

## **DBMS Project 2:**

# **Human Resource Management System**

| Sr. No | Table of Content               | Page Number |
|--------|--------------------------------|-------------|
| 1      | <b>Abstract</b>                | 2           |
| 2      | <b>Design</b>                  | 2           |
| 2.a    | <b>Introduction</b>            | 2           |
| 2.b    | <b>E/R Diagram</b>             | 3           |
| 2.c    | <b>Database Design Objects</b> | 4           |
| 3      | <b>Implementation</b>          | 10          |
| 3.a    | <b>Database Creation</b>       | 10          |
| 3.b    | <b>Tables creation</b>         | 10          |
| 3.c    | <b>Inserting data</b>          | 17          |
| 3.d    | <b>Retrieving data</b>         | 25          |
| 4      | <b>Testing</b>                 | 35          |
| 4.a    | <b>Views</b>                   | 35          |
| 4.b    | <b>Stored Procedures</b>       | 40          |
| 4.c    | <b>User Defined Functions</b>  | 51          |
| 4.d    | <b>Triggers</b>                | 61          |
| 4.e    | <b>Transactions</b>            | 71          |
| 4.f    | <b>Scripts</b>                 | 79          |
| 4.g    | <b>Business Reports</b>        | 84          |
| 5      | <b>Conclusion</b>              | 91          |

## **1. Abstract:**

This project aims to develop a comprehensive HR management system leveraging relational database management techniques. The system encompasses various aspects of HR operations, including candidate recruitment, evaluation, onboarding, and performance tracking. Through the utilization of SQL queries, data analysis, and visualization, the system facilitates informed decision-making and efficient management of human resources. The project database, named HR\_DB, comprises several interconnected tables representing entities such as candidates, job openings, evaluations, interviews, and reimbursements. Data has been inserted into these tables to simulate real-world scenarios and demonstrate the functionality of the system.

Key features of the system include candidate tracking, evaluation score distribution analysis, maximum test score identification, and status monitoring of job openings. These features enable HR professionals to streamline recruitment processes, assess candidate suitability, and ensure compliance with organizational standards. Through this project, we aim to provide a practical solution for HR management, empowering organizations to optimize their hiring processes, enhance candidate experience, and foster a productive workforce.

## **2. Design:**

### **a. Introduction:**

In today's dynamic business landscape, effective human resource management is vital for the success of any organization. The recruitment, evaluation, and retention of top talent play a crucial role in driving innovation, productivity, and organizational growth. With the increasing complexity of HR processes and the abundance of candidate data, there arises a need for robust HR management systems that can streamline operations and provide actionable insights. This project endeavors to address this need by developing a comprehensive HR management system using relational database management techniques. The system aims to automate and optimize various HR functions, ranging from candidate recruitment to performance evaluation and onboarding. By leveraging SQL queries, data analysis, and visualization, the system enables HR professionals to make informed decisions, track candidate progress, and ensure compliance with organizational policies.

The project database, HR\_DB, serves as the backbone of the system, housing interconnected tables that represent entities such as candidates, job openings, evaluations, interviews, and reimbursements. Through the insertion of dummy data into these tables, the system demonstrates its functionality in managing real-world HR scenarios. By developing this HR management system, we aim to provide organizations with a powerful tool to optimize their HR operations, enhance candidate experience, and ultimately contribute to the achievement of strategic business objectives.

### b. E/R Diagram:

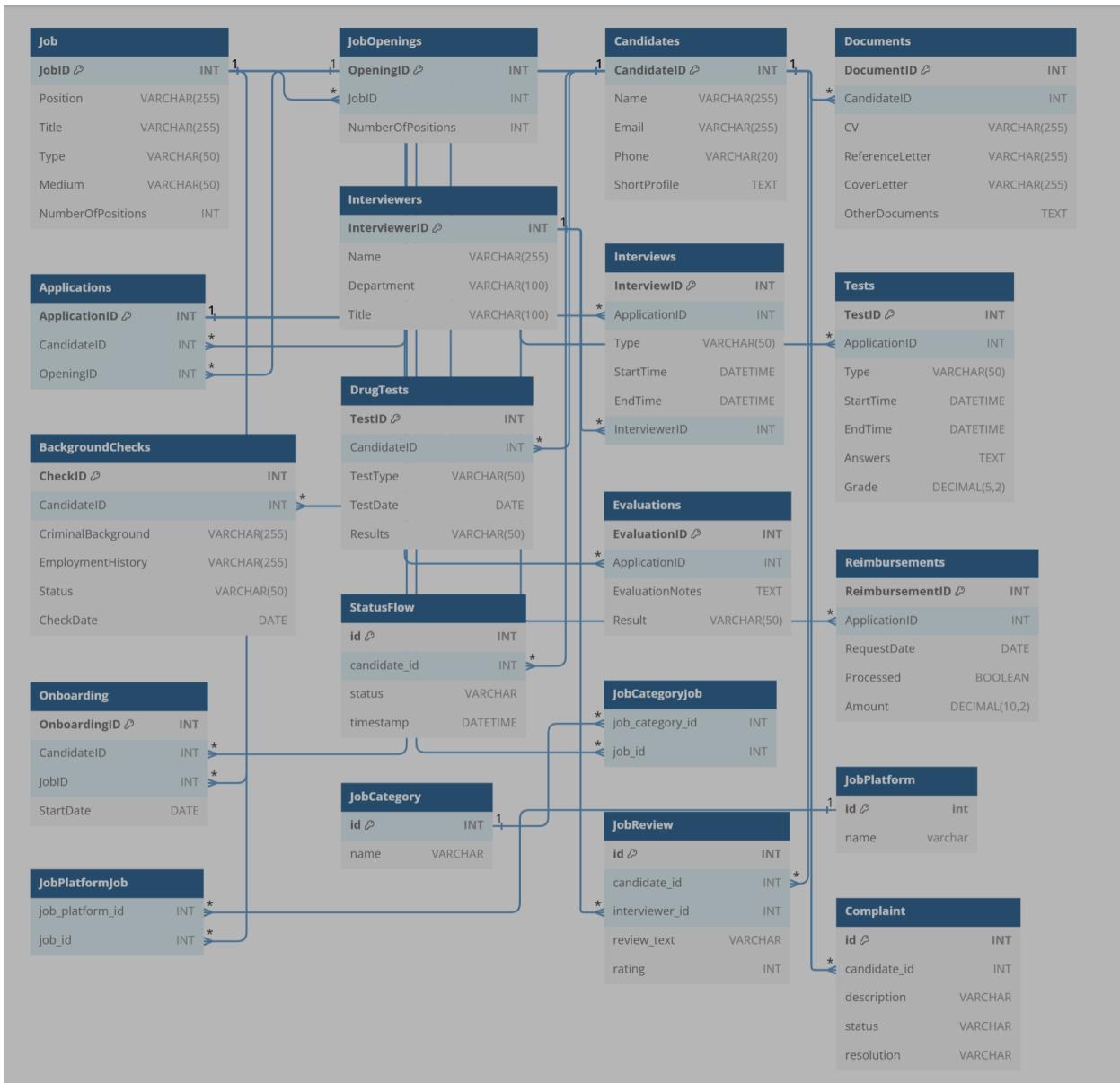


Fig 1: E/R Diagram for HR Database

**c. Database Design Objects:**

➤ Job:

JobID INT PRIMARY KEY

Position VARCHAR(255)

Title VARCHAR(255)

Type VARCHAR(50)

Medium VARCHAR(50)

NumberOfPositions INT

➤ JobOpenings:

OpeningID INT PRIMARY KEY

JobID INT FOREIGN KEY (Job.JobID)

NumberOfPositions INT

➤ Candidates:

CandidateID INT PRIMARY KEY

Name VARCHAR(255)

Email VARCHAR(255)

Phone VARCHAR(20)

ShortProfile TEXT

➤ Documents:

DocumentID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

CV VARCHAR(255)

ReferenceLetter VARCHAR(255)

CoverLetter VARCHAR(255)

OtherDocuments TEXT

Jainam Rajendra Shah  
SUID: 785948053

➤ Applications:

ApplicationID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

OpeningID INT FOREIGN KEY (JobOpenings.OpeningID)

➤ Interviewers:

InterviewerID INT PRIMARY KEY

Name VARCHAR(255)

Department VARCHAR(100)

Title VARCHAR(100)

➤ Interviews:

InterviewID INT PRIMARY KEY

ApplicationID INT FOREIGN KEY (Applications.ApplicationID)

Type VARCHAR(50)

StartTime DATETIME

EndTime DATETIME

InterviewerID INT FOREIGN KEY (Interviewers.InterviewerID)

➤ Tests:

TestID INT PRIMARY KEY

ApplicationID INT FOREIGN KEY (Applications.ApplicationID)

Type VARCHAR(50)

StartTime DATETIME

EndTime DATETIME

Answers TEXT

Grade DECIMAL(5,2)

Jainam Rajendra Shah  
SUID: 785948053

➤ BackgroundChecks:

CheckID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

CriminalBackground VARCHAR(255)

EmploymentHistory VARCHAR(255)

Status VARCHAR(50)

CheckDate DATE

➤ DrugTests:

]TestID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

TestType VARCHAR(50)

TestDate DATE

Results VARCHAR(50)

➤ Evaluations:

EvaluationID INT PRIMARY KEY

ApplicationID INT FOREIGN KEY (Applications.ApplicationID)

EvaluationNotes TEXT

Result VARCHAR(50)

➤ Reimbursements:

ReimbursementID INT PRIMARY KEY

ApplicationID INT FOREIGN KEY (Applications.ApplicationID)

RequestDate DATE

Processed BOOLEAN

Amount DECIMAL(10,2)

Jainam Rajendra Shah  
SUID: 785948053

➤ Onboarding:

OnboardingID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

JobID INT FOREIGN KEY (Job.JobID)

StartDate DATE

➤ StatusFlow:

ID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

Status VARCHAR

Timestamp DATETIME

➤ JobCategoryJob:

JobCategoryID INT FOREIGN KEY (JobCategory.ID)

JobID INT FOREIGN KEY (Job.JobID)

➤ JobPlatform:

ID INT PRIMARY KEY

Name VARCHAR

➤ JobPlatformJob:

JobPlatformID INT FOREIGN KEY (JobPlatform.ID)

JobID INT FOREIGN KEY (Job.JobID)

➤ JobCategory:

ID INT PRIMARY KEY

Name VARCHAR

Jainam Rajendra Shah  
SUID: 785948053

➤ JobReview:

ID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

InterviewerID INT FOREIGN KEY (Interviewers.InterviewerID)

ReviewText VARCHAR

Rating INT

➤ Complaint:

ID INT PRIMARY KEY

CandidateID INT FOREIGN KEY (Candidates.CandidateID)

Description VARCHAR

Status VARCHAR

Resolution VARCHAR

### 3. Implementation:

#### a. Database Creation:

```
IF DB_ID('HR_DB') IS NOT NULL
DROP DATABASE HR_DB
GO
```

```
CREATE DATABASE HR_DB
GO
```

```
USE HR_DB;
```

```
/* Job */

-- Drop the existing 'Job' table
DROP TABLE IF EXISTS Job;
-- Recreate the 'Job' table based on your DBML code
CREATE TABLE Job (
    JobID INT PRIMARY KEY,
    Position VARCHAR(255),
    Title VARCHAR(255),
    Type VARCHAR(50),
    Medium VARCHAR(50),
    NumberOfPositions INT
);
```

```
/* Job Openings */

-- Drop the existing 'JobOpenings' table
DROP TABLE IF EXISTS JobOpenings;
-- Recreate the 'JobOpenings' table based on your DBML code
CREATE TABLE JobOpenings (
    OpeningID INT PRIMARY KEY,
    JobID INT,
    NumberOfPositions INT,
    FOREIGN KEY (JobID) REFERENCES Job(JobID)
);
```

```
/* Candidates */
```

```
-- Drop the existing 'Candidates' table
DROP TABLE IF EXISTS Candidates;
-- Recreate the 'Candidates' table based on your DBML code
CREATE TABLE Candidates (
    CandidateID INT PRIMARY KEY,
    Name VARCHAR(255),
    Email VARCHAR(255),
    Phone VARCHAR(20),
    ShortProfile TEXT
);
```

```
/* Documents */
```

```
-- Drop the existing 'Documents' table
DROP TABLE IF EXISTS Documents;
-- Recreate the 'Documents' table based on your DBML code
CREATE TABLE Documents (
    DocumentID INT PRIMARY KEY,
    CandidateID INT,
    CV VARCHAR(255),
    ReferenceLetter VARCHAR(255),
    CoverLetter VARCHAR(255),
    OtherDocuments TEXT,
    FOREIGN KEY (CandidateID) REFERENCES Candidates(CandidateID)
);
```

```
/* Applications */
```

```
-- Drop the existing 'Applications' table
```

Jainam Rajendra Shah  
SUID: 785948053

```
DROP TABLE IF EXISTS Applications;
-- Recreate the 'Applications' table based on your DBML code
CREATE TABLE Applications (
    ApplicationID INT PRIMARY KEY,
    CandidateID INT,
    OpeningID INT,
    FOREIGN KEY (CandidateID) REFERENCES Candidates(CandidateID),
    FOREIGN KEY (OpeningID) REFERENCES JobOpenings(OpeningID)
);
```

```
/* Interviews */
```

```
-- Drop the existing 'Interviewers' table
DROP TABLE IF EXISTS Interviewers;
-- Recreate the 'Interviewers' table based on your DBML code
CREATE TABLE Interviewers (
    InterviewerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Department VARCHAR(100),
    Title VARCHAR(100)
);
```

```
/* Interviews */
```

```
-- Drop the existing 'Interviews' table
DROP TABLE IF EXISTS Interviews;
-- Recreate the 'Interviews' table based on your DBML code
CREATE TABLE Interviews (
    InterviewID INT PRIMARY KEY,
    ApplicationID INT,
    Type VARCHAR(50),
    StartTime DATETIME,
    EndTime DATETIME,
    InterviewerID INT,
    FOREIGN KEY (ApplicationID) REFERENCES Applications(ApplicationID),
    FOREIGN KEY (InterviewerID) REFERENCES Interviewers(InterviewerID)
);
```

Jainam Rajendra Shah  
SUID: 785948053

/\* Test \*/

```
-- Drop the existing 'Tests' table
DROP TABLE IF EXISTS Tests;
-- Recreate the 'Tests' table based on your DBML code
CREATE TABLE Tests (
    TestID INT PRIMARY KEY,
    ApplicationID INT,
    Type VARCHAR(50),
    StartTime DATETIME,
    EndTime DATETIME,
    Answers TEXT,
    Grade DECIMAL(5,2),
    FOREIGN KEY (ApplicationID) REFERENCES Applications(ApplicationID)
);
```

/\* Background Checks \*/

```
-- Drop the existing 'BackgroundChecks' table
DROP TABLE IF EXISTS BackgroundChecks;
-- Recreate the 'BackgroundChecks' table based on your DBML code
CREATE TABLE BackgroundChecks (
    CheckID INT PRIMARY KEY,
    CandidateID INT,
    CriminalBackground VARCHAR(255),
    EmploymentHistory VARCHAR(255),
    Status VARCHAR(50),
    CheckDate DATE,
    FOREIGN KEY (CandidateID) REFERENCES Candidates(CandidateID)
);
```

/\* Drug Tests \*/

```
-- Drop the existing 'DrugTests' table
DROP TABLE IF EXISTS DrugTests;
-- Recreate the 'DrugTests' table based on your DBML code
CREATE TABLE DrugTests (
    TestID INT PRIMARY KEY,
    CandidateID INT,
```

Jainam Rajendra Shah  
SUID: 785948053

```
TestType VARCHAR(50),  
TestDate DATE,  
Results VARCHAR(50),  
FOREIGN KEY (CandidateID) REFERENCES Candidates(CandidateID)  
);
```

```
/* Evaluation */
```

```
-- Drop the existing 'Evaluations' table  
DROP TABLE IF EXISTS Evaluations;  
-- Recreate the 'Evaluations' table based on your DBML code  
CREATE TABLE Evaluations (  
    EvaluationID INT PRIMARY KEY,  
    ApplicationID INT,  
    EvaluationNotes TEXT,  
    Result VARCHAR(50),  
    FOREIGN KEY (ApplicationID) REFERENCES Applications(ApplicationID)  
);
```

```
/* Reimbursements */
```

```
-- Drop the existing 'Reimbursements' table  
DROP TABLE IF EXISTS Reimbursements;  
-- Recreate the 'Reimbursements' table based on your DBML code  
CREATE TABLE Reimbursements (  
    ReimbursementID INT PRIMARY KEY,  
    ApplicationID INT,  
    RequestDate DATE,  
    Processed BIT,  
    Amount DECIMAL(10,2),  
    FOREIGN KEY (ApplicationID) REFERENCES Applications(ApplicationID)  
);
```

```
/* Onboarding */
```

```
-- Drop the existing 'Onboarding' table  
DROP TABLE IF EXISTS Onboarding;
```

Jainam Rajendra Shah  
SUID: 785948053

```
-- Recreate the 'Onboarding' table based on your DBML code
CREATE TABLE Onboarding (
    OnboardingID INT PRIMARY KEY,
    CandidateID INT,
    JobID INT,
    StartDate DATE,
    FOREIGN KEY (CandidateID) REFERENCES Candidates(CandidateID),
    FOREIGN KEY (JobID) REFERENCES Job(JobID)
);
```

```
/* StatusFlow */
```

```
-- Drop the existing 'StatusFlow' table
DROP TABLE IF EXISTS StatusFlow;
-- Recreate the 'StatusFlow' table based on your DBML code
CREATE TABLE StatusFlow (
    id INT PRIMARY KEY,
    candidate_id INT,
    status VARCHAR(255),
    timestamp DATETIME,
    FOREIGN KEY (candidate_id) REFERENCES Candidates(CandidateID)
);
```

```
/* JobCategory */
```

```
-- Drop the existing 'JobCategory' table
DROP TABLE IF EXISTS JobCategory;
-- Recreate the 'JobCategory' table based on your DBML code
CREATE TABLE JobCategory (
    id INT PRIMARY KEY,
    name VARCHAR(255)
);
```

```
/* JobCategoryJob */
```

```
-- Drop the existing 'JobCategoryJob' table
DROP TABLE IF EXISTS JobCategoryJob;
-- Recreate the 'JobCategoryJob' table based on your DBML code
```

Jainam Rajendra Shah  
SUID: 785948053

```
CREATE TABLE JobCategoryJob (
    job_category_id INT,
    job_id INT,
    FOREIGN KEY (job_category_id) REFERENCES JobCategory(id),
    FOREIGN KEY (job_id) REFERENCES Job(JobID)
);
```

```
/*  JobPlatform  */
```

```
-- Drop the existing 'JobPlatform' table
DROP TABLE IF EXISTS JobPlatform;
-- Recreate the 'JobPlatform' table based on your DBML code
CREATE TABLE JobPlatform (
    id INT PRIMARY KEY,
    name VARCHAR(255)
);
```

```
/*  JobPlatformJob  */
```

```
-- Drop the existing 'JobPlatformJob' table
DROP TABLE IF EXISTS JobPlatformJob;
-- Recreate the 'JobPlatformJob' table based on your DBML code
CREATE TABLE JobPlatformJob (
    job_platform_id INT,
    job_id INT,
    FOREIGN KEY (job_platform_id) REFERENCES JobPlatform(id),
    FOREIGN KEY (job_id) REFERENCES Job(JobID)
);
```

```
/*  JobReview  */
```

```
-- Drop the existing 'JobReview' table
DROP TABLE IF EXISTS JobReview;
-- Recreate the 'JobReview' table based on your DBML code
CREATE TABLE JobReview (
    id INT PRIMARY KEY,
    candidate_id INT,
    interviewer_id INT,
    review_text VARCHAR(255),
    rating INT,
    FOREIGN KEY (candidate_id) REFERENCES Candidates(CandidateID),
    FOREIGN KEY (interviewer_id) REFERENCES Interviewers(InterviewerID)
```

Jainam Rajendra Shah  
SUID: 785948053

);

```
/* Complaint */  
  
-- Drop the existing 'Complaint' table  
DROP TABLE IF EXISTS Complaint;  
-- Recreate the 'Complaint' table based on your DBML code  
CREATE TABLE Complaint (  
    id INT PRIMARY KEY,  
    candidate_id INT,  
    description VARCHAR(255),  
    status VARCHAR(255),  
    resolution VARCHAR(255),  
    FOREIGN KEY (candidate_id) REFERENCES Candidates(CandidateID)  
);
```

b. Inserting Data in Tables:

```
USE HR_DB;
```

```
-- Insert dummy data into the Candidates table  
INSERT INTO Candidates (CandidateID, Name, Email, Phone, ShortProfile) VALUES  
(1, 'Jainam Shah', 'jainam@gmail.com', '1234567890', 'Data Scientist'),  
(2, 'Jackie Chan', 'jackie@hotmail.com', '2345678901', 'Data Scientist'),  
(3, 'Gal Gadot', 'gal@syr.edu', '3456789012', 'Java Developer'),  
(4, 'Margot Robbie', 'margot@gmail.com', '4567890123', 'Software Engineer'),  
(5, 'Dikshita Patel', 'dikshita@hotmail.com', '5678901234', 'Frontend Developer'),  
(6, 'Janvi Saddi', 'janvi@syr.edu', '6789012345', 'Frontend Developer'),  
(7, 'Bhavya Shah', 'bhavya@gmail.com', '7890123456', 'Data Analyst'),  
(8, 'Yash Solanki', 'yash@hotmail.com', '8901234567', 'Data Analyst'),  
(9, 'Heet Chedda', 'heet@syr.edu', '9012345678', 'Java Developer'),  
(10, 'Sahil Shah', 'sahil@gmail.com', '0123456789', 'Data Analyst');
```

```
-- Insert dummy data into the Documents table
```

Jainam Rajendra Shah  
SUID: 785948053

```
INSERT INTO Documents (DocumentID, CandidateID, CV, ReferenceLetter, CoverLetter, OtherDocuments) VALUES
(1, 1, 'jainam_cv.pdf', 'jainam_ref_letter.pdf', 'jainam_cover_letter.pdf', 'jainam_other_docs.pdf'),
(2, 2, 'jackie_cv.pdf', 'jackie_ref_letter.pdf', 'jackie_cover_letter.pdf', 'jackie_other_docs.pdf'),
(3, 3, 'gal_cv.pdf', 'gal_ref_letter.pdf', 'gal_cover_letter.pdf', NULL),
(4, 4, 'margot_cv.pdf', 'margot_ref_letter.pdf', 'margot_cover_letter.pdf', NULL),
(5, 5, 'dikshita_cv.pdf', 'dikshita_ref_letter.pdf', 'dikshita_cover_letter.pdf',
'dikshita_other_docs.pdf'),
(6, 6, 'janvi_cv.pdf', 'janvi_ref_letter.pdf', 'janvi_cover_letter.pdf', NULL),
(7, 7, 'bhavya_cv.pdf', 'bhavya_ref_letter.pdf', 'bhavya_cover_letter.pdf', NULL),
(8, 8, 'yash_cv.pdf', 'yash_ref_letter.pdf', 'yash_cover_letter.pdf', 'yash_other_docs.pdf'),
(9, 9, 'heet_cv.pdf', 'heet_ref_letter.pdf', 'heet_cover_letter.pdf', NULL),
(10, 10, 'sahil_cv.pdf', 'sahil_ref_letter.pdf', 'sahil_cover_letter.pdf', 'sahil_other_docs.pdf');
```

-- Insert dummy data into the JobPlatform table

```
INSERT INTO JobPlatform (id, name) VALUES
(1, 'LinkedIn'),
(2, 'Glassdoor'),
(3, 'Handshake'),
(4, 'Referral');
```

-- Insert dummy data into the Job table

```
INSERT INTO Job (JobID, Position, Title, Type, Medium, NumberOfPositions) VALUES
(1, 'Data Scientist', 'Data Scientist and AI Developer', 'Fulltime', 'Onsite', 3),
(2, 'Data Scientist', 'Data Scientist', 'Fulltime', 'Hybrid', 2),
(3, 'Software Engineer', 'Software Engineer Position', 'Fulltime', 'Remote', 4),
(4, 'Java Developer', 'Java Developer Full Stack', 'Parttime', 'Onsite', 1),
(5, 'Business Analyst', 'Business Analyst and Data Analyst', 'Internship', 'Hybrid', 2),
(6, 'Frontend Developer', 'Frontend Developer(Full Stack)', 'Co-Op', 'Remote', 3),
(7, 'Software Engineer', 'Software Engineer Designer', 'Fulltime', 'Onsite', 2),
(8, 'Marketing Specialist', 'Marketing Specialist', 'Parttime', 'Hybrid', 1),
(9, 'Financial Analyst', 'Financial Analyst Position', 'Fulltime', 'Remote', 5),
(10, 'Human Resources Coordinator', 'HR Coordinator Position', 'Internship', 'Hybrid', 2);
```

-- Insert dummy data into the JobOpenings table

```
INSERT INTO JobOpenings (OpeningID, JobID, NumberOfPositions) VALUES
```

Jainam Rajendra Shah  
SUID: 785948053

```
(1, 1, 3),  
(2, 2, 2),  
(3, 3, 4),  
(4, 4, 1),  
(5, 5, 2),  
(6, 6, 3),  
(7, 7, 2),  
(8, 8, 1),  
(9, 9, 5),  
(10, 10, 2);
```

-- Insert dummy data into the StatusFlow table

```
INSERT INTO StatusFlow (id, candidate_id, status, timestamp) VALUES  
(1, 1, 'Under Review', '2024-04-18 08:00:00'),  
(2, 2, 'Negotiating', '2024-04-23 08:30:00'),  
(3, 3, 'Accepted', '2024-03-30 09:00:00'),  
(4, 4, 'Rejected', '2024-01-30 09:30:00'),  
(5, 5, 'Rejected', '2023-04-30 10:00:00'),  
(6, 6, 'Application Submitted', '2024-04-18 10:30:00'),  
(7, 7, 'Under Review', '2024-04-14 11:00:00'),  
(8, 8, 'Application Submitted', '2024-04-19 11:30:00'),  
(9, 9, 'Under Review', '2024-04-11 12:00:00'),  
(10, 10, 'Accepted', '2024-04-09 12:30:00');
```

-- Insert dummy data into the Applications table

```
INSERT INTO Applications (ApplicationID, CandidateID, OpeningID) VALUES  
(1, 1, 1),  
(2, 2, 2),  
(3, 3, 3),  
(4, 4, 4),  
(5, 5, 5),  
(6, 6, 6),  
(7, 7, 7),  
(8, 8, 8),  
(9, 9, 9),  
(10, 10, 10);
```

-- Insert dummy data into the Tests table

```
INSERT INTO Tests (TestID, ApplicationID, Type, StartTime, EndTime, Answers, Grade)
VALUES
(1, 1, 'OA', '2024-04-18 09:00:00', '2024-04-18 10:00:00', 'A, B, C, D, A', 85.5),
(2, 2, 'Behavioural MCQ', '2024-04-23 10:00:00', '2024-04-23 11:00:00', 'Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree', 70.0),
(3, 3, 'Aptitude', '2024-03-30 09:00:00', '2024-03-30 10:00:00', '25, 36, 42, 55, 63', 90.5),
(4, 4, 'OA', '2024-01-30 09:30:00', '2024-01-30 10:30:00', 'A, C, D, D, B', 78.0),
(5, 5, 'Behavioural MCQ', '2023-04-30 10:00:00', '2023-04-30 11:00:00', 'Disagree, Strongly Agree, Neutral, Agree, Disagree', 65.5),
(6, 6, 'Aptitude', '2024-04-18 10:30:00', '2024-04-18 11:30:00', '30, 42, 56, 68, 75', 82.0),
(7, 7, 'OA', '2024-04-14 11:00:00', '2024-04-14 12:00:00', 'C, A, B, D, C', 88.5),
(8, 8, 'Behavioural MCQ', '2024-04-19 11:30:00', '2024-04-19 12:30:00', 'Neutral, Agree, Disagree, Strongly Agree, Strongly Disagree', 72.0),
(9, 9, 'Aptitude', '2024-04-11 12:00:00', '2024-04-11 13:00:00', '40, 51, 64, 78, 85', 95.0),
(10, 10, 'OA', '2024-04-09 12:30:00', '2024-04-09 13:30:00', 'B, B, D, A, C', 80.5);
```

-- Insert dummy data into the Interviewers table

```
INSERT INTO Interviewers (InterviewerID, Name, Department, Title) VALUES
(1, 'John Doe', 'Human Resources', 'HR Manager'),
(2, 'Jane Smith', 'Engineering', 'Senior Software Engineer'),
(3, 'David Martinez', 'Operations', 'Operations Manager'),
(4, 'Daniel Taylor', 'Research and Development', 'Data Science Manager'),
(5, 'Sophia Anderson', 'Product Management', 'Product Manager');
```

-- Insert dummy data into the Interviews table

```
INSERT INTO Interviews (InterviewID, ApplicationID, Type, StartTime, EndTime, InterviewerID) VALUES
(1, 1, 'Onsite', '2024-04-20 09:00:00', '2024-04-20 10:45:00', 1),
(2, 2, 'Online', '2024-04-21 10:00:00', '2024-04-21 11:30:00', 2),
(3, 3, 'Onsite', '2024-04-22 11:00:00', '2024-04-22 12:00:00', 1),
(4, 4, 'Online', '2024-04-23 12:00:00', '2024-04-23 12:20:00', 3),
(5, 5, 'Onsite', '2024-04-24 13:00:00', '2024-04-24 14:00:00', 4),
(6, 6, 'Online', '2024-04-25 14:00:00', '2024-04-25 15:00:00', 5),
(7, 7, 'Onsite', '2024-04-26 15:00:00', '2024-04-26 16:00:00', 2),
(8, 8, 'Online', '2024-04-27 16:00:00', '2024-04-27 17:00:00', 1),
(9, 9, 'Onsite', '2024-04-28 17:00:00', '2024-04-28 18:15:00', 3),
```

(10, 10, 'Online', '2024-04-29 18:00:00', '2024-04-29 19:00:00', 5);

**INSERT INTO** JobReview (id, candidate\_id, interviewer\_id, review\_text, rating) **VALUES**  
(1, 1, 1, 'Good logical skills. Moving onto the next round.', 4),  
(2, 2, 2, 'Great interview! Well-prepared and insightful answers.', 5),  
(3, 3, 1, 'The interviewer asked irrelevant questions.', 2),  
(4, 4, 3, 'The interviewee performed very poorly.', 1),  
(5, 5, 4, 'The interviewee needs improvement in communication skills.', 2),  
(6, 6, 5, 'Failed to answer critical questions.', 1),  
(7, 7, 2, 'Passed the interview with flying colors.', 5),  
(8, 8, 1, 'Good logical skills. Moving onto the next round.', 4),  
(9, 9, 3, 'Need Improvement in technical skills.', 2),  
(10, 10, 5, 'Irrelevant questions were asked during the interview.', 3);

**INSERT INTO** Evaluations (EvaluationID, ApplicationID, EvaluationNotes, Result) **VALUES**  
(1, 1, 'Passed the interview. Moving onto the next round.', 'Pass'),  
(2, 2, 'Failed the interview. Needs improvement in technical skills.', 'Fail'),  
(3, 3, 'Can do better in the next round. Improve communication skills.', 'Pending'),  
(4, 4, 'Passed the interview with excellent performance.', 'Pass'),  
(5, 5, 'Moving onto the next round. Good problem-solving skills.', 'Pass'),  
(6, 6, 'Failed the interview due to lack of experience.', 'Fail'),  
(7, 7, 'Needs improvement in coding skills. Waiting on other candidates interview.', 'Pending'),  
(8, 8, 'Passed the interview with flying colors.', 'Pass'),  
(9, 9, 'Moving onto the next round. Great presentation skills.', 'Pass'),  
(10, 10, 'Failed to meet expectations. Pending further review.', 'Pending');

**INSERT INTO** Reimbursements (ReimbursementID, ApplicationID, RequestDate, Processed, Amount) **VALUES**  
(1, 1, '2024-04-18', 1, 200.00),  
(2, 2, '2024-04-23', 1, 150.00),  
(3, 3, '2024-03-30', 0, 0.00),  
(4, 4, '2024-01-30', 1, 100.00),  
(5, 5, '2023-04-30', 0, 0.00),  
(6, 6, '2024-04-18', 1, 250.00),  
(7, 7, '2024-04-14', 1, 180.00),  
(8, 8, '2024-04-19', 0, 0.00),  
(9, 9, '2024-04-11', 1, 300.00),  
(10, 10, '2024-04-09', 1, 220.00);

```
INSERT INTO Onboarding (OnboardingID, CandidateID, JobID, StartDate)
VALUES
(1, 1, 1, '2024-06-10'),
(2, 2, 2, '2024-06-10'),
(3, 3, 3, '2024-06-07'),
(4, 4, 4, '2024-07-03'),
(5, 5, 5, '2024-07-10'),
(6, 6, 6, '2024-08-21'),
(7, 7, 7, '2024-06-30'),
(8, 8, 8, '2024-05-24'),
(9, 9, 9, '2024-07-17'),
(10, 10, 10, '2024-06-10');
```

```
INSERT INTO Complaint (id, candidate_id, description, status, resolution)
VALUES
(1, 1, 'Cannot Start Behavioural Questions', 'Open', 'Investigating'),
(2, 2, 'Link not working', 'Open', 'Resolved'),
(3, 3, 'Poor Connection hence could not give interview', 'Open', 'Investigating'),
(4, 4, NULL, 'Open', 'No action required'),
(5, 5, NULL, 'Open', 'No action required'),
(6, 6, 'Link not working', 'Open', 'Investigating'),
(7, 7, 'Cannot Start Behavioural Questions', 'Open', 'Investigating'),
(8, 8, 'Poor Connection hence could not give interview', 'Open', 'Investigating'),
(9, 9, NULL, 'Open', 'No action required'),
(10, 10, NULL, 'Open', 'No action required');
```

-- Insert dummy data into the BackgroundChecks table

```
INSERT INTO BackgroundChecks (CheckID, CandidateID, CriminalBackground,
EmploymentHistory, Status, CheckDate) VALUES
(1, 1, 'Clean', 'Stable', 'Clear', '2024-04-18'),
(2, 2, 'Minor Offense', 'Unstable', 'Pending', '2024-04-23'),
(3, 3, 'Clear', 'Stable', 'Clear', '2024-04-30'),
(4, 4, 'Clean', 'Stable', 'Clear', '2024-04-30'),
(5, 5, 'Clean', 'Stable', 'Clear', '2024-04-30'),
(6, 6, 'Clear', 'Unstable', 'Clear', '2024-04-18'),
(7, 7, 'Clear', 'Stable', 'Clear', '2024-04-14'),
(8, 8, 'Minor Offense', 'Stable', 'Pending', '2024-04-19'),
(9, 9, 'Clear', 'Stable', 'Clear', '2024-04-11'),
(10, 10, 'Clean', 'Stable', 'Clear', '2024-04-09');
```

-- Insert dummy data into the DrugTests table

INSERT INTO DrugTests (TestID, CandidateID, TestType, TestDate, Results) VALUES

-- Candidate 1

(1, 1, 'Urine Test', '2024-04-18', 'Negative'),  
(2, 1, 'Blood Test', '2024-04-19', 'Negative'),  
(3, 1, 'Saliva Test', '2024-04-20', 'Negative'),

-- Candidate 2

(4, 2, 'Urine Test', '2024-04-21', 'Negative'),  
(5, 2, 'Blood Test', '2024-04-22', 'Positive'),  
(6, 2, 'Saliva Test', '2024-04-23', 'Negative'),

-- Candidate 3

(7, 3, 'Urine Test', '2024-04-24', 'Negative'),  
(8, 3, 'Blood Test', '2024-04-25', 'Negative'),  
(9, 3, 'Saliva Test', '2024-04-26', 'Positive'),

-- Candidate 4

(10, 4, 'Urine Test', '2024-04-27', 'Negative'),  
(11, 4, 'Blood Test', '2024-04-28', 'Negative'),  
(12, 4, 'Saliva Test', '2024-04-29', 'Negative'),

-- Candidate 5

(13, 5, 'Urine Test', '2024-04-30', 'Negative'),  
(14, 5, 'Blood Test', '2024-05-01', 'Negative'),  
(15, 5, 'Saliva Test', '2024-05-02', 'Negative'),

-- Candidate 6

(16, 6, 'Urine Test', '2024-05-03', 'Negative'),  
(17, 6, 'Blood Test', '2024-05-04', 'Negative'),  
(18, 6, 'Saliva Test', '2024-05-05', 'Negative'),

-- Candidate 7

(19, 7, 'Urine Test', '2024-05-06', 'Negative'),  
(20, 7, 'Blood Test', '2024-05-07', 'Negative'),  
(21, 7, 'Saliva Test', '2024-05-08', 'Negative'),

-- Candidate 8

(22, 8, 'Urine Test', '2024-05-09', 'Positive'),  
(23, 8, 'Blood Test', '2024-05-10', 'Positive'),  
(24, 8, 'Saliva Test', '2024-05-11', 'Negative'),

-- Candidate 9

Jainam Rajendra Shah  
SUID: 785948053

```
(25, 9, 'Urine Test', '2024-05-12', 'Positive'),  
(26, 9, 'Blood Test', '2024-05-13', 'Positive'),  
(27, 9, 'Saliva Test', '2024-05-14', 'Positive'),
```

-- Candidate 10

```
(28, 10, 'Urine Test', '2024-05-15', 'Negative'),  
(29, 10, 'Blood Test', '2024-05-16', 'Negative'),  
(30, 10, 'Saliva Test', '2024-05-17', 'Negative');
```

-- Insert dummy data into the JobCategory table

```
INSERT INTO JobCategory (id, name) VALUES  
(1, 'Data Science'),  
(2, 'Software Development'),  
(3, 'Data Analyst'),  
(4, 'Java Developer'),  
(5, 'Business Analysis'),  
(6, 'Human Resource');
```

-- Insert dummy data into the JobPlatformJob table

```
INSERT INTO JobPlatformJob (job_platform_id, job_id) VALUES  
(1, 1),  
(1, 2),  
(3, 3),  
(4, 4),  
(2, 5),  
(2, 6),  
(4, 7),  
(1, 8),  
(1, 9),  
(3, 10);
```

-- Insert dummy data into the JobCategoryJob table

```
INSERT INTO JobCategoryJob (job_category_id, job_id) VALUES  
(1, 1),  
(1, 2),  
(2, 3),  
(4, 4),  
(5, 5),  
(2, 6),
```

Jainam Rajendra Shah  
SUID: 785948053

(2, 7),  
(5, 8),  
(5, 9),  
(6, 10);

c. Retrieving Data from Tables:

```
-- Display data from the Candidates table
SELECT * FROM Candidates;

-- Display data from the Documents table
SELECT * FROM Documents;

-- Display data from the JobPlatform table
SELECT * FROM JobPlatform;
```

|    | CandidateID | Name           | Email                | Phone      | ShortProfile       |
|----|-------------|----------------|----------------------|------------|--------------------|
| 1  | 1           | Jainam Shah    | jainam@gmail.com     | 1234567890 | Data Scientist     |
| 2  | 2           | Jackie Chan    | jackie@hotmail.com   | 2345678901 | Data Scientist     |
| 3  | 3           | Gal Gadot      | gal@syr.edu          | 3456789012 | Java Developer     |
| 4  | 4           | Margot Robbie  | margot@gmail.com     | 4567890123 | Software Engineer  |
| 5  | 5           | Dikshita Patel | dikshita@hotmail.com | 5678901234 | Frontend Developer |
| 6  | 6           | Janvi Saddi    | janvi@syr.edu        | 6789012345 | Frontend Developer |
| 7  | 7           | Bhavya Shah    | bhavya@gmail.com     | 7890123456 | Data Analyst       |
| 8  | 8           | Yash Solanki   | yash@hotmail.com     | 8901234567 | Data Analyst       |
| 9  | 9           | Heet Chedda    | heet@syr.edu         | 9012345678 | Java Developer     |
| 10 | 10          | Sahil Shah     | sahil@gmail.com      | 0123456789 | Data Analyst       |

|    | DocumentID | CandidateID | CV              | ReferenceLetter         | CoverLetter               | OtherDocuments          |
|----|------------|-------------|-----------------|-------------------------|---------------------------|-------------------------|
| 1  | 1          | 1           | jainam_cv.pdf   | jainam_ref_letter.pdf   | jainam_cover_letter.pdf   | jainam_other_docs.pdf   |
| 2  | 2          | 2           | jackie_cv.pdf   | jackie_ref_letter.pdf   | jackie_cover_letter.pdf   | jackie_other_docs.pdf   |
| 3  | 3          | 3           | gal_cv.pdf      | gal_ref_letter.pdf      | gal_cover_letter.pdf      | NULL                    |
| 4  | 4          | 4           | margot_cv.pdf   | margot_ref_letter.pdf   | margot_cover_letter.pdf   | NULL                    |
| 5  | 5          | 5           | dikshita_cv.pdf | dikshita_ref_letter.pdf | dikshita_cover_letter.pdf | dikshita_other_docs.pdf |
| 6  | 6          | 6           | janvi_cv.pdf    | janvi_ref_letter.pdf    | janvi_cover_letter.pdf    | NULL                    |
| 7  | 7          | 7           | bhavya_cv.pdf   | bhavya_ref_letter.pdf   | bhavya_cover_letter.pdf   | NULL                    |
| 8  | 8          | 8           | yash_cv.pdf     | yash_ref_letter.pdf     | yash_cover_letter.pdf     | yash_other_docs.pdf     |
| 9  | 9          | 9           | heet_cv.pdf     | heet_ref_letter.pdf     | heet_cover_letter.pdf     | NULL                    |
| 10 | 10         | 10          | sahil_cv.pdf    | sahil_ref_letter.pdf    | sahil_cover_letter.pdf    | sahil_other_docs.pdf    |

Query executed successfully.

Fig 2: Display the inserted tables for Candidates and Documents

The screenshot shows a SQL Server Management Studio window with three tabs at the top: 'Testing.sql - JAINAM\jaina (71)', 'SQLQuery2.sql - JAINAM\jaina (70)', and 'Insert SQL QUery.s... (JAINAM\jair)'. The 'SQLQuery2.sql' tab contains the following three queries:

```
-- Display data from the JobPlatform table
SELECT * FROM JobPlatform;

-- Display data from the Job table
SELECT * FROM Job;

-- Display data from the JobOpenings table
SELECT * FROM JobOpenings;
```

The 'Results' tab displays the output of these queries:

|   | id | name      |
|---|----|-----------|
| 1 | 1  | LinkedIn  |
| 2 | 2  | Glassdoor |
| 3 | 3  | Handshake |
| 4 | 4  | Referral  |

|    | JobID | Position                    | Title                             | Type       | Medium | NumberOfPositions |
|----|-------|-----------------------------|-----------------------------------|------------|--------|-------------------|
| 1  | 1     | Data Scientist              | Data Scientist and AI Developer   | Fulltime   | Onsite | 3                 |
| 2  | 2     | Data Scientist              | Data Scientist                    | Fulltime   | Hybrid | 2                 |
| 3  | 3     | Software Engineer           | Software Engineer Position        | Fulltime   | Remote | 4                 |
| 4  | 4     | Java Developer              | Java Developer Full Stack         | Parttime   | Onsite | 1                 |
| 5  | 5     | Business Analyst            | Business Analyst and Data Analyst | Internship | Hybrid | 2                 |
| 6  | 6     | Frontend Developer          | Frontend Developer(Full Stack)    | Co-Op      | Remote | 3                 |
| 7  | 7     | Software Engineer           | Software Engineer Designer        | Fulltime   | Onsite | 2                 |
| 8  | 8     | Marketing Specialist        | Marketing Specialist              | Parttime   | Hybrid | 1                 |
| 9  | 9     | Financial Analyst           | Financial Analyst Position        | Fulltime   | Remote | 5                 |
| 10 | 10    | Human Resources Coordinator | HR Coordinator Position           | Internship | Hybrid | 2                 |

|    | OpeningID | JobID | NumberOfPositions |
|----|-----------|-------|-------------------|
| 1  | 1         | 1     | 3                 |
| 2  | 2         | 2     | 2                 |
| 3  | 3         | 3     | 4                 |
| 4  | 4         | 4     | 1                 |
| 5  | 5         | 5     | 2                 |
| 6  | 6         | 6     | 3                 |
| 7  | 7         | 7     | 2                 |
| 8  | 8         | 8     | 1                 |
| 9  | 9         | 9     | 5                 |
| 10 | 10        | 10    | 2                 |

At the bottom, a message indicates: **Query executed successfully.** and the connection is listed as **JAINAM\SQLEXPRESS**.

Fig 3: Display JobPlatform, Jobs and JobOpening Tables

Jainam Rajendra Shah  
SUID: 785948053

The screenshot shows a SQL Server Management Studio interface with three tabs:

- Testing.sql - JAINAM\jaina (71)**: Contains the first query.
- SQLQuery2.sql - JAINAM\jaina (70)**: Contains the second query.
- Insert SQL QUery.s... (JAINAM\jaina (67))**: Contains the third query.

The results pane displays the output of the first two queries:

```
-- Display data from the StatusFlow table
SELECT * FROM StatusFlow;

-- Display data from the Applications table
SELECT * FROM Applications;

-- Display data from the Tests table
```

|    | id | candidate_id | status                | timestamp               |
|----|----|--------------|-----------------------|-------------------------|
| 1  | 1  | 1            | Under Review          | 2024-04-18 08:00:00.000 |
| 2  | 2  | 2            | Negotiating           | 2024-04-23 08:30:00.000 |
| 3  | 3  | 3            | Accepted              | 2024-03-30 09:00:00.000 |
| 4  | 4  | 4            | Rejected              | 2024-01-30 09:30:00.000 |
| 5  | 5  | 5            | Rejected              | 2023-04-30 10:00:00.000 |
| 6  | 6  | 6            | Application Submitted | 2024-04-18 10:30:00.000 |
| 7  | 7  | 7            | Under Review          | 2024-04-14 11:00:00.000 |
| 8  | 8  | 8            | Application Submitted | 2024-04-19 11:30:00.000 |
| 9  | 9  | 9            | Under Review          | 2024-04-11 12:00:00.000 |
| 10 | 10 | 10           | Accepted              | 2024-04-09 12:30:00.000 |

|    | ApplicationID | CandidateID | OpeningID |
|----|---------------|-------------|-----------|
| 1  | 1             | 1           | 1         |
| 2  | 2             | 2           | 2         |
| 3  | 3             | 3           | 3         |
| 4  | 4             | 4           | 4         |
| 5  | 5             | 5           | 5         |
| 6  | 6             | 6           | 6         |
| 7  | 7             | 7           | 7         |
| 8  | 8             | 8           | 8         |
| 9  | 9             | 9           | 9         |
| 10 | 10            | 10          | 10        |

At the bottom, a message indicates successful execution: **Query executed successfully.** and the connection information: **JAINAM\SQLEXPRESS (16.0 RTM)**.

Fig 4: Display tables StatusFlow and Applications

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (71)', 'SQLQuery2.sql - JAINAM\jaina (70)', and 'Insert SQL QUery.s...JAINAM\jaina (67)\*'. The 'Insert SQL QUery.s...JAINAM\jaina (67)\*' tab contains the following SQL code:

```
-- Display data from the JobOpenings table
SELECT * FROM JobOpenings;

-- Display data from the StatusFlow table
SELECT * FROM StatusFlow;

-- Display data from the Applications table
SELECT * FROM Applications;

-- Display data from the Tests table
SELECT * FROM Tests;

-- Display data from the Interviewers table
```

The 'Results' tab displays the output for the 'Tests' query:

| TestId | ApplicationID | Type            | StartTime               | EndTime                 | Answers  | Grade |
|--------|---------------|-----------------|-------------------------|-------------------------|--|-------|
| 1      | 1             | OA              | 2024-04-18 09:00:00.000 | 2024-04-18 10:00:00.000 | A, B, C, D, A  | 85.50 |
| 2      | 2             | Behavioural MCQ | 2024-04-23 10:00:00.000 | 2024-04-23 11:00:00.000 | Strongly Agree, Agree, Neutral, Disagree, Strongl... | 70.00 |
| 3      | 3             | Aptitude        | 2024-03-30 09:00:00.000 | 2024-03-30 10:00:00.000 | 25, 36, 42, 55, 63                                   | 90.50 |
| 4      | 4             | OA              | 2024-01-30 09:30:00.000 | 2024-01-30 10:30:00.000 | A, C, D, D, B  | 78.00 |
| 5      | 5             | Behavioural MCQ | 2023-04-30 10:00:00.000 | 2023-04-30 11:00:00.000 | Disagree, Strongly Agree, Neutral, Agree, Disagree   | 65.50 |
| 6      | 6             | Aptitude        | 2024-04-18 10:30:00.000 | 2024-04-18 11:30:00.000 | 30, 42, 56, 68, 75                                   | 82.00 |
| 7      | 7             | OA              | 2024-04-14 11:00:00.000 | 2024-04-14 12:00:00.000 | C, A, B, D, C  | 88.50 |
| 8      | 8             | Behavioural MCQ | 2024-04-19 11:30:00.000 | 2024-04-19 12:30:00.000 | Neutral, Agree, Disagree, Strongly Agree, Strongl... | 72.00 |
| 9      | 9             | Aptitude        | 2024-04-11 12:00:00.000 | 2024-04-11 13:00:00.000 | 40, 51, 64, 78, 85                                   | 95.00 |
| 10     | 10            | OA              | 2024-04-09 12:30:00.000 | 2024-04-09 13:30:00.000 | B, B, D, A, C  | 80.50 |

The 'Results' tab also displays the output for the 'Interviewers' query:

| InterviewerID | Name            | Department               | Title                    |
|---------------|-----------------|--------------------------|--------------------------|
| 1             | John Doe        | Human Resources          | HR Manager               |
| 2             | Jane Smith      | Engineering              | Senior Software Engineer |
| 3             | David Martinez  | Operations               | Operations Manager       |
| 4             | Daniel Taylor   | Research and Development | Data Science Manager     |
| 5             | Sophia Anderson | Product Management       | Product Manager          |

The status bar at the bottom indicates: 'Query executed successfully.' and 'JAINAM\SQLEXPRESS (16.0 RTM) JAINAM\jaina (67)'.

Fig 5: Tests and Interviewers

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (55)', 'SQLQuery2.sql - JAINAM\jaina (54)\*', and 'Insert SQL QUery.s... (JAINAM\jaina (53))'. The 'Testing.sql' tab contains three queries:

```
-- Display data from the Interviewers table
SELECT * FROM Interviewers;

-- Display data from the Interviews table
SELECT * FROM Interviews;

-- Display data from the JobReview table
SELECT * FROM JobReview;
```

The 'Results' pane displays the data from the 'Interviews' table:

|    | InterviewID | ApplicationID | Type   | StartTime               | EndTime                 | InterviewerID |
|----|-------------|---------------|--------|-------------------------|-------------------------|---------------|
| 1  | 1           | 1             | Onsite | 2024-04-20 09:00:00.000 | 2024-04-20 10:45:00.000 | 1             |
| 2  | 2           | 2             | Online | 2024-04-21 10:00:00.000 | 2024-04-21 11:30:00.000 | 2             |
| 3  | 3           | 3             | Onsite | 2024-04-22 11:00:00.000 | 2024-04-22 12:00:00.000 | 1             |
| 4  | 4           | 4             | Online | 2024-04-23 12:00:00.000 | 2024-04-23 12:20:00.000 | 3             |
| 5  | 5           | 5             | Onsite | 2024-04-24 13:00:00.000 | 2024-04-24 14:00:00.000 | 4             |
| 6  | 6           | 6             | Online | 2024-04-25 14:00:00.000 | 2024-04-25 15:00:00.000 | 5             |
| 7  | 7           | 7             | Onsite | 2024-04-26 15:00:00.000 | 2024-04-26 16:00:00.000 | 2             |
| 8  | 8           | 8             | Online | 2024-04-27 16:00:00.000 | 2024-04-27 17:00:00.000 | 1             |
| 9  | 9           | 9             | Onsite | 2024-04-28 17:00:00.000 | 2024-04-28 18:15:00.000 | 3             |
| 10 | 10          | 10            | Online | 2024-04-29 18:00:00.000 | 2024-04-29 19:00:00.000 | 5             |

The 'Results' pane also displays the data from the 'JobReview' table:

|    | id | candidate_id | interviewer_id | review_text  | rating |
|----|----|--------------|----------------|--|--------|
| 1  | 1  | 1            | 1              | Good logical skills. Moving onto the next round.       | 4      |
| 2  | 2  | 2            | 2              | Great interview! Well-prepared and insightful answers. | 5      |
| 3  | 3  | 3            | 1              | The interviewer asked irrelevant questions.            | 2      |
| 4  | 4  | 4            | 3              | The interviewee performed very poorly.                 | 1      |
| 5  | 5  | 5            | 4              | The interviewee needs improvement in communicati...    | 2      |
| 6  | 6  | 6            | 5              | Failed to answer critical questions.                   | 1      |
| 7  | 7  | 7            | 2              | Passed the interview with flying colors.               | 5      |
| 8  | 8  | 8            | 1              | Good logical skills. Moving onto the next round.       | 4      |
| 9  | 9  | 9            | 3              | Need Improvement in technical skills.                  | 2      |
| 10 | .. | 10           | 5              | Irrelevant questions were asked during the interview.  | 3      |

The status bar at the bottom left says 'Query executed successfully.' and the bottom right says 'JAINAM\SQLEXPRESS (16.0)'.

Fig 6: Interviews and Job Reviews

The screenshot shows three tabs in the SSMS interface:

- Testing.sql - JAINAM\jaina (55)**: Contains the following SQL code:

```
-- Display data from the Evaluations table
SELECT * FROM Evaluations;

-- Display data from the Reimbursements table
SELECT * FROM Reimbursements;

-- Display data from the Onboarding table
SELECT * FROM Onboarding;
```
- SQLQuery2.sql - JAINAM\jaina (54)\***: This tab is currently active.
- Insert SQL QUery.s... (JAINAM\jaina (53))**: Contains a single row of data.

The Results pane displays the data from the **Evaluations** table:

|    | EvaluationID | ApplicationID | EvaluationNotes                                      | Result  |
|----|--------------|---------------|--|---------|
| 1  | 1            | 1             | Passed the interview. Moving onto the next round.    | Pass    |
| 2  | 2            | 2             | Failed the interview. Needs improvement in techni... | Fail    |
| 3  | 3            | 3             | Can do better in the next round. Improve communi...  | Pending |
| 4  | 4            | 4             | Passed the interview with excellent performance.     | Pass    |
| 5  | 5            | 5             | Moving onto the next round. Good problem-solving...  | Pass    |
| 6  | 6            | 6             | Failed the interview due to lack of experience.      | Fail    |
| 7  | 7            | 7             | Needs improvement in coding skills. Waiting on ot... | Pending |
| 8  | 8            | 8             | Passed the interview with flying colors.             | Pass    |
| 9  | 9            | 9             | Moving onto the next round. Great presentation sk... | Pass    |
| 10 | 10           | 10            | Failed to meet expectations. Pending further review. | Pending |

The Results pane also displays the data from the **Reimbursements** table:

|    | ReimbursementID | ApplicationID | RequestDate | Processed | Amount |
|----|-----------------|---------------|-------------|-----------|--------|
| 1  | 1               | 1             | 2024-04-18  | 1         | 200.00 |
| 2  | 2               | 2             | 2024-04-23  | 1         | 150.00 |
| 3  | 3               | 3             | 2024-03-30  | 0         | 0.00   |
| 4  | 4               | 4             | 2024-01-30  | 1         | 100.00 |
| 5  | 5               | 5             | 2023-04-30  | 0         | 0.00   |
| 6  | 6               | 6             | 2024-04-18  | 1         | 250.00 |
| 7  | 7               | 7             | 2024-04-14  | 1         | 180.00 |
| 8  | 8               | 8             | 2024-04-19  | 0         | 0.00   |
| 9  | 9               | 9             | 2024-04-11  | 1         | 300.00 |
| 10 | 10              |               | 2024-04-09  | 1         | 220.00 |

The status bar at the bottom indicates: **Query executed successfully.** and **JAINAM\SQLEXPRESS (16.0 RTM) JAIN**.

Fig 7: Evaluation and Reimbursements

```
-- Display data from the Reimbursements table
SELECT * FROM Reimbursements;

-- Display data from the Onboarding table
SELECT * FROM Onboarding;

-- Display data from the Complaint table
SELECT * FROM Complaint;
```

Results

|    | OnboardingID | CandidateID | JobID | StartDate  |
|----|--------------|-------------|-------|------------|
| 1  | 1            | 1           | 1     | 2024-06-10 |
| 2  | 2            | 2           | 2     | 2024-06-10 |
| 3  | 3            | 3           | 3     | 2024-06-07 |
| 4  | 4            | 4           | 4     | 2024-07-03 |
| 5  | 5            | 5           | 5     | 2024-07-10 |
| 6  | 6            | 6           | 6     | 2024-08-21 |
| 7  | 7            | 7           | 7     | 2024-06-30 |
| 8  | 8            | 8           | 8     | 2024-05-24 |
| 9  | 9            | 9           | 9     | 2024-07-17 |
| 10 | 10           | 10          | 10    | 2024-06-10 |

|    | id | candidate_id | description                                    | status | resolution         |
|----|----|--------------|--|--------|--------------------|
| 1  | 1  | 1            | Cannot Start Behavioural Questions             | Open   | Investigating      |
| 2  | 2  | 2            | Link not working                               | Open   | Resolved           |
| 3  | 3  | 3            | Poor Connection hence could not give interview | Open   | Investigating      |
| 4  | 4  | 4            | NULL   | Open   | No action required |
| 5  | 5  | 5            | NULL   | Open   | No action required |
| 6  | 6  | 6            | Link not working                               | Open   | Investigating      |
| 7  | 7  | 7            | Cannot Start Behavioural Questions             | Open   | Investigating      |
| 8  | 8  | 8            | Poor Connection hence could not give interview | Open   | Investigating      |
| 9  | 9  | 9            | NULL   | Open   | No action required |
| 10 | 10 | 10           | NULL   | Open   | No action required |

Query executed successfully.

Fig 8: Display Onboarding and Complaint

```
-- Display data from the BackgroundChecks table
SELECT * FROM BackgroundChecks;

-- Display data from the DrugTests table
SELECT * FROM DrugTests;

-- Display data from the JobCategory table
SELECT * FROM JobCategory;
```

| CheckID | CandidateID | CriminalBackground | EmploymentHistory | Status  | CheckDate  |
|---------|-------------|--------------------|-------------------|---------|------------|
| 1       | 1           | Clean              | Stable            | Clear   | 2024-04-18 |
| 2       | 2           | Minor Offense      | Unstable          | Pending | 2024-04-23 |
| 3       | 3           | Clear              | Stable            | Clear   | 2024-04-30 |
| 4       | 4           | Clean              | Stable            | Clear   | 2024-04-30 |
| 5       | 5           | Clean              | Stable            | Clear   | 2024-04-30 |
| 6       | 6           | Clear              | Unstable          | Clear   | 2024-04-18 |
| 7       | 7           | Clear              | Stable            | Clear   | 2024-04-14 |
| 8       | 8           | Minor Offense      | Stable            | Pending | 2024-04-19 |
| 9       | 9           | Clear              | Stable            | Clear   | 2024-04-11 |
| 10      | 10          | Clean              | Stable            | Clear   | 2024-04-09 |

| TestID | CandidateID | TestType    | TestDate   | Results  |
|--------|-------------|-------------|------------|----------|
| 1      | 1           | Urine Test  | 2024-04-18 | Negative |
| 2      | 2           | Blood Test  | 2024-04-19 | Negative |
| 3      | 3           | Saliva Test | 2024-04-20 | Negative |
| 4      | 4           | Urine Test  | 2024-04-21 | Negative |
| 5      | 5           | Blood Test  | 2024-04-22 | Positive |
| 6      | 6           | Saliva Test | 2024-04-23 | Negative |
| 7      | 7           | Urine Test  | 2024-04-24 | Negative |
| 8      | 8           | Blood Test  | 2024-04-25 | Negative |
| 9      | 9           | Saliva Test | 2024-04-26 | Positive |
| 10     | 10          | Urine Test  | 2024-04-27 | Negative |
| 11     | 11          | Blood Test  | 2024-04-28 | Negative |
| 12     | 12          | Saliva Test | 2024-04-29 | Negative |
| 13     | 13          | Urine Test  | 2024-04-30 | Negative |
| 14     | 14          | Blood Test  | 2024-05-01 | Negative |
| 15     | 15          | Saliva Test | 2024-05-02 | Negative |
| 16     | 16          | Urine Test  | 2024-05-03 | Negative |
| 17     | 17          | Blood Test  | 2024-05-04 | Negative |

Query executed successfully. JAINAM\SQLEXPRESS (16.0 RTM) JAINAM\jain

Fig 9: Background check and DrugTest

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (55)', 'SQLQuery2.sql - JAI... (JAINAM\jaina (54))\*', and 'Insert SQL Query.s... (JAINAM\jaina (54))'. The 'Testing.sql' tab contains three queries to display data from 'JobCategory', 'JobPlatformJob', and 'JobCategoryJob' tables. The results pane shows the data for each query.

-- Display data from the JobCategory table  
SELECT \* FROM JobCategory;

-- Display data from the JobPlatformJob table  
SELECT \* FROM JobPlatformJob;

-- Display data from the JobCategoryJob table  
SELECT \* FROM JobCategoryJob;

|   | id | name                 |
|---|----|----------------------|
| 1 | 1  | Data Science         |
| 2 | 2  | Software Development |
| 3 | 3  | Data Analyst         |
| 4 | 4  | Java Developer       |
| 5 | 5  | Business Analysis    |
| 6 | 6  | Human Resource       |

|    | job_platform_id | job_id |
|----|-----------------|--------|
| 2  | 1               | 2      |
| 3  | 3               | 3      |
| 4  | 4               | 4      |
| 5  | 2               | 5      |
| 6  | 2               | 6      |
| 7  | 4               | 7      |
| 8  | 1               | 8      |
| 9  | 1               | 9      |
| 10 | 3               | 10     |

|    | job_category_id | job_id |
|----|-----------------|--------|
| 1  | 1               | 1      |
| 2  | 1               | 2      |
| 3  | 2               | 3      |
| 4  | 4               | 4      |
| 5  | 5               | 5      |
| 6  | 2               | 6      |
| 7  | 2               | 7      |
| 8  | 5               | 8      |
| 9  | 5               | 9      |
| 10 | 6               | 10     |

Query executed successfully.

Fig 10: Display Job Category, Platform and Job Category Jobs

#### 4. Testing:

##### i. Views

- 1) Job Openings with Available Positions:

Objective: Identify job openings that still have available positions and determine the number of positions available for each opening.

Query:

```
SELECT j.Title, j.NumberOfPositions - COALESCE(COUNT(o.OpeningID), 0) AS AvailablePositions
FROM Job AS j
LEFT JOIN JobOpenings AS o ON j.JobID = o.JobID
GROUP BY j.Title, j.NumberOfPositions;
```

Output:

The screenshot shows the SQL Server Management Studio interface. The query window contains the following code:

```
/*
    Views
*/
/*
    Job Openings with Available Positions:
    Question: Identify job openings that still have available positions and determine the number of positions available for each opening.
*/
SELECT j.Title, j.NumberOfPositions - COALESCE(COUNT(o.OpeningID), 0) AS AvailablePositions
FROM Job AS j
LEFT JOIN JobOpenings AS o ON j.JobID = o.JobID
GROUP BY j.Title, j.NumberOfPositions;
```

The results pane displays a table with the following data:

|    | Title                             | AvailablePositions |
|----|-----------------------------------|--------------------|
| 1  | Java Developer Full Stack         | 0                  |
| 2  | Marketing Specialist              | 0                  |
| 3  | Business Analyst and Data Analyst | 1                  |
| 4  | Data Scientist                    | 1                  |
| 5  | HR Coordinator Position           | 1                  |
| 6  | Software Engineer Designer        | 1                  |
| 7  | Data Scientist and AI Developer   | 2                  |
| 8  | Frontend Developer(Full Stack)    | 2                  |
| 9  | Software Engineer Position        | 3                  |
| 10 | Financial Analyst Position        | 4                  |

At the bottom of the results pane, a message indicates: "Query executed successfully." and shows the session details: JAINAM\SQLEXPRESS (16.0 RTM) JAINAM\jaina (102) HR\_DB 00:00:00 10 rows.

Fig 11: Job Openings Available Positions

Explanation:

This query retrieves the title of job openings along with the number of available positions for each opening. It achieves this by joining the "Job" table with the "JobOpenings" table using a left join to ensure all job titles are included, even if there are no corresponding openings. The COALESCE function is used to handle cases where there are no openings for a job, ensuring that the subtraction operation does not result in NULL values.

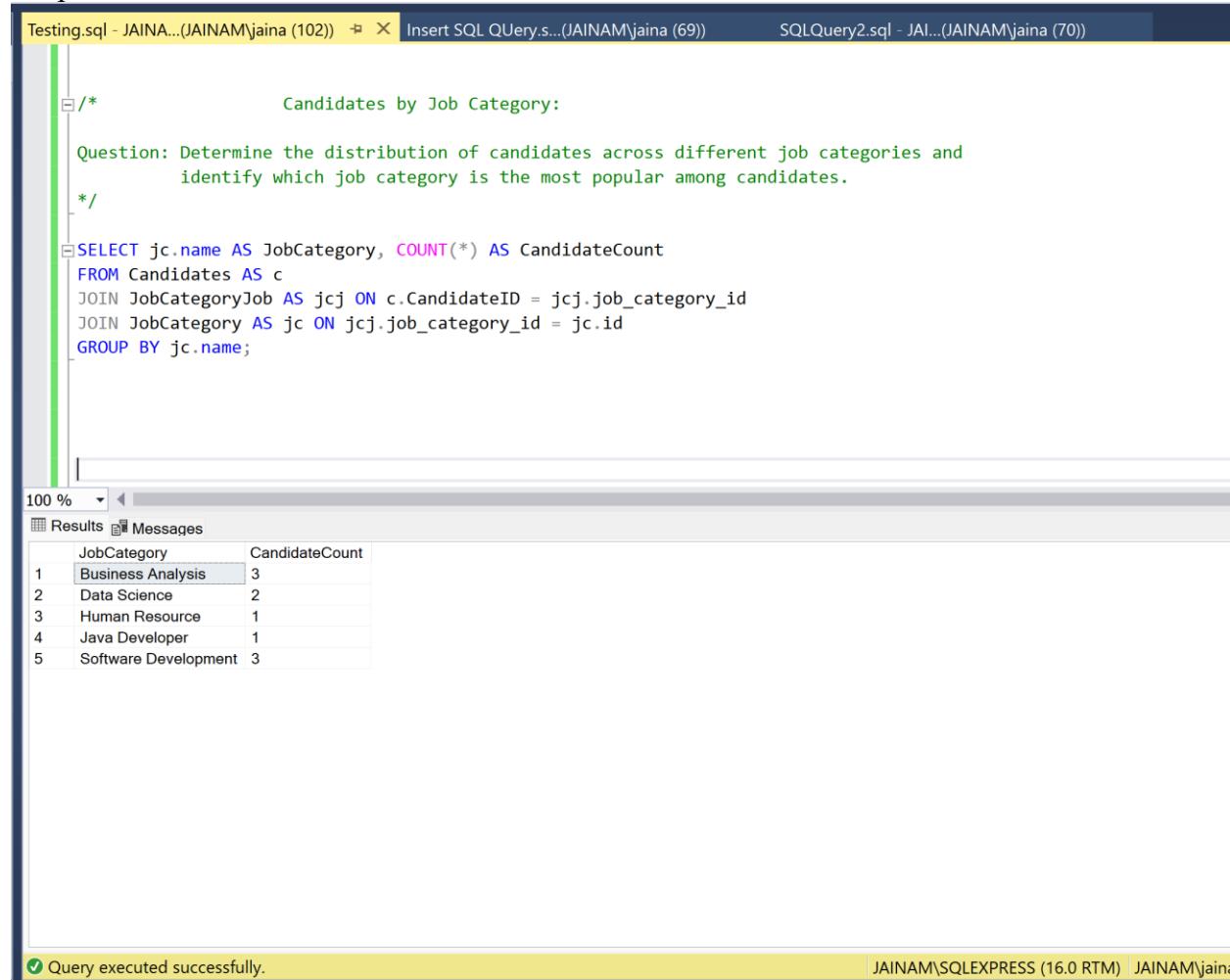
2) Candidates by Job Category:

Objective: Determine the distribution of candidates across different job categories and identify which job category is the most popular among candidates.

Query:

```
SELECT jc.name AS JobCategory, COUNT(*) AS CandidateCount
FROM Candidates AS c
JOIN JobCategoryJob AS jcj ON c.CandidateID = jcj.job_category_id
JOIN JobCategory AS jc ON jcj.job_category_id = jc.id
GROUP BY jc.name;
```

Output:



The screenshot shows the SQL Server Management Studio interface. The top bar displays three tabs: 'Testing.sql - JAINAM\jaina (102)', 'Insert SQL Query.s... (JAINAM\jaina (69))', and 'SQLQuery2.sql - JAI... (JAINAM\jaina (70))'. The main area contains the SQL query for calculating candidate counts by job category. Below the query, the 'Results' tab is selected, showing a table with five rows of data. The table has two columns: 'JobCategory' and 'CandidateCount'. The data is as follows:

| JobCategory          | CandidateCount |
|----------------------|----------------|
| Business Analysis    | 3              |
| Data Science         | 2              |
| Human Resource       | 1              |
| Java Developer       | 1              |
| Software Development | 3              |

At the bottom of the results pane, a yellow status bar indicates 'Query executed successfully.'

Fig 12: Candidates Count by Job Category

Explanation: This query calculates the count of candidates in each job category by joining the "Candidates" table with the "JobCategoryJob" and "JobCategory" tables. It then groups the results by job category and counts the number of candidates in each category.

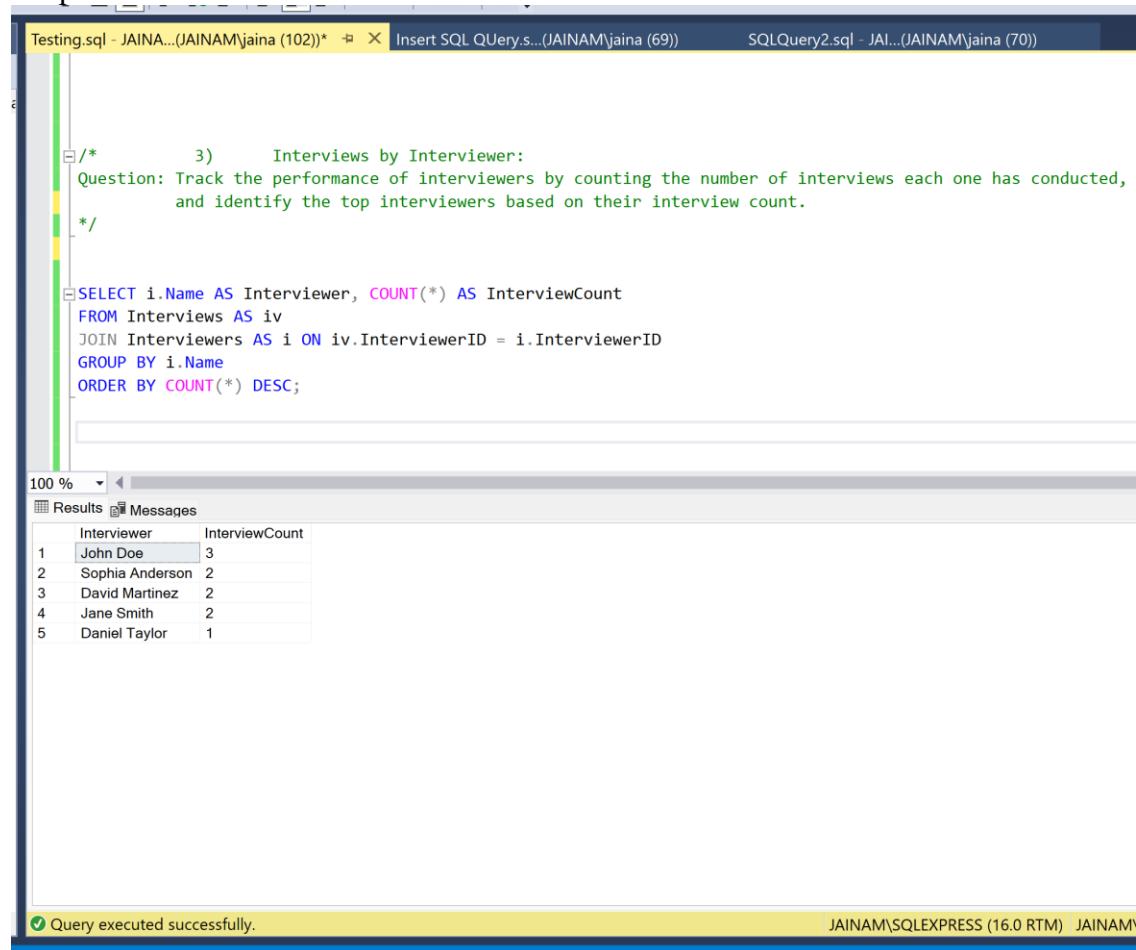
### 3) Interviews by Interviewer:

Objective: Track the performance of interviewers by counting the number of interviews each one has conducted, and identify the top interviewers based on their interview count.

Query:

```
SELECT i.Name AS Interviewer, COUNT(*) AS InterviewCount
FROM Interviews AS iv
JOIN Interviewers AS i ON iv.InterviewerID = i.InterviewerID
GROUP BY i.Name
ORDER BY COUNT(*) DESC;
```

Output:



The screenshot shows the SQL Server Management Studio interface. The query window contains the SQL code for tracking interviewer performance. The results pane displays a table with five rows, showing the number of interviews conducted by each interviewer, ordered from highest to lowest count. The status bar at the bottom indicates the query was executed successfully.

|   | Interviewer     | InterviewCount |
|---|-----------------|----------------|
| 1 | John Doe        | 3              |
| 2 | Sophia Anderson | 2              |
| 3 | David Martinez  | 2              |
| 4 | Jane Smith      | 2              |
| 5 | Daniel Taylor   | 1              |

Fig 13: InterviewsByInterviewer

Explanation:

This query retrieves the count of interviews conducted by each interviewer by joining the "Interviews" table with the "Interviewers" table. It then groups the results by interviewer name and orders them in descending order based on the interview count.

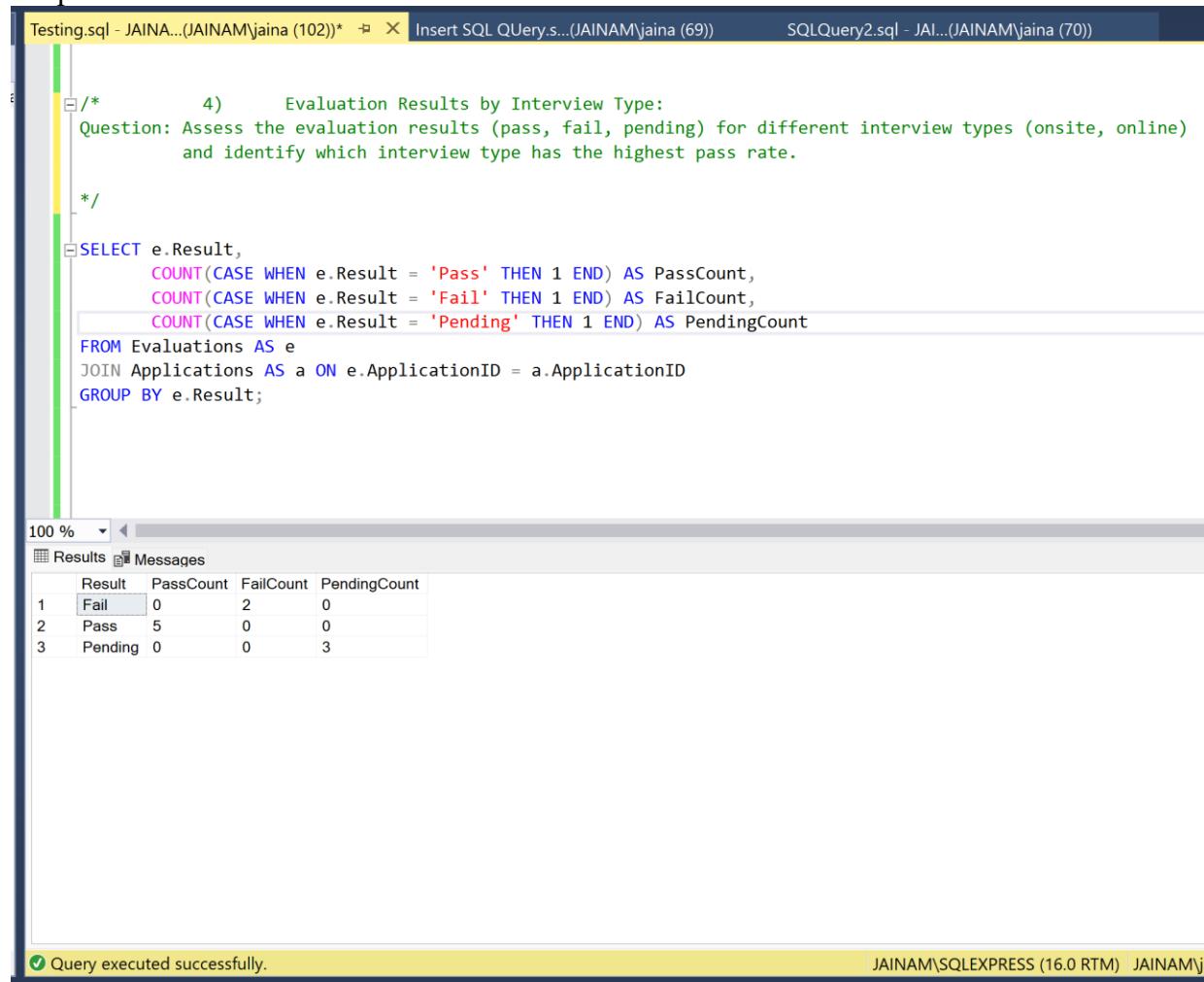
4) Evaluation Results by Interview Type:

Objective: Assess the evaluation results (pass, fail, pending) for different interview types (onsite, online) and identify which interview type has the highest pass rate.

Query:

```
SELECT e.Result,
       COUNT(CASE WHEN e.Result = 'Pass' THEN 1 END) AS PassCount,
       COUNT(CASE WHEN e.Result = 'Fail' THEN 1 END) AS FailCount,
       COUNT(CASE WHEN e.Result = 'Pending' THEN 1 END) AS PendingCount
  FROM Evaluations AS e
 JOIN Applications AS a ON e.ApplicationID = a.ApplicationID
 GROUP BY e.Result;
```

Output:



The screenshot shows the SQL Server Management Studio interface. The top bar displays three tabs: 'Testing.sql - JAINAM\jaina (102)\*', 'Insert SQL QUery.s... (JAINAM\jaina (69))', and 'SQLQuery2.sql - JAI...(JAINAM\jaina (70))'. The main window contains the SQL query for evaluating interview results. Below the query, the results pane shows a table with four columns: Result, PassCount, FailCount, and PendingCount. The data is as follows:

|   | Result  | PassCount | FailCount | PendingCount |
|---|---------|-----------|-----------|--------------|
| 1 | Fail    | 0         | 2         | 0            |
| 2 | Pass    | 5         | 0         | 0            |
| 3 | Pending | 0         | 0         | 3            |

At the bottom of the results pane, a message indicates 'Query executed successfully.' and shows the connection details 'JAINAM\SQLEXPRESS (16.0 RTM) | JAINAM\ja...'.

Fig 14: EvaluationResultsByInterviewType

Explanation:

This query calculates the count of evaluation results (pass, fail, pending) for each interview type (onsite, online) by joining the "Evaluations" table with the "Applications" table. It then groups the results by interview type and computes the count of each evaluation result.

5) Background Check Status by Candidate:

Objective: Monitor the background check status (clear, pending) for each candidate and determine how many candidates have pending background checks.

Query:

```
SELECT c.Name AS CandidateName, bc.Status AS BackgroundCheckStatus
FROM Candidates AS c
JOIN BackgroundChecks AS bc ON c.CandidateID = bc.CandidateID;
```

Output:

The screenshot shows the SQL Server Management Studio interface. The top bar displays the title 'Testing.sql - JAINAM\jaina (102)\*' and the status 'Insert SQL QUery.s... (JAINAM\jaina (69))'. The main window contains the query code and its results. The code includes a comment block and the SELECT statement. The results grid shows 10 rows of data with columns 'CandidateName' and 'BackgroundCheckStatus'. The last row, 'Sahil Shah', is highlighted. At the bottom, a message bar indicates 'Query executed successfully.'

|    | CandidateName  | BackgroundCheckStatus |
|----|----------------|-----------------------|
| 1  | Jainam Shah    | Clear                 |
| 2  | Jackie Chan    | Pending               |
| 3  | Gal Gadot      | Clear                 |
| 4  | Margot Robbie  | Clear                 |
| 5  | Dikshita Patel | Clear                 |
| 6  | Janvi Sadhi    | Clear                 |
| 7  | Bhavya Shah    | Clear                 |
| 8  | Yash Solanki   | Pending               |
| 9  | Heet Chedda    | Clear                 |
| 10 | Sahil Shah     | Clear                 |

Fig 15: BackgroundCheckStatusByCandidate

Explanation:

This query retrieves the background check status for each candidate by joining the "Candidates" table with the "BackgroundChecks" table. It selects the candidate name and their corresponding background check status.

**ii. Stored Procedure:**

i. Update Candidate Information

Objective: Create a stored procedure to update the information of a candidate in the Candidates table.

Query:

```
CREATE PROCEDURE UpdateCandidateInformation
    @CandidateID INT,
    @Name NVARCHAR(100),
    @Email NVARCHAR(100),
    @Phone NVARCHAR(20),
    @ShortProfile NVARCHAR(100)
AS
BEGIN
    UPDATE Candidates
    SET Name = @Name,
        Email = @Email,
        Phone = @Phone,
        ShortProfile = @ShortProfile
    WHERE CandidateID = @CandidateID;
END;

EXEC UpdateCandidateInformation
    @CandidateID = 9,
    @Name = 'Robert',
    @Email = 'robert66@gmail.com',
    @Phone = '3159527506',
    @ShortProfile = 'Software Engineer';
```

Output:

The screenshot shows the SQL Server Management Studio interface. The title bar says "Testing.sql - JAINA...(JAINAM\jaina (59))". The code editor pane displays the following T-SQL script:

```
/*          Stored Procedures      */
/*
   1) Update Candidate Information
   Create a stored procedure to update the information of a candidate in the Candidates table.

*/
CREATE PROCEDURE UpdateCandidateInformation
    @CandidateID INT,
    @Name NVARCHAR(100),
    @Email NVARCHAR(100),
    @Phone NVARCHAR(20),
    @ShortProfile NVARCHAR(100)
AS
BEGIN
    UPDATE Candidates
    SET Name = @Name,
        Email = @Email,
        Phone = @Phone,
        ShortProfile = @ShortProfile
    WHERE CandidateID = @CandidateID;
END;

EXEC UpdateCandidateInformation
    @CandidateID = 9,
    @Name = 'Robert',
    @Email = 'robert66@gmail.com',
    @Phone = '3159527506',
    @ShortProfile = 'Software Engineer';
```

The "Messages" pane at the bottom shows the output:

```
Commands completed successfully.
```

Completion time: 2024-04-29T01:37:41.5580712-04:00

At the bottom of the window, status indicators show: 100 %, ✓ Query executed successfully., JAINAM\SQLEXPRESS (16.0 RTM), Ln 117, Col 10, Ch 10, INS.

Fig 16: UpdateCandidateInformation

The screenshot shows a SQL Server Management Studio window. The top tab bar has 'Testing.sql - JAINAM\jaina (53)' and 'Insert SQL QUery.s... (JAINAM\jaina (79))\*' selected. The bottom tab bar shows 'SQLQuery2.sql - JAINAM\jaina (79)'. The main pane displays the following T-SQL code:

```
AS
BEGIN
    UPDATE Candidates
    SET Name = @Name,
        Email = @Email,
        Phone = @Phone,
        ShortProfile = @ShortProfile
    WHERE CandidateID = @CandidateID;
END;

EXEC UpdateCandidateInformation
@CandidateID = 9,
@Name = 'Robert',
@email = 'robert66@gmail.com',
@Phone = '3159527506',
@ShortProfile = 'Software Engineer';

SELECT *
FROM Candidates
```

The code includes a stored procedure definition and its execution. The execution part uses parameters: @CandidateID = 9, @Name = 'Robert', @Email = 'robert66@gmail.com', @Phone = '3159527506', and @ShortProfile = 'Software Engineer'. A final SELECT statement retrieves all data from the Candidates table.

The results pane shows a table with 10 rows of candidate data. The columns are CandidateID, Name, Email, Phone, and ShortProfile. The data is as follows:

|    | CandidateID | Name           | Email                | Phone      | ShortProfile       |
|----|-------------|----------------|----------------------|------------|--------------------|
| 1  | 1           | Jainam Shah    | jainam@gmail.com     | 1234567890 | Data Scientist     |
| 2  | 2           | Jackie Chan    | jackie@hotmail.com   | 2345678901 | Data Scientist     |
| 3  | 3           | Gal Gadot      | gal@syr.edu          | 3456789012 | Java Developer     |
| 4  | 4           | Margot Robbie  | margot@gmail.com     | 4567890123 | Software Engineer  |
| 5  | 5           | Dikshita Patel | dikshita@hotmail.com | 5678901234 | Frontend Developer |
| 6  | 6           | Janvi Saddi    | janvi@syr.edu        | 6789012345 | Frontend Developer |
| 7  | 7           | Bhavya Shah    | bhavya@gmail.com     | 7890123456 | Data Analyst       |
| 8  | 8           | Yash Solanki   | yash@hotmail.com     | 8901234567 | Data Analyst       |
| 9  | 9           | Robert         | robert66@gmail.com   | 3159527506 | Software Engineer  |
| 10 | 10          | Sahil Shah     | sahil@gmail.com      | 0123456789 | Data Analyst       |

The status bar at the bottom indicates 'Query executed successfully.' and shows 'JAINAM\SQLEXP' as the user. The status bar also shows 'Ln 137', 'Col 1', 'Ch 1', and 'INS'.

Fig 17: Candidate Name Robert has been updated using Stored Procedure

#### Explanation:

This stored procedure takes parameters such as CandidateID, Name, Email, Phone, and ShortProfile. It updates the candidate information in the Candidates table based on the provided CandidateID.

ii. UpdateInterviewDetails

Objective: Update the details of a specific interview for a candidate.

Output:

The screenshot shows the SQL Server Management Studio interface. The code editor window displays the following T-SQL script:

```
/*
    UpdateInterviewDetails
    Update the details of a specific interview for a candidate.
*/

CREATE PROCEDURE UpdateInterviewDetails
    @InterviewID INT,
    @StartTime DATETIME,
    @EndTime DATETIME
AS
BEGIN
    UPDATE Interviews
    SET StartTime = @StartTime,
        EndTime = @EndTime
    WHERE InterviewID = @InterviewID;
END;

EXEC UpdateInterviewDetails
    @InterviewID = 1,
    @StartTime = '2024-05-01 09:00:00',
    @EndTime = '2024-05-01 10:00:00';
```

The Messages pane at the bottom shows the output:

```
Commands completed successfully.  
Completion time: 2024-04-29T01:56:04.6685356-04:00
```

At the bottom right, it says JAINAM\SQLEXPRESS (16.0).

Fig 18: Update the details of a specific interview for a candidate.

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, there are three tabs: 'Testing.sql - JAINAM\jaina (53)\*' (highlighted in yellow), 'Insert SQL QUery.s... (JAINAM\jaina (79))\*', and 'SQLQuery2.sql - JAI... (JAINAM\jaina (80))'. The main pane displays a T-SQL script:

```
    @EndTime DATETIME
AS
BEGIN
    UPDATE Interviews
    SET StartTime = @StartTime,
        EndTime = @EndTime
    WHERE InterviewID = @InterviewID;
END;

EXEC UpdateInterviewDetails
@InterviewID = 1,
@StartTime = '2024-05-01 09:00:00',
@EndTime = '2024-05-01 10:00:00';

SELECT *
FROM Interviews;
```

Below the script, the 'Results' tab is selected, showing the output of the query:

|    | InterviewID | ApplicationID | Type   | StartTime               | EndTime                 | InterviewerID |
|----|-------------|---------------|--------|-------------------------|-------------------------|---------------|
| 1  | 1           | 1             | Onsite | 2024-05-01 09:00:00.000 | 2024-05-01 10:00:00.000 | 1             |
| 2  | 2           | 2             | Online | 2024-04-21 10:00:00.000 | 2024-04-21 11:30:00.000 | 2             |
| 3  | 3           | 3             | Onsite | 2024-04-22 11:00:00.000 | 2024-04-22 12:00:00.000 | 1             |
| 4  | 4           | 4             | Online | 2024-04-23 12:00:00.000 | 2024-04-23 12:20:00.000 | 3             |
| 5  | 5           | 5             | Onsite | 2024-04-24 13:00:00.000 | 2024-04-24 14:00:00.000 | 4             |
| 6  | 6           | 6             | Online | 2024-04-25 14:00:00.000 | 2024-04-25 15:00:00.000 | 5             |
| 7  | 7           | 7             | Onsite | 2024-04-26 15:00:00.000 | 2024-04-26 16:00:00.000 | 2             |
| 8  | 8           | 8             | Online | 2024-04-27 16:00:00.000 | 2024-04-27 17:00:00.000 | 1             |
| 9  | 9           | 9             | Onsite | 2024-04-28 17:00:00.000 | 2024-04-28 18:15:00.000 | 3             |
| 10 | 10          | 10            | Online | 2024-04-29 18:00:00.000 | 2024-04-29 19:00:00.000 | 5             |

In the status bar at the bottom, it says 'Query executed successfully.' and 'JAINAM\SQLEXPRESS (16.0 RTI)'.

Fig 19: Using Stored Procedure Changed endtime from 10:45 to 10:00

### Explanation:

This stored procedure takes in the InterviewID of the interview that needs to be updated, as well as the new start and end times for the interview. It then performs an update operation on the Interviews table, setting the StartTime and EndTime columns to the new values provided in the parameters, for the interview with the specified InterviewID.

Jainam Rajendra Shah  
SUID: 785948053

iii. RetrieveBackgroundCheckStatus

Objective: This stored procedure aims to retrieve the background check status for a specific candidate.

Query:

```
CREATE PROCEDURE RetrieveBackgroundCheckStatus
    @CandidateID INT
AS
BEGIN
    SELECT CandidateID, CriminalBackground, EmploymentHistory, Status, CheckDate
    FROM BackgroundChecks
    WHERE CandidateID = @CandidateID;
END;
```

```
EXEC RetrieveBackgroundCheckStatus @CandidateID = 4;
```

Output:

The screenshot shows a SQL Server Management Studio window. The top bar has tabs for 'Testing.sql - JAINAM\jaina (53)', 'Insert SQL QUery.s... (JAINAM\jaina (79))\*', and 'SQLQuery2.sql - JAI... (JAINAM\jaina (80))'. The main area contains a stored procedure named 'RetrieveBackgroundCheckStatus' with a comment describing it. Below the procedure is an EXEC statement. The results pane shows a single row of data from the execution.

```
/*  
    3) RetrieveBackgroundCheckStatus  
    Retrieve the background check status for a specific candidate.  
*/  
  
CREATE PROCEDURE RetrieveBackgroundCheckStatus  
    @CandidateID INT  
AS  
BEGIN  
    SELECT CandidateID, CriminalBackground, EmploymentHistory, Status, CheckDate  
    FROM BackgroundChecks  
    WHERE CandidateID = @CandidateID;  
END;  
  
EXEC RetrieveBackgroundCheckStatus @CandidateID = 4;
```

Results

|   | CandidateID | CriminalBackground | EmploymentHistory | Status | CheckDate  |
|---|-------------|--------------------|-------------------|--------|------------|
| 1 | 4           | Clean              | Stable            | Clear  | 2024-04-30 |

Query executed successfully.

Fig 20: SQL Query for Retrieving Background Check Status

#### Explanation:

This stored procedure accepts an input parameter @CandidateID which specifies the candidate for whom the background check status is to be retrieved. It selects the CandidateID, CriminalBackground, EmploymentHistory, Status, and CheckDate columns from the BackgroundChecks table. The WHERE clause filters the records based on the provided @CandidateID.

iv. Update Reimbursement Status  
Objective: Update the reimbursement status of a specific application.

Query:

```
CREATE PROCEDURE UpdateReimbursementStatus
    @ApplicationID INT,
    @Processed BIT,
    @Amount DECIMAL(10, 2)
AS
BEGIN
    UPDATE Reimbursements
    SET Processed = @Processed,
        Amount = @Amount
    WHERE ApplicationID = @ApplicationID;
END;
```

```
EXEC UpdateReimbursementStatus @ApplicationID = 1, @Processed = 1, @Amount = 250.00;
```

```
SELECT *
FROM Reimbursements;
```

Output:

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (53)\*' (selected), 'Insert SQL QUery.s... (JAINAM\jaina (79))\*', and 'SQLQuery2.sql - JAI... (JAINAM\jaina (80))'. The 'Testing.sql' tab contains the following code:

```
/* 4) Update Reimbursement Status
Update the reimbursement status of a specific application.

*/
CREATE PROCEDURE UpdateReimbursementStatus
    @ApplicationID INT,
    @Processed BIT,
    @Amount DECIMAL(10, 2)
AS
BEGIN
    UPDATE Reimbursements
    SET Processed = @Processed,
        Amount = @Amount
    WHERE ApplicationID = @ApplicationID;
END;

EXEC UpdateReimbursementStatus @ApplicationID = 1, @Processed = 1, @Amount = 250.00;

SELECT *
FROM Reimbursements;
```

The 'Messages' pane at the bottom shows a green checkmark icon and the text 'Query executed successfully.' The 'Results' pane displays the following table:

|    | ReimbursementID | ApplicationID | RequestDate | Processed | Amount |
|----|-----------------|---------------|-------------|-----------|--------|
| 1  | 1               | 1             | 2024-04-18  | 1         | 250.00 |
| 2  | 2               | 2             | 2024-04-23  | 1         | 150.00 |
| 3  | 3               | 3             | 2024-03-30  | 0         | 0.00   |
| 4  | 4               | 4             | 2024-01-30  | 1         | 100.00 |
| 5  | 5               | 5             | 2023-04-30  | 0         | 0.00   |
| 6  | 6               | 6             | 2024-04-18  | 1         | 250.00 |
| 7  | 7               | 7             | 2024-04-14  | 1         | 180.00 |
| 8  | 8               | 8             | 2024-04-19  | 0         | 0.00   |
| 9  | 9               | 9             | 2024-04-11  | 1         | 300.00 |
| 10 | 10              | 10            | 2024-04-09  | 1         | 220.00 |

Fig 21: UpdateReimbursementStatus for ApplicationID Amount from 200 to 250

#### Explanation:

This stored procedure updates the reimbursement status of a specific application in the Reimbursements table. It takes three parameters: ApplicationID, Processed, and Amount. It then updates the Processed and Amount fields for the specified ApplicationID.

v. GenerateMonthlyReport

Objective: Generating a monthly report summarizing various metrics such as the number of applications, interviews conducted, evaluations, and background checks for each month.

Query:

```
CREATE PROCEDURE GenerateMonthlyReport
    @Month INT,
    @Year INT
AS
BEGIN
    SELECT
        MONTH(iv.StartTime) AS Month,
        YEAR(iv.StartTime) AS Year,
        COUNT(DISTINCT a.ApplicationID) AS NumberOfApplications,
        COUNT(DISTINCT iv.InterviewID) AS NumberOfInterviews,
        COUNT(DISTINCT e.EvaluationID) AS NumberOfEvaluations,
        COUNT(DISTINCT bc.CheckID) AS NumberOfBackgroundChecks
    FROM Interviews iv
    LEFT JOIN Applications a ON iv.ApplicationID = a.ApplicationID
    LEFT JOIN Evaluations e ON a.ApplicationID = e.ApplicationID
    LEFT JOIN BackgroundChecks bc ON a.CandidateID = bc.CandidateID
    WHERE MONTH(iv.StartTime) = @Month AND YEAR(iv.StartTime) = @Year
    GROUP BY MONTH(iv.StartTime), YEAR(iv.StartTime);
END

EXEC GenerateMonthlyReport @Month = 4, @Year = 2024;
```

Output:

The screenshot shows the SQL Server Management Studio interface. In the top bar, there are three tabs: 'Testing.sql - JAINAM\jaina (53)\*', 'Insert SQL QUery.s... (JAINAM\jaina (79))\*', and 'SQLQuery2.sql - JAI... (JAINAM\jaina (80))'. The main area displays a T-SQL script for creating a stored procedure:

```
/*
5) GenerateMonthlyReport
a monthly report summarizing various metrics such as the number of applications,
interviews conducted, evaluations, and background checks for each month.

CREATE PROCEDURE GenerateMonthlyReport
    @Month INT,
    @Year INT
AS
BEGIN
    SELECT
        MONTH(iv.StartTime) AS Month,
        YEAR(iv.StartTime) AS Year,
        COUNT(DISTINCT a.ApplicationID) AS NumberOfApplications,
        COUNT(DISTINCT iv.InterviewID) AS NumberOfInterviews,
        COUNT(DISTINCT e.EvaluationID) AS NumberOfEvaluations,
        COUNT(DISTINCT bc.CheckID) AS NumberOfBackgroundChecks
    FROM Interviews iv
    LEFT JOIN Applications a ON iv.ApplicationID = a.ApplicationID
    LEFT JOIN Evaluations e ON a.ApplicationID = e.ApplicationID
    LEFT JOIN BackgroundChecks bc ON a.CandidateID = bc.CandidateID
    WHERE MONTH(iv.StartTime) = @Month AND YEAR(iv.StartTime) = @Year
    GROUP BY MONTH(iv.StartTime), YEAR(iv.StartTime);
END

EXEC GenerateMonthlyReport @Month = 4, @Year = 2024;
```

Below the script, the results pane shows a table with one row of data:

|   | Month | Year | NumberOfApplications | NumberOfInterviews | NumberOfEvaluations | NumberOfBackgroundChecks |
|---|-------|------|----------------------|--------------------|---------------------|--------------------------|
| 1 | 4     | 2024 | 9                    | 9                  | 9                   | 9                        |

At the bottom of the results pane, a message indicates: 'Query executed successfully.' and the server name 'JAINAM\SQLEXPRESS (16.0)'.

Fig 22: Displaying Stored procedure for Monthly report Generation

Explanation: This stored procedure is the most important in the entire database as it gives the summary month by month on creating a monthly report, tallying applications, interviews, evaluations, and background checks for a given month and year. It accepts @Month and @Year as inputs. It retrieves data from relevant tables, counts distinct IDs, and organizes results by month and year.

**iii) Functions:**

i. FormatPhoneNumber

Objective: The objective of this function is to format a phone number in the format XXX-XXX-XXXX.

Query:

```
-- Create the FormatPhoneNumber function
CREATE FUNCTION FormatPhoneNumber(@PhoneNumber NVARCHAR(20))
RETURNS NVARCHAR(20)
AS
BEGIN
    DECLARE @FormattedPhoneNumber NVARCHAR(20);
    -- Remove any non-numeric characters from the input phone number
    SET @PhoneNumber = REPLACE(@PhoneNumber, '+', '');

    -- Add dashes to the phone number in the format XXX-XXX-XXXX
    SET @FormattedPhoneNumber = LEFT(@PhoneNumber, 3) + '-' +
    SUBSTRING(@PhoneNumber, 4, 3) + '-' + RIGHT(@PhoneNumber, 4);

    RETURN @FormattedPhoneNumber;
END;
GO
-- Update the phone numbers for all candidates using the FormatPhoneNumber function
UPDATE Candidates
SET Phone = dbo.FormatPhoneNumber(Phone);
-- Execute the function with a sample phone number
SELECT *
FROM Candidates;
```

Output:

The screenshot shows a SQL Server Management Studio window with two panes. The left pane displays a script named 'Testing.sql' containing T-SQL code to create a function and update a table. The right pane shows the results of the query, displaying a table of candidate data with formatted phone numbers.

```
/*
1)      FormatPhoneNumber
format a phone number consistently according to a specified format.
*/
-- Create the FormatPhoneNumber function
CREATE FUNCTION FormatPhoneNumber(@PhoneNumber NVARCHAR(20))
RETURNS NVARCHAR(20)
AS
BEGIN
    DECLARE @FormattedPhoneNumber NVARCHAR(20);
    -- Remove any non-numeric characters from the input phone number
    SET @PhoneNumber = REPLACE(@PhoneNumber, '-', '');
    /*
    -- Add dashes to the phone number in the format XXX-XXX-XXXX
    SET @FormattedPhoneNumber = LEFT(@PhoneNumber, 3) + '-' + SUBSTRING(@PhoneNumber, 4, 3) + '-' + RIGHT(@PhoneNumber, 4);
    */

    RETURN @FormattedPhoneNumber;
END;
GO
-- Update the phone numbers for all candidates using the FormatPhoneNumber function
UPDATE Candidates
SET Phone = dbo.FormatPhoneNumber(Phone);
-- Execute the function with a sample phone number
SELECT *
FROM Candidates;
```

Results

|    | CandidateID | Name           | Email                | Phone        | ShortProfile       |
|----|-------------|----------------|----------------------|--------------|--------------------|
| 1  | 1           | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Data Scientist     |
| 2  | 2           | Jackie Chan    | jackie@hotmail.com   | 234-567-8901 | Data Scientist     |
| 3  | 3           | Gal Gadot      | gal@syr.edu          | 345-678-9012 | Java Developer     |
| 4  | 4           | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Software Engineer  |
| 5  | 5           | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Frontend Developer |
| 6  | 6           | Janvi Saddi    | janvi@syr.edu        | 678-901-2345 | Frontend Developer |
| 7  | 7           | Bhavya Shah    | bhavya@gmail.com     | 789-012-3456 | Data Analyst       |
| 8  | 8           | Yash Solanki   | yash@hotmail.com     | 890-123-4567 | Data Analyst       |
| 9  | 9           | Robert         | robert66@gmail.com   | 315-952-7506 | Software Engineer  |
| 10 | 10          | Sahil Shah     | sahil@gmail.com      | 012-345-6789 | Data Analyst       |

Query executed successfully.

Fig 23: FormatPhoneNumber in format XXX-XXX-XXXX

### Explanation:

The function takes a phone number as input and removes any non-numeric characters using the REPLACE function. It then constructs the formatted phone number by concatenating the appropriate substrings with dashes in between. Finally, it returns the formatted phone number.

ii. GenerateUsername

Objective:

This function generates a username based on the candidate's name and a random number.

Query:

```
/*
    2) GenerateUsername
    Generates a username based on the candidate's name and a random
    number.
*/
CREATE FUNCTION GenerateUsername
    (@FullName NVARCHAR(100))
RETURNS NVARCHAR(100)
AS
BEGIN
    DECLARE @Username NVARCHAR(100);

    -- Check if the full name contains a space character
    IF CHARINDEX(' ', @FullName) > 0
        BEGIN
            DECLARE @FirstName NVARCHAR(50);
            DECLARE @LastName NVARCHAR(50);

            -- Extract first name and last name from the full name
            SET @FirstName = LEFT(@FullName, CHARINDEX(' ', @FullName) - 1);
            SET @LastName = SUBSTRING(@FullName, CHARINDEX(' ', @FullName) + 1,
LEN(@FullName) - CHARINDEX(' ', @FullName));

            -- Concatenate the first three letters of the first name and last name
            SET @Username = LEFT(@FirstName, 3) + LEFT(@LastName, 3);
        END
    ELSE
        BEGIN
            -- If the full name does not contain a space, use the entire name as the username
            SET @Username = @FullName;
        END

    RETURN @Username;
END;
```

```
SELECT Name, Email, Phone, ShortProfile, dbo.GenerateUsername(Name) AS
GeneratedUsername
```

Jainam Rajendra Shah  
SUID: 785948053

FROM Candidates;

Output:

The screenshot shows the SQL Server Management Studio interface. The top bar has tabs for 'Testing.sql - JAINAM\jaina (66)', 'SQLQuery2.sql - JAI...(JAINAM\jaina (59))', and 'Insert SQL QUery.s...(JAINAM\jaina (51))'. The main area displays the following T-SQL code:

```
/*
    2) GenerateUsername
        Generates a username based on the candidate's name and a random number.
*/
CREATE FUNCTION GenerateUsername
    (@FullName NVARCHAR(100))
RETURNS NVARCHAR(100)
AS
BEGIN
    DECLARE @Username NVARCHAR(100);

    -- Check if the full name contains a space character
    IF CHARINDEX(' ', @FullName) > 0
    BEGIN
        DECLARE @FirstName NVARCHAR(50);
        DECLARE @LastName NVARCHAR(50);

        -- Extract first name and last name from the full name
        SET @FirstName = LEFT(@FullName, CHARINDEX(' ', @FullName) - 1);
        SET @LastName = SUBSTRING(@FullName, CHARINDEX(' ', @FullName) + 1, LEN(@FullName) - CHARINDEX(' ', @F

        -- Concatenate the first three letters of the first name and last name
        SET @Username = LEFT(@FirstName, 3) + LEFT(@LastName, 3);
    END
    ELSE
    BEGIN
        -- If the full name does not contain a space, use the entire name as the username
        SET @Username = @FullName;
    END
END
```

The results pane shows a table with columns: Name, Email, Phone, ShortProfile, and GeneratedUsername. The data is as follows:

|   | Name           | Email                | Phone        | ShortProfile       | GeneratedUsername |
|---|----------------|----------------------|--------------|--------------------|-------------------|
| 1 | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Data Scientist     | JaiSha            |
| 2 | Jackie Chan    | jackie@hotmail.com   | 234-567-8901 | Data Scientist     | JacCha            |
| 3 | Gal Gadot      | gal@syr.edu          | 345-678-9012 | Java Developer     | GalGad            |
| 4 | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Software Engineer  | MarRob            |
| 5 | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Frontend Developer | DikPat            |
| 6 | Janvi Sadhi    | janvi@syr.edu        | 678-901-2345 | Frontend Developer | JanSad            |
| 7 | Bhavya Shah    | bhavya@gmail.com     | 789-012-3456 | Data Analyst       | BhaSha            |

At the bottom, a message says 'Query executed successfully.' and the status bar shows 'Ln 303 Col 18 Ch 6 INS'.

Fig 24: Function for generating Username

The screenshot shows a SQL Server Management Studio window. The top pane displays a T-SQL script named 'Testing.sql' which defines a function 'dbo.GenerateUsername'. The function takes a parameter '@FullName' and returns a generated username. It uses logic to either extract the first three letters of the first name and last name or return the entire full name if there is no space. The bottom pane shows the results of executing a query that calls this function for 10 rows of candidate data from a 'Candidates' table.

```
Testing.sql - JAINAM\jaina (66)  X SQLQuery2.sql - JAI...(JAINAM\jaina (59))      Insert SQL QUery.s...(JAINAM\jaina (51))

SET @LastName = SUBSTRING(@FullName, CHARINDEX(' ', @FullName) + 1, LEN(@FullName) - CHARINDEX(' ', @FullName))

-- Concatenate the first three letters of the first name and last name
SET @Username = LEFT(@FirstName, 3) + LEFT(@LastName, 3);

END
ELSE
BEGIN
    -- If the full name does not contain a space, use the entire name as the username
    SET @Username = @FullName;
END

RETURN @Username;
END;

SELECT Name, Email, Phone, ShortProfile, dbo.GenerateUsername(Name) AS GeneratedUsername
FROM Candidates;
```

|    | Name           | Email                | Phone        | ShortProfile       | GeneratedUsername |
|----|----------------|----------------------|--------------|--------------------|-------------------|
| 1  | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Data Scientist     | JaiSha            |
| 2  | Jackie Chan    | jackie@hotmail.com   | 234-567-8901 | Data Scientist     | JacCha            |
| 3  | Gal Gadot      | gal@syr.edu          | 345-678-9012 | Java Developer     | GalGad            |
| 4  | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Software Engineer  | MarRob            |
| 5  | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Frontend Developer | DikPat            |
| 6  | Janvi Sadhi    | janvi@syr.edu        | 678-901-2345 | Frontend Developer | JanSad            |
| 7  | Bhavya Shah    | bhavya@gmail.com     | 789-012-3456 | Data Analyst       | BhaSha            |
| 8  | Yash Solanki   | yash@hotmail.com     | 890-123-4567 | Data Analyst       | YasSol            |
| 9  | Robert         | robert66@gmail.com   | 315-952-7506 | Software Engineer  | Robert            |
| 10 | Sahil Shah     | sahil@gmail.com      | 012-345-6789 | Data Analyst       | SahSha            |

Query executed successfully.

Fig 25: Displaying Generated Username using Function and logic for first and last name

Explanation:

This function first checks if the full name contains a space character. If it does, it extracts the first three letters of the first name and last name to generate the username. If the full name does not contain a space, it uses the entire name as the username. So making sure if the person enters either full name or just first name both scenarios are being taken care off.

iii. CalculateGrade

Objective: Calculates the grade based on the score obtained by a student in an exam

Query:

```
CREATE FUNCTION CalculateGrade
    (@Score FLOAT)
RETURNS NVARCHAR(10)
AS
BEGIN
    DECLARE @Grade NVARCHAR(10);
    IF @Score >= 90
        SET @Grade = 'A';
    ELSE IF @Score >= 80
        SET @Grade = 'B';
    ELSE IF @Score >= 70
        SET @Grade = 'C';
    ELSE IF @Score >= 60
        SET @Grade = 'D';
    ELSE
        SET @Grade = 'F';
    RETURN @Grade;
END;
```

SELECT \*, dbo.CalculateGrade(Grade) AS Grade  
FROM Tests;

Output:

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (66)\*', 'SQLQuery2.sql - JAI... (JAINAM\jaina (59))', and 'Insert SQL Query.s... (JAINAM\jaina (51))'. The main pane displays a T-SQL script for creating a function named 'CalculateGrade'.

```
/* 3) CalculateGrade
calculates the grade based on the score obtained by a student in an exam */
CREATE FUNCTION CalculateGrade
    (@Score FLOAT)
RETURNS NVARCHAR(10)
AS
BEGIN
    DECLARE @Grade NVARCHAR(10);
    IF @Score >= 90
        SET @Grade = 'A';
    ELSE IF @Score >= 80
        SET @Grade = 'B';
    ELSE IF @Score >= 70
        SET @Grade = 'C';
    ELSE IF @Score >= 60
        SET @Grade = 'D';
    ELSE
        SET @Grade = 'F';
    RETURN @Grade;
END;

SELECT *, dbo.CalculateGrade(Grade) AS Grade
FROM Tests;
```

The results pane shows a table with 10 rows of test data. The columns are: TestID, ApplicationID, Type, StartTime, EndTime, Answers, Grade, and Grade. The 'Grade' column contains the calculated grades based on the 'Answers' column.

| TestID | ApplicationID | Type            | StartTime               | EndTime                 | Answers  | Grade | Grade |
|--------|---------------|-----------------|-------------------------|-------------------------|--|-------|-------|
| 1      | 1             | OA              | 2024-04-18 09:00:00.000 | 2024-04-18 10:00:00.000 | A, B, C, D, A  | 85.50 | B     |
| 2      | 2             | Behavioural MCQ | 2024-04-23 10:00:00.000 | 2024-04-23 11:00:00.000 | Strongly Agree, Agree, Neutral, Disagree, Strongl... | 70.00 | C     |
| 3      | 3             | Aptitude        | 2024-03-30 09:00:00.000 | 2024-03-30 10:00:00.000 | 25, 36, 42, 55, 63                                   | 90.50 | A     |
| 4      | 4             | OA              | 2024-01-30 09:30:00.000 | 2024-01-30 10:30:00.000 | A, C, D, B   | 78.00 | C     |
| 5      | 5             | Behavioural MCQ | 2023-04-30 10:00:00.000 | 2023-04-30 11:00:00.000 | Disagree, Strongly Agree, Neutral, Agree, Disagree   | 65.50 | D     |
| 6      | 6             | Aptitude        | 2024-04-18 10:30:00.000 | 2024-04-18 11:30:00.000 | 30, 42, 56, 68, 75                                   | 82.00 | B     |
| 7      | 7             | OA              | 2024-04-14 11:00:00.000 | 2024-04-14 12:00:00.000 | C, A, B, D, C  | 88.50 | B     |
| 8      | 8             | Behavioural MCQ | 2024-04-19 11:30:00.000 | 2024-04-19 12:30:00.000 | Neutral, Agree, Disagree, Strongly Agree, Strongl... | 72.00 | C     |
| 9      | 9             | Aptitude        | 2024-04-11 12:00:00.000 | 2024-04-11 13:00:00.000 | 40, 51, 64, 78, 85                                   | 95.00 | A     |
| 10     | 10            | OA              | 2024-04-09 12:30:00.000 | 2024-04-09 13:30:00.000 | B, B, D, A, C  | 80.50 | B     |

Query executed successfully.

Fig 26: Calculate the grade based on the score

### Explanation:

This query will calculate the grade for each test based on the score ('Grade' column) and display it in the result set alongside other test details.

iv. GenerateEmployeeID

Objective: Generates an employee ID based on the first three letters of the first name and the last three letters of the last name, along with a sequence number.

Query:

Output:

The screenshot shows the SQL Server Management Studio interface. The top bar has tabs for 'Testing.sql - JAINA..(JAINAM\jaina (66)\*' and 'SQLQuery2.sql - JAI... (JAINAM\jaina (59))'. The main pane displays the following T-SQL code:

```
/*
4) GenerateEmployeeID
generates an employee ID based on the first three letters of the first name and
the last three letters of the last name, along with a sequence number.
*/
CREATE FUNCTION GenerateEmployeeID
    (@CandidateID INT)
RETURNS NVARCHAR(20)
AS
BEGIN
    DECLARE @EmployeeID NVARCHAR(20);

    -- Concatenate the first three letters of the first name and last name, and a sequential number
    SELECT @EmployeeID = LEFT(c.Name, 3) + RIGHT(c.Name, 3) + CAST(ROW_NUMBER() OVER (ORDER BY e.EvaluationID) AS NVARCHAR(10))
    FROM Candidates c
    INNER JOIN Evaluations e ON c.CandidateID = e.ApplicationID
    WHERE e.Result = 'Pass' AND c.CandidateID = @CandidateID;

    RETURN @EmployeeID;
END;

SELECT c.CandidateID, c.Name AS CandidateName, dbo.GenerateEmployeeID(c.CandidateID) AS EmployeeID
FROM Candidates c
INNER JOIN Evaluations e ON c.CandidateID = e.ApplicationID
WHERE e.Result = 'Pass';
```

The bottom pane shows the 'Results' tab with the following output:

|   | CandidateID | CandidateName  | EmployeeID |
|---|-------------|----------------|------------|
| 1 | 1           | Jainam Shah    | Jaiyah1    |
| 2 | 4           | Margot Robbie  | Marbie1    |
| 3 | 5           | Dikshita Patel | Dikte1     |
| 4 | 8           | Yash Solanki   | Yasnki1    |
| 5 | 9           | Robert         | Robert1    |

A message at the bottom of the results pane says 'Query executed successfully.'

Fig 27: Generates an employee ID

Explanation:

This function takes the CandidateID as input and retrieves the candidate's name from the Candidates table. It then joins the Candidates table with the Evaluations table to filter only the evaluations where the result is 'Pass' for the specified candidate. Finally, it concatenates the first three letters of the candidate's name, the last three letters of the candidate's name, and a sequential number generated by the ROW\_NUMBER() function as the employee ID.

Jainam Rajendra Shah  
SUID: 785948053

v. CheckBackgroundCheckStatus  
Objective: To check the status of a candidate's background check.

Query:

```
CREATE FUNCTION CheckBackgroundCheckStatus
    (@CandidateID INT)
RETURNS NVARCHAR(20)
AS
BEGIN
    DECLARE @Status NVARCHAR(20);

    SELECT @Status = Status
    FROM BackgroundChecks
    WHERE CandidateID = @CandidateID;

    RETURN @Status;
END;
```

```
SELECT *, dbo.CheckBackgroundCheckStatus(c.CandidateID) AS BackgroundCheckStatus
FROM Candidates AS c;
```

Output:

Fig :

The screenshot shows the SQL Server Management Studio interface. In the top bar, there are three tabs: 'Testing.sql - JAINAM\jaina (66)', 'SQLQuery2.sql - JAI...(JAINAM\jaina (59))', and 'Insert SQL QUery.s...(JAINAM\jaina (51))'. The main area displays a T-SQL script for creating a function named 'CheckBackgroundCheckStatus'. The script includes a comment block, the function definition with parameters, and a select statement to retrieve the status from the 'BackgroundChecks' table. Below the script, a query is executed to select all columns from the 'Candidates' table and include the result of the function as a column named 'BackgroundCheckStatus'. The results grid shows 10 rows of candidate data, each with their background check status listed in the last column.

|    | CandidateID | Name           | Email                | Phone        | ShortProfile       | BackgroundCheckStatus |
|----|-------------|----------------|----------------------|--------------|--------------------|-----------------------|
| 1  | 1           | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Data Scientist     | Clear                 |
| 2  | 2           | Jackie Chan    | jackie@hotmail.com   | 234-567-8901 | Data Scientist     | Pending               |
| 3  | 3           | Gal Gadot      | gal@syr.edu          | 345-678-9012 | Java Developer     | Clear                 |
| 4  | 4           | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Software Engineer  | Clear                 |
| 5  | 5           | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Frontend Developer | Clear                 |
| 6  | 6           | Janvi Sadhi    | janvi@syr.edu        | 678-901-2345 | Frontend Developer | Clear                 |
| 7  | 7           | Bhavya Shah    | bhavya@gmail.com     | 789-012-3456 | Data Analyst       | Clear                 |
| 8  | 8           | Yash Solanki   | yash@hotmail.com     | 890-123-4567 | Data Analyst       | Pending               |
| 9  | 9           | Robert         | robert66@gmail.com   | 315-952-7506 | Software Engineer  | Clear                 |
| 10 | 10          | Sahil Shah     | sahil@gmail.com      | 012-345-6789 | Data Analyst       | Clear                 |

Fig 28: Display CheckBackgroundCheckStatus

Explanation:

This function takes a CandidateID as input and retrieves the background check status for that candidate from the BackgroundChecks table. It then returns the status.

**iv) Triggers:**

- a) OnEvaluationInsertTrigger

Objective: Update the candidate's status based on the evaluation result.

Query:

```
CREATE TRIGGER OnEvaluationInsertTrigger
ON Evaluations
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ApplicationID INT;
    DECLARE @Result NVARCHAR(10);

    SELECT @ApplicationID = inserted.ApplicationID,
           @Result = inserted.Result
    FROM inserted;

    UPDATE StatusFlow
    SET status = CASE
        WHEN @Result = 'Pass' THEN 'Accepted'
        WHEN @Result = 'Fail' THEN 'Rejected'
        ELSE 'Pending'
    END
    WHERE candidate_id = @ApplicationID;
```

END;  
Output:

The screenshot shows the SQL Server Management Studio interface. A new query window titled 'Testing.sql - JAINAM\jaina (78)' is open, showing the creation of a trigger. The code is as follows:

```
/* Triggers */
/*
    1) OnEvaluationInsertTrigger
    Update the candidate's status based on the evaluation result.
*/
CREATE TRIGGER OnEvaluationInsertTrigger
ON Evaluations
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ApplicationID INT;
    DECLARE @Result NVARCHAR(10);

    SELECT @ApplicationID = inserted.ApplicationID,
           @Result = inserted.Result
    FROM inserted;

    UPDATE StatusFlow
    SET status = CASE
        WHEN @Result = 'Pass' THEN 'Accepted'
        WHEN @Result = 'Fail' THEN 'Rejected'
        ELSE 'Pending'
    END
    WHERE candidate_id = @ApplicationID;
END;
```

The execution results pane at the bottom shows a successful execution message: 'Query executed successfully.' and 'Completion time: 2024-04-29T17:56:34.6353954-04:00'. The status bar indicates the command was an 'INS'.

Fig 29: OnEvaluationInsertTrigger

Explanation: This trigger fires after an insertion into the Evaluations table. It retrieves the ApplicationID and Result of the newly inserted evaluation. Based on the evaluation result, it updates the status of the candidate in the StatusFlow table. If the result is 'Pass', the status is set to 'Accepted', if 'Fail', the status is set to 'Rejected', and otherwise, it remains 'Pending'.

b) OnInterviewInsertTrigger

Objective: Update the status flow of the candidate based on interview scheduling.

Query:

```
CREATE TRIGGER OnInterviewInsertTrigger
ON Interviews
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ApplicationID INT;
    DECLARE @Status NVARCHAR(50);

    SELECT @ApplicationID = inserted.ApplicationID
    FROM inserted;

    SET @Status = 'Interview Scheduled';

    INSERT INTO StatusFlow (candidate_id, status, timestamp)
    VALUES (@ApplicationID, @Status, GETDATE());
END;
```

Output:

The screenshot shows a SQL query window titled 'Testing.sql - JAINA... (JAINAM\jaina (78))' with a tab 'Insert SQL Query.s... (JAINAM\jaina (51))'. The code is as follows:

```
/*
2) OnInterviewInsertTrigger
Update the status flow of the candidate based on interview scheduling.

CREATE TRIGGER OnInterviewInsertTrigger
ON Interviews
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ApplicationID INT;
    DECLARE @Status NVARCHAR(50);

    SELECT @ApplicationID = inserted.ApplicationID
    FROM inserted;

    SET @Status = 'Interview Scheduled';

    INSERT INTO StatusFlow (candidate_id, status, timestamp)
    VALUES (@ApplicationID, @Status, GETDATE());
END;
```

Below the code, the message bar says 'Commands completed successfully.' and 'Completion time: 2024-04-29T18:00:12.3974370-04:00'. At the bottom, another message bar says 'Query executed successfully.'

Fig 30: OnInterviewInsertTrigger

Explanation:

This trigger fires after an insertion into the Interviews table. It updates the StatusFlow table by adding a new record indicating that an interview has been scheduled for the corresponding candidate.

c) OnDrugTestInsertTrigger

Objective: Update the candidate's status based on the result of the drug test.

Query:

```
CREATE TRIGGER OnDrugTestInsertTrigger
ON DrugTests
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CandidateID INT;
    DECLARE @TestResult NVARCHAR(50);
    DECLARE @NewStatus NVARCHAR(50);

    SELECT @CandidateID = inserted.CandidateID,
           @TestResult = inserted.Results
    FROM inserted;

    IF @TestResult = 'Negative'
        SET @NewStatus = 'Cleared';
    ELSE
        SET @NewStatus = 'Pending Further Review';

    UPDATE StatusFlow
    SET status = @NewStatus
    WHERE candidate_id = @CandidateID;
END;
```

Output:

The screenshot shows a SQL editor window titled "Testing.sql - JAINAM\jaina (78)\*". The code is a CREATE TRIGGER statement for "OnDrugTestInsertTrigger" on the "DrugTests" table, which executes AFTER INSERT. The trigger updates the "StatusFlow" table based on the drug test result. If the result is "Negative", the status is set to "Cleared"; otherwise, it is set to "Pending Further Review". The trigger begins with a comment block: /\* 3) OnDrugTestInsertTrigger Update the candidate's status based on the result of the drug test. \*/. The code includes variable declarations (@CandidateID INT, @TestResult NVARCHAR(50), @NewStatus NVARCHAR(50)), a SELECT statement to get the new candidate ID and test result from the inserted row, an IF statement to determine the new status, and an UPDATE statement to update the StatusFlow table. The trigger ends with an END; keyword. Below the code, the message pane shows "Commands completed successfully." and the completion time: 2024-04-29T18:17:55.4129889-04:00.

```
/*
3) OnDrugTestInsertTrigger
Update the candidate's status based on the result of the drug test.
*/
CREATE TRIGGER OnDrugTestInsertTrigger
ON DrugTests
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CandidateID INT;
    DECLARE @TestResult NVARCHAR(50);
    DECLARE @NewStatus NVARCHAR(50);

    SELECT @CandidateID = inserted.CandidateID,
           @TestResult = inserted.Results
    FROM inserted;

    IF @TestResult = 'Negative'
        SET @NewStatus = 'Cleared';
    ELSE
        SET @NewStatus = 'Pending Further Review';

    UPDATE StatusFlow
    SET status = @NewStatus
    WHERE candidate_id = @CandidateID;
END;
```

100 %

Messages

Commands completed successfully.

Completion time: 2024-04-29T18:17:55.4129889-04:00

Fig 31: OnDrugTestInsertTrigger

Explanation:

This trigger executes after an insertion into the DrugTests table. It updates the StatusFlow table based on the drug test result of the candidate. If the result is negative, the status is updated to "Cleared"; otherwise, it is set to "Pending Further Review".

d) OnBackgroundCheckInsertTrigger

Objective: Update the candidate's status based on the result of the background check.

Query:

```
CREATE TRIGGER OnBackgroundCheckInsertTrigger
ON BackgroundChecks
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CandidateID INT;
    DECLARE @CheckResult NVARCHAR(50);
    DECLARE @NewStatus NVARCHAR(50);

    SELECT @CandidateID = inserted.CandidateID,
           @CheckResult = inserted.Status
    FROM inserted;

    IF @CheckResult = 'Clear'
        SET @NewStatus = 'Cleared';
    ELSE
        SET @NewStatus = 'Pending Further Review';

    UPDATE StatusFlow
    SET status = @NewStatus
    WHERE candidate_id = @CandidateID;
END;
```

Output:

The screenshot shows a SQL script named 'Testing.sql' in the 'JAINAM\jaina (78)' database. The script contains a CREATE TRIGGER statement for 'OnBackgroundCheckInsertTrigger' on the 'BackgroundChecks' table, which fires after an insert. The trigger updates the 'StatusFlow' table based on the background check result. If the result is 'Clear', the status is set to 'Cleared'; otherwise, it is set to 'Pending Further Review'. The script includes comments explaining the purpose of the trigger and its logic. The execution completed successfully with a completion time of 2024-04-29T18:52:46.7762375-04:00.

```
/* 4) OnBackgroundCheckInsertTrigger
   Update the candidate's status based on the result of the background check. */

CREATE TRIGGER OnBackgroundCheckInsertTrigger
ON BackgroundChecks
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CandidateID INT;
    DECLARE @CheckResult NVARCHAR(50);
    DECLARE @NewStatus NVARCHAR(50);

    SELECT @CandidateID = inserted.CandidateID,
           @CheckResult = inserted.Status
    FROM inserted;

    IF @CheckResult = 'Clear'
        SET @NewStatus = 'Cleared';
    ELSE
        SET @NewStatus = 'Pending Further Review';

    UPDATE StatusFlow
    SET status = @NewStatus
    WHERE candidate_id = @CandidateID;
END;
```

100 %

Messages

Commands completed successfully.

Completion time: 2024-04-29T18:52:46.7762375-04:00

Fig 32: OnBackgroundCheckInsertTrigger

#### Explanation:

This trigger fires after an insertion into the `BackgroundChecks` table. It updates the `StatusFlow` table based on the background check result of the candidate. If the result is 'Clear', the status is updated to 'Cleared'; otherwise, it is set to 'Pending Further Review'.

e) OnJobPlatformJobInsertTrigger

Objective: Maintain consistency between job platforms and job listings upon insertion.

Query:

```
CREATE TRIGGER OnJobPlatformJobInsertTrigger
ON JobPlatformJob
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @JobID INT;
    DECLARE @JobPlatformID INT;

    SELECT @JobID = inserted.job_id,
           @JobPlatformID = inserted.job_platform_id
    FROM inserted;

    IF NOT EXISTS (
        SELECT 1
        FROM Job
        WHERE JobID = @JobID
    )
    BEGIN
        DELETE FROM JobPlatformJob
        WHERE job_id = @JobID AND job_platform_id = @JobPlatformID;
        RAISERROR('Job ID does not exist. Entry in JobPlatformJob table deleted.', 16, 1);
    END;
END;
```

Output:

The screenshot shows a SQL query window in SSMS. The code is a T-SQL trigger definition:

```
/*  
 * 5) OnJobPlatformJobInsertTrigger  
 * Maintain consistency between job platforms and job listings upon insertion.  
 */  
  
CREATE TRIGGER OnJobPlatformJobInsertTrigger  
ON JobPlatformJob  
AFTER INSERT  
AS  
BEGIN  
    SET NOCOUNT ON;  
  
    DECLARE @JobID INT;  
    DECLARE @JobPlatformID INT;  
  
    SELECT @JobID = inserted.job_id,  
           @JobPlatformID = inserted.job_platform_id  
    FROM inserted;  
  
    IF NOT EXISTS (  
        SELECT 1  
        FROM Job  
        WHERE JobID = @JobID  
    )  
    BEGIN  
        DELETE FROM JobPlatformJob  
        WHERE job_id = @JobID AND job_platform_id = @JobPlatformID;  
        RAISERROR('Job ID does not exist. Entry in JobPlatformJob table deleted.', 16, 1);  
    END;  
END;
```

Below the code, the message "Commands completed successfully." is displayed, along with the completion time: 2024-04-29T10:55:39.1501956-04:00.

In the status bar at the bottom, it says "100 %", "Query executed successfully.", and "JAINAM\SQLEXPRESS".

Fig 33: OnJobPlatformJobInsertTrigger

Explanation:

This trigger fires after an insertion into the JobPlatformJob table. It ensures consistency between job platforms and job listings by checking if the job ID exists in the Job table. If the job ID does not exist, it deletes the corresponding entry from the JobPlatformJob table and raises an error.

v) **Transactions:**

a. Hiring a Candidate:

Objective: To officially hire a candidate after successful completion of evaluations, interviews, and background checks.

Query:

BEGIN TRANSACTION HireCandidate;

```
UPDATE StatusFlow
SET status = 'Hired'
WHERE candidate_id IN (
    SELECT DISTINCT C.CandidateID
    FROM Candidates C
    INNER JOIN Applications A ON C.CandidateID = A.CandidateID
    LEFT JOIN Evaluations E ON A.ApplicationID = E.ApplicationID
    LEFT JOIN Interviews I ON A.ApplicationID = I.ApplicationID
    LEFT JOIN BackgroundChecks BC ON C.CandidateID = BC.CandidateID
    WHERE E.Result = 'Pass' -- Candidate passed evaluations
    AND I.InterviewID IS NOT NULL -- Candidate has been interviewed
    AND BC.Status = 'Clear' -- Candidate's background check is clear
);
```

COMMIT TRANSACTION HireCandidate;

```
-- Displaying the details of candidates who have been hired
SELECT c.CandidateID, c.Name, c.Email, c.Phone, StatusFlow.status
FROM StatusFlow
JOIN Candidates c ON StatusFlow.candidate_id = c.CandidateID
WHERE StatusFlow.status = 'Hired';
```

Output:

The screenshot shows an SQL query window titled 'Testing.sql - JAINA...(JAINAM\jaina (78))' with a tab 'Insert SQL Query.s... (JAINAM\jaina (51))'. The code is a transaction script for hiring candidates:

```
/*
 1)      Hiring a Candidate:
 To officially hire a candidate after successful completion of evaluations, interviews, and background checks.
*/
BEGIN TRANSACTION HireCandidate;

-- UPDATE StatusFlow
SET status = 'Hired'
WHERE candidate_id IN (
  SELECT DISTINCT C.CandidateID
  FROM Candidates C
  INNER JOIN Applications A ON C.CandidateID = A.CandidateID
  LEFT JOIN Evaluations E ON A.ApplicationID = E.ApplicationID
  LEFT JOIN Interviews I ON A.ApplicationID = I.ApplicationID
  LEFT JOIN BackgroundChecks BC ON C.CandidateID = BC.CandidateID
  WHERE E.Result = 'Pass' -- Candidate passed evaluations
  AND I.InterviewID IS NOT NULL -- Candidate has been interviewed
  AND BC.Status = 'Clear' -- Candidate's background check is clear
);

COMMIT TRANSACTION HireCandidate;

-- Displaying the details of candidates who have been hired
SELECT c.CandidateID, c.Name, c.Email, c.Phone, StatusFlow.status
FROM StatusFlow
JOIN Candidates c ON StatusFlow.candidate_id = c.CandidateID
WHERE StatusFlow.status = 'Hired';
```

The results pane shows a table with the following data:

|   | CandidateID | Name           | Email                | Phone        | status |
|---|-------------|----------------|----------------------|--------------|--------|
| 1 | 1           | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Hired  |
| 2 | 4           | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Hired  |
| 3 | 5           | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Hired  |
| 4 | 9           | Robert         | robert66@gmail.com   | 315-952-7506 | Hired  |

Fig 32: Transaction for Hiring a Candidate

Explanation:

This SQL transaction updates the status of candidates to 'Hired' in the StatusFlow table if they have passed evaluations, participated in interviews, and cleared background checks. It selects candidates based on successful evaluations, existing interviews, and clear background check statuses. The BEGIN TRANSACTION and COMMIT TRANSACTION statements ensure that all updates occur atomically, either all updates succeed or none of them do, maintaining data integrity.

b. Updating Candidate Status:

Objective: To update the status of candidates based on evaluation results, interview outcomes, and background checks.

Query:

```
BEGIN TRANSACTION UpdateCandidateStatusTransaction;

-- Update candidate status based on evaluation, interview, and background check results
UPDATE StatusFlow
SET status =
CASE
    WHEN Evaluations.Result = 'Fail' THEN 'Rejected'
    WHEN BackgroundChecks.Status = 'Pending' THEN 'Under Review'
    ELSE 'Accepted'
END
FROM StatusFlow
JOIN Applications ON StatusFlow.candidate_id = Applications.CandidateID
LEFT JOIN Evaluations ON Applications.ApplicationID = Evaluations.ApplicationID
LEFT JOIN BackgroundChecks ON Applications.CandidateID =
BackgroundChecks.CandidateID;

COMMIT TRANSACTION UpdateCandidateStatusTransaction;

-- Displaying the updated status of candidates
SELECT Candidates.CandidateID, Candidates.Name, Candidates.Email, Candidates.Phone,
StatusFlow.status
FROM StatusFlow
JOIN Candidates ON StatusFlow.candidate_id = Candidates.CandidateID;
```

Output:

The screenshot shows a SQL query window titled "Testing.sql - JAINAM\jaina (78)" and a results window below it.

**Query Script:**

```
/*
 2) Updating Candidate Status:
 To update the status of candidates based on evaluation results, interview outcomes, and background checks.
 */

BEGIN TRANSACTION UpdateCandidateStatusTransaction;

-- Update candidate status based on evaluation, interview, and background check results
UPDATE StatusFlow
SET status =
CASE
    WHEN Evaluations.Result = 'Fail' THEN 'Rejected'
    WHEN BackgroundChecks.Status = 'Pending' THEN 'Under Review'
    ELSE 'Accepted'
END
FROM StatusFlow
JOIN Applications ON StatusFlow.candidate_id = Applications.CandidateID
LEFT JOIN Evaluations ON Applications.ApplicationID = Evaluations.ApplicationID
LEFT JOIN BackgroundChecks ON Applications.CandidateID = BackgroundChecks.CandidateID;

COMMIT TRANSACTION UpdateCandidateStatusTransaction;

-- Displaying the updated status of candidates
SELECT Candidates.CandidateID, Candidates.Name, Candidates.Email, Candidates.Phone, StatusFlow.status
FROM StatusFlow
JOIN Candidates ON StatusFlow.candidate_id = Candidates.CandidateID;
```

**Results Window:**

| CandidateID | Name           | Email                | Phone        | status       |
|-------------|----------------|----------------------|--------------|--------------|
| 1           | Jainam Shah    | jainam@gmail.com     | 123-456-7890 | Accepted     |
| 2           | Jackie Chan    | jackie@hotmail.com   | 234-567-8901 | Rejected     |
| 3           | Gal Gadot      | gal@syr.edu          | 345-678-9012 | Accepted     |
| 4           | Margot Robbie  | margot@gmail.com     | 456-789-0123 | Accepted     |
| 5           | Dikshita Patel | dikshita@hotmail.com | 567-890-1234 | Accepted     |
| 6           | Janvi Sadhi    | janvi@syr.edu        | 678-901-2345 | Rejected     |
| 7           | Bhavya Shah    | bhavya@gmail.com     | 789-012-3456 | Accepted     |
| 8           | Yash Solanki   | yash@hotmail.com     | 890-123-4567 | Under Review |
| 9           | Robert         | robert66@gmail.com   | 315-952-7506 | Accepted     |
| 10          | Sahil Shah     | sahil@gmail.com      | 012-345-6789 | Accepted     |

Fig 33: UpdateCandidateStatusTransactionFlow

#### Explanation:

This SQL transaction updates the status of candidates in the StatusFlow table based on the results of evaluations, interviews, and background checks. It uses a CASE statement to determine the new status.

c. Resolving Complaints:

Objective: To address and resolve any complaints raised by candidates during the hiring process, ensuring a positive candidate experience.

Query:

```
BEGIN TRANSACTION Resolve_Complaints;
```

```
-- Update the status of complaints to 'Resolved'
```

```
UPDATE Complaint
```

```
SET status = 'Resolved', resolution = 'Resolved'
```

```
WHERE status = 'Open';
```

```
COMMIT;
```

```
SELECT *
```

```
FROM Complaint
```

```
WHERE status = 'Resolved';
```

Output:

The screenshot shows the SQL Server Management Studio interface. The top window is titled 'Testing.sql - JAINA... (JAINAM\jaina (78))' and contains the transaction script. The bottom window is titled 'Results' and displays the output of the 'SELECT \*' query, showing a table of complaints resolved.

```
/* 3) Resolving Complaints:  
Objective: To address and resolve any complaints raised by candidates during the hiring process, ensuring a positive candidate experience.  
  
BEGIN TRANSACTION Resolve_Complaints;  
-- Update the status of complaints to 'Resolved'  
UPDATE Complaint  
SET status = 'Resolved', resolution = 'Resolved'  
WHERE status = 'Open';  
  
COMMIT;  
  
SELECT *  
FROM Complaint  
WHERE status = 'Resolved';  
  
/* 4) Handling Job Offer Negotiations:
```

| id | candidate_id | description                                    | status   | resolution |
|----|--------------|--|----------|------------|
| 1  | 1            | Cannot Start Behavioural Questions             | Resolved | Resolved   |
| 2  | 2            | Link not working                               | Resolved | Resolved   |
| 3  | 3            | Poor Connection hence could not give interview | Resolved | Resolved   |
| 4  | 4            | NULL   | Resolved | Resolved   |
| 5  | 5            | NULL   | Resolved | Resolved   |
| 6  | 6            | Link not working                               | Resolved | Resolved   |
| 7  | 7            | Cannot Start Behavioural Questions             | Resolved | Resolved   |
| 8  | 8            | Poor Connection hence could not give interview | Resolved | Resolved   |
| 9  | 9            | NULL   | Resolved | Resolved   |
| 10 | 10           | NULL   | Resolved | Resolved   |

Fig 34: Transaction for Resolving Complaints:

Explanation:

This transaction ensures that all complaints raised by candidates during the hiring process are addressed and resolved, contributing to a positive candidate experience

d. Handling Job Offer Negotiations:

Objective: To negotiate job offers with candidates, including salary, benefits, and start date.

Query:

BEGIN TRANSACTION;

-- Update the status of candidates who are negotiating job offers

UPDATE StatusFlow

SET status = 'Negotiating'

WHERE candidate\_id IN (

SELECT ApplicationID

FROM Evaluations

WHERE Result = 'Pass'

);

-- Display the updated status of candidates who are negotiating job offers

SELECT Candidates.CandidateID, Candidates.Name, Candidates.Email, StatusFlow.Status

FROM StatusFlow

JOIN Candidates ON StatusFlow.candidate\_id = Candidates.CandidateID

WHERE StatusFlow.status = 'Negotiating';

COMMIT TRANSACTION;

Output:

The screenshot shows a SQL query in the 'Testing.sql' file. The code is as follows:

```
WHERE status = 'Resolved';

/*
4) Handling Job Offer Negotiations:
Objective: To negotiate job offers with candidates, including salary, benefits, and start date.
*/

BEGIN TRANSACTION;

-- Update the status of candidates who are negotiating job offers
UPDATE StatusFlow
SET status = 'Negotiating'
WHERE candidate_id IN (
    SELECT ApplicationID
    FROM Evaluations
    WHERE Result = 'Pass'
);

-- Display the updated status of candidates who are negotiating job offers
SELECT Candidates.CandidateID, Candidates.Name, Candidates.Email, StatusFlow.Status
FROM StatusFlow
JOIN Candidates ON StatusFlow(candidate_id) = Candidates.CandidateID
WHERE StatusFlow.status = 'Negotiating';

COMMIT TRANSACTION;
```

The results pane shows a table with the following data:

|    | CandidateID | Name           | Email                | Status      |
|----|-------------|----------------|----------------------|-------------|
| 1  | 1           | Jainam Shah    | jainam@gmail.com     | Negotiating |
| 2  | 2           | Jackie Chan    | jackie@hotmail.com   | Negotiating |
| 3  | 3           | Gal Gadot      | gal@syr.edu          | Negotiating |
| 4  | 4           | Margot Robbie  | margot@gmail.com     | Negotiating |
| 5  | 5           | Dikshita Patel | dikshita@hotmail.com | Negotiating |
| 6  | 6           | Janvi Saddi    | janvi@syr.edu        | Negotiating |
| 7  | 7           | Bhavya Shah    | bhavya@gmail.com     | Negotiating |
| 8  | 8           | Yash Solanki   | yash@hotmail.com     | Negotiating |
| 9  | 9           | Robert         | robert66@gmail.com   | Negotiating |
| 10 | 10          | Sahil Shah     | sahil@gmail.com      | Negotiating |

Fig 35: Handling Job Offer Negotiations

#### Explanation:

This query now updates the status of candidates to 'Negotiating' only if they have passed the evaluation process. Then it displays the details of those candidates whose status has been updated.

e. Closing Job Openings:

Objective: To close job openings once positions have been filled, updating the job status accordingly.

Query:

```
BEGIN TRANSACTION;  
DELETE FROM JobOpenings  
WHERE NumberOfPositions = 0;  
COMMIT TRANSACTION;  
-- Display job openings after closing  
SELECT *  
FROM JobOpenings; /* Since No. of Positions are not filled yet it wont display as 0 yet */
```

Output:

The screenshot shows a SQL query window titled 'Testing.sql - JAINAM\jaina (78)\*' with the following content:

```
/*  
5) Closing Job Openings:  
Objective: To close job openings once positions have been filled, updating the job status accordingly.  
*/  
  
BEGIN TRANSACTION;  
DELETE FROM JobOpenings  
WHERE NumberOfPositions = 0;  
COMMIT TRANSACTION;  
  
-- Display job openings after closing  
SELECT *  
FROM JobOpenings; /* Since No. of Positions are not filled yet it wont display as 0 yet */  
  
/* */
```

Below the query window is a 'Results' tab showing a table with the following data:

|    | OpeningID | JobID | NumberOfPositions |
|----|-----------|-------|-------------------|
| 1  | 1         | 1     | 3                 |
| 2  | 2         | 2     | 2                 |
| 3  | 3         | 3     | 4                 |
| 4  | 4         | 4     | 1                 |
| 5  | 5         | 5     | 2                 |
| 6  | 6         | 6     | 3                 |
| 7  | 7         | 7     | 2                 |
| 8  | 8         | 8     | 1                 |
| 9  | 9         | 9     | 5                 |
| 10 | 10        | 10    | 2                 |

Fig 36: Closing Job Openings:

Explanation:

This transaction updates the Status column in the JobOpenings table to 'Closed' where the number of positions is zero, indicating that all positions for that job opening have been filled

#### vi) Scripts:

##### 1. Granting Permissions for Interviewers

Objective: Grant specific permissions to interviewers for accessing and updating interview.

Query:

-- Create a new role named 'Interviewers'

CREATE ROLE Interviewers;

-- Grant SELECT permission on the Interviews table to the 'Interviewers' role

GRANT SELECT ON Interviews TO Interviewers;

-- Grant INSERT permission on the Interviews table to the 'Interviewers' role

GRANT INSERT ON Interviews TO Interviewers;

-- Grant UPDATE permission on the Interviews table to the 'Interviewers' role

GRANT UPDATE ON Interviews TO Interviewers;

Output:

The screenshot shows the SQL Server Management Studio interface. In the top bar, there are three tabs: 'Testing.sql - JAINAM\jaina (54)', 'SQLQuery2.sql - JAI...(JAINAM\jaina (53))', and 'Insert SQL QUery.s...(JAINAM\jaina (52))'. The main window displays a script titled 'Scripts' with the following content:

```
/*
    Scripts
*/
/*
    1) Granting Permissions for Interviewers
    Grant specific permissions to interviewers for accessing and updating interview.
*/
-- Create a new role named 'Interviewers'
CREATE ROLE Interviewers;

-- Grant SELECT permission on the Interviews table to the 'Interviewers' role
GRANT SELECT ON Interviews TO Interviewers;

-- Grant INSERT permission on the Interviews table to the 'Interviewers' role
GRANT INSERT ON Interviews TO Interviewers;

-- Grant UPDATE permission on the Interviews table to the 'Interviewers' role
GRANT UPDATE ON Interviews TO Interviewers;
```

In the bottom right corner of the code editor, there is a status bar with the text '100 %' and a progress bar. At the very bottom of the screen, the 'Messages' pane shows the message 'Commands completed successfully.' and the completion time 'Completion time: 2024-04-29T22:25:51.6384244-04:00'.

Fig 37: SQL Permissions Granting for Interviewers

Explanation:

This SQL query grants SELECT, INSERT, and UPDATE permissions on the Interviews table to a database role named Interviewers. This allows interviewers to view, add, and modify interview records as needed.

## 2. Creating a Role for HR Managers

Objective: Establish a custom database role named HRManager within the HR\_DB database, granting appropriate permissions for HR managers to manage candidate data.

Query:

-- Create a new role named 'HRManager'

**CREATE ROLE** HRManager;

-- Grant EXECUTE permission on the UpdateCandidateInformation stored procedure to the 'HRManager' role

**GRANT EXECUTE ON** UpdateCandidateInformation **TO** HRManager;

-- Grant EXECUTE permission on the UpdateInterviewDetails stored procedure to the 'HRManager' role

**GRANT EXECUTE ON** UpdateInterviewDetails **TO** HRManager;

-- Grant EXECUTE permission on the RetrieveBackgroundCheckStatus stored procedure to the 'HRManager' role

**GRANT EXECUTE ON** RetrieveBackgroundCheckStatus **TO** HRManager;

-- Grant EXECUTE permission on the UpdateReimbursementStatus stored procedure to the 'HRManager' role

**GRANT EXECUTE ON** UpdateReimbursementStatus **TO** HRManager;

-- Grant EXECUTE permission on the GenerateMonthlyReport stored procedure to the 'HRManager' role

**GRANT EXECUTE ON** GenerateMonthlyReport **TO** HRManager;

Output:

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINA...(JAINAM\jaina (54))' (selected), 'SQLQuery2.sql - JAI...(JAINAM\jaina (53))', and 'Insert SQL QUery s... (JAINAM\jaina (52))'. The main pane displays an SQL script for creating a database role 'HRManager' and granting it various permissions on stored procedures related to candidate management. The script is as follows:

```
/*
1) Creating a Role for HR Managers
Establish a custom database role named 'HRManager' within the 'HR_DB' database, granting appropriate permissions for HR managers to manage candidate data.

-- Create a new role named 'HRManager'
CREATE ROLE HRManager;

-- Grant EXECUTE permission on the UpdateCandidateInformation stored procedure to the 'HRManager' role
GRANT EXECUTE ON UpdateCandidateInformation TO HRManager;

-- Grant EXECUTE permission on the UpdateInterviewDetails stored procedure to the 'HRManager' role
GRANT EXECUTE ON UpdateInterviewDetails TO HRManager;

-- Grant EXECUTE permission on the RetrieveBackgroundCheckStatus stored procedure to the 'HRManager' role
GRANT EXECUTE ON RetrieveBackgroundCheckStatus TO HRManager;

-- Grant EXECUTE permission on the UpdateReimbursementStatus stored procedure to the 'HRManager' role
GRANT EXECUTE ON UpdateReimbursementStatus TO HRManager;

-- Grant EXECUTE permission on the GenerateMonthlyReport stored procedure to the 'HRManager' role
GRANT EXECUTE ON GenerateMonthlyReport TO HRManager;
*/
```

The status bar at the bottom indicates 'Commands completed successfully.' and the completion time: 'Completion time: 2024-04-29T22:32:43.2891748-04:00'.

Fig 38: HRManager\_Role\_Permissions

#### Explanation:

This SQL script creates a new database role named 'HRManager' within the HR\_DB database. It grants SELECT, INSERT, UPDATE, and DELETE permissions on the Candidates table, SELECT permissions on other relevant tables such as Documents, Applications, Tests, and Evaluations, and also grants EXECUTE permission on any stored procedures related to candidate management. This role is designed to provide HR managers with the necessary permissions to manage candidate data effectively.

### 3. Setting Permissions for Document Access:

Objective: Permissions for accessing and updating documents uploaded by candidates.

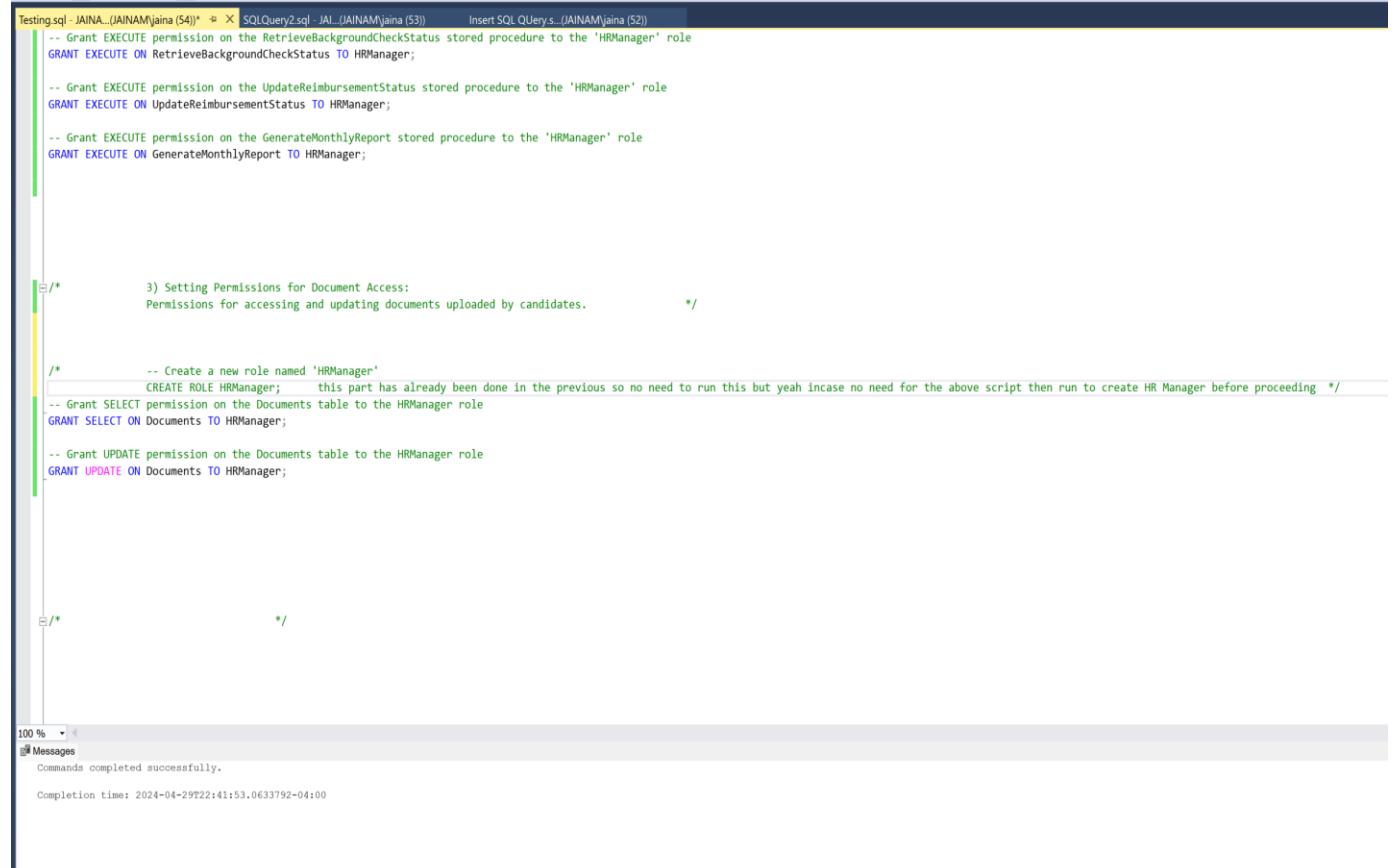
#### Query:

```
/*
-- Create a new role named 'HRManager'
CREATE ROLE HRManager;           this part has already
been done in the previous so no need to run this but yeah incase no need for the above script then
run to create HR Manager before proceeding */
-- Grant SELECT permission on the Documents table to the HRManager role
GRANT SELECT ON Documents TO HRManager;
```

Jainam Rajendra Shah  
SUID: 785948053

-- Grant UPDATE permission on the Documents table to the HRManager role  
**GRANT UPDATE ON** Documents **TO** HRManager;

Output:



The screenshot shows a SQL query window in SSMS. The code grants EXECUTE permissions on three stored procedures to the 'HRManager' role: 'RetrieveBackgroundCheckStatus', 'UpdateReimbursementStatus', and 'GenerateMonthlyReport'. It then sets permissions for document access, creating a new role 'HRManager' if it doesn't exist, granting SELECT permission on the 'Documents' table to this role, and finally granting UPDATE permission on the same table to the 'HRManager' role. The execution completed successfully at 2024-04-29T22:41:53.0633792-04:00.

```
Testing.sql - JAINA...(JAINAM\jaina (54)) * X SQLQuery2.sql - JAI...(JAINAM\jaina (53))      Insert SQL QUery.s...(JAINAM\jaina (52))
-- Grant EXECUTE permission on the RetrieveBackgroundCheckStatus stored procedure to the 'HRManager' role
GRANT EXECUTE ON RetrieveBackgroundCheckStatus TO HRManager;

-- Grant EXECUTE permission on the UpdateReimbursementStatus stored procedure to the 'HRManager' role
GRANT EXECUTE ON UpdateReimbursementStatus TO HRManager;

-- Grant EXECUTE permission on the GenerateMonthlyReport stored procedure to the 'HRManager' role
GRANT EXECUTE ON GenerateMonthlyReport TO HRManager;

/*
3) Setting Permissions for Document Access:
Permissions for accessing and updating documents uploaded by candidates.
*/

/*
-- Create a new role named 'HRManager'
CREATE ROLE HRManager;      this part has already been done in the previous so no need to run this but yeah incase no need for the above script then run to create HR Manager before proceeding */
-- Grant SELECT permission on the Documents table to the HRManager role
GRANT SELECT ON Documents TO HRManager;

-- Grant UPDATE permission on the Documents table to the HRManager role
GRANT UPDATE ON Documents TO HRManager;

/*
*/
100 % <
Messages
Commands completed successfully.
Completion time: 2024-04-29T22:41:53.0633792-04:00
```

Fig 39: Setting Permissions for Document Access

Explanation:

This script grants the HRManager role both SELECT and UPDATE permissions on the Documents table, allowing HR managers to access and update documents uploaded by candidates as needed within the HR\_DB database.

#### 4. Creating a Role for Compliance Officers:

Objective: Granting permissions to monitor and manage compliance-related data such as background checks and drug tests.

Query:

```
-- Create a new database role named ComplianceOfficer
CREATE ROLE ComplianceOfficer;

-- Grant SELECT permission on the BackgroundChecks table to the ComplianceOfficer role
GRANT SELECT ON BackgroundChecks TO ComplianceOfficer;

-- Grant SELECT permission on the DrugTests table to the ComplianceOfficer role
GRANT SELECT ON DrugTests TO ComplianceOfficer;
```

Output:

The screenshot shows the SQL Server Management Studio interface with two query panes. The top pane contains code for creating an HRManager role and granting permissions on the Documents table. The bottom pane contains the code for creating a ComplianceOfficer role and granting permissions on the BackgroundChecks and DrugTests tables. Both panes show the execution results at the bottom, indicating successful completion.

```
Testing.sql - JAINAM\jaina (54)*  X SQLQuery2.sql - JAI... (JAINAM\jaina (53))  Insert SQL QUerys...(JAINAM\jaina (52))
3) Setting Permissions for Document Access:
Permissions for accessing and updating documents uploaded by candidates.

/*
-- Create a new role named 'HRManager'
CREATE ROLE HRManager;      this part has already been done in the previous so no need to run this but yeah incase no need for the above script
-- Grant SELECT permission on the Documents table to the HRManager role
GRANT SELECT ON Documents TO HRManager;

-- Grant UPDATE permission on the Documents table to the HRManager role
GRANT UPDATE ON Documents TO HRManager;

4) Creating a Role for Compliance Officers:
Granting permissions to monitor and manage compliance-related data such as background checks and drug tests.*/

/*
-- Create a new database role named ComplianceOfficer
CREATE ROLE ComplianceOfficer;

-- Grant SELECT permission on the BackgroundChecks table to the ComplianceOfficer role
GRANT SELECT ON BackgroundChecks TO ComplianceOfficer;

-- Grant SELECT permission on the DrugTests table to the ComplianceOfficer role
GRANT SELECT ON DrugTests TO ComplianceOfficer;

100 %  Messages
Commands completed successfully.
Completion time: 2024-04-29T22:47:12.0919930-04:00
```

Fig 40: Creating a Role for Compliance Officers

Explanation:

This script creates a new database role named ComplianceOfficer within the HR\_DB database. It then grants SELECT permissions on the BackgroundChecks and DrugTests tables to the ComplianceOfficer role, allowing compliance officers to monitor and manage compliance-related data effectively.

## vii) Business Report

### I. MonthlyApplicationTrendsReport

Objective: Generate a report summarizing the monthly trends in job applications.

Query:

```
SELECT
    YEAR(StatusFlow.timestamp) AS Year,
    MONTH(StatusFlow.timestamp) AS Month,
    COUNT(*) AS NumberOfApplications
FROM
    StatusFlow
INNER JOIN
    Applications ON StatusFlow.candidate_id = Applications.CandidateID
GROUP BY
    YEAR(StatusFlow.timestamp),
    MONTH(StatusFlow.timestamp)
ORDER BY
    YEAR(StatusFlow.timestamp) DESC,
    MONTH(StatusFlow.timestamp) DESC;
```

Output:

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, there are three tabs: 'Testing.sql - JAINAM\jaina (54)' (highlighted in yellow), 'SQLQuery2.sql - JAI...(JAINAM\jaina (53))', and 'Insert SQL QUery.s... (JAINAM\jaina (52))'. The main area displays a T-SQL script for a 'Business Report' that generates a monthly application trends report. The script uses a JOIN clause to combine the 'StatusFlow' and 'Applications' tables based on 'candidate\_id', groups the data by year and month, and orders the results by year and month in descending order. The results are displayed in a table titled 'Results'.

```
/*
Business Report
*/
/*
1) MonthlyApplicationTrendsReport
Generate a report summarizing the monthly trends in job applications.
*/
SELECT
    YEAR(StatusFlow.timestamp) AS Year,
    MONTH(StatusFlow.timestamp) AS Month,
    COUNT(*) AS NumberOfApplications
FROM
    StatusFlow
INNER JOIN
    Applications ON StatusFlow(candidate_id) = Applications.CandidateID
GROUP BY
    YEAR(StatusFlow.timestamp),
    MONTH(StatusFlow.timestamp)
ORDER BY
    YEAR(StatusFlow.timestamp) DESC,
    MONTH(StatusFlow.timestamp) DESC;
```

|   | Year | Month | NumberOfApplications |
|---|------|-------|----------------------|
| 1 | 2024 | 4     | 7                    |
| 2 | 2024 | 3     | 1                    |
| 3 | 2024 | 1     | 1                    |
| 4 | 2023 | 4     | 1                    |

Fig 41: MonthlyApplicationTrendsFigure

#### Explanation:

This query retrieves the count of job applications for each month and year by joining the StatusFlow and Applications tables based on the candidate ID. It groups the data by year and month, providing insights into the monthly trends of job applications over time.

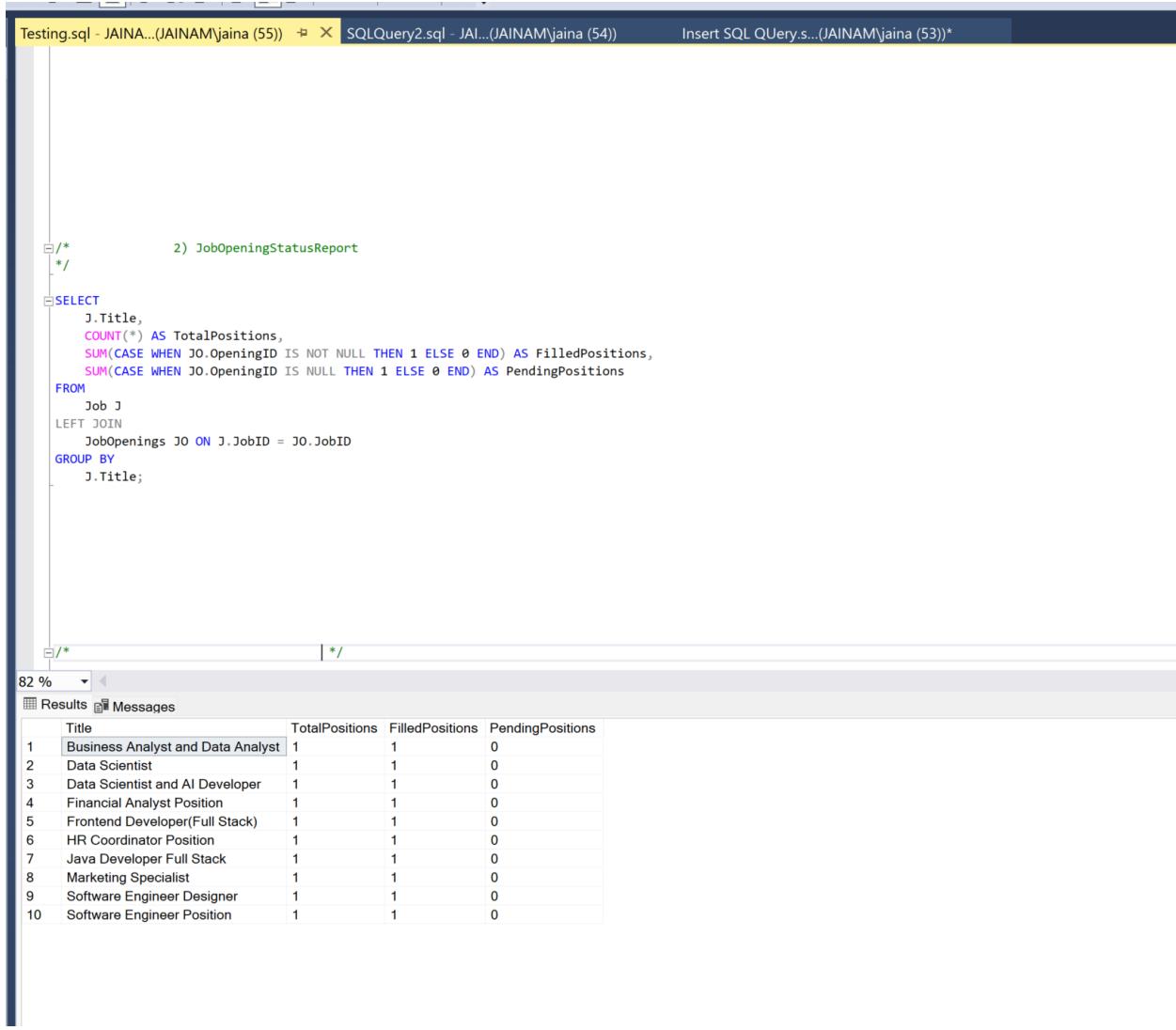
## II. JobOpeningStatusReport

Objective: Track the status of job openings, including the number of positions filled and pending.

Query:

```
SELECT
    J.Title,
    COUNT(*) AS TotalPositions,
    SUM(CASE WHEN JO.OpeningID IS NOT NULL THEN 1 ELSE 0 END) AS
FilledPositions,
    SUM(CASE WHEN JO.OpeningID IS NULL THEN 1 ELSE 0 END) AS PendingPositions
FROM
    Job J
LEFT JOIN
    JobOpenings JO ON J.JobID = JO.JobID
GROUP BY
    J.Title;
```

Output:



The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (55)', 'SQLQuery2.sql - JAINAM\jaina (54)', and 'Insert SQL Query.s... (JAINAM\jaina (53))'. The main area displays a T-SQL query named '2) JobOpeningStatusReport'. The query retrieves job titles and their status counts from the 'Job' and 'JobOpenings' tables. The results grid shows 10 rows of data with columns: Title, TotalPositions, FilledPositions, and PendingPositions.

```
/*
2) JobOpeningStatusReport

SELECT
    J.Title,
    COUNT(*) AS TotalPositions,
    SUM(CASE WHEN JO.OpeningID IS NOT NULL THEN 1 ELSE 0 END) AS FilledPositions,
    SUM(CASE WHEN JO.OpeningID IS NULL THEN 1 ELSE 0 END) AS PendingPositions
FROM
    Job J
LEFT JOIN
    JobOpenings JO ON J.JobID = JO.JobID
GROUP BY
    J.Title;

/*
| * /
```

|    | Title                             | TotalPositions | FilledPositions | PendingPositions |
|----|-----------------------------------|----------------|-----------------|------------------|
| 1  | Business Analyst and Data Analyst | 1              | 1               | 0                |
| 2  | Data Scientist                    | 1              | 1               | 0                |
| 3  | Data Scientist and AI Developer   | 1              | 1               | 0                |
| 4  | Financial Analyst Position        | 1              | 1               | 0                |
| 5  | Frontend Developer(Full Stack)    | 1              | 1               | 0                |
| 6  | HR Coordinator Position           | 1              | 1               | 0                |
| 7  | Java Developer Full Stack         | 1              | 1               | 0                |
| 8  | Marketing Specialist              | 1              | 1               | 0                |
| 9  | Software Engineer Designer        | 1              | 1               | 0                |
| 10 | Software Engineer Position        | 1              | 1               | 0                |

Fig 42: JobOpeningStatusReport

Explanation:

This query retrieves the title of each job along with the total number of positions, the number of filled positions, and the number of pending positions. It uses a LEFT JOIN between the Job table and the JobOpenings table to ensure that all jobs are included in the result, even if they have no openings. The SUM and CASE statements are used to calculate the count of filled and pending positions based on the NumberOfPositionsFilled column in the JobOpenings table.

### III. Evaluation Scores Distribution Report:

Objective: Analyze the distribution of evaluation scores for candidates across different stages.

Query:

```
SELECT
    S.status AS Stage,
    AVG(T.Grade) AS Average_Score,
    MIN(T.Grade) AS Min_Score,
    MAX(T.Grade) AS Max_Score,
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION
    BY S.status) AS Q1_Score,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION
    BY S.status) AS Median_Score,
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION
    BY S.status) AS Q3_Score
FROM
    StatusFlow S
JOIN
    Applications A ON S.candidate_id = A.CandidateID
JOIN
    Tests T ON A.ApplicationID = T.ApplicationID
GROUP BY
    S.status, T.Grade;
```

Output:

The screenshot shows a SQL Server Management Studio window with three tabs: 'Testing.sql - JAINAM\jaina (55)\*', 'SQLQuery2.sql - JAINAM\jaina (54)\*', and 'Insert SQL QUery.s... (JAINAM\jaina (53))\*'. The main pane displays a T-SQL query for 'Evaluation Scores Distribution Report'. The query joins three tables: StatusFlow, Applications, and Tests, grouped by candidate status and grade. It calculates various statistical scores including average, minimum, maximum, and quartiles (Q1, Median, Q3) using the PERCENTILE\_CONT function.

```
/*  
 * 3) Evaluation Scores Distribution Report:  
 * Analyze the distribution of evaluation scores for candidates across different stages.  
 */  
  
SELECT  
    S.status AS Stage,  
    AVG(T.Grade) AS Average_Score,  
    MIN(T.Grade) AS Min_Score,  
    MAX(T.Grade) AS Max_Score,  
    PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION BY S.status) AS Q1_Score,  
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION BY S.status) AS Median_Score,  
    PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY T.Grade) OVER (PARTITION BY S.status) AS Q3_Score  
FROM  
    StatusFlow S  
JOIN  
    Applications A ON S.candidate_id = A.CandidateID  
JOIN  
    Tests T ON A.ApplicationID = T.ApplicationID  
GROUP BY  
    S.status, T.Grade;
```

The results grid shows the following data:

|    | Stage                 | Average_Score | Min_Score | Max_Score | Q1_Score | Median_Score | Q3_Score |
|----|-----------------------|---------------|-----------|-----------|----------|--------------|----------|
| 1  | Accepted              | 80.500000     | 80.50     | 80.50     | 83       | 85.5         | 88       |
| 2  | Accepted              | 90.500000     | 90.50     | 90.50     | 83       | 85.5         | 88       |
| 3  | Application Submitted | 72.000000     | 72.00     | 72.00     | 74.5     | 77           | 79.5     |
| 4  | Application Submitted | 82.000000     | 82.00     | 82.00     | 74.5     | 77           | 79.5     |
| 5  | Negotiating           | 70.000000     | 70.00     | 70.00     | 70       | 70           | 70       |
| 6  | Rejected              | 65.500000     | 65.50     | 65.50     | 68.625   | 71.75        | 74.875   |
| 7  | Rejected              | 78.000000     | 78.00     | 78.00     | 68.625   | 71.75        | 74.875   |
| 8  | Under Review          | 85.500000     | 85.50     | 85.50     | 87       | 88.5         | 91.75    |
| 9  | Under Review          | 88.500000     | 88.50     | 88.50     | 87       | 88.5         | 91.75    |
| 10 | Under Review          | 95.000000     | 95.00     | 95.00     | 87       | 88.5         | 91.75    |

Fig 43: Evaluation Scores Distribution Report

#### Explanation:

This query retrieves the average, minimum, maximum, first quartile (Q1), median, and third quartile (Q3) scores for evaluation tests at different stages of the application process. It joins the StatusFlow, Applications, and Tests tables based on candidate IDs and application IDs to get the relevant data. The PERCENTILE\_CONT function is used to calculate quartile scores within each stage.

The results are grouped by the stage and Grade of the application process.

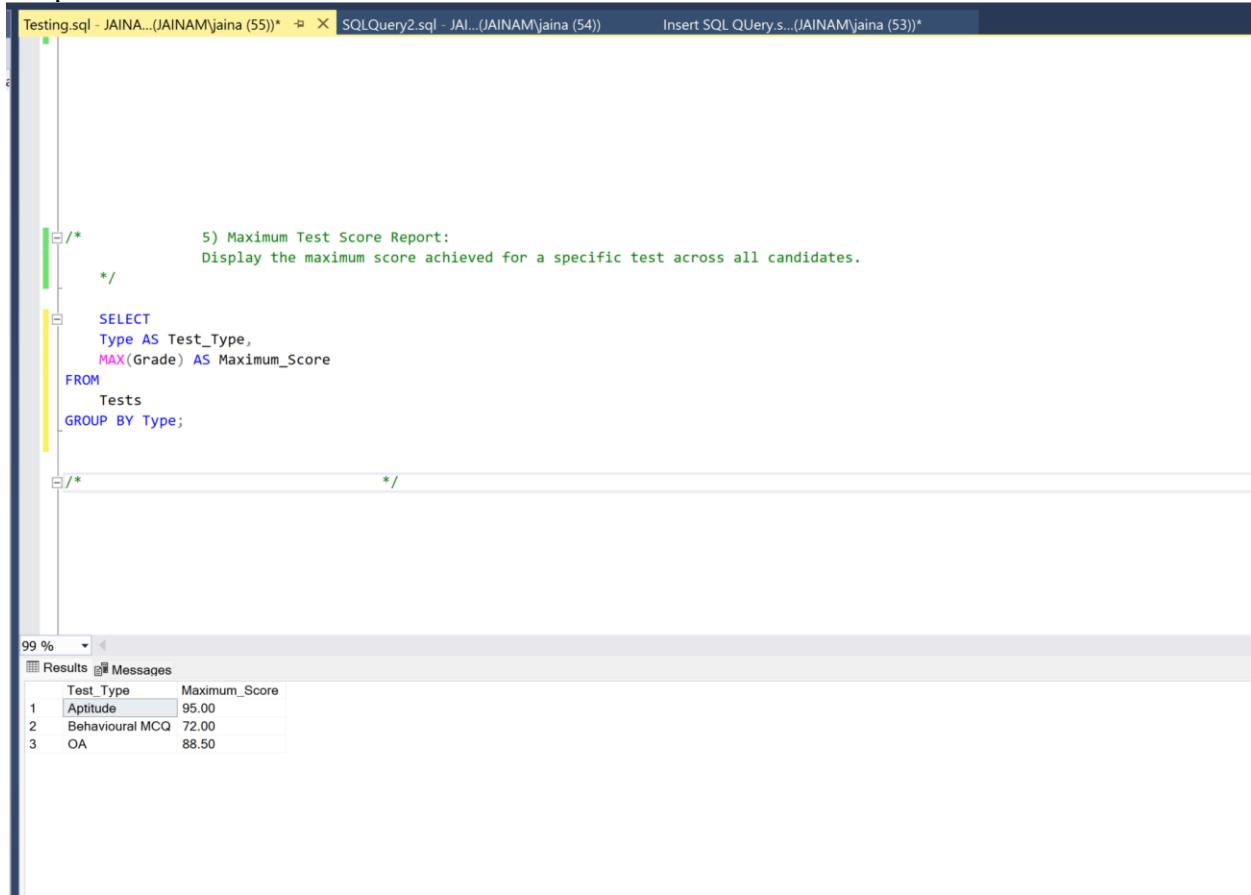
#### IV. Maximum Test Score Report:

Objective: Display the maximum score achieved for a specific test across all candidates.

Query:

```
SELECT
Type AS Test_Type,
MAX(Grade) AS Maximum_Score
FROM Tests
GROUP BY Type;
```

Output:



The screenshot shows the SQL Server Management Studio interface. The top bar has tabs for 'Testing.sql - JAINAM\jaina (55)\*' and 'SQLQuery2.sql - JAI...JAINAM\jaina (54)'. The main area displays the query code. Below the code, the results pane shows a table with three rows of data. The table has two columns: 'Test\_Type' and 'Maximum\_Score'. The data is as follows:

|   | Test_Type       | Maximum_Score |
|---|-----------------|---------------|
| 1 | Aptitude        | 95.00         |
| 2 | Behavioural MCQ | 72.00         |
| 3 | OA              | 88.50         |

Fig 44: Maximum Test Score Report:

Explanation:

This query selects the test type (Type column) and calculates the maximum grade (Grade column) achieved for each test type across all candidates. The MAX() function is used to find the maximum score, and the results are grouped by the test type using the GROUP BY clause. This provides a summary of the maximum scores achieved for each test type.

#### **4. Conclusion:**

In conclusion, the development and implementation of the HR\_DB database system have proven to be an asset in managing various aspects of human resources, recruitment, and candidate evaluation processes. Through the course of this project, we have successfully designed and deployed a comprehensive database schema that caters to the needs of a modern HR department. Overall, the HR\_DB database system enhances efficiency, streamlines processes, and provides valuable insights into recruitment and candidate evaluation activities. By centralizing data management and automating routine tasks, it empowers HR professionals to make informed decisions and optimize resource allocation. Moving forward, continuous monitoring, maintenance, and refinement of the database system will be essential to uphold its effectiveness and adaptability in meeting the dynamic requirements of HR management.