DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES1201800218 Vinay.V.Kirpalani

Summary of Example chosen

A database management system for maintaining a record of the essential details that are involved in the process of hiring in a single company has been implemented. The implementation below can be used to improve the effectiveness with which the process of hiring takes place in the company.

This model can be used to avoid patterns in the interview process, by retrieval of essential data and detailed analysis of the same. The retrieval of essential data is done by writing complex queries in structured query language.

In a case when there are multiple candidates interviewing for a particular job

This model can be used to identify the better candidate. In addition to this, this model can also be used to determine the most trending IT skills in the market.

This model makes it easier for the company to assess the way different interviewers are conducting interviews. This model also makes it easier to keep the interviews strictly aligned to the job requirement. This model also standardizes the interviews in addition to simplification by reducing the dependence of questions asked on the interviewer. The process of interviewing

Can therefore be made more systematic and competitive using the model implemented.

In the model implemented below we have the following relations.

- Candidate -> contains various essential details about a candidate.
- Job details -> contains job id and job title
- Employee ->contains various essential details about an employee
- Skills -> contains skill name, level, candidate id
- INTERVIEW -> stores the essential interview data such as interviewer id, candidate id, interview id, employee id, RESULT, score, start date
- INTERVIEWER -> stores interviewer id and employee id.
- MAP -> stores INTERVIEWID , QUESTIONID
- Question -> Question id, Question description
- Question tags -> Question id , Tag
- Question difficulty -> Question id , difficulty
- Question explanation ->Question id , explanation
- Job role -> contains job title and job description

The model implemented below has predefined complex sql queries using which essential data is retrieved. Which can later on be analysed .

The model implemented below also acts as a REAL TIME APPLICATION in the following manner.

- The model takes parameters from the user and queries correlated data, or data whose dependence on this parameter Is being studied.
- Sql Queries are written in python using the pyodbc and tabulate module.
- This allows the usage and implementation of an interactive user interface
- The user interface makes it easy for the user to update ,insert ,delete data from the database.

- The model allows the user to update the database in easier way using the built user interface.
- The model supports custom queries which can be written in a user interface by the user in a easier way.
- This model can also be used to load data sets as a sql database in a easier way.

INTRODUCTION

The application was built using python3 with Microsoft SQL as the backend server for the database to execute transactions. The application has a simple interface and requires the user to either type out the parameters required for execution of complex sql queries or for update operations. This application also enables **predefined query execution**. The user has to give input which indicates which predefined query must be run. This application also enables users to write custom queries on the interface itself. Pythons ODBC , pyodbc , tabulate modules were used to implement the same. As many as 43 SQL queries have been written in this implementation.

DATA MODEL

```sql
CREATE TABLE JOB_DETAILS
(
    JOB_ID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    LOCATION VARCHAR(100) NOT NULL,
    JOB_TITLE VARCHAR(100) NOT NULL,
    FILLED bit NOT NULL default 'FALSE'
);
```

- Primary Key: JOB_ID
- Candidate Keys: JOB_ID,(JOB_TITLE,LOCATION)
- Auto Increment feature used here.
- Default value of filled is 0

```
CREATE TABLE JOB_ROLE
(
    JOB_TITLE VARCHAR(100) PRIMARY KEY,
    JOB_DESCRIPTION  VARCHAR(8000)
)
```

Primary Key: JOB_TITLE

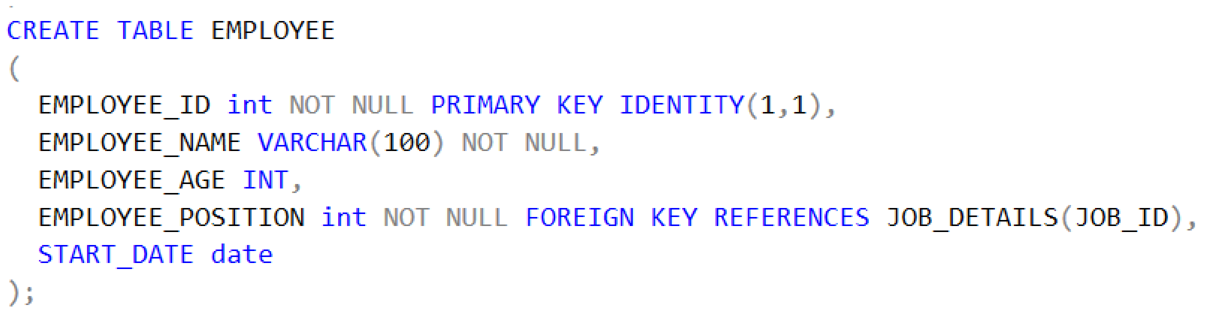
```
CREATE TABLE CANDIDATE
(
    CANDIDATE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    STATUS VARCHAR(100) CHECK (STATUS IN ('ONGOING','REJECTED','ACCEPTED','entry recieved')),
    EXPERIENCE VARCHAR(100),
    NAME VARCHAR(100) NOT NULL,
    AGE INT NOT NULL,
    EDUCATION VARCHAR(8000),
    ROLE int FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID) ON DELETE SET NULL
);
```

- Primary key: Candidate_ID
- Foreign Key: Role
- Referential Integrity constraint implemented here
- Check constraint implemented here

```
CREATE TABLE SKILLS
(
  CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE CASCADE,
  SKILL_NAME VARCHAR(100) NOT NULL,
  LEVEL INT,
  PRIMARY KEY(CANDIDATE_ID, SKILL_NAME)
);
```

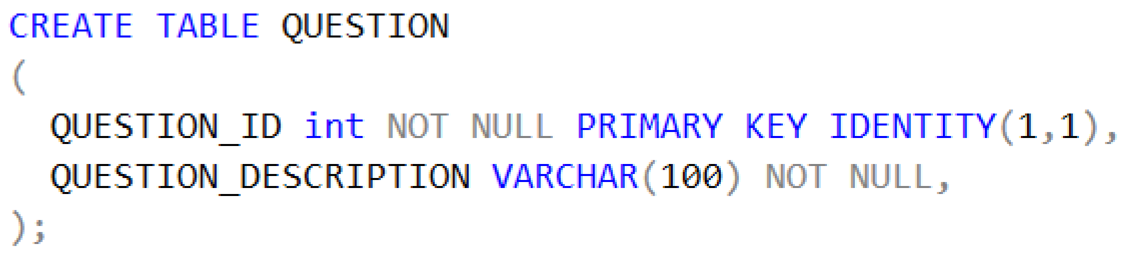
- Primary Key: (candidate_id,SKILL_NAME)
- Foreign Key: Candidate_ID

- Referential integrity constraint applied here.



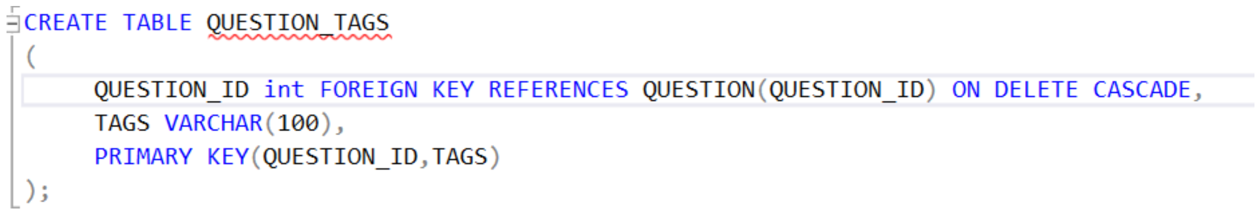
```
CREATE TABLE EMPLOYEE
(
  EMPLOYEE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  EMPLOYEE_NAME VARCHAR(100) NOT NULL,
  EMPLOYEE_AGE INT,
  EMPLOYEE_POSITION int NOT NULL FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID),
  START_DATE date
);
```

- Primary Key: Employee_ID
- Foreign Key: EMPLOYEE_POSITION
- Referential Integrity constraint applied
- Auto increment used here.



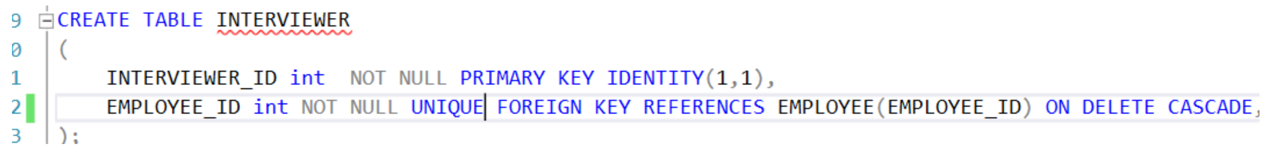
```
CREATE TABLE QUESTION
(
  QUESTION_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  QUESTION_DESCRIPTION VARCHAR(100) NOT NULL,
);
```

- Primary Key: Question_ID
- Auto increment used here.



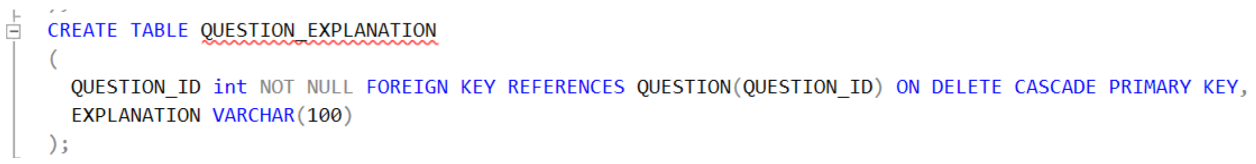
```
CREATE TABLE QUESTION_TAGS
(
    QUESTION_ID int FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE CASCADE,
    TAGS VARCHAR(100),
    PRIMARY KEY(QUESTION_ID,TAGS)
);
```

- Primary Key: (QUESTION_ID,TAGS)
- Foreign Key: Question_ID
- Referntial integrity constraint applied



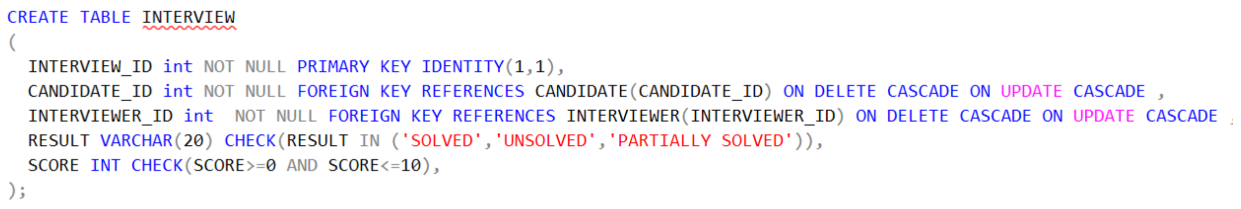
```
CREATE TABLE INTERVIEWER
(
    INTERVIEWER_ID int  NOT NULL PRIMARY KEY IDENTITY(1,1),
    EMPLOYEE_ID int NOT NULL UNIQUE FOREIGN KEY REFERENCES EMPLOYEE(EMPLOYEE_ID) ON DELETE CASCADE,
);
```

- Primary Key: INTERVIEWER_ID
- Candidate Key: EMPLOYEE_ID
- Foreign Key: EMPLOYEE_ID
- Referential integrity constraint applied



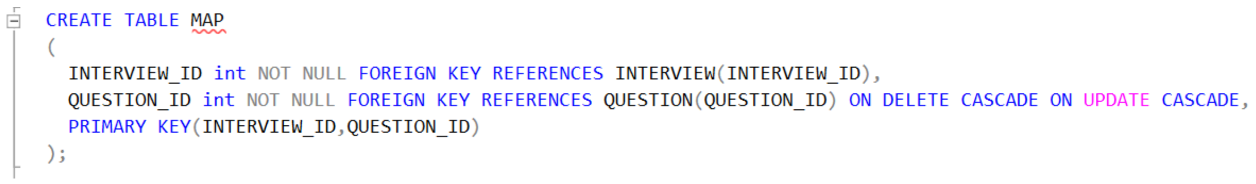
```sql
CREATE TABLE QUESTION_EXPLANATION
(
  QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE CASCADE PRIMARY KEY,
  EXPLANATION VARCHAR(100)
);
```

- Primary Key: QUESTION_ID
- Foreign Key: QUESTION_ID
- Referential integrity constraint applied



```sql
CREATE TABLE INTERVIEW
(
  INTERVIEW_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE CASCADE ON UPDATE CASCADE ,
  INTERVIEWER_ID int  NOT NULL FOREIGN KEY REFERENCES INTERVIEWER(INTERVIEWER_ID) ON DELETE CASCADE ON UPDATE CASCADE ,
  RESULT VARCHAR(20) CHECK(RESULT IN ('SOLVED','UNSOLVED','PARTIALLY SOLVED')),
  SCORE INT CHECK(SCORE>=0 AND SCORE<=10),
);
```

- Primary Key: INTERVIEW_ID
- Foreign Key: CANDIDATE_ID
- Foreign Key: INTERVIEWER_ID
- Referential integrity constraint applied
- Check constraint applied

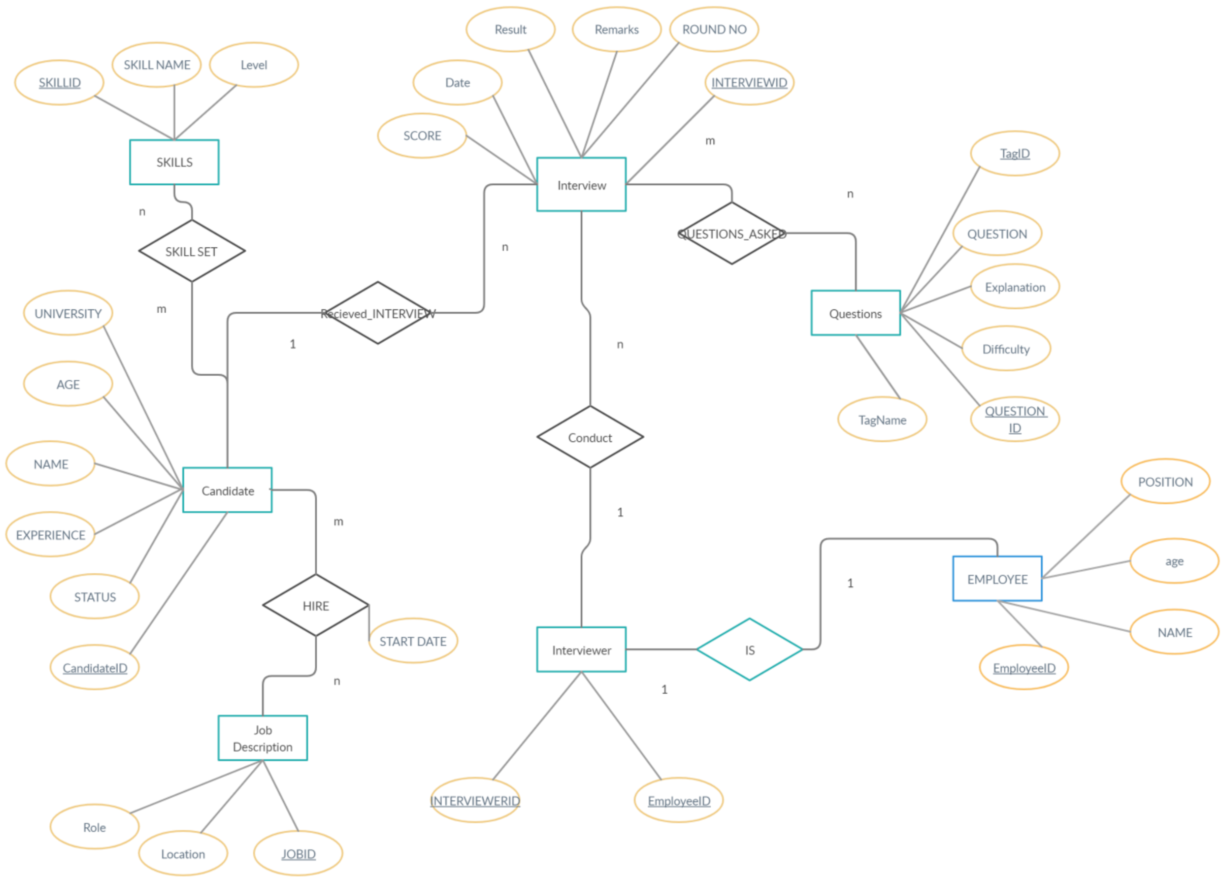


```sql
CREATE TABLE MAP
(
  INTERVIEW_ID int NOT NULL FOREIGN KEY REFERENCES INTERVIEW(INTERVIEW_ID),
  QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY(INTERVIEW_ID,QUESTION_ID)
);
```

- Primary Key : (QUESTION_ID,INTERVIEW_ID)
- Foreign Key: QUESTION_ID

  Referential integrity constraint applied here.

ER MODEL



SKILLID
SKILL NAME
Level
SKILLS
n
SKILL SET
m
Result
Remarks
ROUND NO
Date
INTERVIEWID
SCORE
Interview
m
n
QUESTIONS_ASKED
TagID
QUESTION
Explanation
Difficulty
QUESTION ID
TagName
Questions
n
Recieved_INTERVIEW
1
UNIVERSITY
AGE
NAME
EXPERIENCE
STATUS
CandidateID
Candidate
n
Conduct
1
m
HIRE
START DATE
n
Job Description
Role
Location
JOBID
Interviewer
IS
1
1
EMPLOYEE
POSITION
age
NAME
EmployeeID
INTERVIEWERID
EmployeeID

Functional Dependencies and Normal Form:

```
CREATE TABLE JOB_DETAILS
(
    JOB_ID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    LOCATION VARCHAR(100) NOT NULL,
    JOB_TITLE VARCHAR(100) NOT NULL,
    FILLED bit NOT NULL default 'FALSE'
);
```

- Primary Key: JOB_ID
- Candidate Keys: JOB_ID,(JOB_TITLE,LOCATION)

Functional Dependencies →

1) JOB_ID -> LOCATION,JOB_TITLE,FILLED
2) JOB_TITLE,LOCATION -> JOB_ID,FILLED

Normal form → BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

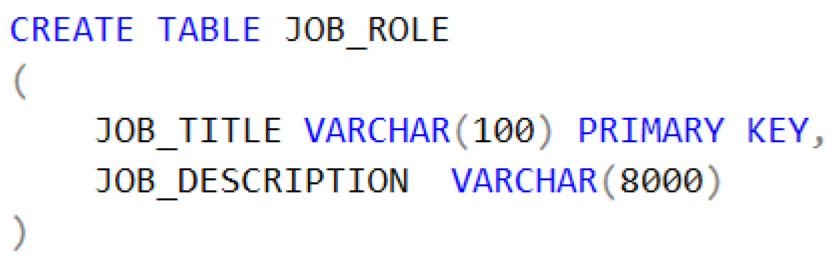
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```sql
CREATE TABLE JOB_ROLE
(
    JOB_TITLE VARCHAR(100) PRIMARY KEY,
    JOB_DESCRIPTION  VARCHAR(8000)
)
```

- Primary Key → JOB_TITLE

Functional Dependencies :

JOB_TITLE -> JOB_DESCRIPTION

Normal Form → BCNF

Why is this table in BCNF?

Answer) Normal Forms: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

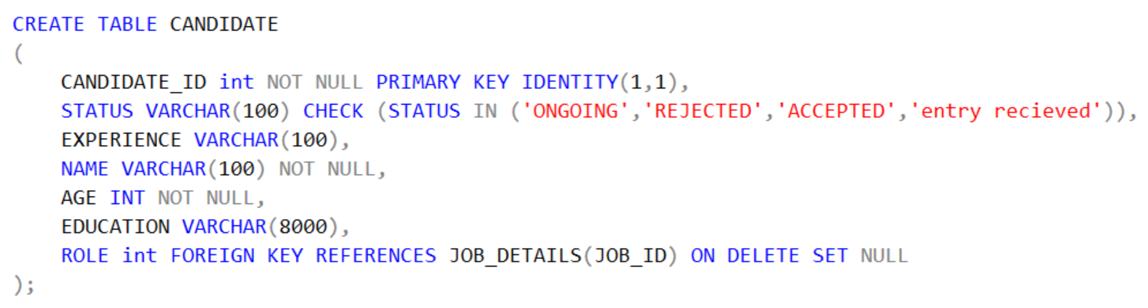
DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) The relation JOB_ROLE has only atomic values , Therefore JOB_ROLE is in 1st Normal Form.

2) The absence of partial dependencies tells us that the above relation is in 2NF.

3) Absence of Transitive dependencies for non-prime attributes tells us that the above table is in 3rd Normal Form.

4) Every Functional Dependency of JOB_ROLE is of the type X → Y where X is the super key of the table , therefore the above table is in BCNF.



```
CREATE TABLE CANDIDATE
(
    CANDIDATE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    STATUS VARCHAR(100) CHECK (STATUS IN ('ONGOING','REJECTED','ACCEPTED','entry recieved')),
    EXPERIENCE VARCHAR(100),
    NAME VARCHAR(100) NOT NULL,
    AGE INT NOT NULL,
    EDUCATION VARCHAR(8000),
    ROLE int FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID) ON DELETE SET NULL
);
```

CONSTRAINTS:

- Check constraint used here.
- Referential integrity constraint used here.

Primary Key: Candidate_ID

Functional Dependencies:

- Candidate_ID -> STATUS,EXPERIENCE,NAME,AGE,EDUCATION,ROLE

Normal Form → BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

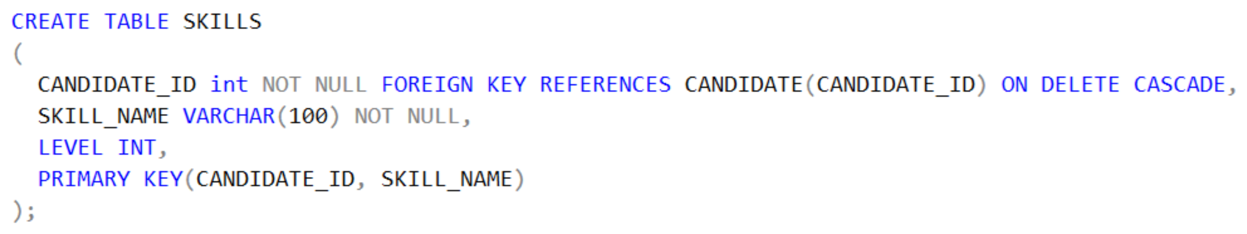
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE SKILLS
(
  CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE CASCADE,
  SKILL_NAME VARCHAR(100) NOT NULL,
  LEVEL INT,
  PRIMARY KEY(CANDIDATE_ID, SKILL_NAME)
);
```

Primary Key: (CANDIDATE_ID, SKILL_NAME)

Functional Dependencies →

- CANDIDATE_ID, SKILL_NAME -> LEVEL

Normal Form: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

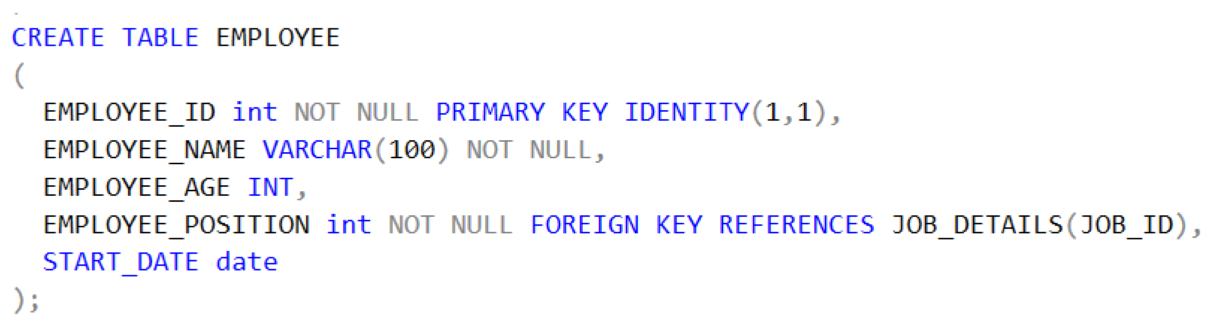
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE EMPLOYEE
(
  EMPLOYEE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  EMPLOYEE_NAME VARCHAR(100) NOT NULL,
  EMPLOYEE_AGE INT,
  EMPLOYEE_POSITION int NOT NULL FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID),
  START_DATE date
);
```

Primary Key: EMPLOYEE_ID

Functional dependencies:

- EMPLOYEE_ID -> EMPLOYEE_NAME, EMPLOYEE_AGE, EMPLOYEE_POSITION

Normal Forms: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

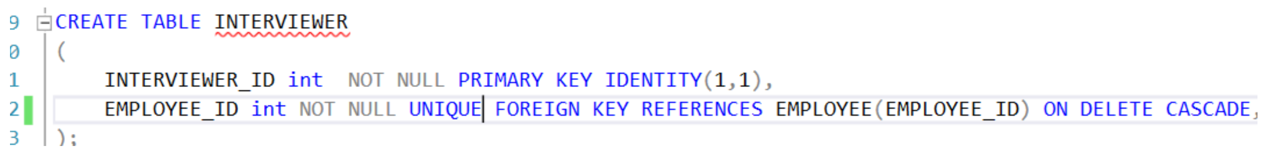
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE INTERVIEWER
(
    INTERVIEWER_ID int  NOT NULL PRIMARY KEY IDENTITY(1,1),
    EMPLOYEE_ID int NOT NULL UNIQUE FOREIGN KEY REFERENCES EMPLOYEE(EMPLOYEE_ID) ON DELETE CASCADE,
);
```

Primary Key : INTERVIEWER_ID

Candidate Key : EMPLOYEE_ID

Functional Dependencies :

- INTERVIEWER_ID -> EMPLOYEE_ID
- EMPLOYEE_ID -> INTERVIEWER_ID

Normal Form: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

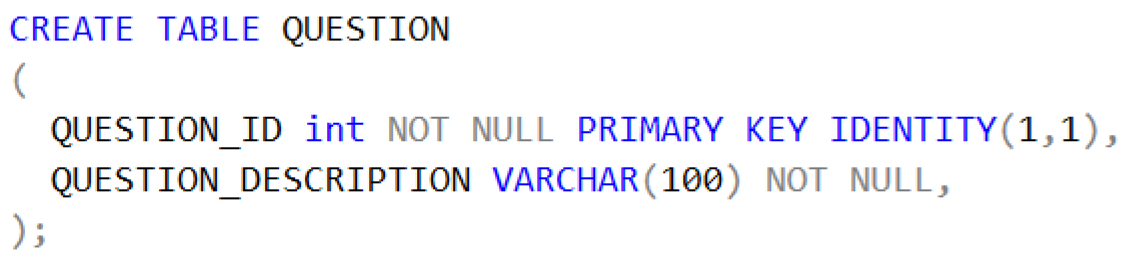
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE QUESTION
(
  QUESTION_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  QUESTION_DESCRIPTION VARCHAR(100) NOT NULL,
);
```

Primary key : QUESTION_ID

Functional Dependencies:

- QUESTION_ID -> QUESTION_DESCRIPTION

Normal Form: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

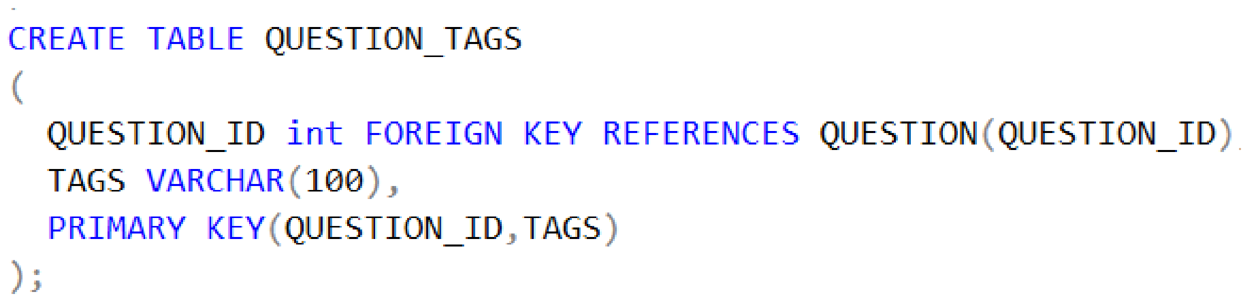
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

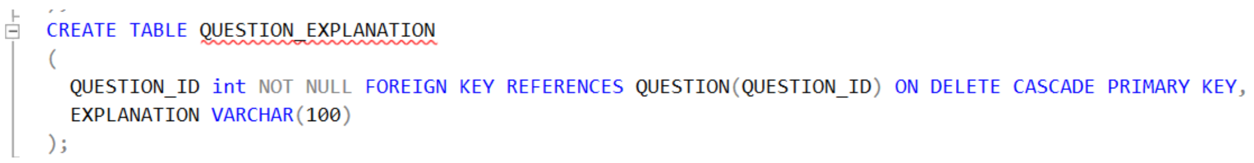
5) Therefore the relation above is in BCNF.



```
CREATE TABLE QUESTION_TAGS
(
  QUESTION_ID int FOREIGN KEY REFERENCES QUESTION(QUESTION_ID)
  TAGS VARCHAR(100),
  PRIMARY KEY(QUESTION_ID,TAGS)
);
```

Primary Key: (QUESTION_ID,TAGS)

Normal Form: BCNF



```
CREATE TABLE QUESTION_EXPLANATION
(
  QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE CASCADE PRIMARY KEY,
  EXPLANATION VARCHAR(100)
);
```

Primary Key: QUESTION_ID

Functional Dependencies:

- QUESTION_ID -> EXPLANATION

Normal Form: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

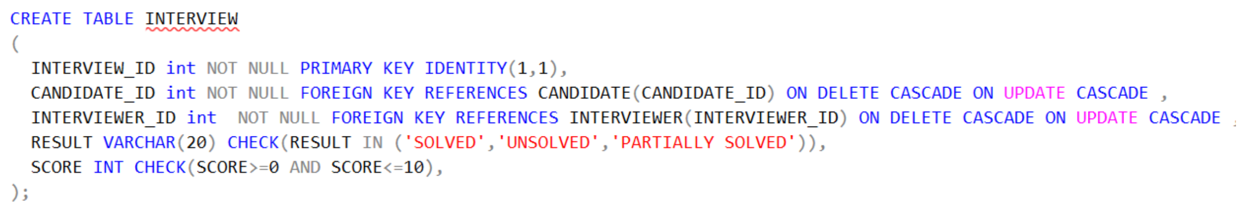
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE INTERVIEW
(
  INTERVIEW_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
  CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE CASCADE ON UPDATE CASCADE ,
  INTERVIEWER_ID int  NOT NULL FOREIGN KEY REFERENCES INTERVIEWER(INTERVIEWER_ID) ON DELETE CASCADE ON UPDATE CASCADE ,
  RESULT VARCHAR(20) CHECK(RESULT IN ('SOLVED','UNSOLVED','PARTIALLY SOLVED')),
  SCORE INT CHECK(SCORE>=0 AND SCORE<=10),
);
```

(Check Constrained used here)

<u>Primary Key</u>: INTERVIEW_ID

Functional Dependencies

- INTERVIEW_ID -> CANDIDATE_ID,INTERVIEWER_ID,RESULT,SCORE

Normal Form: BCNF

Why is this table in BCNF?

Answer: A table is in BCNF if it is in 3rd Normal form and if in every functional dependency X → Y, X is the super key of the table.

DETAILED EXPLANATION OF HOW THE ABOVE TABLE IS IN BCNF

1) All the Attributes in the Relation have atomic values. This Tells us that the relation is in 1st Normal Form.

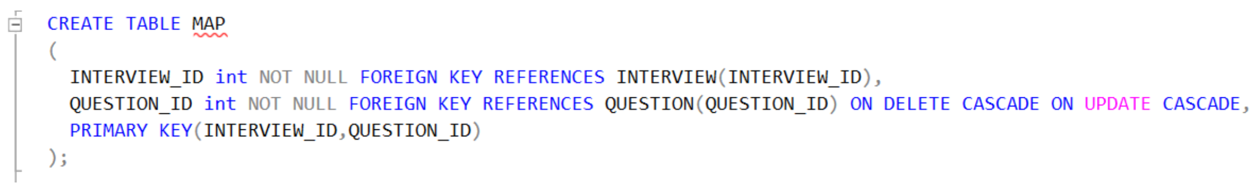
2) Absence of any partial dependencies tells us that this table is in 2nd Normal form.

3) A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.

In this relation above there is absence of transitive dependency for non-prime attributes, therefore the relation shown above is in 3rd Normal Form.

4) Every dependency for the relation above is of the form X → Y, where X is a super key.

5) Therefore the relation above is in BCNF.



```
CREATE TABLE MAP
(
  INTERVIEW_ID int NOT NULL FOREIGN KEY REFERENCES INTERVIEW(INTERVIEW_ID),
  QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY(INTERVIEW_ID,QUESTION_ID)
);
```

Primary Key: (INTERVIEW_ID,QUESTION_ID)

Normal Form : BCNF

TRIGGERS →

A **trigger** is a special type of stored procedure that automatically runs when an event occurs in the database server, In this database management system 2 triggers have been implemented as described below.

1) **Make_Status** : When a candidate initially applies for a job his details are stored in the relation candidate. In all cases when a candidate initially applies for a job, the candidate's application status is 'entry received'. So each time rather than entering a status into the relation candidate as 'entry received' for applicants at a initial stage, we can use a trigger to do the same.

   What does this trigger make_status do ?

   - When a tuple is inserted into the candidate relation, after the insert operation , the trigger sets the status as 'entry received'.
   - This way we don't need to explicitly enter the status as 'entry received' of a candidate who's application is still at an initial stage.

2) **Make_employee** :

   - When a candidate clears interview for a particular job , then the candidate is selected for the respective job. When the candidate is selected for a particular job his status in the candidate table is made as accepted
   - If a candidate is selected for a particular job then this candidate is now an employee of the company.
   - So now the details of this candidate have to added into the employee table. That is a tuple having details of this candidate now must be added in the employee table.
   - Rather than manually adding a tuple to the employee relation each time a candidate gets selected for a job, we can do this operation using a trigger.

What does this trigger make_employee do ?

- Each time a candidate's status is updated as 'accepted', the candidate's details are taken and a corresponding tuple in the employee relation is added.
- An employee id is also generated for this new employee.
- In addition to these features make_employee does one more important operation. The job for which the candidate is selected , its availability status is set to as filled automatically by this trigger. Therefore this particular job opening is not available after a candidate has been selected for the same job.

DDL (create table scripts) →

```
CREATE TABLE JOB_DETAILS
  (
    JOB_ID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    LOCATION VARCHAR(100) NOT NULL,
    JOB_TITLE VARCHAR(100) NOT NULL,
    FILLED bit NOT NULL default 'FALSE'
  );
  CREATE TABLE JOB_ROLE
  (
    JOB_TITLE VARCHAR(100) PRIMARY KEY,
    JOB_DESCRIPTION  VARCHAR(8000)
  )
  CREATE TABLE CANDIDATE
  (
    CANDIDATE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    STATUS VARCHAR(100) CHECK (STATUS IN ('ONGOING','REJECTED','ACCEPTED','entry
recieved')),
    EXPERIENCE VARCHAR(100),
    NAME VARCHAR(100) NOT NULL,
    AGE INT NOT NULL,
    EDUCATION VARCHAR(8000),
    ROLE int FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID) ON DELETE SET NULL
  );
  CREATE TABLE SKILLS
  (
    CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE
CASCADE,
    SKILL_NAME VARCHAR(100) NOT NULL,
    LEVEL INT,
    PRIMARY KEY(CANDIDATE_ID, SKILL_NAME)
  );
  CREATE TABLE EMPLOYEE
  (
    EMPLOYEE_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    EMPLOYEE_NAME VARCHAR(100) NOT NULL,
    EMPLOYEE_AGE INT,
    EMPLOYEE_POSITION int NOT NULL FOREIGN KEY REFERENCES JOB_DETAILS(JOB_ID),
    START_DATE date
  );


  CREATE TABLE INTERVIEWER
  (
    INTERVIEWER_ID int  NOT NULL PRIMARY KEY IDENTITY(1,1),
    EMPLOYEE_ID int NOT NULL FOREIGN KEY REFERENCES EMPLOYEE(EMPLOYEE_ID) ON DELETE
CASCADE,
  );
  CREATE TABLE QUESTION
  (
    QUESTION_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    QUESTION_DESCRIPTION VARCHAR(100) NOT NULL,
  );
  CREATE TABLE QUESTION_TAGS
  (
    QUESTION_ID int FOREIGN KEY REFERENCES QUESTION(QUESTION_ID),
    TAGS VARCHAR(100),
    PRIMARY KEY(QUESTION_ID,TAGS)
  );
  CREATE TABLE QUESTION_EXPLANATION
  (
    QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE
CASCADE PRIMARY KEY,
    EXPLANATION VARCHAR(100)
  );
  CREATE TABLE QUESTION_DIFFICULTY
  (
    QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE
CASCADE PRIMARY KEY,
    DIFFICULTY VARCHAR(8000)
  );
  CREATE TABLE INTERVIEW
  (
    INTERVIEW_ID int NOT NULL PRIMARY KEY IDENTITY(1,1),
    CANDIDATE_ID int NOT NULL FOREIGN KEY REFERENCES CANDIDATE(CANDIDATE_ID) ON DELETE
CASCADE ON UPDATE CASCADE ,
    INTERVIEWER_ID int  NOT NULL FOREIGN KEY REFERENCES INTERVIEWER(INTERVIEWER_ID) ON
DELETE CASCADE ON UPDATE CASCADE ,
    RESULT VARCHAR(20) CHECK(RESULT IN ('SOLVED','UNSOLVED','PARTIALLY SOLVED')),
    SCORE INT CHECK(SCORE>=0 AND SCORE<=10),
  );
  CREATE TABLE MAP
  (
    INTERVIEW_ID int NOT NULL FOREIGN KEY REFERENCES INTERVIEW(INTERVIEW_ID),
    QUESTION_ID int NOT NULL FOREIGN KEY REFERENCES QUESTION(QUESTION_ID) ON DELETE
CASCADE ON UPDATE CASCADE,
    PRIMARY KEY(INTERVIEW_ID,QUESTION_ID)
     );
```

SQL Queries

Problem Statement: prepare a summary for each interviewer, which will consist of

- INTERVIEWER_ID
- NAME
- Number of candidates interviewed
- Average score of candidates interviewed by an interviewer
- Average difficulty of interviews taken by an interview

Query:

```
SELECT INTERVIEWER.INTERVIEWER_ID, EMPLOYEE.EMPLOYEE_NAME,COUNT(CANDIDATE_ID) AS
NUM_CANDIDATES_INTERVIEWED,AVG(SCORE) as AVG_CANDIDATE_SCORE, AVG(CASE WHEN
DIFFICULTY='hard' THEN 8 WHEN DIFFICULTY='challenge' THEN 10 WHEN DIFFICULTY='medium'
THEN 6 WHEN DIFFICULTY='easy' THEN 4 WHEN DIFFICULTY='beginner' THEN 2 END) AS
AVG_DIFFICULTY FROM INTERVIEW JOIN INTERVIEWER ON
INTERVIEW.INTERVIEWER_ID=INTERVIEWER.INTERVIEWER_ID JOIN MAP ON
MAP.INTERVIEW_ID=INTERVIEW.INTERVIEW_ID JOIN QUESTION_DIFFICULTY ON
MAP.QUESTION_ID=QUESTION_DIFFICULTY.QUESTION_ID JOIN EMPLOYEE ON
INTERVIEWER.EMPLOYEE_ID=EMPLOYEE.EMPLOYEE_ID GROUP BY INTERVIEWER.INTERVIEWER_ID,
EMPLOYEE.EMPLOYEE_NAME HAVING AVG(SCORE) IS NOT NULL ORDER BY INTERVIEWER.INTERVIEWER_ID
ASC;
```

Application:

a company would like to do a evaluation of the effectiveness of its interviews. A company would like to therefore analyse the interviewer statistics and various interviewer data with respect to the interview's an interviewer takes.

Problem Statement: For each job_title

- Query the number of distinct locations at which this job_title exists.
- Query the number of available positions for this job_title

Query:

```
SELECT JOB_DETAILS.JOB_TITLE,COUNT(DISTINCT JOB_DETAILS.LOCATION) AS NUM_LOCATIONS,
COUNT(JOB_DETAILS.FILLED)-COUNT(CASE WHEN JOB_DETAILS.FILLED=1 THEN 1 END) AS VACANCIES
from JOB_DETAILS JOIN JOB_ROLE ON JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE GROUP BY
JOB_DETAILS.JOB_TITLE;
```

Application: When a candidate is looking for a job on a online portal, this query can be used to add the different filters a candidate is looking for.

Problem Statement:

Given a candidate id Query the following

- Name, age, education of the candidate
- Average difficulty of the interviews taken by this candidate
- Average score of the candidate

QUERY:

```
SELECT NAME,AGE,EDUCATION,SUM(CASE WHEN DIFFICULTY='hard' THEN 8 WHEN
DIFFICULTY='challenge' THEN 10 WHEN DIFFICULTY='medium' THEN 6 WHEN DIFFICULTY='easy'
THEN 4 WHEN DIFFICULTY='beginner' THEN 2 END)/COUNT(*) AS INTERVIEW_DIFFICULTY,
SUM(SCORE)/COUNT(*) AS AVG_SCORE FROM ((CANDIDATE JOIN INTERVIEW ON
CANDIDATE.CANDIDATE_ID=INTERVIEW.CANDIDATE_ID) JOIN MAP ON
MAP.INTERVIEW_ID=INTERVIEW.INTERVIEW_ID) JOIN QUESTION_DIFFICULTY ON
QUESTION_DIFFICULTY.QUESTION_ID=MAP.QUESTION_ID WHERE CANDIDATE.CANDIDATE_ID=(?) GROUP BY
NAME, AGE, EDUCATION;
```

Application→

1) A company would like to create a summary for the interviewing process for each candidate, in order to evaluate his selection or rejection.

Problem Statement: for each interview id , list the average difficulty of that interview.

Query:

```
/* MAP */
/* Difficulty of interview */
SELECT INTERVIEW.INTERVIEW_ID, AVG(CASE WHEN DIFFICULTY='hard' THEN 8 WHEN
DIFFICULTY='challenge' THEN 10 WHEN DIFFICULTY='medium' THEN 6 WHEN DIFFICULTY='easy'
THEN 4 WHEN DIFFICULTY='beginner' THEN 2 END) AS DIFFICULTY_SCORE FROM (INTERVIEW JOIN
MAP ON INTERVIEW.INTERVIEW_ID=MAP.INTERVIEW_ID) JOIN QUESTION_DIFFICULTY ON
QUESTION_DIFFICULTY.QUESTION_ID=MAP.QUESTION_ID GROUP BY INTERVIEW.INTERVIEW_ID;
```

APPLICATION: A company would like to keep a record of the difficulty of the interviews taken by the company.

Problem Statement : for each question , list the number of interviewers who have asked this question in some interview before

Query:

```
/* interview id */
/* Distribution of questions asked in interviews */
SELECT MAP.QUESTION_ID, COUNT(INTERVIEW.INTERVIEW_ID) AS NUMBER_OF_INTERVIEWERS FROM
INTERVIEW JOIN MAP ON INTERVIEW.INTERVIEW_ID=MAP.INTERVIEW_ID GROUP BY MAP.QUESTION_ID;
```

APPLICATION : this is a wide ranging application as each company would like to keep a record of how many times a particular interviewer has asked a particular question in order to make the interviews less predictable

Problem Statement : Given a interview ID, Query all the details of the questions asked in that interview.

That is given interviewId Query

- Question id
- Question Description

Query:

```
/* List of questions asked in interview by interview id */

SELECT QUESTION.QUESTION_ID,QUESTION.QUESTION_DESCRIPTION FROM INTERVIEW JOIN MAP ON
INTERVIEW.INTERVIEW_ID=MAP.INTERVIEW_ID JOIN QUESTION ON
MAP.QUESTION_ID=QUESTION.QUESTION_ID WHERE INTERVIEW.INTERVIEW_ID=(?);
```

Application : A company would like to maintain a detailed record of the questions asked in a interview.

Problem statement: Query the candidate id, number of interviewers that interview a candidate, average score of the candidate

Condition : Query only when the average score of the candidate is greater than a given lower bound.

Query:

```
/* Filter candidates by average score */
SELECT CANDIDATE.CANDIDATE_ID, NAME, COUNT(INTERVIEW_ID) AS NUM_INTERVIEWS, AVG(SCORE) AS
AVG_SCORE FROM CANDIDATE JOIN INTERVIEW ON CANDIDATE.CANDIDATE_ID=INTERVIEW.CANDIDATE_ID
WHERE SCORE IS NOT NULL GROUP BY CANDIDATE.CANDIDATE_ID, NAME HAVING AVG(SCORE)>(?);
/* lower bound */
```

Application: Each company has to find ways to shortlist candidates , when the number of applicants are large.

Problem Statement: Query a count of the number of candidates for whom the interview is pending

Query:

```
/* Distribution of interview results */
SELECT RESULT, COUNT(*) AS NUM_CANDIDATES FROM INTERVIEW WHERE RESULT IS NOT NULL GROUP
BY RESULT;
```

Application: A company has to keep track of number of interviews pending at any moment of time in order to keep the interview process efficient.

Problem statement:

Query the details of the pending interviews

Query:

```
SELECT INTERVIEW_ID, CANDIDATE_ID, INTERVIEWER_ID FROM INTERVIEW WHERE RESULT IS NULL;
```

APPLICATION

A company would like to maintain a record of details of interviews pending in addition to the count queried earlier.

Problem Statement: Given an interviewerid , query details of all interviews taken by that interviewer.

Query:

```
  SELECT INTERVIEW_ID, CANDIDATE_ID, SCORE, RESULT FROM INTERVIEW WHERE
INTERVIEWER_ID=(?);
```

Problem Statement: Given a candidateID , query the interview details of the interview given by this candidate.

Query:

```
      SELECT INTERVIEW_ID, INTERVIEWER_ID, SCORE, RESULT FROM INTERVIEW WHERE
CANDIDATE_ID=(?);
```

Problem Statement: List the Questions for which a explanation does not exist in the database.

Query:
```
SELECT QUESTION.QUESTION_ID, QUESTION_DESCRIPTION FROM QUESTION JOIN QUESTION_EXPLANATION
ON QUESTION.QUESTION_ID=QUESTION_EXPLANATION.QUESTION_ID WHERE EXPLANATION IS NULL;
```

Problem Statement:
Query the number of questions of different levels of difficulty.

Query:
```
SELECT DIFFICULTY, COUNT(DIFFICULTY) AS NUM_QUESTIONS FROM QUESTION_DIFFICULTY GROUP BY
DIFFICULTY;
```

Applications:
A company needs to maintain a count of different level of questions, in order to evaluate the effectiveness of its interviewing process.

Problem Statement: Given a Question ID , Query all the details about the question.

Query:

```
SELECT QUESTION_DESCRIPTION, EXPLANATION, DIFFICULTY, TAGS FROM ((QUESTION JOIN
QUESTION_EXPLANATION ON QUESTION.QUESTION_ID=QUESTION_EXPLANATION.QUESTION_ID) JOIN
QUESTION_DIFFICULTY ON QUESTION.QUESTION_ID=QUESTION_DIFFICULTY.QUESTION_ID) JOIN
QUESTION_TAGS ON QUESTION.QUESTION_ID=QUESTION_TAGS.QUESTION_ID WHERE
QUESTION.QUESTION_ID=(?);
```

Problem Statement: Query For each interviewer the number of candidates whose status is same as a given status(input).

Query:
```
SELECT INTERVIEWER.INTERVIEWER_ID, EMPLOYEE_NAME, COUNT(CANDIDATE.CANDIDATE_ID) AS
NUM_CANDIDATES FROM ((INTERVIEWER JOIN EMPLOYEE ON
INTERVIEWER.EMPLOYEE_ID=EMPLOYEE.EMPLOYEE_ID) JOIN INTERVIEW ON
INTERVIEWER.INTERVIEWER_ID=INTERVIEW.INTERVIEWER_ID) JOIN CANDIDATE ON
INTERVIEW.CANDIDATE_ID=CANDIDATE.CANDIDATE_ID WHERE CANDIDATE.STATUS=(?) GROUP BY
INTERVIEWER.INTERVIEWER_ID,EMPLOYEE_NAME ORDER BY INTERVIEWER.INTERVIEWER_ID;
```

Problem Statement: Query the number of questions in each difficulty level that are asked by a particular interviewer

Query:
```
SELECT QUESTION_DIFFICULTY.DIFFICULTY, COUNT(*) AS NUM_QUESTIONS FROM (((INTERVIEWER JOIN
INTERVIEW ON INTERVIEWER.INTERVIEWER_ID=INTERVIEW.INTERVIEWER_ID) JOIN MAP ON
INTERVIEW.INTERVIEW_ID=MAP.INTERVIEW_ID) JOIN QUESTION_DIFFICULTY ON
MAP.QUESTION_ID=QUESTION_DIFFICULTY.QUESTION_ID) WHERE INTERVIEWER.INTERVIEWER_ID=(?)
GROUP BY QUESTION_DIFFICULTY.DIFFICULTY;
```

Application:
Using this query a company can identify the number of difficult, easy ,challenge ,medium questions that a particular interviewer asks.

Problem Statement:
Query the details of the candidates that have been interviewed by a particular interviewer.

Query:
```
SELECT EMPLOYEE_NAME,INTERVIEW.CANDIDATE_ID,NAME,AGE,STATUS FROM ((INTERVIEWER JOIN
EMPLOYEE ON INTERVIEWER.EMPLOYEE_ID=EMPLOYEE.EMPLOYEE_ID) JOIN INTERVIEW ON
INTERVIEWER.INTERVIEWER_ID=INTERVIEW.INTERVIEW_ID) JOIN CANDIDATE ON
INTERVIEW.CANDIDATE_ID=CANDIDATE.CANDIDATE_ID WHERE INTERVIEWER.INTERVIEWER_ID=(?);
```

Problem Statement: Query all the interviewer details given a interviewer id.

Query:
```
SELECT INTERVIEWER_ID,INTERVIEWER.EMPLOYEE_ID,EMPLOYEE_NAME,EMPLOYEE_AGE,JOB_TITLE FROM
(INTERVIEWER JOIN EMPLOYEE ON INTERVIEWER.EMPLOYEE_ID=EMPLOYEE.EMPLOYEE_ID) JOIN
JOB_DETAILS ON EMPLOYEE.EMPLOYEE_POSITION=JOB_DETAILS.JOB_ID WHERE INTERVIEWER_ID=(?);
```

Problem Statement:
Query details about all the employees who are at a particular position

Query:
```
SELECT EMPLOYEE_ID,EMPLOYEE_NAME FROM (EMPLOYEE JOIN JOB_DETAILS ON
EMPLOYEE_POSITION=JOB_ID) JOIN JOB_ROLE ON JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE WHERE
JOB_ROLE.JOB_TITLE=(?);
```

Problem Statement:
Query for all the different positions the number of employees that are there.

Query:
```
SELECT JOB_DETAILS.JOB_TITLE,COUNT(*) AS NUM_EMPLOYEES FROM JOB_DETAILS JOIN JOB_ROLE ON
JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE GROUP BY JOB_DETAILS.JOB_TITLE;
```

Application:
A company can use the query above to evaluate the number of new job openings that it needs to announce.

Problem Statement:
Given a EmployeeID, Query all the details about the employee.

Query:
```
SELECT
EMPLOYEE_NAME,EMPLOYEE_AGE,JOB_DETAILS.JOB_ID,JOB_DETAILS.JOB_TITLE,JOB_DESCRIPTION FROM
(EMPLOYEE JOIN JOB_DETAILS ON EMPLOYEE_POSITION=JOB_ID) JOIN JOB_ROLE ON
JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE WHERE EMPLOYEE_ID=(?);
```

Problem Statement:
Query the details of all the candidates with a particular skill.

Condition:
The level of expertise of that candidate in that skill must be above a certain value which is given as input.

Query:
```
SELECT CANDIDATE.CANDIDATE_ID,NAME,LEVEL FROM SKILLS JOIN CANDIDATE ON
CANDIDATE.CANDIDATE_ID=SKILLS.CANDIDATE_ID WHERE SKILLS.SKILL_NAME=(?) AND LEVEL>(?);
```

Problem Statement: Query all the skills of a particular candidate, given the candidate’s candidate_id.

Query:
```
SELECT NAME,SKILL_NAME,LEVEL FROM SKILLS JOIN CANDIDATE ON
CANDIDATE.CANDIDATE_ID=SKILLS.CANDIDATE_ID WHERE CANDIDATE.CANDIDATE_ID=(?);
```

Problem Statement: Given a lower bound and a upper bound for age, query all the candidates who are in this age group.

Query:

```
  SELECT CANDIDATE_ID, NAME FROM CANDIDATE WHERE AGE BETWEEN (?) AND (?);
```

Problem Statement:
Given a candidate id query all the details with respect to that candidate

Query:

```
SELECT NAME, AGE, EDUCATION, EXPERIENCE, STATUS, JOB_TITLE FROM CANDIDATE JOIN
JOB_DETAILS ON CANDIDATE.ROLE=JOB_DETAILS.JOB_ID WHERE CANDIDATE_ID=(?);
```

Problem Statement:
Query the number of Total job applicants for every location.

Query:

```
SELECT LOCATION, COUNT(CANDIDATE_ID) AS NUM_CANDIDATES FROM CANDIDATE JOIN JOB_DETAILS ON
CANDIDATE.ROLE=JOB_DETAILS.JOB_ID GROUP BY LOCATION;
```

Problem Statement:
Query the number of candidates whose status is

- ‘entry received’
- ‘ongoing’
- ‘accepted’
- ‘rejected’

Query:

```
 SELECT STATUS, COUNT(STATUS) AS NUM_CANDIDATES FROM CANDIDATE GROUP BY STATUS;
```

Problem Statement:
Query the total number of available jobs

Query:

```
SELECT COUNT(*) AS NUM_VACANT FROM JOB_DETAILS WHERE FILLED=0;
```

Problem Statement:
Given a location, Query details of all the jobs available

Query:

```
SELECT JOB_ID,JOB_DETAILS.JOB_TITLE,JOB_DESCRIPTION,FILLED FROM JOB_DETAILS JOIN JOB_ROLE
ON JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE WHERE LOCATION=(?);
```

Application:
If a candidate would like to search for jobs specific to a particular location then this Query can be used.

Problem statement:
Given a job id query all the job details corresponding to this job id.

Query:
```
SELECT JOB_DETAILS.JOB_TITLE,JOB_DESCRIPTION,LOCATION,FILLED FROM JOB_DETAILS JOIN
JOB_ROLE ON JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE WHERE JOB_ID=(?);
```

Problem statement:
Query details of all the jobs

Query:
```
SELECT JOB_ID,JOB_DETAILS.JOB_TITLE,JOB_DESCRIPTION,LOCATION,FILLED FROM JOB_DETAILS JOIN
JOB_ROLE ON JOB_DETAILS.JOB_TITLE=JOB_ROLE.JOB_TITLE;
```

Problem statement:
Query the number of locations at which a particular job title exists.

Query:
```
SELECT COUNT(LOCATION) AS NUMBER_OF_LOCATIONS,JOB_TITLE FROM JOB_DETAILS GROUP BY
JOB_TITLE;
```

Problem Statement:
Query the number of jobs at a specific location.

Query:
```
SELECT COUNT(JOB_ID) AS NUMBER_OF_JOBS,LOCATION FROM JOB_DETAILS GROUP BY LOCATION;
```

Conclusion:

A database management system which stores necessary details about the job hiring process of a company was successfully implemented. A large number of complex queries on the implemented database were written. We can summarize the queries written as

- Number of jobs in each location
- Location count for each job title
- All job details
- Job details for job id
- Job details for location
- Number of available jobs
- Status summary of applications
- Number of applicants per location
- Candidate details for candidate id
- Candidates by age group
- Skills of candidate by candidate id
- Candidates with skill above specific level
- Employee details by employee id
- Number of employees at different position
- Employees at particular position
- **Interviewer details by interviewer id**
- **List of candidates interviewed by interviewer**
- **Distribution of question difficulties by interviewer id**
- **Number of candidates at different stages by interviewer**
- **View question details for question id**
- **Number of questions by difficulty**
- **List of questions without explanations**
- **List interviews of candidate**
- **List interviews of interviewer**
- **Scheduled (but not yet conducted) interviews**
- **Distribution of interview results**
- **Filter candidates by average score**
- **List of questions asked in interview by interview id**
- **Distribution of questions asked in interviews**
- **Difficulty of interview**
- **Summary of candidate**
- **Summary statistics for job title**
- **Summary of interviewers**

However there are some limitations to the model implemented above, some of them are:

1) Absence of micro level statistics such as time taken by candidate to solve a particular question.
2) Absence of a voice controlled Query system
3)The model above, almost simulates the hiring process except for the feature of having multiple rounds, which is a prominent feature in interviews taken.

4)Absence of Interviewer remarks in INTERVIEW relation.

5) Absence of a web portal to display new job postings or taking details of new candidates.

6)The queries take too long to execute in case of existence of large quantities of data , therefore the queries must be further optimized.

7)THE MODEL CAN BE USED ONLY BY A SINGLE USER AT A TIME.

8)The model doesn’t allow concurrency, therefore increasing time of execution.