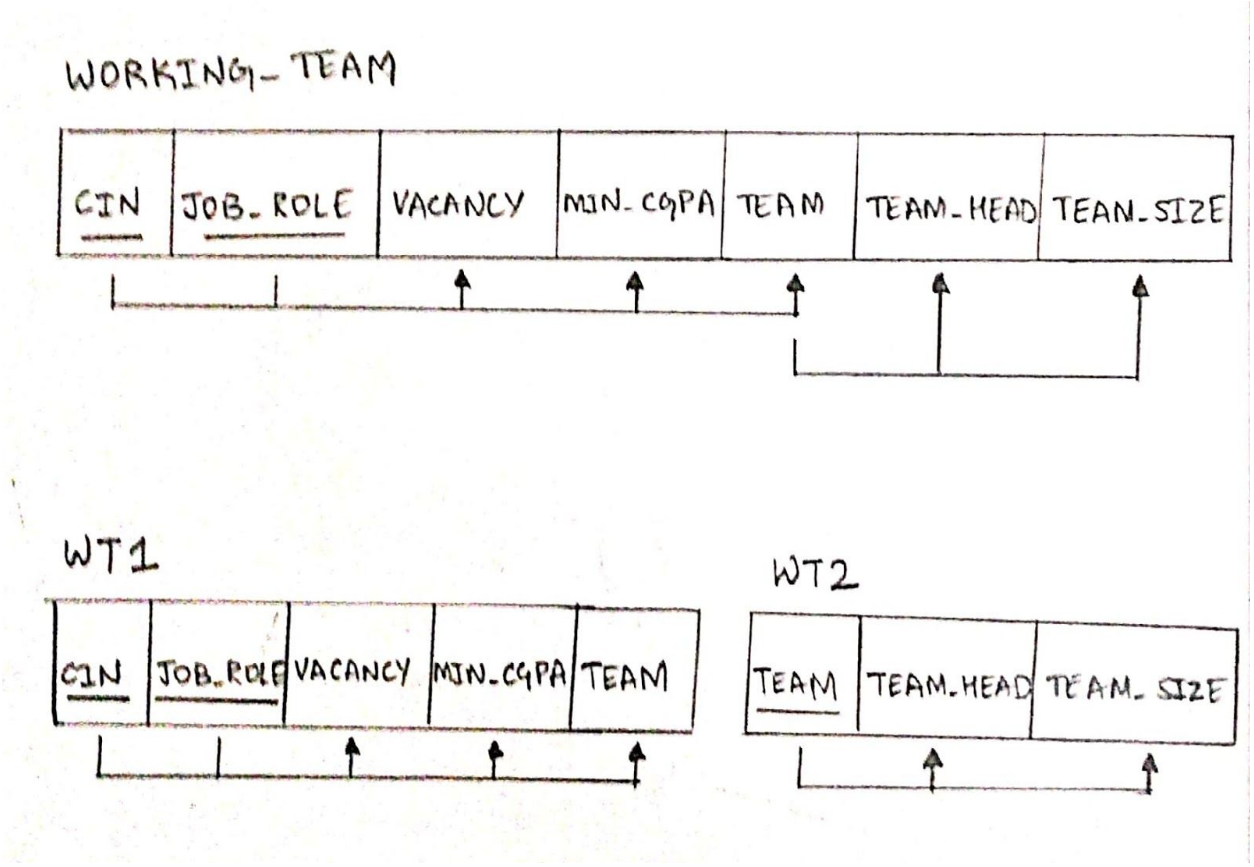


CIN → INFO, but the primary key is {*CIN, JOB_ROLE*}

The non-prime attribute *INFO* is not *fully dependent* on the primary key of the JOB_REQUIREMENTS relation. Hence addition of *INFO* attribute in JOB_REQUIREMENTS relations violates second NF.

VIOLATION OF THIRD NORMAL FORM

Consider the relation WORKING_TEAM



In every company, there are multiple teams based on the *JOB_ROLES*. Once a student is selected, he is assigned to a team for the projects.

The functional dependencies are:

- ***{CIN, JOB_ROLE} → TEAM***
- ***TEAM → {TEAM_HEAD, TEAM_SIZE}***

Here, the attributes TEAM_HEAD and TEAM_SIZE are *transitively dependent* on the primary key. This violates the third NF. The relations have to be split up into two sub-relations, namely WT1 and WT2 to obtain the third normal form.

BAD RELATIONAL SCHEMA DESIGN

When COMPANY and JOB_REQUIREMENTS relations are combined, it suffers from INSERT and UPDATE ANOMALIES. *SPURIOUS TUPLES* are also generated.

COMPANY_JOBS (CIN, JOB_ROLE, NAME, INFO, URL, VACANCY, MIN_CGPA)

- A new company record is inserted into the above table. This company might not have any job requirements currently. Yet, we will have to fill in redundant information or assign NULL values. This leads to an *insertion anomaly*.
- A company can have multiple job requirements. This leads to the repetition of information in INFO and URL columns.
- While performing join operation with other relations, it leads the creation of *spurious tuples*.

DDL

The following SQL commands will create the relations required for the database:

```
DROP TABLE IF EXISTS STUDENT_LOGIN CASCADE;

CREATE TABLE STUDENT_LOGIN(
SRN     CHAR(13) NOT NULL,
PASSWORD    VARCHAR(30) NOT NULL,
ACTIVE          INTEGER NOT NULL DEFAULT 0,
CONSTRAINT PK_LOGIN PRIMARY KEY (SRN) );

DROP TABLE IF EXISTS COMPANY_LOGIN CASCADE;

CREATE TABLE COMPANY_LOGIN(
CIN     CHAR(5) NOT NULL,   --STRICTLY 5 DIGITS
PASSWORD    VARCHAR(30) NOT NULL,
CONSTRAINT PK_C_LOGIN PRIMARY KEY (CIN) );

DROP TABLE IF EXISTS STUDENT CASCADE;

CREATE TABLE STUDENT(
SRN     CHAR(13) NOT NULL,
NAME  VARCHAR(30) NOT NULL,
BRANCH        CHAR(3) NOT NULL,
EMAIL VARCHAR(30) NOT NULL,
CGPA  DECIMAL(4,2) NOT NULL,
G_YEAR INTEGER NOT NULL CHECK (G_YEAR > 2020),
CONSTRAINT PK_STU PRIMARY KEY (SRN),
CONSTRAINT FK_STU_ID FOREIGN KEY(SRN) REFERENCES STUDENT_LOGIN(SRN) ON
DELETE CASCADE ON UPDATE CASCADE );

DROP TABLE IF EXISTS COMPANY CASCADE;

CREATE TABLE COMPANY(
CIN     CHAR(5) NOT NULL,
NAME  VARCHAR(30) NOT NULL,
INFO VARCHAR(100) NOT NULL,
URL VARCHAR(100),
CONSTRAINT PK_COM PRIMARY KEY(CIN),
CONSTRAINT FK_COM_CID FOREIGN KEY(CIN) REFERENCES COMPANY_LOGIN(CIN)
ON DELETE CASCADE ON UPDATE CASCADE );

DROP TABLE IF EXISTS JOB_REQUIREMENTS CASCADE;

CREATE TABLE JOB_REQUIREMENTS(
CIN     CHAR(5) NOT NULL,
JOB_ROLE     VARCHAR(30) NOT NULL,
VACANCY INTEGER NOT NULL,
MIN_CGPA DECIMAL(4,2) NOT NULL DEFAULT 05.00,
CONSTRAINT FK_CR_CID FOREIGN KEY(CIN) REFERENCES COMPANY_LOGIN(CIN) ON
DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT PK_CR PRIMARY KEY(CIN,JOB_ROLE) );

DROP TABLE IF EXISTS INTERVIEW_SCHEDULE CASCADE;

CREATE TABLE INTERVIEW_SCHEDULE(
SRN    CHAR(13) NOT NULL,
CIN     CHAR(5) NOT NULL,
JOB_ROLE     VARCHAR(30) NOT NULL,
DATE  DATE NOT NULL,
VENUE          VARCHAR(30) NOT NULL,
CONSTRAINT FK_SCH_SID FOREIGN KEY(SRN) REFERENCES STUDENT_LOGIN(SRN)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT FK_SCH_CID_SDATE FOREIGN KEY(CIN,JOB_ROLE) REFERENCES
JOB_REQUIREMENTS(CIN,JOB_ROLE) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT PK_SCH PRIMARY KEY(SRN,DATE) );

DROP TABLE IF EXISTS RESULTS CASCADE;
```

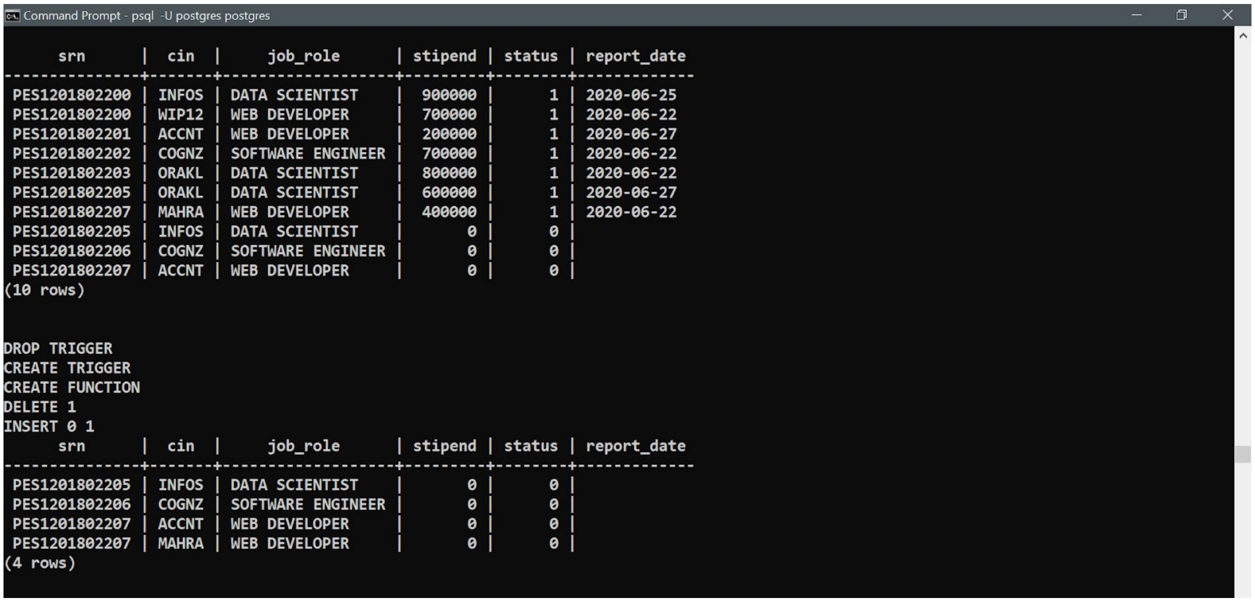
```
CREATE TABLE RESULTS(
SRN    CHAR(13) NOT NULL,
CIN     CHAR(5) NOT NULL,
JOB_ROLE     VARCHAR(30) NOT NULL,
STIPEND INTEGER NOT NULL,
STATUS INTEGER DEFAULT 0,
REPORT_DATE        DATE ,
CONSTRAINT CH_RE_OFF CHECK(STATUS = 1 OR STATUS = 0),
CONSTRAINT FK_RE_SID FOREIGN KEY(SRN) REFERENCES STUDENT_LOGIN(SRN) ON
DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT FK_SCH_CID_SDATE FOREIGN KEY(CIN,JOB_ROLE) REFERENCES
JOB_REQUIREMENTS(CIN,JOB_ROLE) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT PK_RE PRIMARY KEY(SRN,CIN,JOB_ROLE) );
```

Triggers

1) **Remove all such entries from RESULTS relation where a student has accepted the offer (*STATUS* = 1)**

```
SELECT * FROM RESULTS;

DROP TRIGGER IF EXISTS ACCEPTED_STUDENTS ON RESULTS CASCADE;

CREATE TRIGGER ACCEPTED_STUDENTS AFTER INSERT ON RESULTS
FOR EACH ROW EXECUTE PROCEDURE MODIFIERFUNC();

CREATE OR REPLACE FUNCTION MODIFIERFUNC() RETURNS TRIGGER AS
$ACCEPTED_STUDENTS$
  BEGIN
    DELETE FROM RESULTS WHERE STATUS=1;
    RETURN NEW;
  END;
$ACCEPTED_STUDENTS$ LANGUAGE PLPGSQL;

DELETE FROM RESULTS WHERE SRN = 'PES1201802207' AND CIN='MAHRA';
INSERT INTO RESULTS (SRN,CIN,JOB_ROLE,STIPEND,STATUS)
VALUES('PES1201802207','MAHRA','WEB DEVELOPER',0,0);

SELECT * FROM RESULTS;
```

OUTPUT :

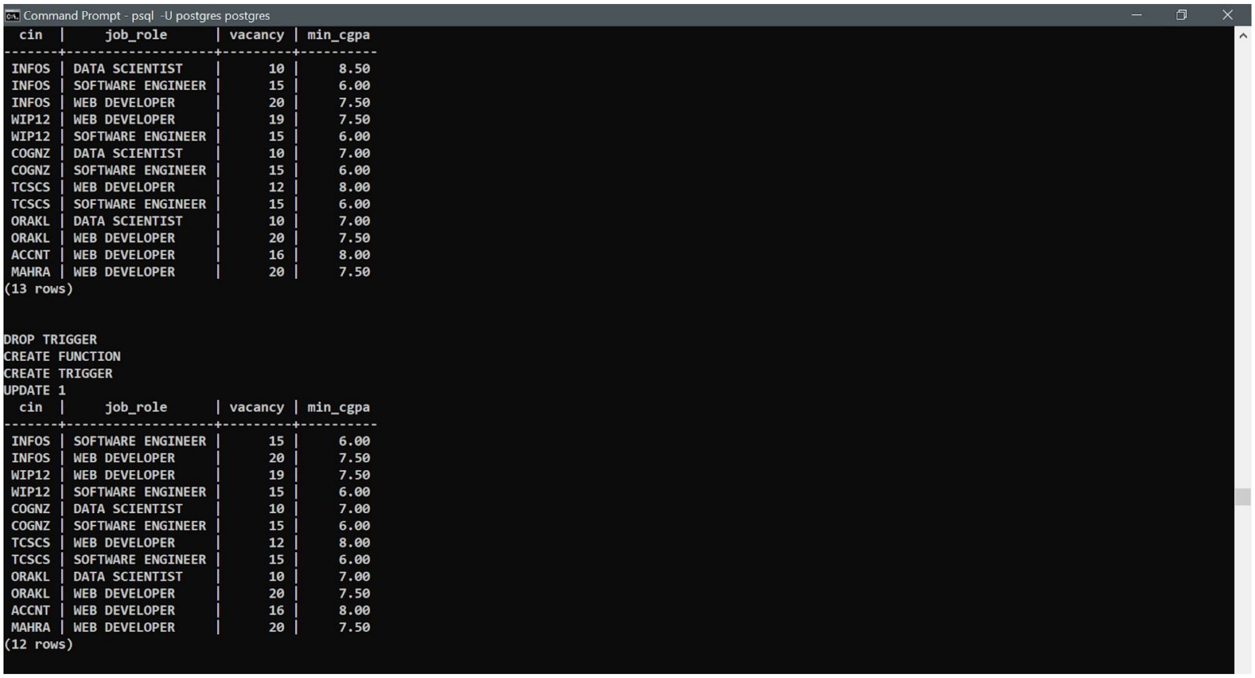


```
     srn      |  cin  |     job_role      | stipend | status | report_date
--------------+-------+-------------------+---------+--------+-------------
 PES1201802200 | INFOS | DATA SCIENTIST    |  900000 |      1 | 2020-06-25
 PES1201802200 | WIP12 | WEB DEVELOPER     |  700000 |      1 | 2020-06-22
 PES1201802201 | ACCNT | WEB DEVELOPER     |  200000 |      1 | 2020-06-27
 PES1201802202 | COGNZ | SOFTWARE ENGINEER |  700000 |      1 | 2020-06-22
 PES1201802203 | ORAKL | DATA SCIENTIST    |  800000 |      1 | 2020-06-22
 PES1201802205 | ORAKL | DATA SCIENTIST    |  600000 |      1 | 2020-06-27
 PES1201802207 | MAHRA | WEB DEVELOPER     |  400000 |      1 | 2020-06-22
 PES1201802205 | INFOS | DATA SCIENTIST    |       0 |      0 |
 PES1201802206 | COGNZ | SOFTWARE ENGINEER |       0 |      0 |
 PES1201802207 | ACCNT | WEB DEVELOPER     |       0 |      0 |
(10 rows)


DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
DELETE 1
INSERT 0 1
     srn      |  cin  |     job_role      | stipend | status | report_date
--------------+-------+-------------------+---------+--------+-------------
 PES1201802205 | INFOS | DATA SCIENTIST    |       0 |      0 |
 PES1201802206 | COGNZ | SOFTWARE ENGINEER |       0 |      0 |
 PES1201802207 | ACCNT | WEB DEVELOPER     |       0 |      0 |
 PES1201802207 | MAHRA | WEB DEVELOPER     |       0 |      0 |
(4 rows)
```

2) **Remove the entries from JOB_REQUIREMENTS relation if the vacancies have been filled**

```
SELECT * FROM JOB_REQUIREMENTS;

DROP TRIGGER IF EXISTS NO_VACANCIES ON JOB_REQUIREMENTS CASCADE;

CREATE OR REPLACE FUNCTION VACANCIESFUNC() RETURNS TRIGGER
  LANGUAGE PLPGSQL AS
$$BEGIN
      DELETE FROM JOB_REQUIREMENTS
      WHERE VACANCY = 0;

      RAISE NOTICE 'SORRY, NO VACANCIES AVAILABLE';
  RETURN OLD;
END;$$;

CREATE TRIGGER NO_VACANCIES AFTER UPDATE ON JOB_REQUIREMENTS
  FOR EACH ROW EXECUTE PROCEDURE VACANCIESFUNC();

UPDATE JOB_REQUIREMENTS
SET VACANCY = 0
WHERE CIN='INFOS' AND JOB_ROLE='DATA SCIENTIST';

SELECT * FROM JOB_REQUIREMENTS;
```

OUTPUT :



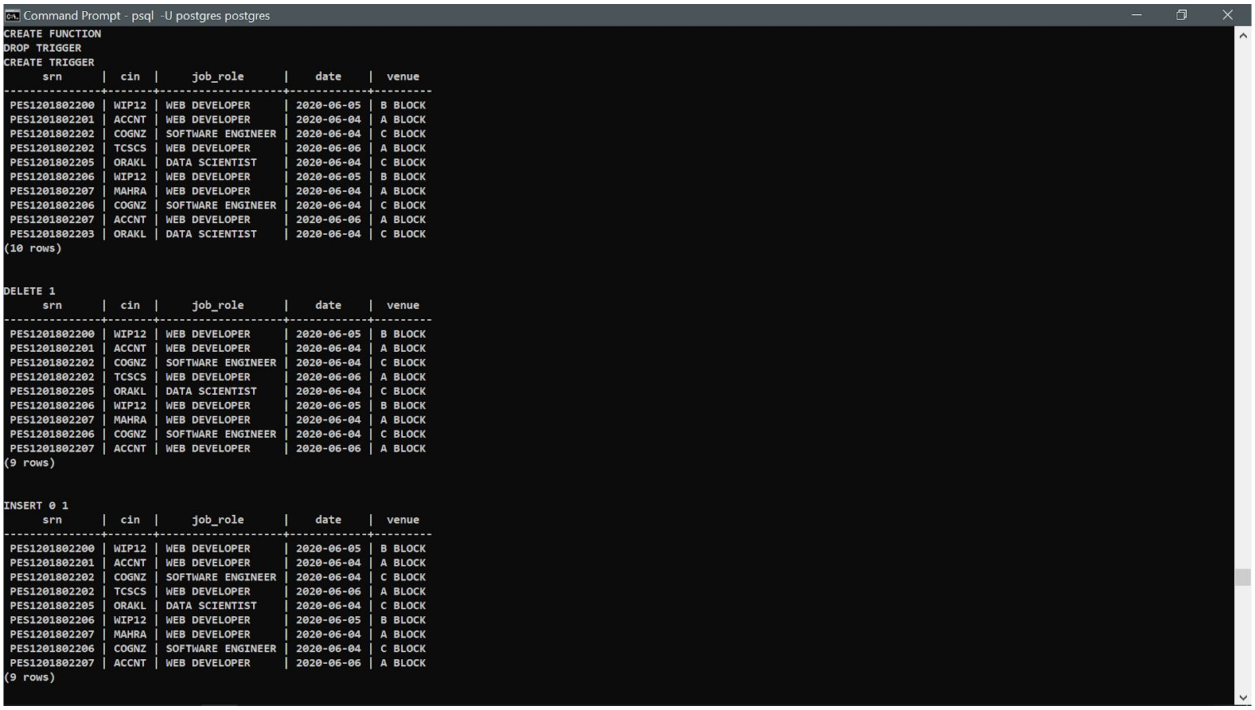
```
  cin  |     job_role      | vacancy | min_cgpa
-------+-------------------+---------+----------
 INFOS | DATA SCIENTIST    |      10 |     8.50
 INFOS | SOFTWARE ENGINEER |      15 |     6.00
 INFOS | WEB DEVELOPER     |      20 |     7.50
 WIP12 | WEB DEVELOPER     |      19 |     7.50
 WIP12 | SOFTWARE ENGINEER |      15 |     6.00
 COGNZ | DATA SCIENTIST    |      10 |     7.00
 COGNZ | SOFTWARE ENGINEER |      15 |     6.00
 TCSCS | WEB DEVELOPER     |      12 |     8.00
 TCSCS | SOFTWARE ENGINEER |      15 |     6.00
 ORAKL | DATA SCIENTIST    |      10 |     7.00
 ORAKL | WEB DEVELOPER     |      20 |     7.50
 ACCNT | WEB DEVELOPER     |      16 |     8.00
 MAHRA | WEB DEVELOPER     |      20 |     7.50
(13 rows)


DROP TRIGGER
CREATE FUNCTION
CREATE TRIGGER
UPDATE 1
  cin  |     job_role      | vacancy | min_cgpa
-------+-------------------+---------+----------
 INFOS | SOFTWARE ENGINEER |      15 |     6.00
 INFOS | WEB DEVELOPER     |      20 |     7.50
 WIP12 | WEB DEVELOPER     |      19 |     7.50
 WIP12 | SOFTWARE ENGINEER |      15 |     6.00
 COGNZ | DATA SCIENTIST    |      10 |     7.00
 COGNZ | SOFTWARE ENGINEER |      15 |     6.00
 TCSCS | WEB DEVELOPER     |      12 |     8.00
 TCSCS | SOFTWARE ENGINEER |      15 |     6.00
 ORAKL | DATA SCIENTIST    |      10 |     7.00
 ORAKL | WEB DEVELOPER     |      20 |     7.50
 ACCNT | WEB DEVELOPER     |      16 |     8.00
 MAHRA | WEB DEVELOPER     |      20 |     7.50
(12 rows)
```

3) Create a trigger to delete old interview schedules

```
CREATE OR REPLACE FUNCTION DELETE_SCHEDULE()
  RETURNS TRIGGER AS
$BODY$
    BEGIN
      DELETE FROM INTERVIEW_SCHEDULE WHERE DATE < NOW();
      RETURN NEW;
    END;
$BODY$
  LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS OLD_SCHEDULE ON INTERVIEW_SCHEDULE CASCADE;

CREATE TRIGGER OLD_SCHEDULE
  AFTER INSERT
  ON INTERVIEW_SCHEDULE
  FOR EACH ROW
  EXECUTE PROCEDURE DELETE_SCHEDULE();

SELECT * FROM INTERVIEW_SCHEDULE;

DELETE FROM INTERVIEW_SCHEDULE
```

```
WHERE SRN='PES1201802203' AND DATE='04/06/2020';

SELECT * FROM INTERVIEW_SCHEDULE;

INSERT INTO INTERVIEW_SCHEDULE (SRN,CIN,JOB_ROLE,DATE,VENUE) VALUES
('PES1201802203','ORAKL','DATA SCIENTIST','04/04/2020','C BLOCK');

SELECT * FROM INTERVIEW_SCHEDULE;
```

OUTPUT :



```
CREATE FUNCTION
DROP TRIGGER
CREATE TRIGGER
      srn      |  cin  |     job_role      |    date    |  venue
---------------+-------+-------------------+------------+---------
 PES1201802200 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802201 | ACCNT | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802202 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802202 | TCSCS | WEB DEVELOPER     | 2020-06-06 | A BLOCK
 PES1201802205 | ORAKL | DATA SCIENTIST    | 2020-06-04 | C BLOCK
 PES1201802206 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802207 | MAHRA | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802206 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802207 | ACCNT | WEB DEVELOPER     | 2020-06-06 | A BLOCK
 PES1201802203 | ORAKL | DATA SCIENTIST    | 2020-06-04 | C BLOCK
(10 rows)


DELETE 1
      srn      |  cin  |     job_role      |    date    |  venue
---------------+-------+-------------------+------------+---------
 PES1201802200 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802201 | ACCNT | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802202 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802202 | TCSCS | WEB DEVELOPER     | 2020-06-06 | A BLOCK
 PES1201802205 | ORAKL | DATA SCIENTIST    | 2020-06-04 | C BLOCK
 PES1201802206 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802207 | MAHRA | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802206 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802207 | ACCNT | WEB DEVELOPER     | 2020-06-06 | A BLOCK
(9 rows)


INSERT 0 1
      srn      |  cin  |     job_role      |    date    |  venue
---------------+-------+-------------------+------------+---------
 PES1201802200 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802201 | ACCNT | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802202 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802202 | TCSCS | WEB DEVELOPER     | 2020-06-06 | A BLOCK
 PES1201802205 | ORAKL | DATA SCIENTIST    | 2020-06-04 | C BLOCK
 PES1201802206 | WIP12 | WEB DEVELOPER     | 2020-06-05 | B BLOCK
 PES1201802207 | MAHRA | WEB DEVELOPER     | 2020-06-04 | A BLOCK
 PES1201802206 | COGNZ | SOFTWARE ENGINEER | 2020-06-04 | C BLOCK
 PES1201802207 | ACCNT | WEB DEVELOPER     | 2020-06-06 | A BLOCK
(9 rows)
```

SQL Queries

1) Students who have applied for the interview for the position of WEB DEVELOPER

```
SELECT S.SRN,S.NAME,I.JOB_ROLE
FROM STUDENT S, INTERVIEW_SCHEDULE I
WHERE S.SRN = I.SRN AND JOB_ROLE='WEB DEVELOPER';
```

2) List out the *companies*, *job roles* and *number of positions* available for students who have at least 8 CGPA

```
SELECT C.NAME AS COMPANY,J.JOB_ROLE AS JOB_ROLE,J.VACANCY AS VACANCIES,J.MIN_CGPA AS MINIMUM_CGPA
FROM COMPANY C, JOB_REQUIREMENTS J
WHERE C.CIN = J.CIN AND J.MIN_CGPA >= 8.00
ORDER BY J.MIN_CGPA DESC;
```

3) Find the total number of positions that are offered by a company for all the job roles

```
SELECT C.NAME AS COMPANY,SUM(VACANCY) AS TOTAL_VACANCIES
FROM COMPANY C, JOB_REQUIREMENTS J
WHERE C.CIN = J.CIN
GROUP BY C.NAME
ORDER BY COMPANY ;
```

4) Find the average starting salary for the batch of *2022* per department

```
SELECT S.BRANCH AS DEPARTMENT, AVG(R.STIPEND) AS AVERAGE_PACKAGE
FROM STUDENT S, RESULTS R
WHERE S.SRN = R.SRN AND S.G_YEAR = 2022
GROUP BY S.BRANCH
ORDER BY AVERAGE_PACKAGE DESC;
```

5) Students whose interview is scheduled in B Block with the company INFOSYS

```
SELECT S.SRN,S.NAME,S.BRANCH
FROM STUDENT AS S
WHERE EXISTS (
   SELECT *
   FROM INTERVIEW_SCHEDULE AS I
   WHERE I.SRN = S.SRN AND I.CIN='INFOS' AND I.VENUE='B BLOCK' )
ORDER BY S.SRN;
```

6) List out the students who had an interview scheduled with WIPRO but did not receive their *results*

```
SELECT S.SRN, S.NAME
FROM STUDENT AS S
WHERE S.SRN IN  (
   (
      SELECT I.SRN
      FROM INTERVIEW_SCHEDULE I
      WHERE I.CIN='WIP12'
   )
   EXCEPT
   (
      SELECT R.SRN
      FROM RESULTS R
      WHERE R.CIN='WIP12'
   )
);
```

7) Companies registered with the university but did not give any job requirements

```
(
   SELECT DISTINCT(C.NAME) AS COMPANY
   FROM COMPANY C LEFT OUTER JOIN JOB_REQUIREMENTS J ON C.CIN = J.CIN
)
EXCEPT
(
   SELECT DISTINCT(C.NAME) AS COMPANY
   FROM COMPANY C RIGHT OUTER JOIN JOB_REQUIREMENTS J ON C.CIN = J.CIN
);
```

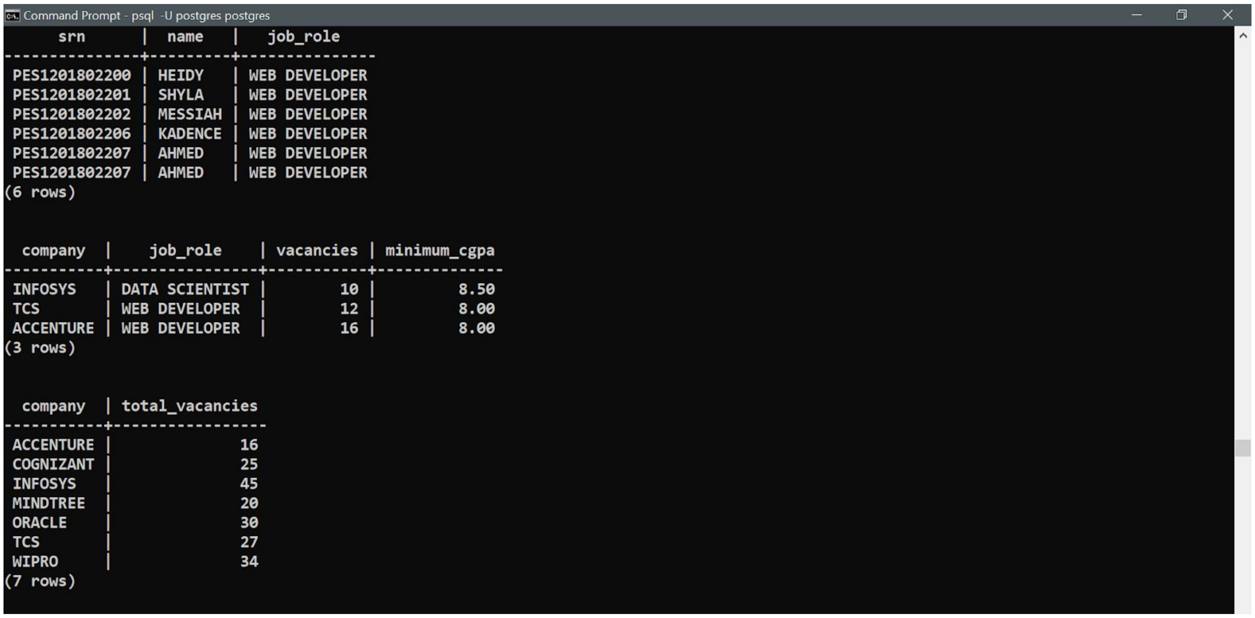
8) Students who did not register for the interview

```
SELECT S.SRN,S.NAME,S.BRANCH
FROM STUDENT S
WHERE S.SRN NOT IN (
   SELECT S.SRN
   FROM STUDENT S RIGHT OUTER JOIN INTERVIEW_SCHEDULE I ON S.SRN = I.SRN
);
```

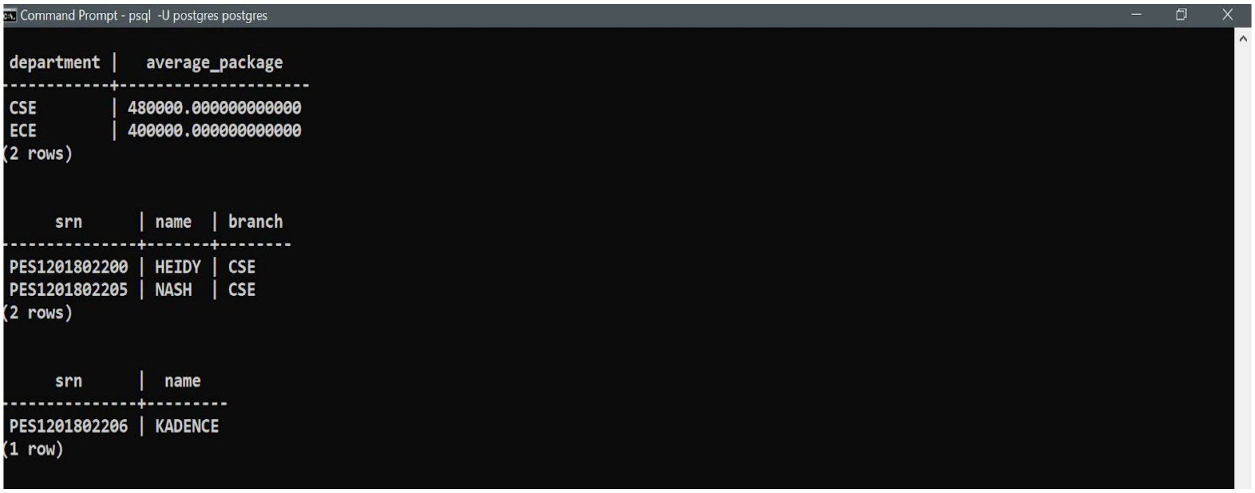
9) List out the students who have their stipend more than 500000 but less than 800000

```
SELECT S.NAME,S.BRANCH,R.JOB_ROLE,R.STIPEND
FROM STUDENT S,RESULTS R
WHERE S.SRN = R.SRN AND R.STIPEND BETWEEN 500000 AND 800000
ORDER BY BRANCH,STIPEND DESC;
```

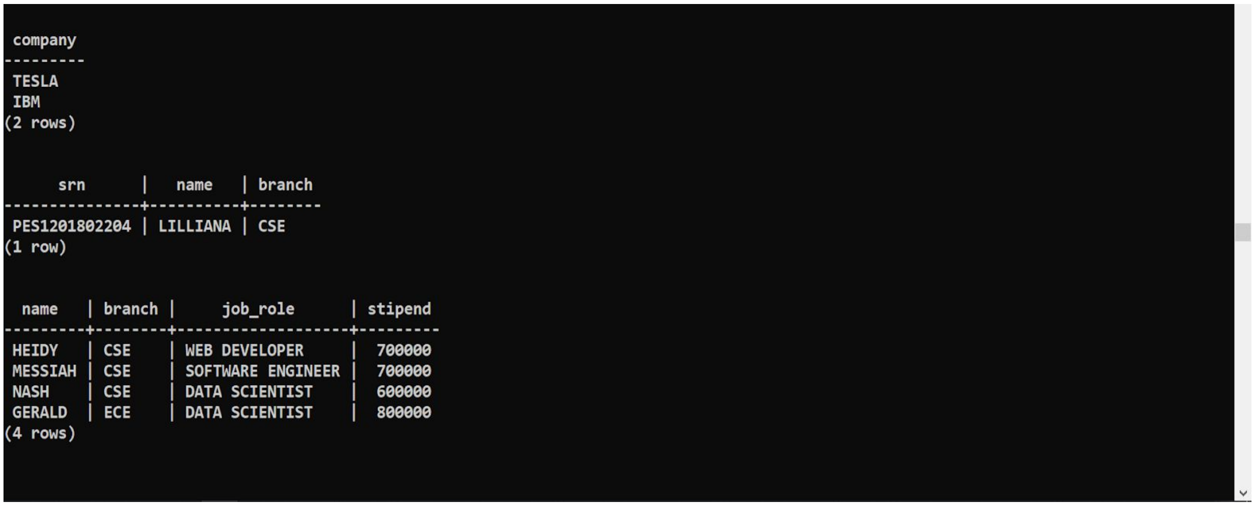
OUTPUT OF SQL QUERIES



```
     srn       |  name   |   job_role
---------------+---------+---------------
 PES1201802200 | HEIDY   | WEB DEVELOPER
 PES1201802201 | SHYLA   | WEB DEVELOPER
 PES1201802202 | MESSIAH | WEB DEVELOPER
 PES1201802206 | KADENCE | WEB DEVELOPER
 PES1201802207 | AHMED   | WEB DEVELOPER
 PES1201802207 | AHMED   | WEB DEVELOPER
(6 rows)


  company  |    job_role    | vacancies | minimum_cgpa
-----------+----------------+-----------+--------------
 INFOSYS   | DATA SCIENTIST |        10 |         8.50
 TCS       | WEB DEVELOPER  |        12 |         8.00
 ACCENTURE | WEB DEVELOPER  |        16 |         8.00
(3 rows)


  company  | total_vacancies
-----------+-----------------
 ACCENTURE |              16
 COGNIZANT |              25
 INFOSYS   |              45
 MINDTREE  |              20
 ORACLE    |              30
 TCS       |              27
 WIPRO     |              34
(7 rows)
```



```
 department |   average_package
------------+---------------------
 CSE        | 480000.000000000000
 ECE        | 400000.000000000000
(2 rows)


     srn       | name  | branch
---------------+-------+--------
 PES1201802200 | HEIDY | CSE
 PES1201802205 | NASH  | CSE
(2 rows)


     srn       |  name
---------------+---------
 PES1201802206 | KADENCE
(1 row)
```



```
 company
---------
 TESLA
 IBM
(2 rows)


     srn       |   name   | branch
---------------+----------+--------
 PES1201802204 | LILLIANA | CSE
(1 row)


  name   | branch |      job_role      | stipend
---------+--------+-------------------+---------
 HEIDY   | CSE    | WEB DEVELOPER      |  700000
 MESSIAH | CSE    | SOFTWARE ENGINEER  |  700000
 NASH    | CSE    | DATA SCIENTIST     |  600000
 GERALD  | ECE    | DATA SCIENTIST     |  800000
(4 rows)
```

Test for Lossless Join Property

While performing decomposition of the relations, non-additive (or lossless) join property should be preserved. No spurious tuples should be generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition. The lossless join property is always defined with respect to a specific set of functional dependencies.

The decomposition is said to be lossless if:

1. Union of both the sub relations must contain all the attributes that are present in the original relation R.
2. Intersection of both the sub relations must not be null.
3. Intersection of both the sub relations must be a super key of either R_1 or R_2 or both.

Consider the relation **COMPANY_JOBS** with the following FDs:

COMPANY_JOBS(CIN,NAME,INFO,URL,JOB_ROL,VACANCY,MIN_CGPA)

- ***cin → {name, info, url}***
- ***{cin, job_role} → {vacancy, min_cgpa}***

Decomposing the relation COMPANY_JOBS into **COMPANY(CIN, NAME, INFO, URL)** and **JOB_REQUIREMENTS(CIN, JOB_ROLE, VACANCY, MIN_CGPA)** to prove the lossless join property.

1. COMPANY(CIN, NAME, INFO, URL) **U** JOB_REQUIREMENTS(CIN, JOB_ROLE, VACANCY, MIN_CGPA) **=** COMPANY_JOBS(CIN,NAME,INFO,URL,JOB_ROL,VACANCY,MIN_CGPA)

   Clearly, union of attributes of the sub relations result in all the attributes of the relation COMPANY_JOBS.

2. COMPANY(CIN, NAME, INFO, URL) ∩ JOB_REQUIREMENTS(CIN, JOB_ROLE, VACANCY, MIN_CGPA) **=** CIN

   Clearly, intersection of the attributes of the sub relations is not null.

3. COMPANY(CIN, NAME, INFO, URL) ∩ JOB_REQUIREMENTS(CIN, JOB_ROLE, VACANCY, MIN_CGPA) **=** CIN

   The *closure* of attribute *CIN* is,

CIN^{+} = {CIN, NAME, INFO, URL}

The attribute *CIN* can determine all the attributes of the sub relation COMPANY. Thus, it is a *super key* of the sub relation COMPANY. Clearly, intersection of the sub relations is a super key of one of the sub relations.

Thus, we conclude that the decomposition is lossless.

Conclusion

The Placement Management System provides a transparent yet secure interface for the communication between the students and the companies. Students can interact with their faculty members regarding the job positions offered by the companies and make a sound decision for their careers. The system provides facilities like insertion and deletion of records from the database. It also provides triggers and functions to automate certain processes. Eligible students receive their offer letter from the companies through this interface. This system ensures that the process of placement activities on the campus run smoothly and effectively.

The interface can offer various levels of abstraction between the students and the companies like providing faculty member details who trained the students for a specific project, list of projects and achievements of the students, storing the academic and non-academic certificates in the database in the form of images and many more will be the future enhancements of this system.

A user interface will be the plus point for this system as it will make the process interactive and interesting. Future improvements will include the involvement of placement officers as the admin so that the companies do not face any issues during the interview process. A web app can be developed and deployed on the PESU Academy to integrate the process with other existing processes of the University.