ITI Online Examination System

Track: Power BI Development

Branch: Cairo University

Done by:

Dalia Hosny
Hala Farid
Sandra Essa
Nadda Khaled

Under the Supervision of:

Eng. Ramy Mohamed

Feb. 2024

Acknowledgment

We would like to express our sincere gratitude to all those who supported us throughout the completion of our graduation project at Information Technology Institute.

First and foremost, we extend our heartfelt appreciation to our project supervisor **ENG. Ramy Mohamed** for their invaluable guidance, encouragement, and insightful feedback that greatly contributed to the success of this project.

We are also grateful to the faculty members of the Information Technology Institute for their expertise and assistance during the course of our studies.

Thank you,

Team Members.

Table Of Content

1. Project Overview

This project was done as a graduation project for the "information technology institute" (ITI),

The Examination System includes a **Web Application** that is designed and developed for students to take exams, submit answers and then see their exam result either passed or failed according to percentage.

To use the application, the student must log in with an email and password that has been stored before in the database.

The exams are generated randomly by Stored Procedure called Exam_Generation, there is another Procedure called Exam_Answer that is responsible for storing the Students' answers in database, and finally a Procedure called Exam_Correction which is responsible for correcting the answers and calculating the students' percentage in this exam.

2. Tools, Languages and Frameworks we used:

- **Draw.io (online website).**
  To design both Entity Relationship Diagram (ERD) & Database schema.
- **SQL Server Management Studio.**
  For database implementation.
- **SQL Server Reporting Service and Visual Studio.**
  For implementing Reports.
- **SQL Server Management Studio and SQL Server Integration Service.**
  For Implementing Datawarehouse.
- **Microsoft Power BI Desktop.**
  For designing Dashboards.
- **Microsoft Fabric (Power BI Service) & Microsoft Power BI Desktop.**
  For connecting to Paginated Reports created by SSRS.
- **HTML, CSS, JAVASCRIPT.**
  For designing website Front-End.
- **ASP.NET Core 6.**
- **Entity Framework Core version 6.0.2 package.**
- **Entity Framework Core.SqlServer Version 6.0.26 package.**
- **EntityFrameworkCore.Tools Version 6.0.26 package.**
- **EF Core Power Tools.**
  For designing website Back-End.

Business Case: Online Examination System

We are developing an online examination system, on which students can exercise many exams in different topics and get their exam percentage instantly.

Entities Overview:

- **Branches:** Our system centralizes the management of branches across different locations, each with a unique identifier, name, location, and founding date. Additionally, branches are led by competent managers and offer various educational tracks facilitated by instructors.
- **Students:** Students are the core beneficiaries of our system, each identified by personal details including name, contact information, gender, and age. They are affiliated with specific faculties, acquire certificates, and participate in rounds or intakes. Moreover, students engage in track enrollment, examination taking, and question answering activities.
- **Instructors:** Our system empowers instructors with the necessary tools to effectively manage educational tracks and courses. Instructors possess essential information such as contact details, qualifications, and teaching experience. Additionally, they oversee the delivery of courses, evaluate student performance, and provide mentorship to facilitate academic growth.
- **Tracks:** Educational tracks represent specialized learning paths designed to equip students with industry-relevant skills and knowledge. Each track is managed by a designated instructor and encompasses a cohort of students, courses, and graduates.
- **Courses:** Courses form the building blocks of educational tracks, offering structured learning experiences on specific subjects. They are delivered by instructors and cover various topics, assessments, and evaluations to gauge student proficiency.
- **Exams:** Examinations serve as pivotal assessments within courses, enabling students to demonstrate their understanding and competence. Exams are characterized by unique identifiers, names, dates, levels, durations, and full marks, with each exam comprising multiple questions.
- **Questions:** Questions are integral components of exams, featuring textual prompts, correct answers, difficulty levels, and types (e.g., multiple-choice, true/false). They are associated with specific courses and exams, allowing for comprehensive assessment across various subjects.
- **Graduates:** Graduates represent successful completers of educational tracks, equipped with industry-ready skills and certifications. They transition into the workforce, securing employment opportunities within reputable companies, contributing to organizational growth and success.

4. ERD

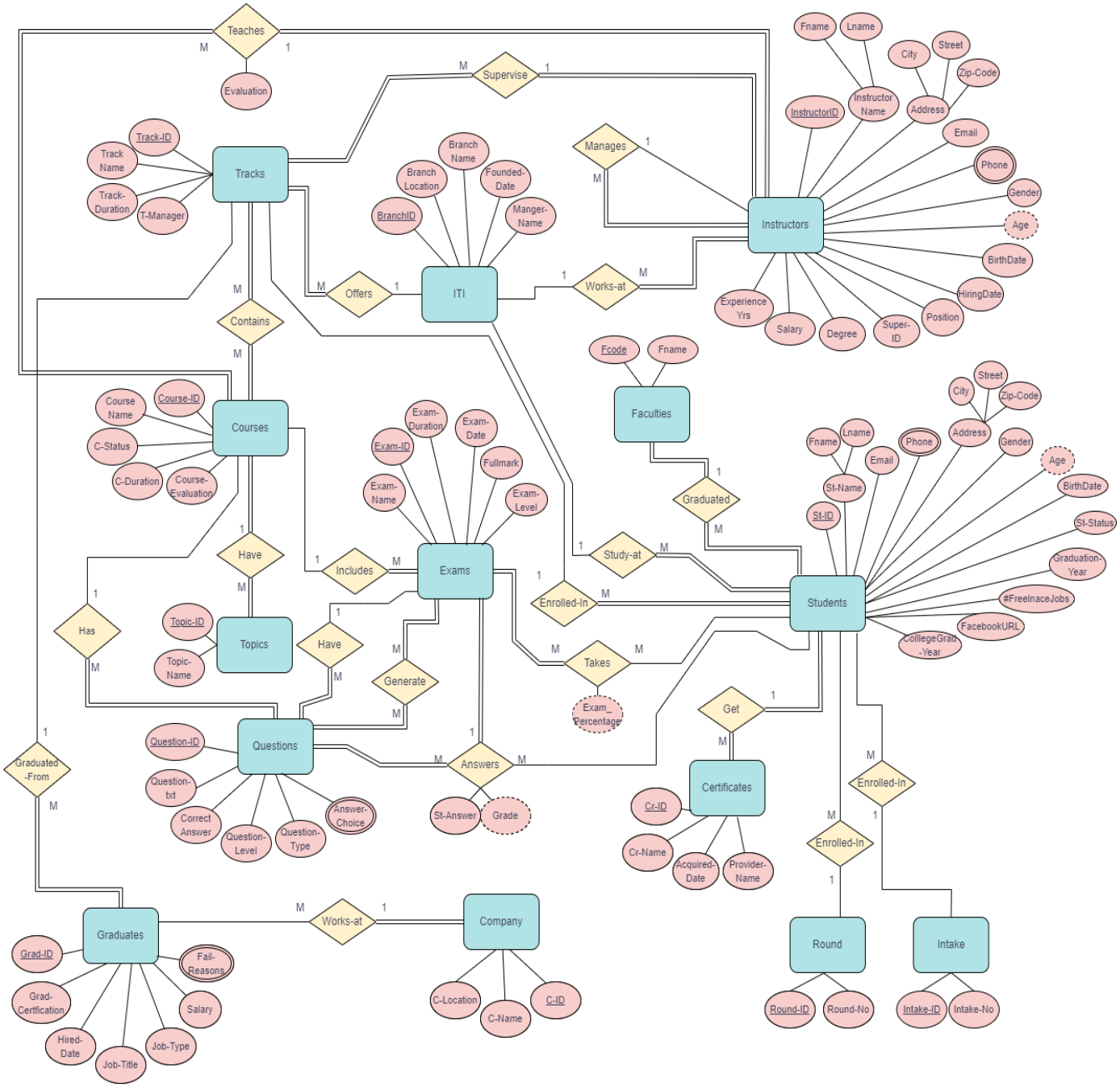


Figure1. Entity Relationship Diagram

5. Mapping

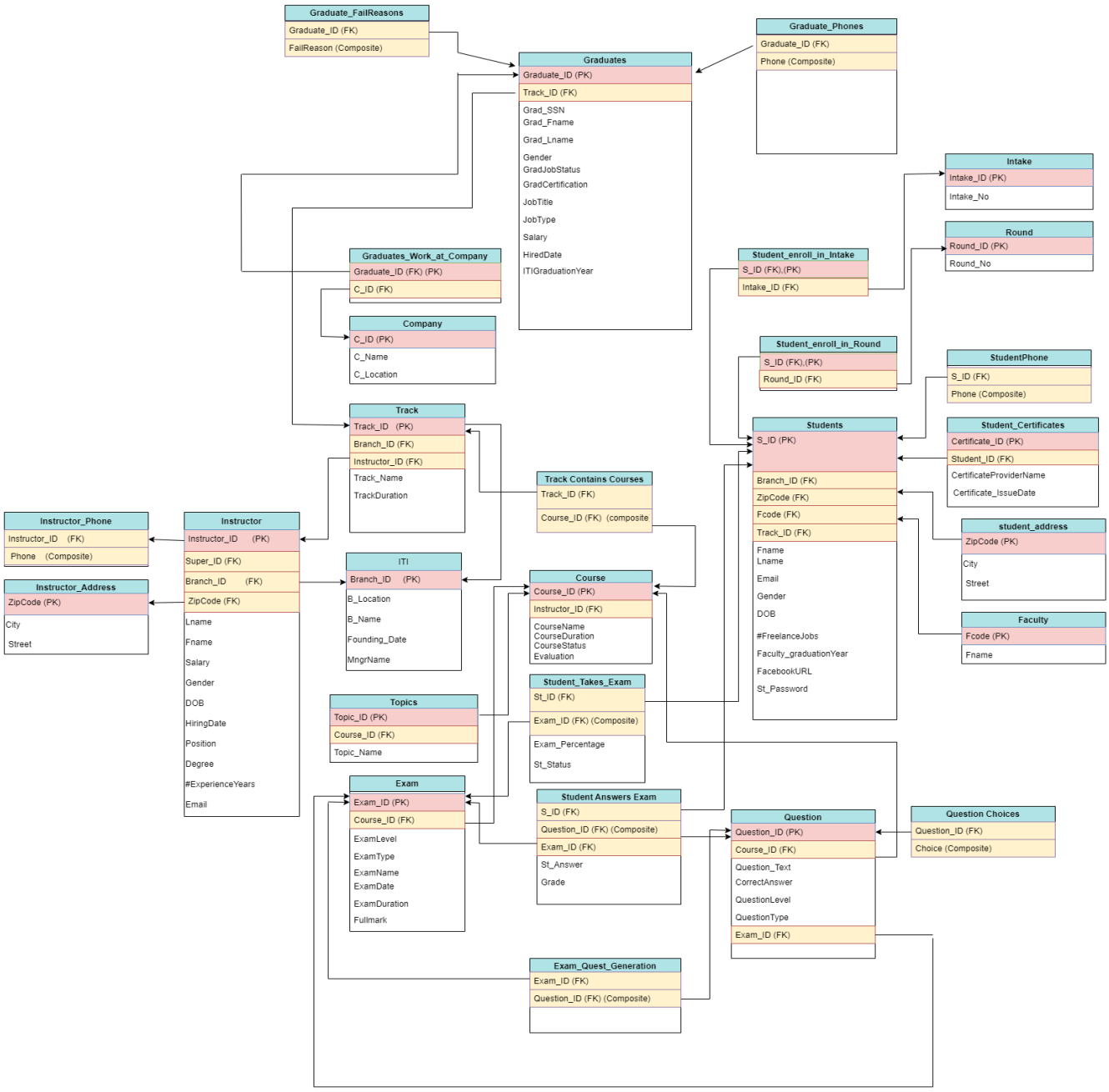


Figure2. Schema

6. SnowFlake Schema

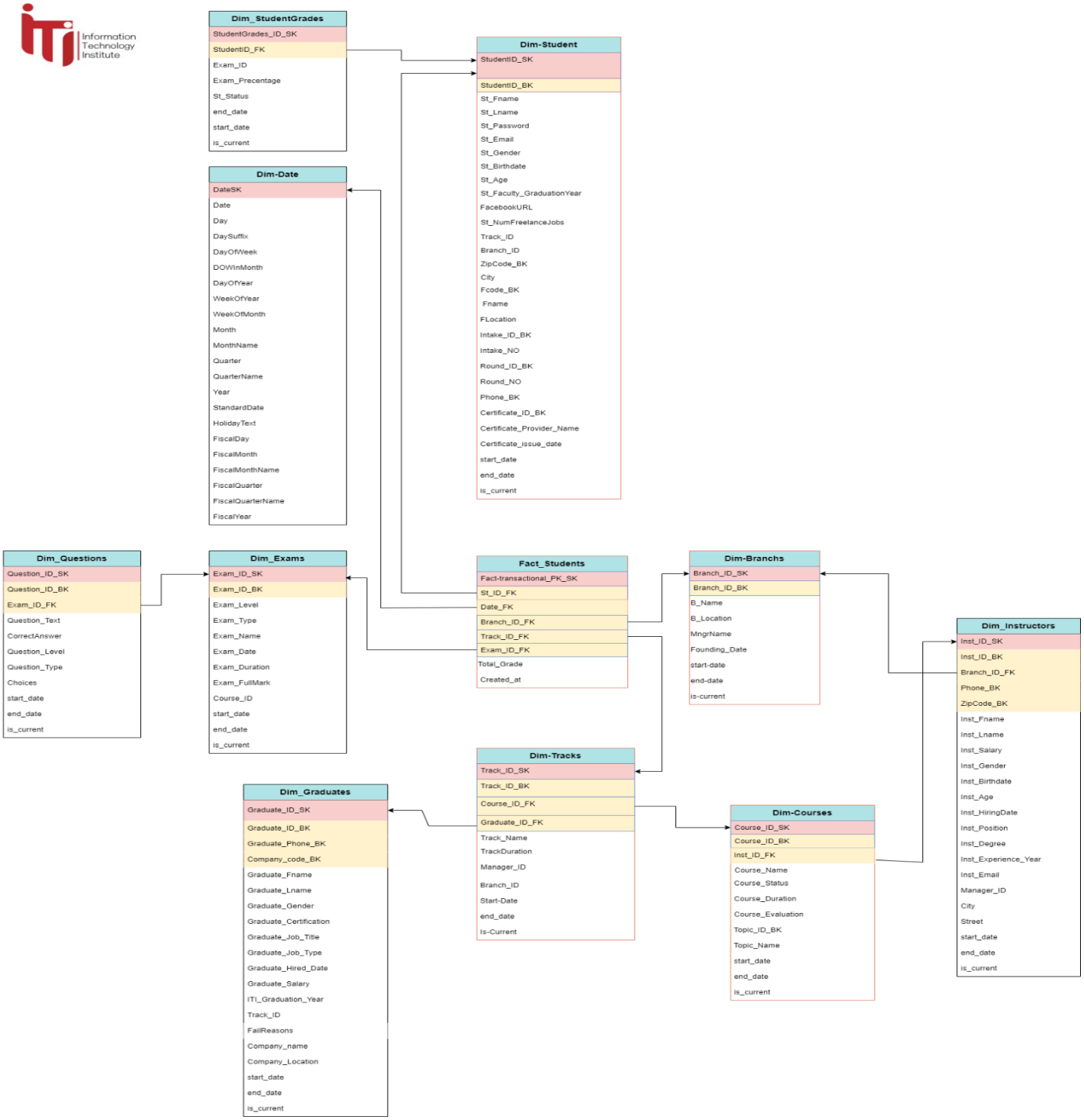


Figure3. Snowflake Schema

7. Database Dictionary

7.1.Table: StudentsInfo

Column Name	Data Type	Constraints
St_ID	Int	Primary Key, Not Null
St_Fname	varchar(50)	Accept Null Value
St_Lname	varchar(50)	Accept Null Value
St_Gender	varchar(50)	Accept Null Value
St_Birthdate	Date	Accept Null Value
St_NumFreelanceJobs	Int	Accept Null Value
St_Email	varchar(100)	Accept Null Value
St_Age	Int	Accept Null Value
St_Faculty_GraduationYear	Int	Accept Null Value
St_ITI_GraduationYear	Int	Accept Null Value
FacebookURL	varchar(400)	Accept Null Value
Branch_ID	Int	Foreign Key, Accept Null Value
ZipCode	Int	Foreign Key, Accept Null Value
Fcode	varchar(200)	Foreign Key, Accept Null Value
Track_ID	Int	Foreign Key, Accept Null Value

• Age is derived attribute and calculated according to this equation:
Age = (year (getdate ())-year (Birthdate))

7.2. Table: Instructors

Column Name	Data Type	Constraints
Inst_ID	Int	Primary Key, Not Null
Inst_Fname	varchar(50)	Accept Null Value
Inst_Lname	varchar(50)	Accept Null Value
Inst_Email	varchar(250)	Accept Null Value
Inst_Gender	varchar(20)	Accept Null Value
Inst_Birthdate	Date	Accept Null Value
Inst_Salary	Int	Accept Null Value
Inst_Age		
Inst_HiringDate	Date	Accept Null Value
Inst_Position	varchar(50)	Accept Null Value
Inst_Degree	varchar(50)	Accept Null Value

Inst_Experience_Year	Int	Accept Null Value
Manager_ID	Int	Foreign Key, Accept Null Value
Branch_ID	Int	Foreign Key, Accept Null Value
ZipCode	Int	Foreign Key, Accept Null Value

• Age is derived attribute and calculated according to this equation:

Age = (year (getdate ())-year (Birthdate))

7.3. Table: Tracks

Column Name	Data Type	Constraints
Track_ID	Int	Primary Key, Not Null
Track_Name	varchar(250)	Accept Null Value
Track_Duartion	varchar(100)	Accept Null Value
Branch_ID	Int	Foreign Key, Accept Null Value
Manager_ID	Int	Foreign Key, Accept Null Value

7.4. Table: Branches

Column Name	Data Type	Constraints
Branch_ID	Int	Primary key, Not Null
B_Name	varchar(100)	Accept Null Value
B_Location	varchar(100)	Accept Null Value
Founding_date	Date	Accept Null Value
MngrName	varchar(200)	Accept Null Value

7.5. Table: Courses

Column Name	Data Type	Constraints
Course_ID	Int	Primary Key, Not Null
Course_Name	varchar(250)	Accept Null Value
Course_Duration	varchar(250)	Accept Null Value
Course_Evaluation	varchar(250)	Accept Null Value
Course_Status	varchar(50)	Accept Null Value
Inst_ID	Int	Foreign Key, Accept Null Value

7.6. Table: Topics

Column Name	Data Type	Constraints
Topic_ID	Int	Primary Key, Not Null
TopicName	varchar(250)	Accept Null Value
Course_ID	Int	Foreign key, Accept Null Value

7.7. Table: Exams

Column Name	Data Type	Constraints
Exam_ID	Int	Primary Key, Not Null
Exam_Level	varchar(50)	Accept Null Value
Exam_Type	varchar(50)	Accept Null Value
Exam_Name	varchar(50)	Accept Null Value
Exam_Date	Date	Accept Null Value
Exam_Duration	varchar(50)	Accept Null Value
Exam_Fullmark	varchar(50)	Accept Null Value
Grade	Int	Accept Null Value
Course_ID	Int	Foreign Key, Accept Null Value

7.8.Table: Questions

Column Name	Data Type	Constraints
Question_ID	Int	Primary Key, Not Null
Question_Text	varchar(250)	Accept Null Value
Question_Level	varchar(50)	Accept Null Value
CorrectAnswer	varchar(250)	Accept Null Value
Question_Type	varchar(50)	Accept Null Value

7.9.Table: Faculty

Column Name	Data Type	Constraints
Fcode	varchar(200)	Primary Key, Not Null
Fname	varchar(250)	Accept Null Value
FLocation	varchar(250)	Accept Null Value

7.10.Table: Graduates

Column Name	Data Type	Constraints
Graduate_ID	Int	Primary Key, Not Null
Graduate_Certification	varchar(50)	Accept Null Value
Graduate_Job_Title	varchar(250)	Accept Null Value
Graduate_Job_Type	varchar(100)	Accept Null Value
Graduate_Hired_Date	Date	Accept Null Value
Graduate_Salary	Money	Accept Null Value
Track_ID	Int	Foreign Key, Accept Null Value

7.11.Table: Intake

Column Name	Data Type	Constraints
Intake_ID	Int	Primary Key, Not Null
Intake_NO	varchar(100)	Accept Null Value

7.12.Table: Round

Column Name	Data Type	Constraints
Round_ID	Int	Primary Key, Not Null
Round_NO	varchar(100)	Accept Null Value

7.13.Table: Student_Certificate

Column Name	Data Type	Constraints
Certificate_ID	Int	Primary Key, Not Null
certificate_Provider_Name	varchar(250)	Accept Null Value
certificate_issue_date	Date	Accept Null Value
St_ID	Int	Foreign Key, Accept Null Value

7.14.Table: Graduate_company

Column Name	Data Type	Constraints
Company_code	varchar(250)	Primary Key, Not Null
Company_name	varchar(250)	Accept Null Value
Company_Location	varchar(250)	Accept Null Value

Tables Created Due to Attributes Types

7.15.Table: Student_address

• According to Composite Attribute: Address

Column Name	Data Type	Constraints
ZipCode	Int	Primary Key, Not null
City	varchar(250)	Accept Null Value
Street	varchar(250)	Accept Null Value

7.16.Table: Student_Phone

• According to Multivalued Attribute: Phone

Column Name	Data Type	Constraints	
St_ID	Int	Foreign Key, Not Null	Composite Primary Key
Phone	varchar(20)	Not Null	

7.17.Table: Instructor_address

• According to Composite Attribute: Address

Column Name	Data Type	Constraints
ZipCode	Int	Primary Key, Not Null
City	varchar(250)	Accept Null Value
Street	varchar(250)	Accept Null Value

7.18.Table: Instructor_Phone

• According to Multivalued Attribute: Phone

Column Name	Data Type	Constraints	
Inst_ID	Int	Foreign Key, Not Null	Composite Primary Key
Phone	varchar(20)	Not null	

7.19.Table: Graduate_Fail_Reasons

• According to Multivalued Attribute: Fail_Reasons

Column Name	Data Type	Constraints	
Graduate_ID	Int	Foreign Key, Not Null	Composite Primary Key
FailReasons	varchar(250)	Not null	

7.20.Table: Question_Choices

• According to Multivalued Attribute: Answer_Choices

Column Name	Data Type	Constraints	
Question_ID	Int	Foreign Key, Not Null	Composite Primary Key
Choice	varchar(250)	Not null	

Tables Created Due To Relationship Between Entities

7.21.Table: Student_Takes_Exam

• According To Many To Many Relationship Between Students And Exams

Column Name	Data Type	Constraints	
St_ID	Int	Foreign Key, Not Null	Composite Primary Key
Exam_ID	Int	Foreign Key, Not Null	
Exam_Percentage	decimal(5, 1)	Accept Null Value	
St_Status	varchar(50)	Accept Null Value	

7.22.Table: Student_Answers_Exam

• According To Ternary Relationship Between Students, Exams, And Questions

Column Name	Data Type	Constraints	
St_ID	Int	Foreign Key, Not Null	Composite Primary Key
Question_ID	Int	Foreign Key, Not Null	
Exam_ID	Int	Foreign Key, Accept Null Value	
St_Answer	varchar(250)	Accept Null Value	
Grade	Int	Accept Null Value	

7.23.Table: Student_enroll_Round

• According To One To Many Partial Participation Relationship Between Intake and Students

Column Name	Data Type	Constraints
St_ID	Int	Primary Key, Not Null
Round_ID	Int	Foreign Key, Accept Null Value

7.24.Table: Student_enroll_intake

• According To One To Many Partial Participation Relationship Between Round and Students

Column Name	Data Type	Constraints
St_ID	Int	Primary Key, Not Null
Intake_ID	Int	Foreign Key, Accept Null Value

7.25.Table: GraduatesWorkCompany

- According To One To Many Partial Participation Relationship Between Graduates and Graduates_Company

Column Name	Data Type	Constraints
Graduate_ID	Int	Primary Key, Not Null
Company_code	varchar(250)	Foreign Key, Accept Null Value

7.26.Table: Track_Contains_Courses

- According To Many To Many Relationship Between Tracks And Courses

Column Name	Data Type	Constraints	
Track_ID	Int	Foreign Key, Not Null	Composite Primary Key
Course_ID	Int	Foreign Key, Not Null	

7.27.Table: Exam_Quest_Generation

- According To Many To Many Relationship Between Exams And Questions

Column Name	Data Type	Constraints	
Exam_ID	Int	Foreign Key, Not null	Composite primary key
Question_ID	Int	Foreign Key, Not null	

8. Stored Procedures for Tables

8.1. Basic Stored Procedure

8.1.1. Insert Stored Procedure

1) InsertBranches

The "InsertBranches" stored procedure inserts branch details into the "Branches" table, checking for existing entries based on name and location. If no duplicate is found, it adds the new branch information. Otherwise, it raises an error indicating the existing branch ID. Error handling is included to manage exceptions during execution.

```
--1)Branches
create procedure InsertBranches  @Branch_ID int, @b_name varchar(100), @b_location varchar(100),
                                 @founding_date date, @mngr_name varchar(100)
as
begin
    begin try
        -- Check if branch ID already exists
        if not exists (select 1 from Branches where B_Name = @b_name and B_Location = @b_location)
        begin
            -- Insert a new branch
            insert into Branches (Branch_ID,B_Name, B_Location, Founding_date, MngrName)
            values (@Branch_ID,@b_name, @b_location, @founding_date, @mngr_name);
        end
        else
        begin
            -- Branch ID already exists, raise an error
            throw 50001, 'Branch ID already exists.', 1;
        end
    end try
    begin catch
        print error_message();
    end catch
end;
```

2) insert_questions

The "insert_questions" stored procedure adds new questions to the "Questions" table, ensuring uniqueness based on Question_ID. It performs security checks on question level and type, only allowing predefined values. If the question passes these checks, it's inserted into the table. Error handling is included to manage exceptions during execution.

```
--2)Question
CREATE PROCEDURE insert_questions  @Question_ID INT, @question_text VARCHAR(250),
                                   @correct_answer VARCHAR(250), @question_level VARCHAR(50),
                                   @question_type VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        -- Check if the question ID already exists
        IF NOT EXISTS (SELECT 1 FROM Questions WHERE Question_ID = @Question_ID)
        BEGIN
            -- Security check for Question_Level
            IF @question_level NOT IN ('Basic', 'Intermediate', 'Advanced')
            BEGIN
                THROW 50002, 'Invalid Question Level. Please choose from: Easy, Intermediate, Advanced.', 1;
            END
            -- Security check for Question_Type
            IF @question_type NOT IN ('T or F', 'MCQ')
            BEGIN
                THROW 50003, 'Invalid Question Type. Please choose from: T/F, Choose.', 1;
            END
            -- Insert the question if it passes security checks
            INSERT INTO Questions (Question_ID, Question_Text, CorrectAnswer, Question_Level, Question_Type)
            VALUES (@Question_ID, @question_text, @correct_answer, @question_level, @question_type);
            PRINT 'Question inserted successfully.';
        END
        ELSE
        BEGIN
            -- Question ID already exists, raise an error
            THROW 50004, 'Question ID already exists.', 1;
        END
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

3) InsertQuestionChoice

The "InsertQuestionChoice" stored procedure adds choices for a given question ID to the "Question_Choices" table. It verifies if the provided Question_ID exists and if the choice already exists for that question. If the checks pass, the choice is inserted into the able. Error handling is implemented to manage exceptions during execution

```
--3)/*Question_Choices*/
create proc InsertQuestionChoice  @Question_ID INT, @Choice VARCHAR(250)
as
begin
    begin try
        -- Check if Question_ID exists in the Questions table
        if  exists (select 1 from Questions where Question_ID = @Question_ID)
        begin
            throw 50005, 'Invalid Question_ID. Please provide a valid Question_ID.', 1;
        end
        -- Check if the choice already exists for the same Question_ID
        if exists (select 1 from Question_Choices where Question_ID = @Question_ID and Choice = @Choice)
        begin
            throw 50006, 'The choice already exists for this question.', 1;
        end
        -- Insert the choice
        insert into Question_Choices (Question_ID, Choice)
        values (@Question_ID, @Choice);
        print 'Choice inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

4) InsertFaculty

The "InsertFaculty" stored procedure inserts new faculty details into the "Faculty" table. It ensures that the provided faculty code starts with 'EGY-UNV' and is unique. If the checks pass, the faculty information is added to the table. Error handling is implemented to manage exceptions during execution.

```
--4)/*Faculty*/
create procedure InsertFaculty  @Fcode varchar(200), @Fname varchar(250), @FLocation varchar(250)
as
begin
    begin try
            -- Check if Fcode starts with EGY-UNV
        if left(@Fcode, 7) != 'EGY-UNV'
        begin
            throw 50007, 'Faculty code must start with EGY-UNV.', 1;
        end
        -- Check if Fcode already exists
        if exists (select 1 from Faculty where Fcode = @Fcode)
        begin
            throw 50008, 'Faculty code already exists.', 1;
        end
        -- Insert the faculty
        insert into Faculty (Fcode, Fname, Flocation)
        values (@Fcode, @Fname, @Flocation);
        print 'Faculty inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

5) **InsertStudentaddress**

The "InsertStudentaddress" stored procedure inserts new student address details into the "Student_address" table. It checks for existing zip codes and streets to ensure uniqueness. If the checks pass, the student address information is added to the table. Error handling is implemented to manage exceptions during execution.

```
--5)/*Student_address*/
create proc InsertStudentaddress @Zipcode int, @City varchar(250), @Street varchar(250)
as
begin
    begin try
        -- Check if zipcode already exists
        if exists (select 1 from Student_address where Zipcode = @Zipcode)
        begin
            throw 50009, 'Zip code already exists.', 1;
        end
                -- Check if street already exists
        if exists (select 1 from Student_address where Street = @Street)
        begin
            throw 50010, 'Street already exists.', 1;
        end
        -- Insert the student address
        insert into Student_address (Zipcode, City, Street)
        values (@Zipcode, @City, @Street);
        print 'Student address inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

6) **InsertInstructoraddress**

The "InsertInstructoraddress" stored procedure inserts new instructor address details into the "Instructor_address" table. It checks for existing zip codes and streets to ensure uniqueness. If the checks pass, the instructor address information is added to the table. Error handling is implemented to manage exceptions during execution.

```
--6)/*Instructor_address*/
create proc InsertInstructoraddress @Zipcode int, @City varchar(250), @Street varchar(250)
as
begin
    begin try
        -- Check if zipcode already exists
        if exists (select 1 from Student_address where Zipcode = @Zipcode)
        begin
            throw 50011, 'Zip code already exists.', 1;
        end
                -- Check if street already exists
        if exists (select 1 from Student_address where Street = @Street)
        begin
            throw 50012, 'Street already exists.', 1;
        end
        -- Insert the student address
        insert into Instructor_address (Zipcode, City, Street)
        values (@Zipcode, @City, @Street);
        print 'Instructor address inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

7) **InsertGraduteCompany**

The "InsertGraduteCompany" stored procedure inserts details of a graduate company into the "Graduate_company" table. It checks if the company location contains 'egypt' and verifies the company code accordingly. If the checks pass and the company code is unique, the information is inserted into the table. Error handling is implemented to manage exceptions during execution.

```
--7)/*Gradute_company*/
alter proc InsertGraduteCompany @company_code varchar(250), @company_name varchar(250),
                                @company_location varchar(250)
as
begin
    begin try
        -- Check if Company_location contains 'egypt'
        if charindex('egypt', lower(@company_location)) > 0
        begin
            -- Company Location contains 'egypt', so Company code should start with 'EGY-CMP'
            if left(@company_code, 7) != 'egy-cmp'
            begin
                throw 50013, 'Company code must start with EGY-CMP for companies located in Egypt.', 1;
            end
        end
        else
        begin
            -- Company Location does not contain 'egypt', so Company code should start with 'INTL-CMP'
            if left(@company_code, 7) != 'intl-cmp'
            begin
                throw 50014, 'Company code must start with INTL-CMP for companies located outside Egypt.', 1;
            end
        end
        -- Check if Company_code already exists
        if exists (select 1 from Graduate_company where company_code = @company_code)
        begin
            throw 50015, 'Company code already exists.', 1;
        end
        -- Insert the graduate company
        insert into Graduate_company (company_code, company_name, company_location)
        values (@company_code, @company_name, @company_location);
        print 'Graduate company inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

8) InsertInstructor

The "InsertInstructor" stored procedure inserts details of an instructor into the "instructors" table. It performs various checks including the existence of the instructor ID, ensuring the salary is above 5000, validating gender, position, degree, manager ID, branch ID, and zipcode. If all checks pass, the information is inserted into the table. Error handling is included to manage exceptions during execution, providing informative messages for encountered errors.

```
--8)/*Instructors*/
CREATE PROCEDURE InsertInstructor @Inst_ID INT, @inst_fname VARCHAR(250), @inst_lname VARCHAR(50),@inst_salary MONEY,
                                  @inst_gender VARCHAR(50),@inst_birthdate DATE, @inst_hiringdate DATE,
                                  @inst_position VARCHAR(50), @inst_degree VARCHAR(50),@inst_experience_year INT,
                                  @inst_email VARCHAR(250), @manager_id INT, @branch_id INT, @zipcode INT
AS
BEGIN
    BEGIN TRY
        -- Check if the inst_id already exists
        IF EXISTS (SELECT 1 FROM instructors WHERE inst_id = @Inst_ID)
        BEGIN
            PRINT 'Instructor ID already exists.';
            RETURN; -- Exit the stored procedure
        END
        -- Check if inst_salary is greater than 5000
        IF @inst_salary <= 5000
        BEGIN
            THROW 50016, 'Instructor salary must be greater than 5000.', 1;
        END
        -- Check if inst_gender is 'm' or 'f'
        IF @inst_gender NOT IN ('m', 'f')
        BEGIN
            THROW 50017, 'Invalid gender. Gender must be "m" or "f".', 1;
        END
        -- Check if inst_position is 'instructor' or 'freelancer'
        IF @inst_position NOT IN ('instructor', 'freelancer')
        BEGIN
            THROW 50018, 'Invalid position. Position must be "instructor" or "freelancer".', 1;
        END
        -- Check if inst_degree is 'bachelor', 'master', or 'phd'
        IF @inst_degree NOT IN ('bachelor', 'master', 'phd')
        BEGIN
            THROW 50019, 'Invalid degree. Degree must be "bachelor", "master", or "phd".', 1;
        END
        -- Check if manager_id is valid (foreign key reference to inst_id)
        IF NOT EXISTS (SELECT 1 FROM instructors WHERE inst_id = @manager_id)
        BEGIN
            THROW 50020, 'Invalid manager ID.', 1;
        END
        -- Check if branch_id is valid (foreign key reference to branches)
        IF NOT EXISTS (SELECT 1 FROM branches WHERE branch_id = @branch_id)
        BEGIN
            THROW 50021, 'Invalid branch ID.', 1;
        END
        -- Check if zipcode is valid (foreign key reference to instructor_address)
        IF NOT EXISTS (SELECT 1 FROM instructor_address WHERE zipcode = @zipcode)
        BEGIN
            THROW 50022, 'Invalid zip code.', 1;
        END
        -- Insert the instructor
        INSERT INTO instructors (inst_id, inst_fname, inst_lname, inst_salary, inst_gender, inst_birthdate, inst_hiringdate, inst_position, inst_degree, inst_experience_year, inst_email, manager_id, branch_id, zipcode)
        VALUES (@Inst_ID, @inst_fname, @inst_lname, @inst_salary, @inst_gender, @inst_birthdate, @inst_hiringdate, @inst_position, @inst_degree, @inst_experience_year, @inst_email, @manager_id, @branch_id, @zipcode);

        PRINT 'Instructor inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

9) InsertInstructorPhone

The "InsertInstructorPhone" stored procedure adds phone numbers for a given instructor ID to the "Instructor_Phone" table. It validates the existence of the instructor ID in the "instructors" table and checks the length of the phone number to ensure it falls within the acceptable range. Additionally, it verifies if the phone number already exists for the instructor. If all checks pass, the phone number is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages.

```
--9)/*Instructor Phone*/
create proc InsertInstructorPhone @inst_id int,@phone varchar(20)
as
begin
    begin try
        -- Check if inst_id exists in the instructors table
        if  not exists (select 1 from instructors where inst_id = @inst_id)
        begin
            throw 50023, 'Invalid instructor ID. Please provide a valid instructor ID.', 1;
        end
        -- Check if the phone number is valid
        if len(@phone) < 11 or len(@phone) > 11
        begin
            throw 50024, 'Invalid phone number. Phone number must be between 7 and 20 characters.', 1;
        end
                -- Check if the phone number already exists for the instructor
        if exists (select 1 from Instructor_Phone where Inst_ID = @inst_id and Phone = @phone)
        begin
            throw 50025, 'Phone number already exists for this instructor.', 1;
        end
        -- Insert the instructor phone number
        insert into instructor_phone (inst_id, phone)
        values (@inst_id, @phone);
        print 'Instructor phone number inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

10)InsertCourse

The "InsertCourse" stored procedure inserts details of a course into the "Courses" table. It checks if the course ID already exists, validates the course status, and ensures the instructor ID exists in the "instructors" table. If all checks pass, the course information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--10)/*Courses*/
CREATE PROCEDURE InsertCourse @course_id INT,@course_name VARCHAR(250),@course_status VARCHAR(50),
                              @course_duration VARCHAR(250),@course_evaluation VARCHAR(250),@inst_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if the course ID already exists
        IF EXISTS (SELECT 1 FROM Courses WHERE Course_ID = @course_id)
        BEGIN
            THROW 50029, 'Course with the same ID already exists.', 1;
        END
        -- Check if course_status is valid
        IF @course_status NOT IN ('offline', 'online')
        BEGIN
            THROW 50026, 'Invalid course status. Course status must be "offline" or "online".', 1;
        END
        -- Check if inst_id exists in the instructors table
        IF NOT EXISTS (SELECT 1 FROM instructors WHERE inst_id = @inst_id)
        BEGIN
            THROW 50027, 'Invalid instructor ID. Please provide a valid instructor ID.', 1;
        END
        -- Insert the course
        INSERT INTO courses (course_id, course_name, course_status, course_duration, course_evaluation, inst_id)
        VALUES (@course_id, @course_name, @course_status, @course_duration, @course_evaluation, @inst_id);
        PRINT 'Course inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

11) InsertTopic

The "InsertTopic" stored procedure adds topics to a specific course in the "topics" table. It verifies if the topic ID already exists, ensures the validity of the provided course ID, and checks if the topic already exists for the given course. If all checks pass, the topic information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--11)Topics
create PROCEDURE InsertTopic @topic_Id INT, @course_id INT, @topic_name VARCHAR(250)
AS
BEGIN
    BEGIN TRY
        -- Check if topic_id already exists
        IF EXISTS (SELECT 1 FROM topics WHERE topic_Id = @topic_Id)
        BEGIN
            THROW 50036, 'Topic ID already exists. Please provide a unique topic ID.', 1;
        END
        -- Check if course_id exists in the courses table
        IF NOT EXISTS (SELECT 1 FROM courses WHERE course_id = @course_id)
        BEGIN
            THROW 50033, 'Invalid course ID. Please provide a valid course ID.', 1;
        END
        -- Check if the topic already exists for the given course
        IF EXISTS (SELECT 1 FROM topics WHERE course_id = @course_id AND topic_name = @topic_name)
        BEGIN
            THROW 50034, 'Topic already exists for this course.', 1;
        END
        -- Insert the topic
        INSERT INTO topics (topic_Id, course_id, topic_name)
        VALUES (@topic_Id, @course_id, @topic_name);
        PRINT 'Topic inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

12)InsertTrack

The "InsertTrack" stored procedure inserts details of a track into the "tracks" table. It checks if the provided Track_ID already exists, validates the track duration, and ensures the validity of the provided branch ID. Additionally, it verifies if the manager ID exists in the "instructors" table if provided. If all checks pass, the track information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--12)/*Tracks*/
create PROCEDURE InsertTrack @Track_Id INT,@track_name VARCHAR(250),@track_duration VARCHAR(100),
                             @branch_id INT,@manager_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if the provided Track_Id already exists
        IF EXISTS (SELECT 1 FROM tracks WHERE Track_Id = @Track_Id)
        BEGIN
            THROW 50038, 'Track ID already exists. Please provide a unique Track ID.', 1;
        END
        -- Check if track_duration is valid
        IF @track_duration NOT IN ('3 months', '9 months')
        BEGIN
            THROW 50035, 'Invalid track duration. Track duration must be "3 months" or "9 months".', 1;
        END
        -- Check if branch_id exists in the branches table
        IF NOT EXISTS (SELECT 1 FROM branches WHERE branch_id = @branch_id)
        BEGIN
            THROW 50036, 'Invalid branch ID. Please provide a valid branch ID.', 1;
        END
        -- Check if manager_id exists in the instructors table
        IF @manager_id IS NOT NULL AND NOT EXISTS (SELECT 1 FROM instructors WHERE inst_id = @manager_id)
        BEGIN
            THROW 50037, 'Invalid manager ID. Please provide a valid manager ID.', 1;
        END
        -- Insert the track
        INSERT INTO tracks (Track_Id, track_name, track_duration, branch_id, manager_id)
        VALUES (@Track_Id, @track_name, @track_duration, @branch_id, @manager_id);
        PRINT 'Track inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

13) InsertIntake

The "InsertIntake" stored procedure inserts details of an intake into the "intake" table. It checks if the provided Intake_ID already exists and validates the format of the intake number to ensure it starts with 'Intake_'. If all checks pass, the intake information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--13)/*Intake*/
CREATE PROCEDURE InsertIntake @Intake_ID INT, @intake_no VARCHAR(100)
AS
BEGIN
    BEGIN TRY
        -- Check if the provided Intake_ID already exists
        IF EXISTS (SELECT 1 FROM intake WHERE Intake_ID = @Intake_ID)
        BEGIN
            THROW 50039, 'Intake with the provided ID already exists. Please provide a unique ID.', 1;
        END
        -- Check if Intake_No starts with 'Intake_'
        IF LEFT(@intake_no, 7) != 'intake_'
        BEGIN
            THROW 50038, 'Invalid Intake_No format. Intake_No must start with "Intake_".', 1;
        END
        -- Insert the intake
        INSERT INTO intake (Intake_ID, intake_no)
        VALUES (@Intake_ID, @intake_no);
        PRINT 'Intake inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

14) InsertRound

The "InsertRound" stored procedure inserts details of a round into the "rounds" table. It checks if the provided Round_ID already exists and validates the format of the round number to ensure it starts with 'Round_'. If all checks pass, the round information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--14)/*Round*/
create PROCEDURE InsertRound @Round_ID int, @round_no varchar(100)
AS
BEGIN
    BEGIN TRY
        -- Check if Round_ID already exists
        IF EXISTS (SELECT 1 FROM rounds WHERE Round_ID = @Round_ID)
        BEGIN
            THROW 50040, 'Round ID already exists. Please provide a different Round ID.', 1;
        END
        -- Check if Round_NO starts with 'Round_'
        IF LEFT(@round_no, 6) != 'round_'
        BEGIN
            THROW 50039, 'Invalid Round_NO format. Round_NO must start with "Round_".', 1;
        END
        -- Insert the round
        INSERT INTO rounds (Round_ID, round_no)
        VALUES (@Round_ID, @round_no);
        PRINT 'Round inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

15)InsertStudentsInfo

The "InsertStudentsInfo" stored procedure inserts details of a student into the "studentsinfo" table. It checks if the provided St_ID already exists and validates the gender, branch ID, zip code, faculty code, and track ID to ensure their existence in respective tables. If all checks pass, the student information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--15)/*Students*/
CREATE PROCEDURE InsertStudentsInfo  @St_ID INT,@st_fname VARCHAR(50),@st_lname VARCHAR(50),@st_email VARCHAR(100),
                                     @st_gender VARCHAR(50),@st_birthdate DATE,@st_faculty_graduationyear INT,
                                     @facebookurl VARCHAR(400),@st_numfreelancejobs INT,@branch_id INT,
                                     @zipcode INT,@fcode VARCHAR(200),@track_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if St_ID already exists
        IF EXISTS (SELECT 1 FROM studentsinfo WHERE St_ID = @St_ID)
        BEGIN
            THROW 50045, 'Student ID already exists. Please provide a unique student ID.', 1;
        END
        -- Check if st_gender is 'M' or 'F'
        IF @st_gender NOT IN ('M', 'F')
        BEGIN
            THROW 50040, 'Invalid gender. Gender must be "M" or "F".', 1;
        END
        -- Check if branch_id exists in the branches table
        IF NOT EXISTS (SELECT 1 FROM branches WHERE branch_id = @branch_id)
        BEGIN
            THROW 50041, 'Invalid branch ID. Please provide a valid branch ID.', 1;
        END
        -- Check if zipcode exists in the student_address table
        IF NOT EXISTS (SELECT 1 FROM student_address WHERE zipcode = @zipcode)
        BEGIN
            THROW 50042, 'Invalid zip code. Please provide a valid zip code.', 1;
        END
        -- Check if fcode exists in the faculty table
        IF NOT EXISTS (SELECT 1 FROM faculty WHERE fcode = @fcode)
        BEGIN
            THROW 50043, 'Invalid faculty code. Please provide a valid faculty code.', 1;
        END
       -- Check if track_id exists in the tracks table
       IF NOT EXISTS (SELECT 1 FROM tracks WHERE track_id = @track_id)
       BEGIN
          THROW 50044, 'Invalid track ID. Please provide a valid track ID.', 1;
       END
       -- Insert the student info
       INSERT INTO studentsinfo (St_ID, st_fname, st_lname, st_email, st_gender, st_birthdate, st_faculty_graduationyear, facebookurl, st_numfreelancejobs, branch_id, zipcode, fcode, track_id)
       VALUES (@St_ID, @st_fname, @st_lname, @st_email, @st_gender, @st_birthdate, @st_faculty_graduationyear, @facebookurl, @st_numfreelancejobs, @branch_id, @zipcode, @fcode, @track_id);
       PRINT 'Student info inserted successfully.';
    END TRY
    BEGIN CATCH
       PRINT ERROR_MESSAGE();
    END CATCH
END;
```

16) InsertStudentPhone

The "InsertStudentPhone" stored procedure inserts phone numbers for a given student ID into the "student_phone" table. It verifies if the student ID exists in the "studentsinfo" table and checks the length of the phone number to ensure it is 11 characters long. If all checks pass, the phone number is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--16)/*Student Phone*/
create proc InsertStudentPhone @st_id int,@phone varchar(20)
as
begin
    begin try
        -- Check if St_ID exists in the StudentsInfo table
        if not exists (select 1 from studentsinfo where st_id = @st_id)
        begin
            throw 50045, 'Invalid student ID. Please provide a valid student ID.', 1;
        end
        -- Check if the phone number is valid
        if len(@phone) < 11 or len(@phone) > 11
        begin
            throw 50046, 'Invalid phone number. Phone number must be 11', 1;
        end
        -- Insert the student phone number
        insert into student_phone (st_id, phone)
        values (@st_id, @phone);
        print 'Student phone number inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

17) InserttrackContainsCourse

The "InserttrackContainsCourse" stored procedure creates associations between tracks and courses in the "track_contains_courses" table. It verifies if both the Track_ID and Course_ID exist in their respective tables. It also ensures that the combination of Track_ID and Course_ID doesn't already exist in the association table. If all checks pass, the association is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--17)/*Track_Contains_Courses*/
create proc InserttrackContainsCourse @track_id int,@course_id int
as
begin
    begin try
        -- Check if Track_ID exists in the Tracks table
        if not exists (select 1 from tracks where track_id = @track_id)
        begin
            throw 50047, 'Invalid track ID. Please provide a valid track ID.', 1;
        end
        -- Check if Course_ID exists in the Courses table
        if not exists (select 1 from courses where course_id = @course_id)
        begin
            throw 50048, 'Invalid course ID. Please provide a valid course ID.', 1;
        end
        -- Check if the combination of Track_ID and Course_ID already exists
        if exists (select 1 from track_contains_courses where track_id = @track_id and course_id = @course_id)
        begin
            throw 50049, 'The specified course is already associated with the track.', 1;
        end
        -- Insert the track-course association
        insert into track_contains_courses (track_id, course_id)
        values (@track_id, @course_id);
        print 'Track-course association inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

18) InsertStudentenrollIntake

The "InsertStudentenrollIntake" stored procedure enrolls a student in a specific intake. It verifies if the student ID exists in the "studentsinfo" table and if the intake ID exists in the "intake" table. Additionally, it checks if the student is already enrolled in a round to prevent duplicate enrollments. If all checks pass, the student's enrollment for the intake is inserted into the "student_enroll_intake" table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--18)/*Student_enroll_in_intake*/
create proc InsertStudentenrollIntake @st_id int,@intake_id int
as
begin
    begin try
        -- Check if St_ID exists in the StudentsInfo table
        if not exists (select 1 from studentsinfo where st_id = @st_id)
        begin
            throw 50050, 'Invalid student ID. Please provide a valid student ID.', 1;
        end
        -- Check if Intake_ID exists in the Intake table
        if not exists (select 1 from intake where intake_id = @intake_id)
        begin
            throw 50051, 'Invalid intake ID. Please provide a valid intake ID.', 1;
        end
                -- Check if student is already enrolled in a round
        if exists (select 1 from student_enroll_round where st_id = @st_id)
        begin
            throw 50052, 'Student is already enrolled in a round.', 1;
        end
        -- Insert the student enrollment for intake
        insert into student_enroll_intake (st_id, intake_id)
        values (@st_id, @intake_id);
        print 'Student enrollment for intake inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

19) InsertStudentenrollRound

The "InsertStudentenrollRound" stored procedure enrolls a student in a specific round. It verifies if the student ID exists in the "studentsinfo" table and if the round ID exists in the "rounds" table. Additionally, it checks if the student is already enrolled in an intake to prevent enrollment in a round. If all checks pass, the student's enrollment for the round is inserted into the "student_enroll_round" table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--19)/*Student_enroll_in_Round*/
CREATE PROCEDURE InsertStudentenrollRound @st_id INT,@round_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if St_ID exists in the StudentsInfo table
        IF NOT EXISTS (SELECT 1 FROM studentsinfo WHERE st_id = @st_id)
        BEGIN
            THROW 50053, 'Invalid student ID. Please provide a valid student ID.', 1;
        END
        -- Check if Round_ID exists in the Rounds table
        IF NOT EXISTS (SELECT 1 FROM rounds WHERE round_id = @round_id)
        BEGIN
            THROW 50054, 'Invalid round ID. Please provide a valid round ID.', 1;
        END
        -- Check if the student is already enrolled in an intake
        IF EXISTS (SELECT 1 FROM student_enroll_intake WHERE st_id = @st_id)
        BEGIN
            THROW 50055, 'Student is already enrolled in an intake. Cannot enroll in a round.', 1;
        END
        -- Insert the student enrollment for round
        INSERT INTO student_enroll_round (st_id, round_id)
        VALUES (@st_id, @round_id);
        PRINT 'Student enrollment for round inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

20)InsertGraduate

The "InsertGraduate" stored procedure inserts details of a graduate into the "graduates" table. It validates the graduate certification to ensure it is either "Completed" or "Not completed" and verifies the job type to be either "Onsite" or "Freelance". Additionally, it checks if the provided track ID exists in the "Tracks" table. If all checks pass, the graduate information is inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--20)/*Graduates*/
create proc InsertGraduate @Graduate_ID int,@graduate_certification varchar(50),@graduate_job_title varchar(250),
                           @graduate_job_type varchar(100),@graduate_hired_date date,@graduate_salary money,
                           @track_id int
as
begin
    begin try
        -- Check if Graduate_Certification is 'Completed' or 'Not completed'
        if @graduate_certification not in ('completed', 'not completed')
        begin
            throw 50058, 'Invalid graduate certification. Certification must be "Completed" or "Not completed".', 1;
        end
        -- Check if Graduate_Job_Type is 'Onsite' or 'Freelance'
        if @graduate_job_type not in ('onsite', 'freelance')
        begin
            throw 50059, 'Invalid job type. Job type must be "Onsite" or "Freelance".', 1;
        end
        -- Check if St_ID exists in the StudentsInfo table
        if not exists (select 1 from Tracks where track_id = @track_id)
        begin
            throw 50060, 'Invalid track ID. Please provide a valid track ID.', 1;
        end
        -- Insert the graduate
        insert into graduates (Graduate_ID,graduate_certification, graduate_job_title, graduate_job_type, graduate_hired_date, graduate_salary, track_id)
        values (@Graduate_ID,@graduate_certification, @graduate_job_title, @graduate_job_type, @graduate_hired_date, @graduate_salary, @track_id);
        print 'Graduate inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

21)InsertGraduatefailReason

The "InsertGraduatefailReason" stored procedure records fail reasons for a graduate who has not completed their certification. It first checks if the provided Graduate_ID exists in the "Graduates" table. Then, it verifies if the graduate's certification status is "Not completed" to ensure fail reasons are recorded only for students with this status. If all checks pass, the fail reasons are inserted into the "Graduate_Fail_Reasons" table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--21)/*Graduate_Fail_Reasons*/

alter proc InsertGraduatefailReason @Graduate_ID int,@failreasons varchar(250)
as
begin
    begin try
        -- Check if Graduate_ID exists in the Graduates table
        if not exists (select 1 from Graduates where Graduate_ID = @Graduate_ID)
        begin
            throw 50061, 'Invalid Graduate_ID. Please provide a valid Graduate_ID.', 1;
        end
        -- Check if the graduates's certification status is "Not completed"
        if exists (select 1 from graduates where  graduate_certification = 'Completed')
        begin
            throw 50062, 'graduates has completed the certification. Fail reasons can only be recorded for students with "Not completed" certification.', 1;
        end
        -- Insert the student's fail reasons
        insert into Graduate_Fail_Reasons (Graduate_ID, failreasons)
        values (@Graduate_ID, @failreasons);
        print 'Graduate  fail reasons inserted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

22)InsertStudentCertificate

The "InsertStudentCertificate" stored procedure inserts details of a certificate obtained by a student into the "student_certificates" table. It first checks if the provided student ID exists in the "studentsinfo" table. Then, it verifies if the Certificate_ID is unique to ensure no duplicate entries are made. If all checks pass, the certificate details including the provider's name and issue date are inserted into the table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--22)/*Student_Certificates*/
create PROCEDURE InsertStudentCertificate @Certificate_ID int, @st_id int,@certificate_provider_name varchar(250),
                                          @certificate_issue_date date
AS
BEGIN
    BEGIN TRY
        -- Check if St_ID exists in the StudentsInfo table
        IF NOT EXISTS (SELECT 1 FROM studentsinfo WHERE st_id = @st_id)
        BEGIN
            THROW 50063, 'Invalid student ID. Please provide a valid student ID.', 1;
        END
        -- Check if Certificate_ID exists in the student_certificates table
        IF EXISTS (SELECT 1 FROM student_certificates WHERE Certificate_ID = @Certificate_ID)
        BEGIN
            THROW 50064, 'Certificate ID already exists. Please provide a unique Certificate ID.', 1;
        END
        -- Insert the student certificate
        INSERT INTO student_certificates (Certificate_ID, st_id, certificate_provider_name, certificate_issue_date)
        VALUES (@Certificate_ID, @st_id, @certificate_provider_name, @certificate_issue_date);
        PRINT 'Student certificate inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

23)InsertGraduateworkatCompany

The "InsertGraduateworkatCompany" stored procedure inserts a record indicating a graduate's employment at a company. It first checks if the provided Graduate_ID exists in the "Graduates" table to ensure the validity of the graduate ID. Then, it verifies if the provided company code exists in the "Graduate_company" table, ensuring the validity of the company code. If both checks pass, the record of the graduate's employment at the company is inserted into the "graduatesworkcompany" table. Error handling is implemented to manage exceptions during execution, providing informative error messages for encountered issues.

```
--23)/*Graduates_work_at_company*/
create PROC InsertGraduateworkatCompany @Graduate_ID INT, @company_code VARCHAR(250)
AS
BEGIN
    BEGIN TRY
        -- Check if Graduate_ID exists in the Graduates table
        IF NOT EXISTS (SELECT 1 FROM Graduates WHERE Graduate_ID = @Graduate_ID)
        BEGIN
            THROW 50064, 'Invalid graduate ID. Please provide a valid graduate ID.', 1;
        END
        -- Check if Company_code exists in the Gradute_company table
        IF NOT EXISTS (SELECT 1 FROM Graduate_company WHERE company_code = @company_code)
        BEGIN
            THROW 50065, 'Invalid company code. Please provide a valid company code.', 1;
        END
        -- Insert the graduate's work at company record
        INSERT INTO graduatesworkcompany (Graduate_ID, company_code)
        VALUES (@Graduate_ID, @company_code);
        PRINT 'Graduate work at company record inserted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

8.1.2. Select Stored Procedure

1) **Selectbranch**

The "selectbranch" stored procedure retrieves data for a specific branch based on the provided branch ID. It first checks if the branch ID exists in the "branches" table. If the branch ID is invalid or does not exist, it throws an error with code 70001 and a corresponding message. If the branch ID is valid, it proceeds to select and display all data associated with that branch from the "branches" table. Error handling is implemented to print any encountered errors during execution for debugging purposes.

```
--1)Branches
create proc selectbranch @branch_id int
as
begin
    begin try
        -- Check if the branch exists
        if not exists (select 1 from branches where branch_id = @branch_id)
        begin
            throw 70001, 'Invalid branch ID. Please provide a valid branch ID.', 1;
        end
        -- Select branch data
        select *
        from branches
        where branch_id = @branch_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

2) **Selectquestionwithchoices**

The "selectquestionwithchoices" stored procedure retrieves data for a specific question along with its associated choices based on the provided question ID. It first checks if the question ID exists in the "questions" table. If the question ID is invalid or does not exist, it throws an error with code 70002 and a corresponding message. If the question ID is valid, it proceeds to select and display all data associated with that question, including its choices, from the "questions" and "question_choices" tables using a left join. Error handling is implemented to print any encountered errors during execution for debugging purposes.

```
--2)Question and choices
create procedure selectquestionwithchoices @question_id int
as
begin
    begin try
        -- Check if the question exists
        if not exists (select 1 from questions where question_id = @question_id)
        begin
            throw 70002, 'Invalid question ID. Please provide a valid question ID.', 1;
        end
        -- Select question data
        select q.*, qc.choice
        from questions q
        left join question_choices qc on q.question_id = qc.question_id
        where q.question_id = @question_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

3) Selectfaculty

This stored procedure verifies the existence of a faculty code in the database. If the provided code is not found, it throws an error indicating an invalid code. Upon success, it retrieves all data associated with the faculty identified by the provided code. It employs error handling to catch any exceptions during execution.

```
--3)Faculty
create procedure selectfaculty @fcode varchar(200)
as
begin
    begin try
        -- Check if the faculty exists
        if not exists (select 1 from faculty where fcode = @fcode)
        begin
            throw 70003, 'Invalid faculty code. Please provide a valid faculty code.', 1;
        end
        -- Select faculty data
        select *
        from faculty
        where fcode = @fcode;
    end try
    begin catch
        print error_message();
    end catch
end;
```

4) Selectstudentaddress

This stored procedure checks if a student address exists in the database based on the provided zip code. If the zip code is not found, it throws an error indicating an invalid zip code. Upon success, it retrieves all data associated with the student address identified by the provided zip code. It includes error handling to capture and display any exceptions encountered during execution.

```
--4)Student_address
create procedure selectstudentaddress @zipcode int
as
begin
    begin try
        -- Check if the student address exists
        if not exists (select 1 from student_address where zipcode = @zipcode)
        begin
            throw 70004, 'Invalid zipcode. Please provide a valid zipcode.', 1;
        end
        -- Select student address data
        select *
        from student_address
        where zipcode = @zipcode;
    end try
    begin catch
        print error_message();
    end catch
end;
```

5) Instructor_address

This procedure verifies the existence of an instructor address in the database based on the provided zip code. If the zip code is not found, it throws an error indicating an invalid zip code. Upon success, it retrieves all data associated with the instructor address identified by the provided zip code. It includes error handling to capture and display any exceptions encountered during execution.

```
--5)Instructor_address
create procedure selectinstructoraddress @zipcode int
as
begin
    begin try
        -- Check if the instructor address exists
        if not exists (select 1 from instructor_address where zipcode = @zipcode)
        begin
            throw 70005, 'Invalid zipcode. Please provide a valid zipcode.', 1;
        end
        -- Select instructor address data
        select *
        from instructor_address
        where zipcode = @zipcode;
    end try
    begin catch
        print error_message();
    end catch
end;
```

6) Selectgradutecompany

This procedure verifies the existence of a graduate company in the database based on the provided company code. If the company code is not found, it throws an error indicating an invalid company code. Upon success, it retrieves all data associated with the graduate company identified by the provided company code. It includes error handling to capture and display any exceptions encountered during execution.

```
--6)Gradute_company
create procedure selectgradutecompany @company_code varchar(250)
as
begin
    begin try
        -- Check if the graduate company exists
        if not exists (select 1 from Graduate_company where company_code = @company_code)
        begin
            throw 70006, 'Invalid company code. Please provide a valid company code.', 1;
        end
        -- Select graduate company data
        select *
        from Graduate_company
        where company_code = @company_code;
    end try
    begin catch
        print error_message();
    end catch
end;
```

7) Selectinstructors

This procedure validates the existence of an instructor in the database based on the provided instructor ID. If the instructor ID is not found, it throws an error indicating an invalid instructor ID. Upon successful validation, it retrieves all data associated with the instructor identified by the provided ID. Error handling is implemented to capture and display any exceptions encountered during execution.

```
--7)Instructors
create procedure selectinstructors @inst_id int
as
begin
    begin try
        -- Check if the instructor exists
        if not exists (select 1 from instructors where inst_id = @inst_id)
        begin
            throw 70007, 'Invalid instructor ID. Please provide a valid instructor ID.', 1;
        end
        -- Select instructor data
        select *
        from instructors
        where inst_id = @inst_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

8) Selectinstructorphone

This procedure verifies the existence of an instructor's phone record in the database based on the provided instructor ID. If no record is found for the given ID, it throws an error indicating invalid instructor ID or phone number. Upon successful validation, it retrieves all data associated with the instructor's phone record identified by the provided ID. Error handling is implemented to capture and display any exceptions encountered during execution.

```
--8)Instructor Phone
create procedure selectinstructorphone @inst_id int
as
begin
    begin try
        -- Check if the instructor phone record exists
        if not exists (select 1 from instructor_phone where inst_id = @inst_id )
        begin
            throw 70008, 'Invalid instructor ID or phone number. Please provide valid information.', 1;
        end
        -- Select instructor phone record
        select *
        from instructor_phone
        where inst_id = @inst_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

9) Selectcourses

This procedure checks for the existence of a course in the database based on the provided course ID. If the course does not exist, it throws an error indicating an invalid course ID. Upon successful validation, it retrieves all data associated with the course identified by the provided ID. Error handling ensures that any exceptions encountered during execution are captured and displayed.

```
--9)Courses
create procedure selectcourses  @course_id int
as
begin
    begin try
        -- Check if the course exists
        if not exists (select 1 from courses where course_id = @course_id)
        begin
            throw 70009, 'Invalid course ID. Please provide a valid course ID.', 1;
        end
        -- Select course data
        select *
        from courses
        where course_id = @course_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

10) Selectexams

This procedure verifies the existence of an exam based on the provided exam ID. If the exam does not exist, it throws an error indicating an invalid exam ID. Upon successful validation, it retrieves all data associated with the exam identified by the provided ID. Error handling ensures that any exceptions encountered during execution are captured and displayed.

```
--10)Exams
create procedure selectexams  @exam_id int
as
begin
    begin try
        -- Check if the exam exists
        if not exists (select 1 from exams where exam_id = @exam_id)
        begin
            throw 70010, 'Invalid exam ID. Please provide a valid exam ID.', 1;
        end
        -- Select exam data
        select *
        from exams
        where exam_id = @exam_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

11) Selecttopics

This procedure checks for the existence of a topic based on the provided topic ID. If the topic does not exist, it throws an error indicating an invalid topic ID. Upon successful validation, it retrieves all data associated with the topic identified by the provided ID. Error handling ensures that any exceptions encountered during execution are captured and displayed.

```
--11)Topics
create procedure selecttopics  @topic_id int
as
begin
    begin try
        -- Check if the topic exists
        if not exists (select 1 from topics where topic_id = @topic_id)
        begin
            throw 70011, 'Invalid topic ID. Please provide a valid topic ID.', 1;
        end
        -- Select topic data
        select *
        from topics
        where topic_id = @topic_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

12) selecttracks

This procedure validates the existence of a track using the provided track ID. If the track does not exist, it throws an error indicating an invalid track ID. Upon successful validation, it retrieves all data associated with the track identified by the provided ID. Error handling ensures that any exceptions encountered during execution are captured and displayed.

```
--12)Tracks
create procedure selecttracks @track_id int
as
begin
    begin try
        -- Check if the track exists
        if not exists (select 1 from tracks where track_id = @track_id)
        begin
            throw 70012, 'Invalid track ID. Please provide a valid track ID.', 1;
        end
        -- Select track data
        select *
        from tracks
        where track_id = @track_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

13) selectintake

This procedure verifies the existence of an intake based on the provided intake ID. If the intake does not exist, it throws an error indicating an invalid intake ID. If the intake exists, it retrieves all associated data related to that intake ID. Error handling ensures proper notification in case of exceptions during execution.

```
--13)Intake
create procedure selectintake @intake_id int
as
begin
    begin try
        -- Check if the intake exists
        if not exists (select 1 from intake where intake_id = @intake_id)
        begin
            throw 70013, 'Invalid intake ID. Please provide a valid intake ID.', 1;
        end
        -- Select intake data
        select *
        from intake
        where intake_id = @intake_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

14) selectrounds

This procedure checks for the existence of a round based on the provided round ID. If the round does not exist, it throws an error indicating an invalid round ID. If the round exists, it retrieves all associated data related to that round ID. Error handling ensures proper notification in case of exceptions during execution.

```
--14)Round
create procedure selectrounds @round_id int
as
begin
    begin try
        -- Check if the round exists
        if not exists (select 1 from rounds where round_id = @round_id)
        begin
            throw 70014, 'Invalid round ID. Please provide a valid round ID.', 1;
        end
        -- Select round data
        select *
        from rounds
        where round_id = @round_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

15) selectstudentsinfo

This procedure verifies the existence of a student based on the provided student ID. If the student does not exist, it throws an error indicating an invalid student ID. If the student exists, it retrieves all associated data related to that student ID. Error handling ensures proper notification in case of exceptions during execution.

```
--15)Students
create procedure selectstudentsinfo  @st_id int
as
begin
    begin try
        -- Check if the student exists
        if not exists (select 1 from studentsinfo where st_id = @st_id)
        begin
            throw 70015, 'Invalid student ID. Please provide a valid student ID.', 1;
        end
        -- Select student data
        select *
        from studentsinfo
        where st_id = @st_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

16) selectstudentphone

This procedure checks for the existence of a phone record associated with the provided student ID. If no phone record is found for the given student ID, it throws an error indicating an invalid student ID. Otherwise, it retrieves and returns all phone data associated with the student ID. Error handling ensures proper notification in case of exceptions during execution.

```
--16)Student Phone
create procedure selectstudentphone @st_id int
as
begin
    begin try
        -- Check if the student phone record exists
        if not exists (select 1 from student_phone where st_id = @st_id)
        begin
            throw 70016, 'Invalid student ID. No phone number found for the provided student ID.', 1;
        end
        -- Select student phone data
        select *
        from student_phone
        where st_id = @st_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

17)Selecttrackcontainscourses

This procedure verifies if the specified track contains any courses. If no courses are found for the given track ID, it throws an error indicating the absence of courses. Otherwise, it selects and returns the track ID along with the associated course IDs and names. Error handling ensures proper notification in case of exceptions during execution.

```
--17)Track_Contains_Courses
create procedure selecttrackcontainscourses  @track_id int
as
begin
    begin try
        -- Check if the track contains the specified course
        if not exists (
            select 1
            from track_contains_courses
            where track_id = @track_id
        )
        begin
            throw 70017, 'The specified track does not contain the specified course.', 1;
        end
        -- Select track-course relationship data
        select t.Track_ID,c.Course_ID,c.Course_Name
        from track_contains_courses t
                left join Courses c on t.Course_ID=c.Course_ID
                where track_id = @track_id ;

    end try
    begin catch
        print error_message();
    end catch
end;
```

18) selectstudentenrollintake

This procedure checks if the provided student ID has any enrollment record in any intake. If no record is found, it throws an error indicating the absence of enrollment. Otherwise, it selects and returns the student enrollment data along with the intake number. Error handling ensures proper notification in case of exceptions during execution.

```
--18)Student_enroll_in_intake
create procedure selectstudentenrollintake @st_id int
as
begin
    begin try
        -- Check if the student enrollment in intake exists
        if not exists (select 1 from student_enroll_intake where st_id = @st_id)
        begin
            throw 70018, 'Invalid student ID. No enrollment record found for the provided student ID.', 1;
        end
        -- Select student enrollment data for intake
        select s.*, i.Intake_NO
        from student_enroll_intake s
                left join Intake i on s.Intake_ID=i.Intake_ID
        where st_id = @st_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

19) selectstudentenrollround

This procedure verifies whether the provided student ID has any enrollment record in any round. If no record is found, it throws an error indicating the absence of enrollment. Otherwise, it selects and returns the student enrollment data along with the round number. Error handling ensures proper notification in case of exceptions during execution.

```
--19)Student_enroll_in_Round
create procedure selectstudentenrollround @st_id int
as
begin
    begin try
        -- Check if the student enrollment in round exists
        if not exists (select 1 from student_enroll_round where st_id = @st_id)
        begin
            throw 70019, 'Invalid student ID. No enrollment record found for the provided student ID.', 1;
        end
        -- Select student enrollment data for round
        select r.*,ro.Round_NO
        from Student_enroll_Round r
                left join Rounds ro on r.Round_ID=ro.Round_ID
        where st_id = @st_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

20) selectstudentanswersexam

This procedure checks if the provided student ID has any recorded answers for exams. If no answers are found, it throws an error indicating the absence of data for the student and exam. Otherwise, it selects and returns the student's answers for the exam. Error handling ensures proper notification in case of exceptions during execution.

```
--20)Student_Answers_Exam
create procedure selectstudentanswersexam @st_id int
as
begin
    begin try
        -- Check if the student's answers for the exam exist
        if not exists (
            select 1
            from student_answers_exam
            where st_id = @st_id
        )
        begin
            throw 70020, 'Invalid student ID or exam ID. No answers found for the provided student and exam.', 1;
        end
        -- Select student's answers for the exam
        select *
        from student_answers_exam
        where st_id = @st_id ;
    end try
    begin catch
        print error_message();
    end catch
end;
```

21) selectgraduates

This procedure checks if the provided graduate ID exists in the database. If no record is found, it throws an error indicating the absence of data for the provided graduate ID. Otherwise, it selects and returns the graduate's data. Error handling ensures proper notification in case of exceptions during execution.

```
--21)Graduates
create procedure selectgraduates @graduate_id int
as
begin
    begin try
        -- Check if the graduate exists
        if not exists (select 1 from graduates where graduate_id = @graduate_id)
        begin
            throw 70021, 'Invalid graduate ID. No record found for the provided graduate ID.', 1;
        end
        -- Select graduate data
        select *
        from graduates
        where graduate_id = @graduate_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

22) selectGraduatesfailreasons

This procedure verifies the existence of fail reasons for a given graduate ID. If no fail reasons are found, it throws an error indicating the absence of such data. Otherwise, it retrieves and returns the fail reasons associated with the provided graduate ID. Error handling ensures appropriate notification in case of exceptions during execution.

```
--22)Graduate_Fail_Reasons
create procedure selectGraduatesfailreasons @graduate_id int
as
begin
    begin try
        -- Check if the graduate's fail reasons exist
        if not exists (
            select 1
            from Graduate_Fail_Reasons
            where graduate_id = @graduate_id
        )
        begin
            throw 70022, 'Invalid graduate ID. No fail reasons found for the provided graduate ID.', 1;
        end
        -- Select fail reasons for the graduate
        select *
        from Graduate_Fail_Reasons
        where graduate_id = @graduate_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

23) Student_Certificates

This procedure verifies the existence of a student certificate based on the provided certificate ID. If no certificate is found, it throws an error indicating the absence of such data. Otherwise, it retrieves and returns the certificate information associated with the provided certificate ID. Error handling ensures appropriate notification in case of exceptions during execution.

```
--23)Student_Certificates
create procedure selectstudentcertificates @certificate_id int
as
begin
    begin try
        -- Check if the certificate exists
        if not exists (select 1 from student_certificates where certificate_id = @certificate_id)
        begin
            throw 70023, 'Invalid certificate ID. No record found for the provided certificate ID.', 1;
        end
        -- Select certificate data
        select *
        from student_certificates
        where certificate_id = @certificate_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

24) selectgraduatesworkatcompany

This stored procedure checks whether a graduate is associated with a company based on the provided graduate ID. If no such association is found, it throws an error indicating the absence of related records. Otherwise, it retrieves and returns the company information where the graduate works. Error handling ensures proper notification in case of exceptions during execution.

```
--24)Graduates_work_at_company
create procedure selectgraduatesworkatcompany @graduate_id int
as
begin
    begin try
        -- Check if the graduate works at a company
        if not exists (
            select 1
            from graduatesworkcompany
            where graduate_id = @graduate_id
        )
        begin
            throw 70024, 'Invalid graduate ID. No record found for the provided graduate ID.', 1;
        end
        -- Select company information where the graduate works
        select *
        from graduatesworkcompany
        where graduate_id = @graduate_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

25) selectStudent_Takes_Exam

This stored procedure verifies whether a student has taken an exam by checking for their record in the table using the provided student ID. If no such record is found, it throws an error indicating the absence of grades for the specified student and exam. If the record exists, it retrieves and returns the student's grade for the exam. Error handling ensures proper notification of any exceptions during execution.

```
--25)Student_Takes_Exam
create procedure selectStudent_Takes_Exam @st_id int
as
begin
    begin try
        -- Check if the student's answers for the exam exist
        if not exists (
            select 1
            from Student_Takes_Exam
            where st_id = @st_id
        )
        begin
            throw 70020, 'Invalid student ID . No Grade found for the provided student and exam.', 1;
        end
        -- Select student's Grade for the exam
        select *
        from Student_Takes_Exam
        where st_id = @st_id ;
    end try
    begin catch
        print error_message();
    end catch
end;
```

8.1.3. Delete Stored Procedure

1)Deletebranch

This stored procedure deletes a branch and its associated records from various tables in the database, provided a valid branch ID. It first checks if the branch exists and if it's not the main branch (Branch number 1). Then, it deletes records related to student certificates, phone numbers, enrollments, exams, graduates, and tracks associated with the branch. After that, it reassigns instructors to Branch number 1, updates their branch IDs, and finally deletes the branch itself. Error handling ensures proper notification of any exceptions during execution.

```
--1)Branches
--assingn instructors and tracks in this branch to branch number 1 "Smart Village_Main branch"
--Cannot delete branch number 1
--when delete branch, assign instructors to main branch

create PROCEDURE deletebranch @branch_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if the branch exists
        IF NOT EXISTS (SELECT 1 FROM Branches WHERE Branch_ID = @branch_id)
        BEGIN
            THROW 60001, 'Invalid branch ID. Please provide a valid branch ID.', 1;
        END
        -- Check if the branch is Branch number 1
        IF @branch_id = 1
        BEGIN
            THROW 60002, 'Branch number 1 cannot be deleted. This is the main branch.', 1;
        END
                -- Delete student certificates associated with the branch
        DELETE FROM Student_Certificates WHERE St_ID IN (SELECT St_ID FROM StudentsInfo WHERE Branch_ID = @branch_id);

                -- Delete students' phone records associated with the branch
        DELETE FROM Student_Phone WHERE St_ID IN (SELECT St_ID FROM StudentsInfo WHERE Branch_ID = @branch_id);

        -- Delete students' round enrollment records associated with the branch
        DELETE FROM Student_enroll_Round WHERE St_ID IN (SELECT St_ID FROM StudentsInfo WHERE Branch_ID = @branch_id);

                -- Delete student intake enrollment records associated with the branch
        DELETE FROM Student_enroll_intake WHERE St_ID IN (SELECT St_ID FROM StudentsInfo WHERE Branch_ID = @branch_id);

                -- Delete students' exam records associated with the branch
        DELETE FROM Student_Takes_Exam WHERE St_ID IN (SELECT St_ID FROM StudentsInfo WHERE Branch_ID = @branch_id);



            -- Delete graduates' phone records associated with the branch
        DELETE FROM Graduate_Phone WHERE Graduate_ID IN (SELECT Graduate_ID FROM Graduates WHERE Track_ID IN (SELECT Track_ID FROM Tracks WHERE Branch_ID = @branch_id));

        -- Delete graduates' work company records associated with the branch
        DELETE FROM GraduatesWorkCompany WHERE Graduate_ID IN (SELECT Graduate_ID FROM Graduates WHERE Track_ID IN (SELECT Track_ID FROM Tracks WHERE Branch_ID = @branch_id));

        -- Delete graduates associated with the branch
        DELETE FROM Graduates WHERE Track_ID IN (SELECT Track_ID FROM Tracks WHERE Branch_ID = @branch_id);

                -- Delete students associated with the branch
        DELETE FROM StudentsInfo WHERE Branch_ID = @branch_id;

                -- Delete tracks associated with the branch
        DELETE FROM Tracks WHERE Branch_ID = @branch_id;

                -- Reassign instructors to Branch number 1
        UPDATE Instructors
        SET Branch_ID = 1
        WHERE Branch_ID = @branch_id;

        -- Delete the branch
        DELETE FROM Branches WHERE Branch_ID = @branch_id;

        PRINT 'Branch, associated tracks, graduates, students, and enrollment records deleted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

2) deletequestionwithchoices

This stored procedure deletes a question along with its associated choices and related records from the database, given a valid question ID. It first checks if the question ID exists in the Questions table. Then, it deletes records associated with the question ID from the Student_Answers_Exam table and the Exam_Quest_Generation table. After that, it deletes choices associated with the question from the question_choices table. Finally, it deletes the question itself from the questions table. Error handling ensures proper notification of any exceptions during execution.

```
--2)/*Question with choices*/
--when you delete Question you should delete choices

ALTER PROCEDURE deletequestionwithchoices @question_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if Question_ID exists in the Questions table
        IF NOT EXISTS (SELECT 1 FROM questions WHERE question_id = @question_id)
        BEGIN
            THROW 60004, 'Invalid question ID. Please provide a valid question ID.', 1;
        END
        -- Delete records associated with the question ID from the Student_Answers_Exam table
        DELETE FROM Student_Answers_Exam WHERE Question_ID = @question_id;

                -- Delete records associated with the question ID from the Exam_Quest_Generation table
        DELETE FROM Exam_Quest_Generation WHERE Question_ID = @question_id;

        -- Delete choices associated with the question
        DELETE FROM question_choices WHERE question_id = @question_id;

        -- Delete the question
        DELETE FROM questions WHERE question_id = @question_id;
        PRINT 'Question, associated choices, and related records deleted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

3) deletefaculty

This stored procedure deletes a faculty record from the database based on the provided faculty code (@fcode). It first checks if the faculty exists in the faculty table. If the faculty does not exist, it throws an error indicating that the provided faculty code is invalid. Additionally, it checks if the faculty has associated students in the studentsinfo table; if so, it throws an error indicating that the faculty cannot be deleted because it has associated students. Finally, if no errors occur, it deletes the faculty record from the faculty table and prints a success message. Error handling ensures proper notification of any exceptions during execution.

```
--3)/*Faculty*/
create procedure deletefaculty @fcode varchar(200)
as
begin
    begin try
        -- Check if the faculty exists
        if not exists (select 1 from faculty where fcode = @fcode)
        begin
            throw 60005, 'Invalid faculty code. Please provide a valid faculty code.', 1;
        end
        -- Check if the faculty has associated students
        if exists (select 1 from studentsinfo where fcode = @fcode)
        begin
            throw 60006, 'Cannot delete faculty. Faculty has associated students.', 1;
        end
        -- Delete the faculty
        delete from faculty
        where fcode = @fcode;
        print 'Faculty deleted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

4) deletestudentaddress

This stored procedure deletestudentaddress deletes a student's address record from the database based on the provided ZIP code (@zipcode). It first checks if the student address exists in the student_address table. If the address does not exist, it throws an error indicating that the provided ZIP code is invalid. Additionally, it checks if the address has associated students in the studentsinfo table; if so, it throws an error indicating that the address cannot be deleted because it has associated students. Finally, if no errors occur, it deletes the address record from the student_address table and prints a success message. Error handling ensures proper notification of any exceptions during execution.

```
--4)/*Student_address*/

create proc deletestudentaddress @zipcode int
as
begin
    begin try
        -- Check if the student address exists
        if not exists (select 1 from student_address where zipcode = @zipcode)
        begin
            throw 60007, 'Invalid zipcode. Please provide a valid zipcode.', 1;
        end
        -- Check if the address has associated students
        if exists (select 1 from studentsinfo where zipcode = @zipcode)
        begin
            throw 60008, 'Cannot delete student address. Address has associated students.', 1;
        end
        -- Delete the address
        delete from student_address
        where zipcode = @zipcode;
        print 'Student address deleted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

5) deleteinstructoraddress

This stored procedure deleteinstructoraddress deletes an instructor's address record from the database based on the provided ZIP code (@zipcode). It first checks if the instructor address exists in the instructor_address table. If the address does not exist, it throws an error indicating that the provided ZIP code is invalid. Additionally, it checks if the address has associated instructors in the instructors table; if so, it throws an error indicating that the address cannot be deleted because it has associated instructors. Finally, if no errors occur, it deletes the address record from the instructor_address table and prints a success message. Error handling ensures proper notification of any exceptions during execution.

```
--5)/*Instructor_address*/
create proc deleteinstructoraddress  @zipcode int
as
begin
    begin try
        -- Check if the instructor address exists
        if not exists (select 1 from instructor_address where zipcode = @zipcode)
        begin
            throw 60009, 'Invalid zipcode. Please provide a valid zipcode.', 1;
        end
        -- Check if the address has associated instructors
        if exists (select 1 from instructors where zipcode = @zipcode)
        begin
            throw 60010, 'Cannot delete instructor address. Address has associated instructors.', 1;
        end
        -- Delete the address
        delete from instructor_address
        where zipcode = @zipcode;
        print 'Instructor address deleted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

6) deletegraduatedcompany

This stored procedure deletegraduatedcompany is responsible for deleting a graduated company record from the database based on the provided company code (@company_code). It first checks if the company exists in the graduate_company table. If the company does not exist, it throws an error indicating that the provided company code is invalid. Additionally, it checks if the company has associated graduates in the Graduatesworkcompany table; if so, it throws an error indicating that the company cannot be deleted because it has associated graduates. Finally, if no errors occur, it deletes the company record from the graduate_company table and prints a success message. Error handling ensures proper notification of any exceptions during execution.

```
--6)Gradute_company
alter proc deletegraduatedcompany  @company_code varchar(250)
as
begin
    begin try
        -- Check if the company exists
        if not exists (select 1 from graduate_company where company_code = @company_code)
        begin
            throw 60011, 'Invalid company code. Please provide a valid company code.', 1;
        end
        -- Check if the company has associated graduates
        if exists (select 1 from Graduatesworkcompany where Company_code = @company_code)
        begin
            throw 60012, 'Cannot delete graduated company. Company has associated graduates.', 1;
        end
        -- Delete the company
        delete from Graduate_company
        where company_code = @company_code;
        print 'Graduated company deleted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

7) deleteinstructor

The deleteinstructor stored procedure removes an instructor's record from the database based on the provided instructor ID (@inst_id). It also reassigns the courses taught by the deleted instructor to a new instructor specified by @new_inst_id.

It first verifies if the instructor exists in the database. If not, it throws an error indicating an invalid instructor ID.

Next, it updates the course records to assign them to the new instructor, ensuring continuity in course assignments.

It then proceeds to delete the instructor's phone records and finally deletes the instructor's record.

The procedure includes error handling to manage any exceptions that may arise during execution. Upon successful completion, it prints a message confirming the deletion of the instructor, associated phone records, and the reassignment of courses.

```
--7)Instructors
alter PROCEDURE deleteinstructor @inst_id INT, @new_inst_id INT -- The ID of the instructor to which courses will be reassigned
AS
BEGIN
    BEGIN TRY
        -- Check if the instructor exists
        IF NOT EXISTS (SELECT 1 FROM instructors WHERE inst_id = @inst_id)
        BEGIN
            THROW 60013, 'Invalid instructor ID. Please provide a valid instructor ID.', 1;
        END
        -- Reassign courses to the new instructor
        UPDATE Courses
        SET Inst_ID = @new_inst_id
        WHERE Inst_ID = @inst_id;
        -- Delete the instructor's phone records
        DELETE FROM instructor_phone
        WHERE inst_id = @inst_id;
        -- Delete the instructor
        DELETE FROM instructors
        WHERE inst_id = @inst_id;
        PRINT 'Instructor, associated phone records, and courses reassigned successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

8) delete_topic

The delete_topic procedure removes a topic based on its ID. First, it confirms the topic's existence; if not found, it throws an error. It then checks if the topic is the sole topic associated with a course; if so, it throws an error as a course must have at least two topics. Finally, if both conditions are met, it deletes the topic from the database. Error handling ensures any exceptions are caught and displayed appropriately.

```
--8)/*Topics*/
create procedure delete_topic  @topic_id int
as
begin
    begin try
        -- Check if Topic_ID exists in the Topics table
        if not exists (select 1 from topics where topic_id = @topic_id)
        begin
            throw 50014, 'Invalid topic ID. Please provide a valid topic ID.', 1;
        end
        -- Check if the topic is the only topic associated with the course
        declare @course_id int;
        select @course_id = course_id from topics where topic_id = @topic_id;

        if (select count(*) from topics where course_id = @course_id) <= 2
        begin
            throw 50015, 'Cannot delete topic. Course must have at least two topic.', 1;
        end
        -- Delete the topic
        delete from topics where topic_id = @topic_id;
        print 'Topic deleted successfully.';
    end try
    begin catch
        print error_message();
    end catch
end;
```

9)deletestudent

The deletestudent procedure deletes a student and associated records from the database based on the provided student ID. It first checks if the student exists; if not, it throws an error. Then, it deletes records related to the student's exam attempts, enrollment in intake and round, certificates, and phone records. Error handling ensures any exceptions are caught and displayed. Upon successful deletion, it prints a confirmation message.

```
--9)/*Students*/
alter PROCEDURE deletestudent @St_ID int
AS
BEGIN
    BEGIN TRY
        -- Check if the student exists
        IF NOT EXISTS (SELECT 1 FROM StudentsInfo WHERE St_ID = @St_ID)
        BEGIN
            THROW 60016, 'Invalid student ID. Please provide a valid student ID.', 1;
        END
        -- Delete records from Student_Takes_Exam associated with the student
        DELETE FROM Student_Takes_Exam WHERE St_ID = @St_ID;

        -- Check if the student is enrolled in any intake
        IF EXISTS (SELECT 1 FROM Student_enroll_intake WHERE St_ID = @St_ID)
        BEGIN
            -- If the student is enrolled in any intake, delete the enrollment record
            DELETE FROM Student_enroll_intake WHERE St_ID = @St_ID;
            PRINT 'Student enrollment in intake deleted.';
        END

        -- Check if the student is enrolled in any round
        IF EXISTS (SELECT 1 FROM Student_enroll_Round WHERE St_ID = @St_ID)
        BEGIN
            -- If the student is enrolled in any round, delete the enrollment record
            DELETE FROM Student_enroll_Round WHERE St_ID = @St_ID;
            PRINT 'Student enrollment in round deleted.';
        END


          -- Delete the Student_Certificates records
          DELETE FROM Student_Certificates WHERE St_ID = @St_ID;

          -- Delete the student's phone records
          DELETE FROM Student_Phone WHERE St_ID = @St_ID;

          -- Delete the student
          DELETE FROM StudentsInfo WHERE St_ID = @St_ID;
          PRINT 'Student and associated records deleted successfully.';
      END TRY
      BEGIN CATCH
          PRINT ERROR_MESSAGE();
      END CATCH
  END;
```

10) delete_graduate

The delete_graduate procedure removes a graduate and all related records from the database, based on the provided graduate ID. It verifies the existence of the graduate and throws an error if not found. Then, it proceeds to delete associated records from tables such as Graduate_Phone, Graduate_Fail_Reasons, and Graduatesworkcompany. Finally, it deletes the graduate's record from the graduates table. Error handling ensures any exceptions are caught and displayed, and upon successful deletion, it prints a confirmation message.

```
--10)/*Graduate*/
alter PROCEDURE delete_graduate @graduate_id INT
AS
BEGIN
    BEGIN TRY
        -- Check if the graduate exists
        IF NOT EXISTS (SELECT 1 FROM graduates WHERE graduate_id = @graduate_id)
        BEGIN
            THROW 60017, 'Invalid graduate ID. Please provide a valid graduate ID.', 1;
        END

        -- Delete records from Graduate_Phone table associated with the graduate
        DELETE FROM Graduate_Phone WHERE Graduate_ID = @graduate_id;

        -- Delete records from Graduate_Fail_Reasons table associated with the graduate
        DELETE FROM Graduate_Fail_Reasons WHERE Graduate_ID = @graduate_id;

        -- Delete records from Graduates_work_at_company table associated with the graduate
        DELETE FROM Graduatesworkcompany WHERE Graduate_ID = @graduate_id;

        -- Delete the graduate
        DELETE FROM graduates WHERE graduate_id = @graduate_id;
        PRINT 'Graduate and associated records deleted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

11) delete_track

The delete_track procedure handles the removal of a track from the database based on the provided track ID. It first validates the existence of the track and throws an error if not found. Subsequently, it checks for associated graduates and takes different actions based on their certification status. If there are graduates with incomplete certification, it removes associated records from tables like Graduatesworkcompany, Graduate_Fail_Reasons, and Graduate_Phone before deleting the graduates themselves.

If no incomplete certifications are found, it directly deletes associated records for graduates and proceeds with deleting the track's records from the Track_Contains_Courses table and finally removes the track itself.

Error handling ensures any exceptions are caught and displayed, while successful deletion prompts a confirmation message.

```
            END
            ELSE
            BEGIN
                -- Remove graduates associated with the track from Graduates_work_at_company
                DELETE FROM Graduatesworkcompany WHERE graduate_id IN (SELECT graduate_id FROM graduates WHERE track_id = @track_id);

                -- Remove graduates associated with the track from Graduate_Phone
                DELETE FROM Graduate_Phone WHERE Graduate_ID IN (SELECT graduate_id FROM graduates WHERE track_id = @track_id);

                -- Remove graduates associated with the track
                DELETE FROM graduates WHERE track_id = @track_id;
            END
        END

        -- Delete records from Track_Contains_Courses
        DELETE FROM Track_Contains_Courses WHERE Track_ID = @track_id;

        -- Delete the track
        DELETE FROM tracks WHERE track_id = @track_id;

        PRINT 'Track deleted successfully.';
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

8.1.4. Update Stored Procedure

1) updatebranch

The updatebranch procedure allows for the modification of branch information, specifically the manager's name, identified by the branch ID provided.It first validates the existence of the branch, throwing an error if not found.Upon successful validation, it proceeds to update the manager's name in the branches table for the specified branch ID.Error handling ensures any exceptions are caught and displayed, providing clarity in case of issues during the update process.

```
--1)Branches
create procedure updatebranch @branch_id int, @mngr_name varchar(100)
as
begin
    begin try
        -- Check if the branch exists
        if not exists (select 1 from branches where branch_id = @branch_id)
        begin
            throw 80001, 'Invalid branch ID. No record found for the provided branch ID.', 1;
        end
        -- Update branch information
        update branches
        set mngrname = @mngr_name
        where branch_id = @branch_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

2) updatequestion

The updatequestion procedure facilitates the modification of question details, such as the correct answer and question level, based on the provided question ID.
It initially verifies the existence of the question, throwing an error if not found.
Upon successful validation, it proceeds to update the correct answer and question level fields in the questions table for the specified question ID.
Error handling ensures any exceptions are caught and printed, ensuring clarity in case of issues during the update process.

```
--2)Questions
create procedure updatequestion @question_id int, @new_correct_answer varchar(250), @new_question_level varchar(50)
as
begin
    begin try
        -- Check if the question exists
        if not exists (select 1 from questions where question_id = @question_id)
        begin
            throw 80002, 'Invalid question ID. No record found for the provided question ID.', 1;
        end
        -- Update question information
        update questions
        set correctanswer = @new_correct_answer,
            question_level = @new_question_level
        where question_id = @question_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

3) updatequestionchoice

The updatequestionchoice procedure allows for the modification of a specific choice for a question identified by its question ID.
It first validates the existence of the question choice based on the provided question ID and choice. If the choice is not found, it throws an error.
Upon successful validation, it updates the specified choice with the new choice provided.
Error handling ensures that any exceptions encountered during the update process are caught and printed for clarity.

```
--3)Question_Choices
create procedure updatequestionchoice @question_id int, @choice varchar(250), @new_choice varchar(250)
as
begin
    begin try
        -- Check if the question choice exists
        if not exists (select 1 from question_choices where question_id = @question_id and choice = @choice)
        begin
            throw 80003, 'Invalid question ID or choice. No record found for the provided question ID and choice.', 1;
        end
        -- Update question choice
        update question_choices
        set choice = @new_choice
        where question_id = @question_id and choice = @choice;
    end try
    begin catch
        print error_message();
    end catch
end;
```

4) updatefaculty

The updatefaculty procedure enables the modification of the name of a faculty identified by its faculty code (fcode).

It first verifies the existence of the faculty in the database based on the provided faculty code.

If the faculty is not found, it throws an error indicating that the update cannot proceed.

Upon successful validation, it updates the faculty's name with the new name provided (fname).

Error handling ensures that any exceptions encountered during the update process are caught and printed for clarity.

```
--4)Faculty
create procedure updatefaculty @fcode varchar(200), @fname varchar(250)
as
begin
    begin try
        -- Check if the faculty exists
        if not exists (select 1 from faculty where fcode = @fcode)
        begin
            throw 80004, 'Faculty not found. Unable to update.', 1;
        end
        -- Update faculty information
        update faculty
        set fname = @fname
        where fcode = @fcode;
    end try
    begin catch
        print error_message();
    end catch
end;
```

5) updategradutescompany

The updategradutescompany procedure facilitates the update of a graduate company's location based on the provided company code (company_code) and the new company location (company_location).

It first verifies if the provided company code exists in the graduate_company table. If not, it throws an error indicating that the company code does not exist.

The procedure then checks if the provided company location contains the word 'egypt'. Depending on the presence of 'egypt', it validates whether the company code starts with the appropriate prefix ('egy-cmp' for companies in Egypt and 'intl-cmp' for international companies). If the validation fails, corresponding errors are thrown.

Finally, upon successful validation, it updates the company location in the graduate_company table with the new location provided.

Error handling ensures that any encountered exceptions during the update process are caught and printed for notification.

```
--5)/*Gradute_company*/
create PROCEDURE updategradutescompany  @company_code VARCHAR(250), @company_location VARCHAR(250)
AS
BEGIN
    BEGIN TRY
        -- Check if the company code exists
        IF NOT EXISTS (SELECT 1 FROM graduate_company WHERE company_code = @company_code)
        BEGIN
            THROW 80005, 'Company code does not exist.', 1;
        END
        -- Convert company location to lowercase for comparison
        SET @company_location = @company_location

        -- Check if the company location contains 'egypt'
        IF CHARINDEX('egypt', @company_location) > 0
        BEGIN
            -- Check if the company code starts with 'egy-cmp'
            IF LEFT(@company_code, 7) <> 'egy-cmp'
            BEGIN
                THROW 80006, 'Company location contains Egypt but company code does not start with ''egy-cmp''.', 1;
            END

        END
        ELSE
        BEGIN
            -- Check if the company code starts with 'intl-cmp'
            IF LEFT(@company_code, 8) <> 'intl-cmp'
            BEGIN
                THROW 80007, 'Company location does not contain Egypt but company code does not start with ''intl-cmp''.', 1;
            END
        END
        -- Update graduate_company table
        UPDATE graduate_company
        SET company_location = @company_location
        WHERE company_code = @company_code;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

6) update_instructor

The update_instructor procedure is designed to update an instructor's information based on the provided instructor ID (inst_id) along with new salary, degree, and years of experience values.

It first verifies if the provided instructor ID exists in the instructors table. If not, it throws an error indicating that the instructor ID does not exist.

Next, it checks if the new instructor salary is greater than 5000. If not, it throws an error indicating that the salary must be more than 5000.

Upon successful validation, it updates the instructors table with the new salary, degree, and experience year values for the specified instructor ID.

Error handling ensures that any encountered exceptions during the update process are caught and printed for notification.

```
--6)Instructors
CREATE PROCEDURE update_instructor @inst_id INT, @new_inst_salary MONEY, @new_inst_degree VARCHAR(50), @new_inst_experience_year INT
AS
BEGIN
    BEGIN TRY
        -- Check if the instructor exists
        IF NOT EXISTS (SELECT 1 FROM instructors WHERE inst_id = @inst_id)
        BEGIN
            THROW 80006, 'Instructor ID does not exist.', 1;
        END

        -- Check if the new instructor salary is more than 5000
        IF @new_inst_salary <= 5000
        BEGIN
            THROW 80008, 'Instructor salary must be more than 5000.', 1;
        END

        -- Update instructors table
        UPDATE instructors
        SET
            inst_salary = @new_inst_salary,
            inst_degree = @new_inst_degree,
            inst_experience_year = @new_inst_experience_year

        WHERE inst_id = @inst_id;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

7) update_instructor_phone

The update_instructor_phone procedure allows for updating an instructor's phone number based on the provided instructor ID (inst_id) and the new phone number (new_phone).

It first verifies if the instructor exists in the instructor_phone table by checking the existence of the provided instructor ID. If the instructor ID does not exist, it throws an error indicating so.

Subsequently, it updates the instructor_phone table by setting the phone number to the new phone number provided for the specified instructor ID.

Error handling ensures that any exceptions encountered during the update process are caught and printed for notification.

```
--7)Instructor Phone
create procedure update_instructor_phone @inst_id int,@new_phone varchar(20)
as
begin
    begin try
        -- Check if the instructor exists
        if not exists (select 1 from instructor_phone where inst_id = @inst_id)
        begin
            throw 80007, 'Instructor ID does not exist.', 1;
        end
        -- Update instructor_phone table
        update instructor_phone
        set phone = @new_phone
        where inst_id = @inst_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

8) update_course

The update_course procedure enables the modification of a course's status and instructor assignment based on the provided course ID (course_id), new course status (new_course_status), and new instructor ID (new_inst_id).

Initially, it validates the existence of the course in the courses table by checking if a record with the given course ID exists. If the course ID is not found, it throws an error indicating that the course ID does not exist.

Subsequently, it updates the courses table by setting the course status and instructor ID to the provided new values for the specified course ID.

Error handling is implemented to catch and print any encountered exceptions during the update process, ensuring appropriate notification of errors.

```
--8)Courses
create procedure update_course  @course_id int, @new_course_status varchar(50), @new_inst_id int
as
begin
    begin try
        -- Check if the course exists
        if not exists (select 1 from courses where course_id = @course_id)
        begin
            throw 80009, 'Course ID does not exist.', 1;
        end
        -- Update the course
        update courses
        set
            course_status = @new_course_status,
            inst_id = @new_inst_id
        where course_id = @course_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

9) update_exam

The update_exam procedure facilitates the modification of an exam's date based on the provided exam ID (exam_id) and the new exam date (new_exam_date).

It begins by verifying the existence of the exam in the exams table. If no record corresponding to the provided exam ID is found, it throws an error indicating that the exam ID does not exist.

Upon confirmation of the exam's existence, the procedure proceeds to update the exams table, setting the exam date to the specified new exam date for the given exam ID.

Error handling is implemented to capture and print any encountered exceptions during the update process, ensuring proper notification of errors.

```
--9)Exams
create procedure update_exam @exam_id int,  @new_exam_date date
as
begin
     begin try
          -- Check if the exam exists
          if not exists (select 1 from exams where exam_id = @exam_id)
          begin
               throw 80010, 'Exam ID does not exist.', 1;
          end
          -- Update the exam
          update exams
          set
               exam_date = @new_exam_date
          where exam_id = @exam_id;
     end try
     begin catch
          print error_message();
     end catch
end;
```

10) update_topic

The update_topic procedure allows for the modification of a topic's name based on the provided
topic ID (topic_id) and the new topic name (new_topic_name).
First, it verifies the existence of the topic in the topics table. If no record corresponding to the provided topic ID is found, it throws an error indicating that the topic ID does not exist.
Upon confirming the topic's existence, the procedure proceeds to update the topics table, setting the topic name to the specified new topic name for the given topic ID.
Error handling is implemented to capture and print any encountered exceptions during the update process, ensuring proper notification of errors.

```
--10)/*Topics*/
create procedure update_topic  @topic_id int,  @new_topic_name varchar(250)
as
begin
    begin try
        -- Check if the topic exists
        if not exists (select 1 from topics where topic_id = @topic_id)
        begin
            throw 80011, 'Topic ID does not exist.', 1;
        end
        -- Update the topic
        update topics
        set topic_name =  @new_topic_name
        where topic_id = @topic_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

11) update_track

The update_track procedure facilitates the modification of track information in the database based on the provided parameters: track_id, track_name, branch_id, and manager_id.

It first verifies if the track exists in the tracks table by checking if a record with the specified track_id exists. If not, it throws an error indicating that the provided track ID does not exist.

Subsequently, it ensures that the provided branch_id exists in the branches table and that the provided manager_id exists in the instructors table. If either of these conditions is not met, respective errors are thrown.

Once all checks pass, the procedure updates the tracks table, setting the track name, branch ID, and manager ID to the provided values for the specified track_id.

Error handling is implemented to catch and print any exceptions that may occur during the update process, providing feedback on encountered errors.

```
--11)Tracks
create procedure update_track @track_id int, @track_name varchar(250),  @branch_id int, @manager_id int
as
begin
    begin try
        -- Check if the track exists
        if not exists (select 1 from tracks where track_id = @track_id)
        begin
            throw 80012, 'Track ID does not exist.', 1;
        end
        -- Check if the branch ID exists
        if not exists (select 1 from branches where branch_id = @branch_id)
        begin
            throw 80013, 'Branch ID does not exist.', 1;
        end
        -- Check if the manager ID exists
        if not exists (select 1 from instructors where inst_id = @manager_id)
        begin
            throw 80014, 'Manager ID does not exist.', 1;
        end
        -- Update the track
        update tracks
        set track_name = @track_name,
            branch_id = @branch_id,
            manager_id = @manager_id
        where track_id = @track_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

12) update_student_info

The update_student_info procedure allows for the modification of student information in the database, specifically the ZIP code, based on the provided parameters: st_id and zipcode.

It first checks if the student exists in the StudentsInfo table by verifying if a record with the specified st_id exists. If not, it throws an error indicating that the provided student ID does not exist.

Subsequently, it checks if the provided ZIP code exists in the Student_address table. If not, it throws an error indicating that the provided ZIP code does not exist.

After ensuring the existence of both the student and the ZIP code, the procedure updates the StudentsInfo table, setting the ZIP code for the specified student to the provided value.

Error handling is implemented to catch and print any exceptions that may occur during the update process, providing feedback on encountered errors.

```
-----------------------------------------------------------------
--12)Students
create procedure update_student_info @st_id int, @zipcode int
as
begin
    begin try
        -- Check if the student exists
        if not exists (select 1 from StudentsInfo where St_ID = @st_id)
        begin
            throw 80015, 'Student ID does not exist.', 1;
        end
        -- Check if the ZIP code exists
        if not exists (select 1 from Student_address where ZipCode = @zipcode)
        begin
            throw 80016, 'ZIP code does not exist.', 1;
        end
        -- Update the student information
        update StudentsInfo
        set ZipCode = @zipcode
        where St_ID = @st_id;
    end try
    begin catch
        print error_message();
    end catch
end;
```

13) update_student_phone

The update_student_phone procedure is designed to update the phone number of a student in the database based on the provided parameters: st_id (student ID) and new_phone (new phone number).

It begins by checking if the provided student ID exists in the StudentsInfo table. If not, it throws an error indicating that the provided student ID does not exist.

Upon confirming the existence of the student, the procedure proceeds to update the phone number in the Student_Phone table associated with the specified student ID to the provided new phone number.

Error handling is implemented to catch and print any exceptions that may occur during the update process, providing feedback on encountered errors.

```
--13)Student Phone
create procedure update_student_phone @st_id int, @new_phone varchar(20)
as
begin
     begin try
          -- Check if the student ID exists
          if not exists (select 1 from StudentsInfo where St_ID = @st_id)
          begin
               throw 80017, 'Student ID does not exist.', 1;
          end
          -- Update the student phone number
          update Student_Phone
          set Phone = @new_phone
          where St_ID = @st_id;
     end try
     begin catch
          print error_message();
     end catch
end;
```

14) update_trackcontainscourses

The update_trackcontainscourses procedure facilitates the modification of course associations within a track in the database. It takes in parameters such as track_id (the ID of the track), course_id (the existing course ID), and new_course_id (the ID of the course to be updated to).

Firstly, it verifies the existence of the provided track_id and course_id. If either is not found in their respective tables (Tracks or Courses), it throws a custom error indicating that the ID does not exist.

Subsequently, if the provided IDs are valid, the procedure updates the course_id associated with the specified track_id to the new_course_id in the Track_Contains_Courses table.

Error handling is implemented to catch any potential exceptions during the execution of the update operation, printing out error messages for diagnostic purposes.

```
--14)/*Track_Contains_Courses*/
create procedure update_trackcontainscourses @track_id int, @course_id int,@new_course_id int
as
begin
    begin try
        -- Check if the track ID exists
        if not exists (select 1 from Tracks where Track_ID = @track_id)
        begin
            throw 80018, 'Track ID does not exist.', 1;
        end
        -- Check if the course ID exists
        if not exists (select 1 from Courses where Course_ID = @course_id)
        begin
            throw 80019, 'Course ID does not exist.', 1;
        end
        -- Update the student phone number
        update Track_Contains_Courses
        set course_id = @new_course_id
        where track_id  = @track_id and course_id= @course_id ;
    end try
    begin catch
        print error_message();
    end catch
end;
```

15) update_graduates_status

The update_graduates_status procedure is designed to modify the certification status of a graduate in the database. It accepts parameters such as @graduate_id (the ID of the graduate) and @graduate_certification (the new certification status).

Initially, the procedure checks if the provided graduate_id exists in the Graduate_Fail_Reasons table, which holds records of graduates who have failed to meet certification requirements. If the ID is found, it proceeds to update the Graduates table, setting the Graduate_Certification field to the new certification status. Additionally, it removes the graduate from the Graduate_Fail_Reasons table.

In case the provided graduate_id does not exist in the Graduate_Fail_Reasons table, it throws a custom error indicating that the ID does not exist in that context.

Error handling is incorporated to catch any potential exceptions during the execution, printing out error messages for debugging purposes.

```
--15)Graduates
create PROCEDURE update_graduates_status @graduate_id INT, @graduate_certification VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        -- Check if the graduate ID exists in Graduate_Fail_Reasons
        IF EXISTS (SELECT 1 FROM Graduate_Fail_Reasons WHERE Graduate_ID = @graduate_id)
        BEGIN
            -- Update the graduates table
            UPDATE Graduates
            SET Graduate_Certification = @graduate_certification
            WHERE Graduate_ID = @graduate_id;

            -- Remove the graduate from the Student_Fail_Reasons table
            DELETE FROM Graduate_Fail_Reasons
            WHERE Graduate_ID = @graduate_id;
        END
        ELSE
        BEGIN
            -- Handle the case where the graduate ID does not exist in Graduate_Fail_Reasons
            THROW 80020, 'Graduate ID does not exist in Graduate_Fail_Reasons.', 1;
        END;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH;
END;
```

16) update_graduates_info

The update_graduates_info procedure is intended to modify the job-related information of a graduate in the database. It accepts parameters such as @graduate_id (the ID of the graduate), @graduate_job_title (the new job title), @graduate_job_type (the type of job), @graduate_hired_date (the date the graduate was hired), and @graduate_salary (the graduate's salary).

Initially, the procedure checks if the provided graduate_id exists in the Graduates table. If the ID is found, it proceeds to update the Graduates table, setting the fields Graduate_Job_Title, Graduate_Job_Type, Graduate_Hired_Date, and Graduate_Salary to the corresponding new values.

In case the provided graduate_id does not exist in the Graduates table, it throws a custom error indicating that the ID does not exist in that context.

Error handling is implemented to catch and print any potential exceptions during execution for debugging purposes.

```
--16)Graduates
create PROCEDURE update_graduates_info @graduate_id INT,  @graduate_job_title VARCHAR(250),@graduate_job_type VARCHAR(100),
                                    @graduate_hired_date DATE, @graduate_salary MONEY
AS
BEGIN
    BEGIN TRY
        -- Check if the graduate ID exists
        IF EXISTS (SELECT 1 FROM Graduates WHERE Graduate_ID = @graduate_id)
        BEGIN
            -- Update the graduates table
            UPDATE Graduates
            SET
                Graduate_Job_Title = @graduate_job_title,
                Graduate_Job_Type = @graduate_job_type,
                Graduate_Hired_Date = @graduate_hired_date,
                Graduate_Salary = @graduate_salary
            WHERE Graduate_ID = @graduate_id;
        END
        ELSE
        BEGIN
            -- Handle the case where the graduate ID does not exist
            THROW 80021, 'Graduate ID does not exist.', 1;
        END;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH
END;
```

17) update_Graduate_fail_reasons

The update_Graduate_fail_reasons procedure facilitates the modification of fail reasons associated with a graduate in the database. It takes @graduate_id as the ID of the graduate whose fail reasons need updating, and @fail_reasons as the new fail reasons to be assigned. Upon execution, the procedure first checks if the provided graduate_id exists in the Graduates table. If the ID is found, it proceeds to update the FailReasons field in the Graduate_Fail_Reasons table for the specified graduate, setting it to the new value provided in @fail_reasons.

If the provided graduate_id does not exist in the Graduates table, it throws a custom error indicating that the ID does not exist.

For error handling, the procedure catches any potential exceptions during execution and prints the corresponding error message for debugging purposes.

```
--17)Graduate_Fail_Reasons
CREATE PROCEDURE update_Graduate_fail_reasons @graduate_id INT, @fail_reasons VARCHAR(250)
AS
BEGIN
    BEGIN TRY
        -- Check if the graduate ID exists
        IF EXISTS (SELECT 1 FROM Graduates WHERE Graduate_ID = @graduate_id)
        BEGIN
            -- Update the fail reasons for the specified graduate
            UPDATE Graduate_Fail_Reasons
            SET FailReasons = @fail_reasons
            WHERE Graduate_ID = @graduate_id;
        END
        ELSE
        BEGIN
            -- Handle the case where the graduate ID does not exist
            THROW 80023, 'Graduate ID does not exist.', 1;
        END;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH;
END;
```

18) update_graduatesworkcompany

The update_graduatesworkcompany procedure facilitates the update of the company code associated with a graduate's work record in the database. It takes @graduate_id as the ID of the graduate whose work company information needs updating, and @company_code as the new company code to be assigned.

Upon execution, the procedure first verifies if the provided graduate_id exists in the Graduates table. If the ID is found, it proceeds to update the Company_code field in the Graduatesworkcompany table for the specified graduate, setting it to the new value provided in @company_code.

If the provided graduate_id does not exist in the Graduates table, it throws a custom error indicating that the ID does not exist.

For error handling, the procedure catches any potential exceptions during execution and prints the corresponding error message for debugging purposes.

```
--18)/*Graduates_work_at_company*/
create PROCEDURE update_graduatesworkcompany  @graduate_id INT,  @company_code VARCHAR(250)
AS
BEGIN
    BEGIN TRY
        -- Check if the graduate ID exists
        IF EXISTS (SELECT 1 FROM Graduates WHERE Graduate_ID = @graduate_id)
        BEGIN
            -- Update the company code for the specified graduate
            UPDATE Graduatesworkcompany
            SET Company_code = @company_code
            WHERE Graduate_ID = @graduate_id;
        END
        ELSE
        BEGIN
            -- Handle the case where the graduate ID does not exist
            THROW 80024, 'Graduate ID does not exist.', 1;
        END;
    END TRY
    BEGIN CATCH
        PRINT ERROR_MESSAGE();
    END CATCH;
END;
```

8.2.Report's Stored Procedure

8.2.1: SP returns the students information according to Branch No.

This stored procedure takes an input parameter @branchid of type integer. Its purpose is to retrieve student information based on the provided Branch_ID parameter.
It checks if the provided @branchid exists in the Branches table using the exists function. If the branch exists, it proceeds to retrieve student information; otherwise, it returns a message indicating that the branch does not exist. f the branch exists, it selects all columns (*) from the StudentsInfo table, joining it with the Branches table based on the Branch_ID column. It then filters the results based on the provided @branchid. In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   --Report that returns student info accoring to Branch_ID parameter
2   create proc studsinfobybranchid @branchid int
3   as
4   begin
5     begin try
6       if exists (select * from Branches where Branch_ID=@branchid)
7       begin
8         select s.*
9         from StudentsInfo s,Branches
10        where s.Branch_ID=Branches.Branch_ID
11        and s.Branch_ID=@branchid
12      end
13      else
14      begin
15        select 'Branch doesnot exist' as 'Branch_ID'
16      end
17  end try
18  begin catch
19    select ERROR_MESSAGE() as error
20  end catch
21  end
```

8.2.2: SP takes the student ID and returns the grades of the student in all courses.

This stored procedure takes a student ID (@StudentID) as input and returns the grades of the student in all courses.

It begins with a begin try block to handle potential errors during execution.

It checks if the provided @StudentID exists in the StudentsInfo table using the exists function. If the student exists, it proceeds to retrieve the grades; otherwise, it returns a message indicating that the student does not exist.If the student exists, it selects the course name and the average grade (Course_Grade) for each course taken by the student. It calculates the average grade by summing up the exam percentages for each course taken by the student and then dividing by 3 (assuming there are 3 exams per course).

It joins multiple tables (Student_Takes_Exam, StudentsInfo, Tracks, Track_Contains_Courses, and Courses) to get the necessary information about the student's grades and courses.It filters the data based on the provided @StudentID.It groups the result by the course name to calculate the average grade for each course.In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   --Report that takes the student ID and returns the grades of the student in all courses. %
2
3   alter PROCEDURE GetStudentGrade @StudentID int
4   AS
5   BEGIN
6   BEGIN TRY
7     IF EXISTS (SELECT * FROM StudentsInfo WHERE St_ID = @StudentID)
8     BEGIN
9
10    SELECT
11    (SUM(STE.Exam_Percentage))/3 Course_Grade , C.Course_Name
12    FROM Student_Takes_Exam STE inner join StudentsInfo S on S.St_ID = STE.St_ID
13    inner join Tracks T on T.Track_ID = S.Track_ID
14    Inner Join Track_Contains_Courses TC on TC.Track_ID = T.Track_ID
15    inner join Courses c on C.Course_ID = TC.Course_ID
16
17    WHERE S.St_ID = @StudentID
18    GROUP BY C.Course_Name
19
20  END;
21  else
22    begin
23      select 'Student does not exist' as 'St_ID'
24    end
25  end try
26  begin catch
27    select ERROR_MESSAGE() as error
28  end catch
29  end
30
31  GetStudentGrade 100
32
```

8.2.3: SP takes the instructor ID and returns the name of the courses that he teaches and the number of student per course.

This stored procedure takes an instructor ID (@instructorId) as input and returns the courses taught by the instructor along with the count of students enrolled in each course.

It begins with a begin try block to handle potential errors during execution. It checks if the provided @instructorId exists in the Instructors table using the exists function. If the instructor exists, it proceeds to retrieve the courses; otherwise, it returns a message indicating that the instructor does not exist.

If the instructor exists, it selects the course name, along with the count of students (StudentCount) enrolled in each course taught by the instructor.

It joins multiple tables (Courses, Instructors, Track_Contains_Courses, Tracks, and StudentsInfo) to get the necessary information about the courses taught by the instructor and the students enrolled in those courses. It filters the data based on the provided @instructorId.

It groups the result by the course name and the instructor ID to calculate the count of students enrolled in each course taught by the instructor.In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   Create PROCEDURE GetInstructorCourses @instructorId int
2   AS
3   BEGIN
4   BEGIN TRY
5     IF EXISTS (SELECT * FROM Instructors WHERE Inst_ID = @instructorId)
6     BEGIN
7
8     SELECT
9     C.Course_Name, COUNT(S.St_ID) AS StudentCount, I.Inst_ID
10    FROM  Courses C INNER JOIN Instructors I on I.Inst_ID = C.Inst_ID
11    INNER JOIN Track_Contains_Courses CT on CT.Course_ID = C.Course_ID
12    INNER JOIN StudentsInfo S ON S.Track_ID = CT.Track_ID
13    WHERE I.Inst_ID = @instructorId
14    GROUP BY C.Course_Name, I.Inst_ID;
15
16  END;
17  else
18    begin
19      select 'Instructor does not exist' as 'Inst_ID'
20    end
21  end try
22  begin catch
23    select ERROR_MESSAGE() as error
24  end catch
25  end
26
27  GetInstructorCourses 21
```

8.2.4: SP takes course ID and returns its topics.

This stored procedure takes a course ID (@courseid) as input and returns the topics related to that course.

It begins with a begin try block to handle potential errors during execution.It checks if the provided @courseid exists in the Courses table using the exists function. If the course exists, it proceeds to retrieve its topics; otherwise, it returns a message indicating that the course does not exist.If the course exists, it selects all columns (*) from the Topics table where the Course_ID matches the provided @courseid.

It joins the Topics table with the Courses table based on the Course_ID column.

It filters the data based on the provided @courseid.In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   --Report that takes course id and returns its topics
2
3   create proc topicsbycourseid @courseid int
4   as
5   begin
6     begin try
7       if exists (select * from Courses where Course_ID=@courseid)
8       begin
9         select t.*
10        from Topics t,Courses
11        where t.Course_ID=Courses.Course_ID
12        and t.Course_ID=@courseid
13      end
14      else
15      begin
16        select 'course doesnot exist' as 'Course_ID'
17      end
18  end try
19  begin catch
20    select ERROR_MESSAGE() as error
21  end catch
22  end
```

8.2.5: SP that takes exam number and returns the Questions in it and chocies.

This stored procedure takes an exam ID (@examId) as input and returns the questions associated with that exam along with their choices.

It begins with a begin try block to handle potential errors during execution.It checks if the provided @examId exists in the Exams table using the exists function. If the exam exists, it proceeds to retrieve its questions; otherwise, it returns a message indicating that the exam does not exist.

If the exam exists, it selects the question ID, question text, and question choices from the relevant tables (Questions, Exam_Quest_Generation, Exams, and Question_Choices).It joins multiple tables based on their relationships to retrieve the necessary information about the questions associated with the exam.It filters the data based on the provided @examId.It orders the result set by the Question_ID.In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   Create PROCEDURE GetExamQS @examId int
2   AS
3   BEGIN
4   BEGIN TRY
5     IF EXISTS (SELECT * FROM Exams WHERE Exam_ID = @examId)
6     BEGIN
7
8     SELECT
9     Q.Question_Text, QC.Choice
10    FROM Exams E inner join Exam_Quest_Generation SE on E.Exam_ID = SE.Exam_ID
11    inner join Questions Q on SE.Question_ID = Q.Question_ID
12    Inner Join Question_Choices QC on Q.Question_ID = QC.Question_ID
13
14    WHERE E.Exam_ID = @examId
15    GROUP BY Q.Question_Text, QC.Choice
16
17  END;
18  else
19    begin
20      select 'Exam does not exist' as 'Exam_ID'
21    end
22  end try
23  begin catch
24    select ERROR_MESSAGE() as error
25  end catch
26  end
27
28  GetExamQS 205
29
```

8.2.6: SP takes exam number and the student ID then returns the Questions in this exam with the student answers.

This stored procedure retrieves the answers given by a specific student (@studentId) for a particular exam (@examId).

It begins with a begin try block to handle potential errors during execution.It checks if the provided @studentId exists in the Student_Answers_Exam table, ensuring the student has participated in exams.If the student has participated in exams, it selects the question text and the student's answer from the relevant tables (Exams, Student_Answers_Exam, Questions, and StudentsInfo).

It joins multiple tables based on their relationships to retrieve the necessary information about the student's answers for the specified exam. It filters the data based on the provided @examId and @studentId. It groups the result by question text and student answer.In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
Create PROCEDURE GetStudentAns @examId int , @studentId int
AS
BEGIN
BEGIN TRY

  IF EXISTS (SELECT * FROM Student_Answers_Exam WHERE St_ID = @studentId)
  BEGIN

  SELECT
  Q.Question_Text, St_Answer
  FROM Exams E inner join Student_Answers_Exam SE on E.Exam_ID = SE.Exam_ID
  inner join Questions Q on SE.Question_ID = Q.Question_ID
  inner join StudentsInfo St on St.St_ID = SE.St_ID

  WHERE E.Exam_ID = @examId AND St.St_ID = @studentId
  GROUP BY Q.Question_Text, St_Answer

END;
else
  begin
    select 'Student did not take exam' as 'St_ID'
  end
end try
begin catch
  select ERROR_MESSAGE() as error
end catch
end

GetStudentAns 3, 5
```

8.3: Exam Generation Related Stored Procedures:

8.3.1: SP for Exam generation

This stored procedure named Exam_Generation is designed to generate an exam based on various parameters such as exam level, type, duration, full marks. The procedure accepts several input parameters including @Exam_ID, @Course_ID, @Exam_Level, @Exam_Type, @Exam_Duration, @Exam_FullMark, @Exam_Name, @Num_TF_Questions, and @Num_MCQ_Questions.

It begins with a begin try block to handle potential errors during execution. It checks if the provided @Course_ID exists in the Courses table. If the course doesn't exist, it returns an error message indicating that the course does not exist.

If the course exists, it inserts the details of the generated exam into the Exams table including the exam ID, level, type, date, duration, full mark, name, and course ID. It inserts True or False (T or F) questions and Multiple Choice Questions (MCQs) into the Exam_Quest_Generation table based on the exam level and the specified number of questions for each type. It selects the questions for the generated exam from the Questions table based on the inserted question IDs in the Exam_Quest_Generation table. The questions are selected based on the exam ID provided as input.

In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
alter procedure Exam_Generation
  @Exam_ID int,
  @Course_ID int,
  @Exam_Level varchar(50),
  @Exam_Type varchar(50),
  @Exam_Duration varchar(100),
  @Exam_FullMark int,
  @Exam_Name varchar(100),
  @Num_TF_Questions int,
  @Num_MCQ_Questions int
as
begin
  declare @Quest_Difficulty varchar(20)
  if @Exam_Level='Basic'
    set @Quest_Difficulty='Basic'
  if @Exam_Level='Intermediate'
    set @Quest_Difficulty='Intermediate'
  if @Exam_Level='Advanced'
    set @Quest_Difficulty='Advanced'

  begin try
    if not exists (select * from Courses where Course_ID=@Course_ID)
      select 'Course doesnot exist' as 'error'
    else
      ----insert the generated exam in exam table---
      insert into Exams(Exam_ID,Exam_Level,Exam_Type,Exam_Date,Exam_Duration,Exam_FullMark,Exam_Name,Course_ID)
      values(@Exam_ID,@Exam_Level,@Exam_Type,GETDATE(),@Exam_Duration,@Exam_FullMark,@Exam_Name,@Course_ID)
```

```
29          --insert T or F questions according to exam_level---
30          insert into Exam_Quest_Generation(Exam_ID,Question_ID)
31          select top (@Num_TF_Questions) @Exam_ID,Question_ID
32          from Questions
33          where Questions.Course_ID=@Course_ID and Questions.Question_Type='T or F'
34          and Questions.Question_Level=@Quest_Difficulty
35          order by NEWID()
36
37          --insert MCQ questions according to exam_level---
38          insert into Exam_Quest_Generation(Exam_ID,Question_ID)
39          select top (@Num_MCQ_Questions) @Exam_ID,Question_ID
40          from Questions
41          where Questions.Course_ID=@Course_ID and Questions.Question_Type='MCQ'
42          and Questions.Question_Level=@Quest_Difficulty
43          order by NEWID()
44
45          -- Select exam model
46                SELECT Q.*
47                FROM Exam_Quest_Generation eq, Questions Q, Exams E
48          WHERE eq.Exam_ID = E.Exam_ID AND eq.Question_ID = Q.Question_ID AND eq.Exam_ID = @Exam_ID
49
50        end try
51        begin catch
52          select ERROR_MESSAGE() as ErrorMessage
53        end catch
54    end
55
56
57    delete from Exam_Quest_Generation where Exam_ID between 202 and 208
58    delete from Exams where Exam_ID between 202 and 208
59
60    Exam_Generation 202,200,'Intermediate','Online','30 minutes',50,'Data Mining and Warehouse',5,5
```

8.3.2: SP for Storing Exam Answers

This stored procedure, named exam_answer, is designed to insert student answers for a specific exam question into the Student_Answers_Exam table.
The procedure accepts four input parameters: @st_ID (student ID), @exam_ID (exam ID), @q_ID (question ID), and @answer (student's answer).
It retrieves the name of the student corresponding to the provided @st_ID from the StudentsInfo table and stores it in the variable @std_name.
It begins with a begin try block to handle potential errors during execution.It checks if the required parameters (@exam_ID, @st_ID, and @q_ID) are not null. If any of these parameters are null, it raises an error with a specified severity level and error message.If all required parameters are provided, it inserts the student's answer into the Student_Answers_Exam table along with the student ID, exam ID, and question ID.
In case of any errors during execution, the catch block is triggered. It retrieves the error message, severity, and state, and then raises an error with the same information.

```
create procedure [dbo].[exam_answer]
    @st_ID  int,
  @exam_ID INT,
  @q_ID int,
  @answer varchar(300)
as
begin
declare @std_name varchar(50)
set @std_name =
      (select StudentsInfo.St_Fname+' '+StudentsInfo.St_Lname from StudentsInfo
      where StudentsInfo.St_ID = @st_ID)
    begin try
        IF (@exam_ID IS NULL OR @st_ID IS NULL OR @q_ID IS NULL )
            RAISERROR('exam_ID, this data are required', 16, 1)
        ELSE
            -- Insert the answers into the answers table
            INSERT INTO Student_Answers_Exam(St_ID,Exam_ID,Question_ID,St_Answer)
            VALUES (@st_ID,@exam_ID,@q_ID,@answer)

    END TRY
    BEGIN CATCH
        DECLARE @ErrorMessage NVARCHAR(4000);
        DECLARE @ErrorSeverity INT;
        DECLARE @ErrorState INT;

        SELECT @ErrorMessage = ERROR_MESSAGE(),
               @ErrorSeverity = ERROR_SEVERITY(),
               @ErrorState = ERROR_STATE();

        RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH;
END;


exam_answer 1,201,8,'To ensure data integrity and consistency by grouping related operations together'
```

8.3.3: SP for Exam Correction

This altered stored procedure, Exam_Correction, is designed to correct and grade exams for a specific student (@St_ID) and exam (@Exam_ID).
It begins with a begin try block to handle potential errors during execution.It checks if the provided exam ID exists in the Exams table and if the provided student ID exists in the StudentsInfo table. If either the exam or the student does not exist, it returns an error message accordingly.

- It updates the Grade column in the Student_Answers_Exam table based on whether the student's answer matches the correct answer stored in the Questions table. Each correct answer receives a grade of 5, while incorrect answers receive a grade of 0.
- It calculates the total correct score for the student's answers.
- It calculates the exam percentage for the student based on the total correct score and the full mark of the exam.
- It determines the student's status (Passed or Failed) based on the exam percentage.
- It inserts a record into the Student_Takes_Exam table with the student's ID, exam ID, exam percentage, and status.

In case of any errors during execution, the catch block is triggered, and it returns the error message using the ERROR_MESSAGE() function.

```
1   alter procedure Exam_Correction @Exam_ID int, @St_ID int
2   as
3
4   begin try
5     if not exists (select * from Exams where Exam_ID=@Exam_ID)
6       select 'exam doesnot exist' as 'error'
7     else if not exists (select * from StudentsInfo where St_ID=@St_ID)
8       select 'student doesnot exist' as 'error'
9     else
10      begin
11
12
13        update Student_Answers_Exam
14        set Grade=case when Student_Answers_Exam.St_Answer=Questions.CorrectAnswer
15                    then 5
16                  else 0
17                  end
18        from Student_Answers_Exam,Questions
19        where Student_Answers_Exam.Question_ID=Questions.Question_ID
20        and Student_Answers_Exam.Exam_ID=@Exam_ID
21        and Student_Answers_Exam.St_ID=@St_ID
22  |
23        declare @total_correct int
24        select @total_correct=SUM(Grade)
25        from Student_Answers_Exam
26        where Student_Answers_Exam.Exam_ID=@Exam_ID
27        and Student_Answers_Exam.St_ID=@St_ID
28
```

```
29          declare @exam_percentage decimal(5,1)
30          select @exam_percentage=(@total_correct*100.0)/Exam_FullMark
31          from Student_Answers_Exam,Exams
32          where student_answers_exam.Exam_ID=Exams.Exam_ID
33          and Student_Answers_Exam.Exam_ID=@Exam_ID and St_ID=@St_ID
34
35          declare @St_Status varchar(20)
36            if @exam_percentage<50
37            set @St_Status='Failed'
38            if @exam_percentage>=50
39            set @St_Status='Passed'
40
41          insert into Student_Takes_Exam (St_ID,Exam_ID,Exam_Percentage,St_Status)
42          values (@St_ID,@Exam_ID,@exam_percentage,@St_Status)
43        end
44    end try
45    begin catch
46      select ERROR_MESSAGE() as errorMessage
47    end catch
48
49    Exam_Correction 201,1
50
```

9. Data Warehouse

In a data warehousing environment, data modelling is crucial for structuring the database schema to efficiently store and analyzing data. The dimension tables serve as the backbone of the data warehouse, providing context and descriptive attributes for the measures stored in the fact tables. Here's a breakdown of the modelled dimensions in the provided schema:

1. Dim_Students:
- Describes student-related attributes such as name, email, gender, birthdate, faculty, address, and contact information.
- Includes references to related entities like branches, tracks, intakes, rounds, certificates, and faculty.

2. Dim_StudentGrades:
- Stores information related to student grades in exams, including exam ID, percentage, and pass/fail status.

3. Dim_Branches:
- Contains details about branches, including name, location, founding date, and manager name.

4. Dim_Instructors:
- Holds information about instructors, including name, salary, gender, birthdate, hiring date, position, degree, experience, email, and contact details.
- References branch information where instructors are associated.

5. Dim_Graduates:
- Encompasses graduate details such as name, gender, certification status, job title, job type, hire date, salary, phone, graduation year, track, and fail reasons.
- Includes information about the companies graduates work for, such as code, name, and location.

6. Dim_Courses:
- Provides data on courses, including name, status, duration, evaluation method, instructor, and related topic.
- References instructors who teach the courses.

7. Dim_Tracks:
- Describes tracks offered, including name, duration, associated course, branch, manager, and related graduates.

8. Dim_Exams:
- Stores exam-related information such as level, type, name, date, duration, full mark, and associated course.

9. Dim_Questions:
- Contains details about questions in exams, including text, correct answer, level, type, and available choices.
- References the corresponding exams.

10. Fact_Students:

This fact table captures transactional data related to student activities within the educational institution.

It records key metrics and measures associated with student performance, progress, and interactions.

The table includes foreign keys referencing dimension tables to establish relationships and provide context for the recorded transactions.

Transactional details such as the student ID, track ID, branch ID, exam ID, and date of the transaction are included to provide context and enable drill-down analysis.

Additional attributes such as creation timestamp (create_at) and source system code are included for audit and data lineage purposes.

1. Start Date:

- **Definition:** The start date field indicates the date when a particular record or transaction becomes effective or valid within the system.
- **Purpose:** It serves as a reference point for establishing the validity period of a record and helps track the timeline of when the data was first introduced or became relevant.

2. End Date:

- **Definition:** The end date field denotes the date when a record or transaction ceases to be valid or effective within the system.
- **Purpose:** It marks the termination point or expiration date of a record, indicating when it is no longer applicable or relevant.

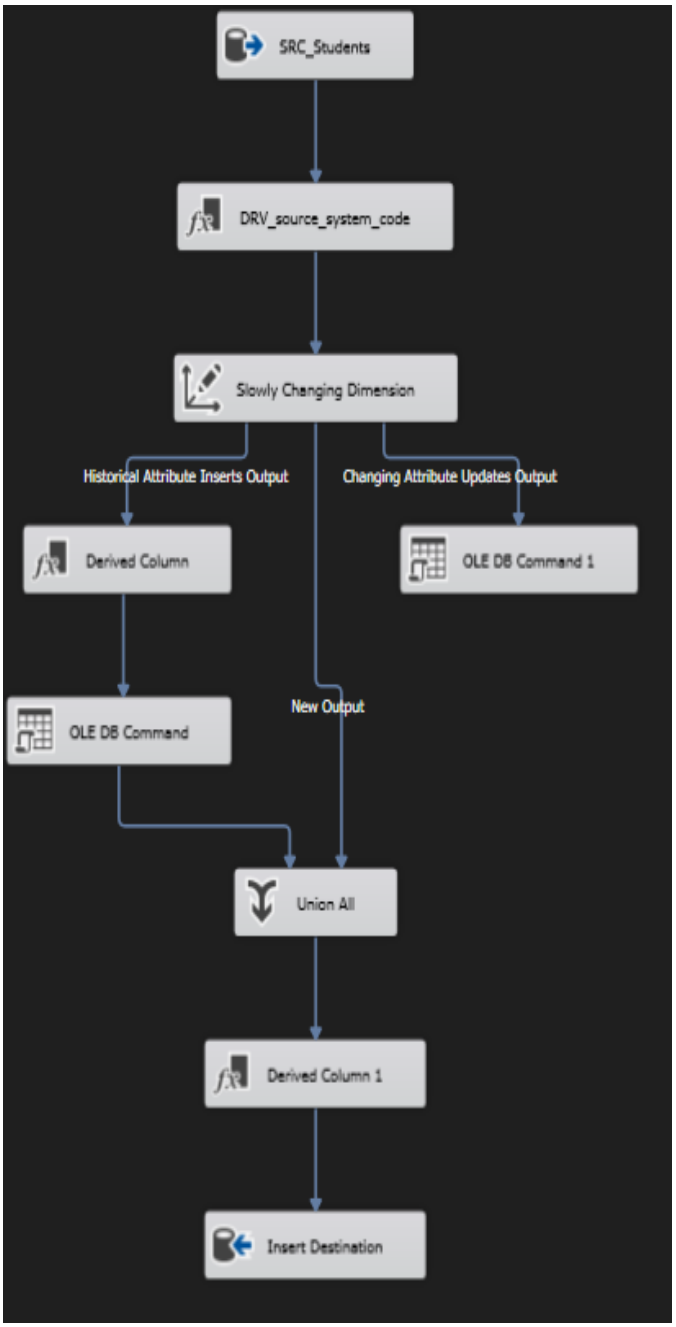
3. Is Current:

- **Definition:** The is current field is a binary indicator (often represented as a Boolean or tinyint value) that flags whether a record is currently active or valid.
- **Purpose:** It provides a simple and efficient way to identify the latest or most recent version of a record without needing to compare start and end dates.

9.2. Slowly Changing Dimensions (SCD) and Column Tracking

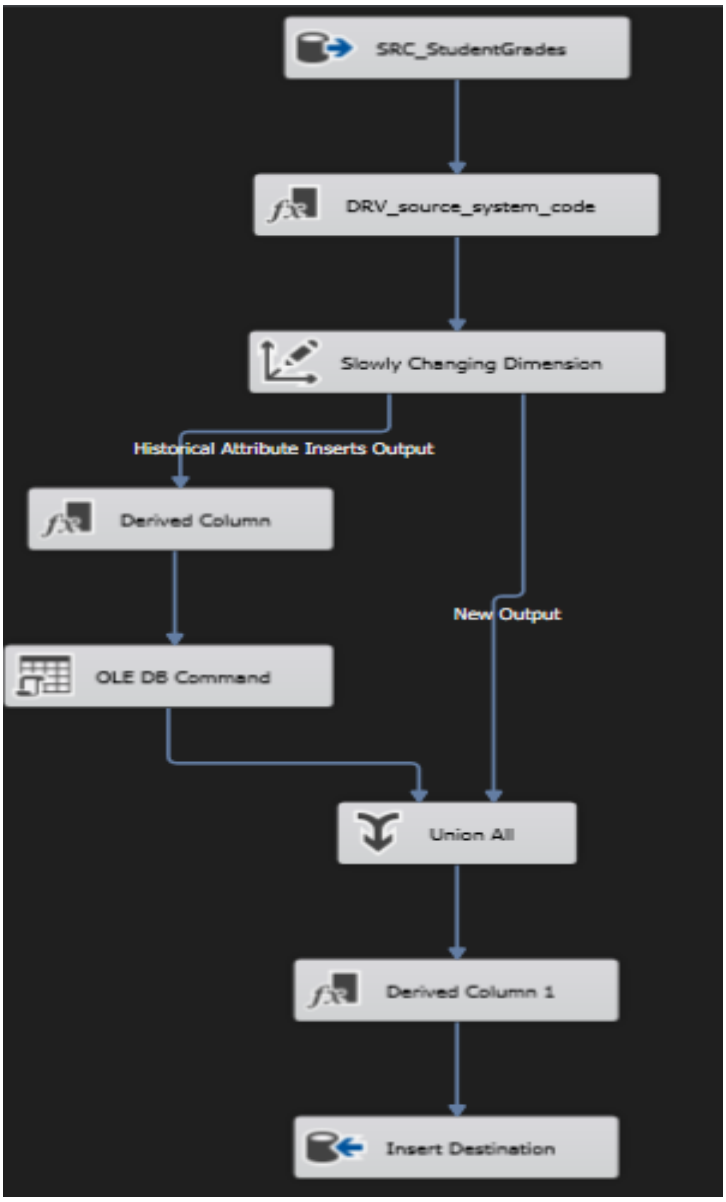
Slowly Changing Dimensions (SCD) refer to techniques used in data warehousing to manage changes to dimension attributes over time. Let's explore how SCD types are applied to your data warehouse schema and identify which columns are changing, historical, and fixed:

1. Dim_Students:



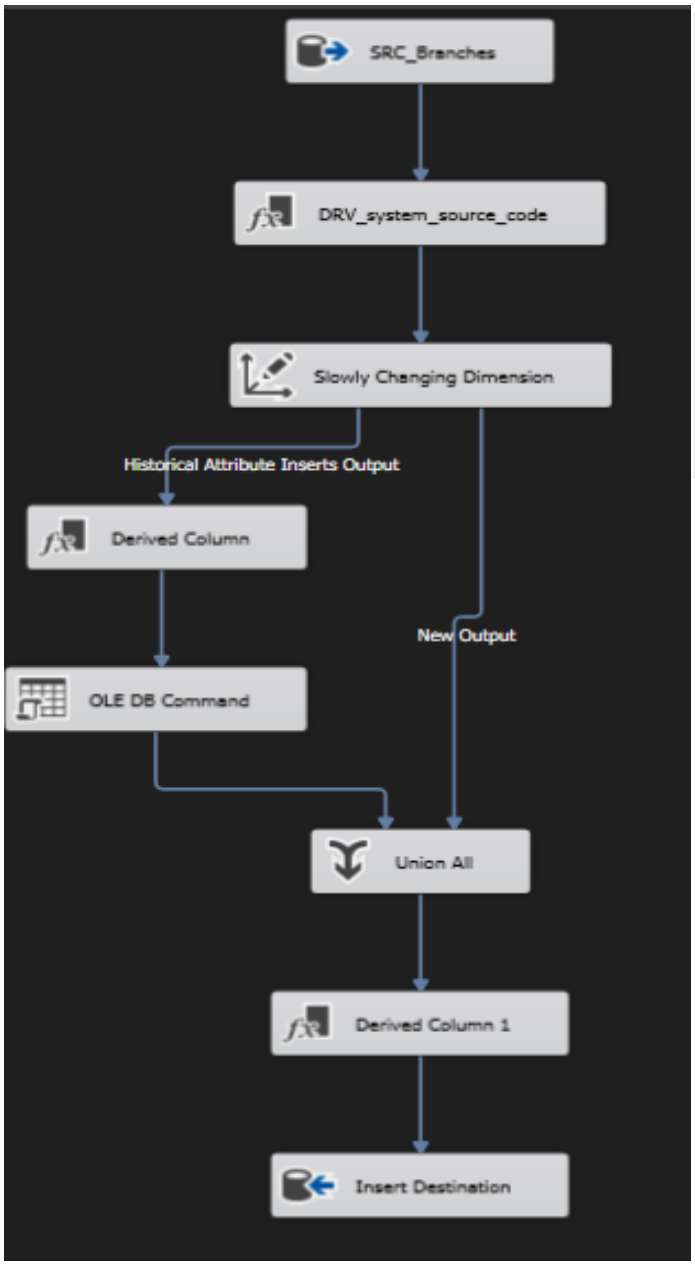
Dimension Columns	Change Type
certificate_issue_date	Changing attribute
certificate_Provider_Name	Fixed attribute
City	Historical attribute
FacebookURL	Historical attribute
Fcode_BK	Fixed attribute
FLocation	Fixed attribute
Fname	Fixed attribute
Intake_ID_BK	Fixed attribute
Intake_NO	Fixed attribute
Phone_BK	Historical attribute
Round_ID_BK	Fixed attribute
Round_NO	Fixed attribute
St_Birthdate	Fixed attribute
St_Email	Historical attribute
St_Faculty_GraduationYear	Fixed attribute
St_Fname	Fixed attribute
St_Gender	Fixed attribute
St_Lname	Fixed attribute
St_NumFreelanceJobs	Historical attribute
St_Password	Historical attribute
Street	Historical attribute
Track_ID	Fixed attribute
ZipCode_BK	Changing attribute

2. Dim_StudentGrades:



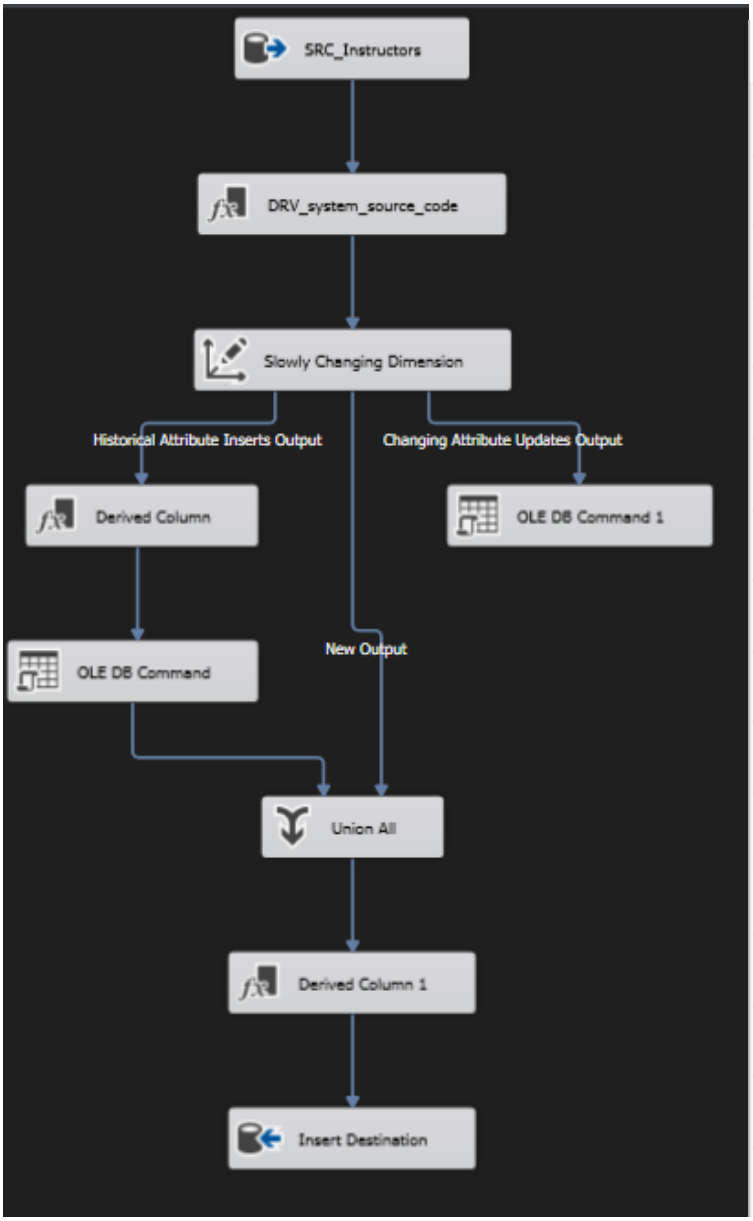
Dimension Columns	Change Type
Exam_ID	Fixed attribute
Exam_Percentage	Historical attribut
source_system_code	Fixed attribute
St_Status	Historical attribut

3. Dim_Branches:



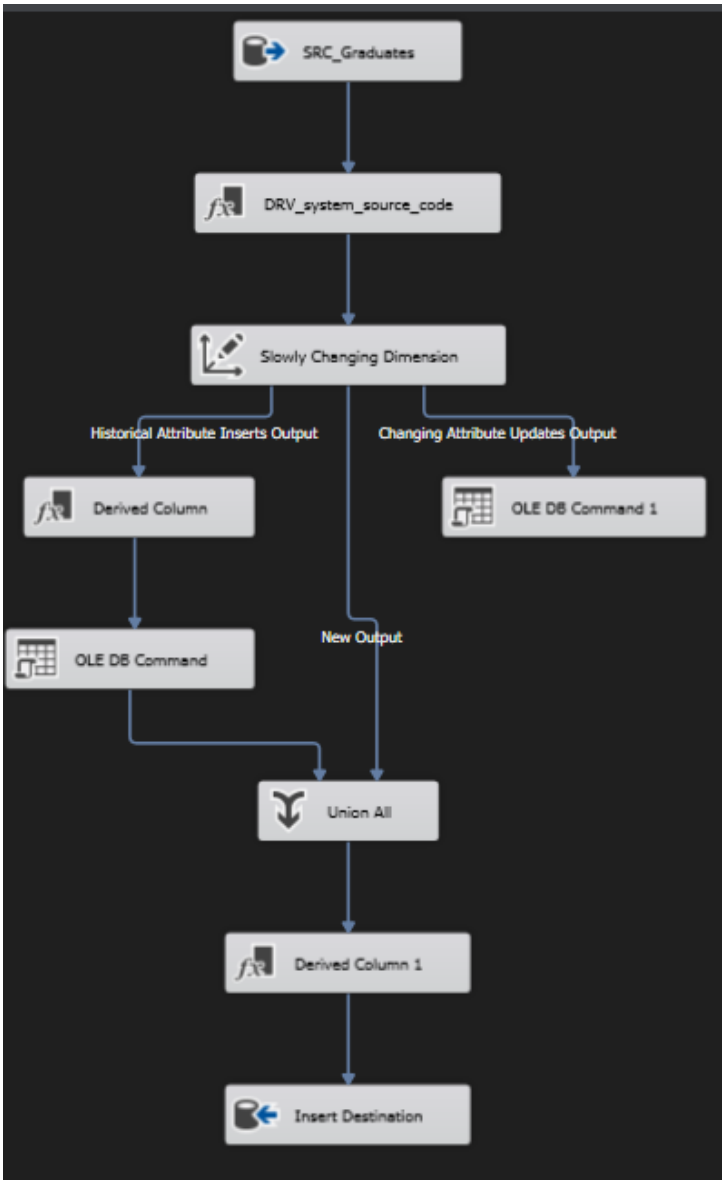
Dimension Columns	Change Type
B_Location	Historical attribute
B_Name	Historical attribute
Founding_date	Fixed attribute
MngrName	Historical attribute

4. Dim_Instructors:



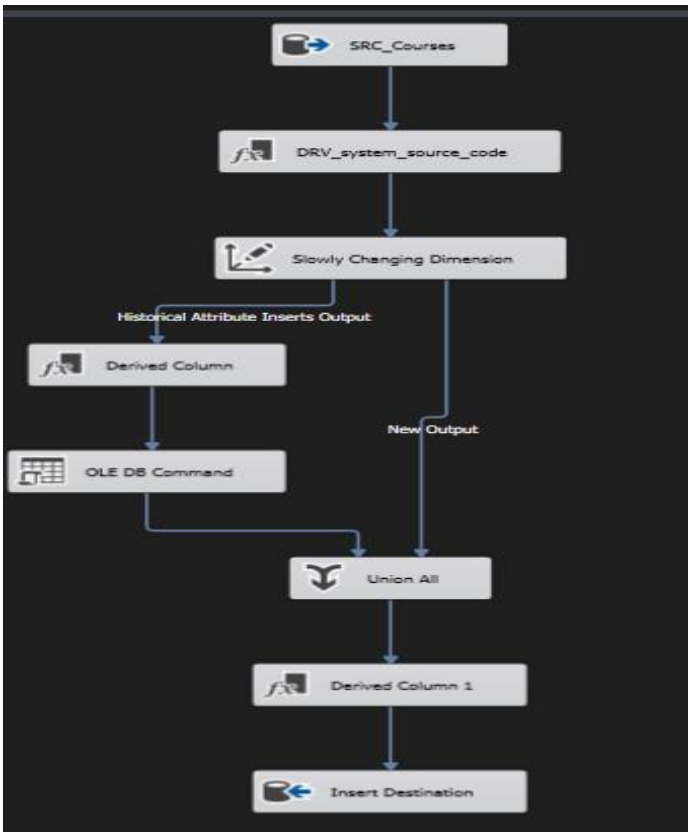
Dimension Columns	Change Type
Branch_ID_FK	Historical attribute
City	Historical attribute
Inst_Birthdate	Fixed attribute
Inst_Degree	Historical attribute
Inst_Email	Historical attribute
Inst_Experience_Year	Changing attribute
Inst_Fname	Fixed attribute
Inst_Gender	Fixed attribute
Inst_HiringDate	Fixed attribute
Inst_Lname	Fixed attribute
Inst_Position	Historical attribute
Inst_Salary	Historical attribute
Manager_ID	Historical attribute
Phone_BK	Historical attribute
source_system_code	Fixed attribute
Street	Historical attribute

5. Dim_Graduates:



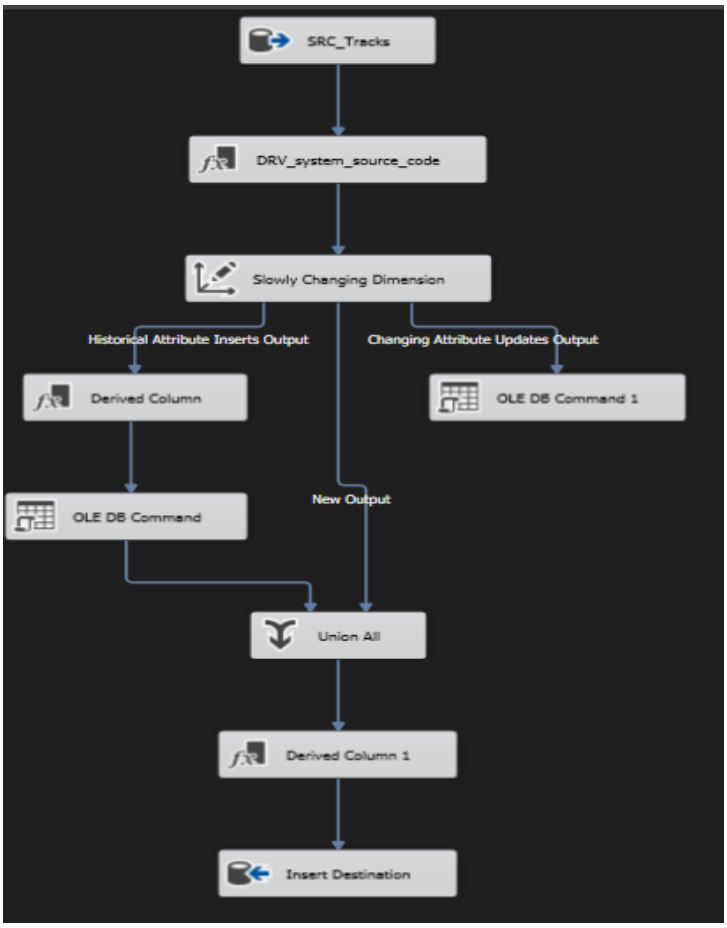
Dimension Columns	Change Type
Company_Location	Historical attribute
Company_name	Historical attribute
FailReasons	Historical attribute
Graduate_Certifica...	Fixed attribute
Graduate_Fname	Fixed attribute
Graduate_Gender	Fixed attribute
Graduate_Hired_D...	Changing attribute
Graduate_Job_Title	Historical attribute
Graduate_Job_Type	Historical attribute
Graduate_Lname	Fixed attribute
Graduate_Phone_BK	Historical attribute
Graduate_Salary	Historical attribute
ITI_Graduation_Year	Fixed attribute
source_system_code	Fixed attribute
Track_ID	Fixed attribute
Company_code_BK	Fixed attribute

6. Dim_Courses:



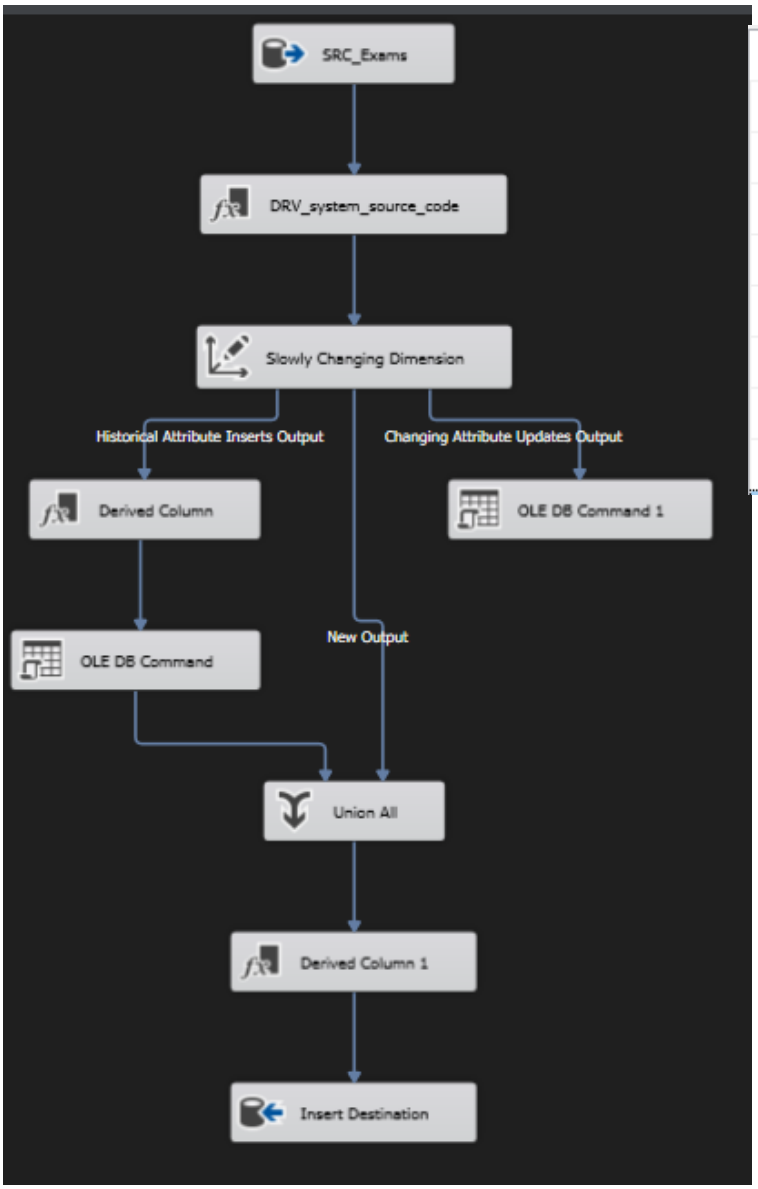
Dimension Columns	Change Type
Course_Duration	Historical attribute
Course_Evaluation	Historical attribute
Course_Name	Fixed attribute
Course_Status	Historical attribute
Inst_ID_FK	Historical attribute
source_system_code	Fixed attribute
Topic_Name	Historical attribute

7. Dim_Tracks:



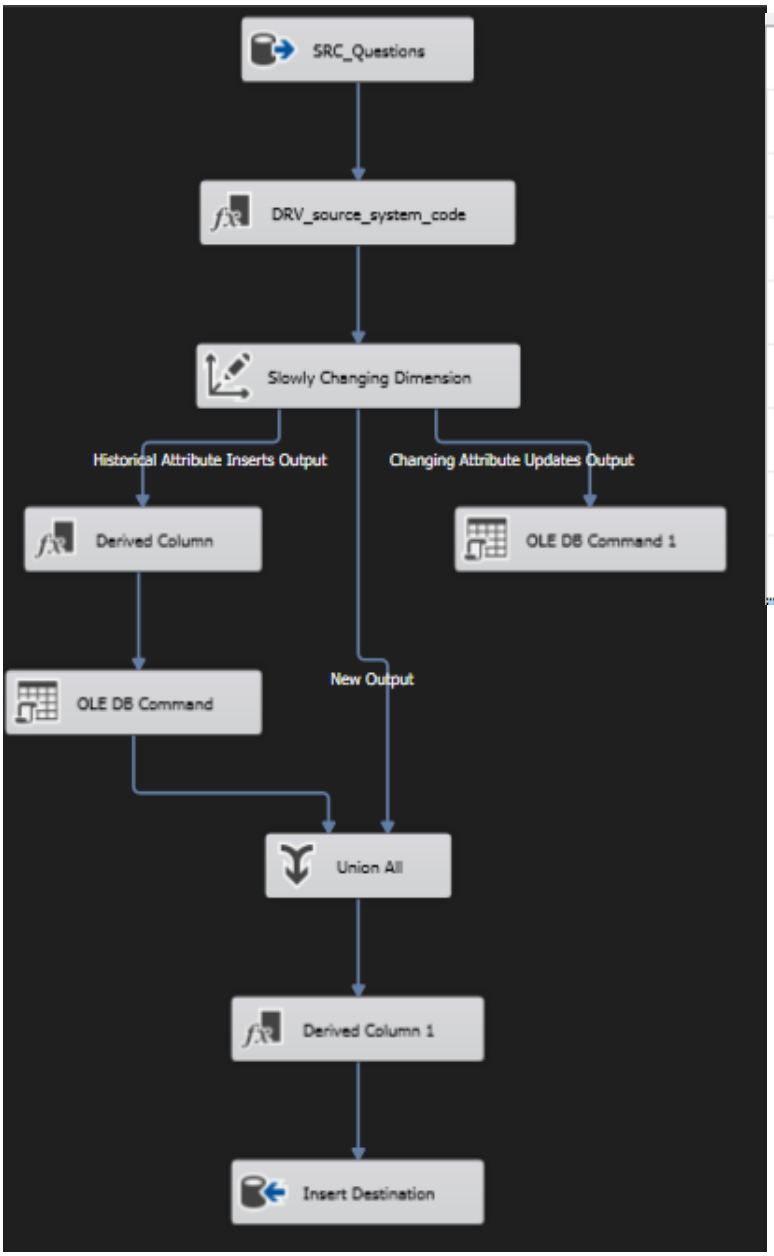
Dimension Columns	Change Type
Branch_ID	Changing attribute
Course_ID_FK	Historical attribute
Graduate_ID_FK	Historical attribute
Manager_ID	Historical attribute
source_system_code	Fixed attribute
Track_Duration	Fixed attribute
Track_Name	Historical attribute

8. Dim_Exams:



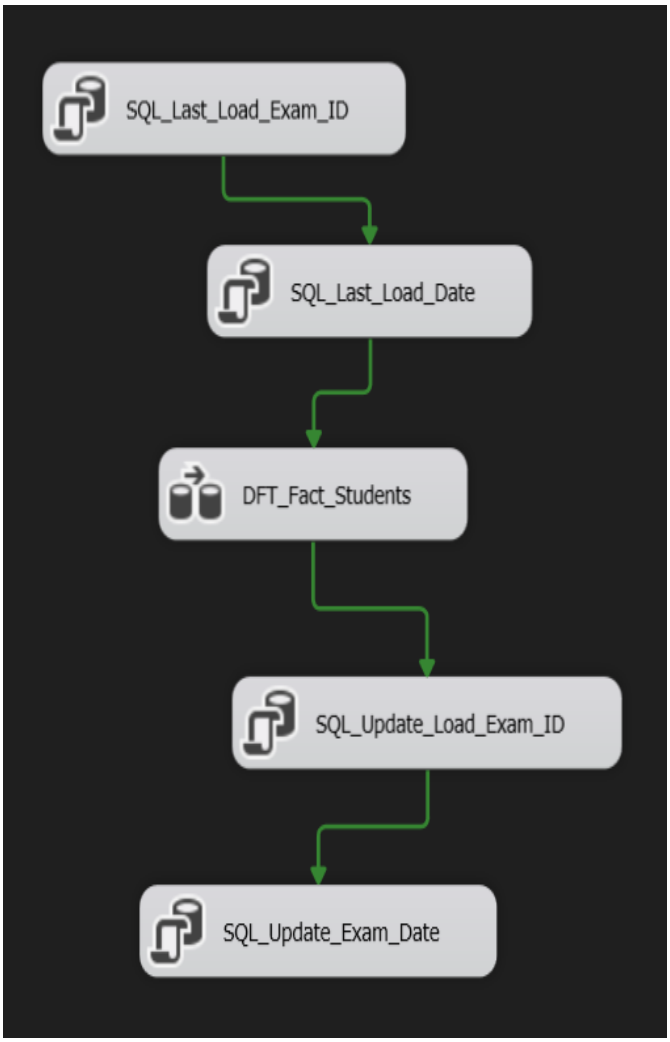
Dimension Columns	Change Type
Course_ID	Changing attribute
Exam_Date	Changing attribute
Exam_Duration	Historical attribute
Exam_FullMark	Historical attribute
Exam_Level	Historical attribute
Exam_Name	Fixed attribute
Exam_Type	Historical attribute
source_system_code	Fixed attribute

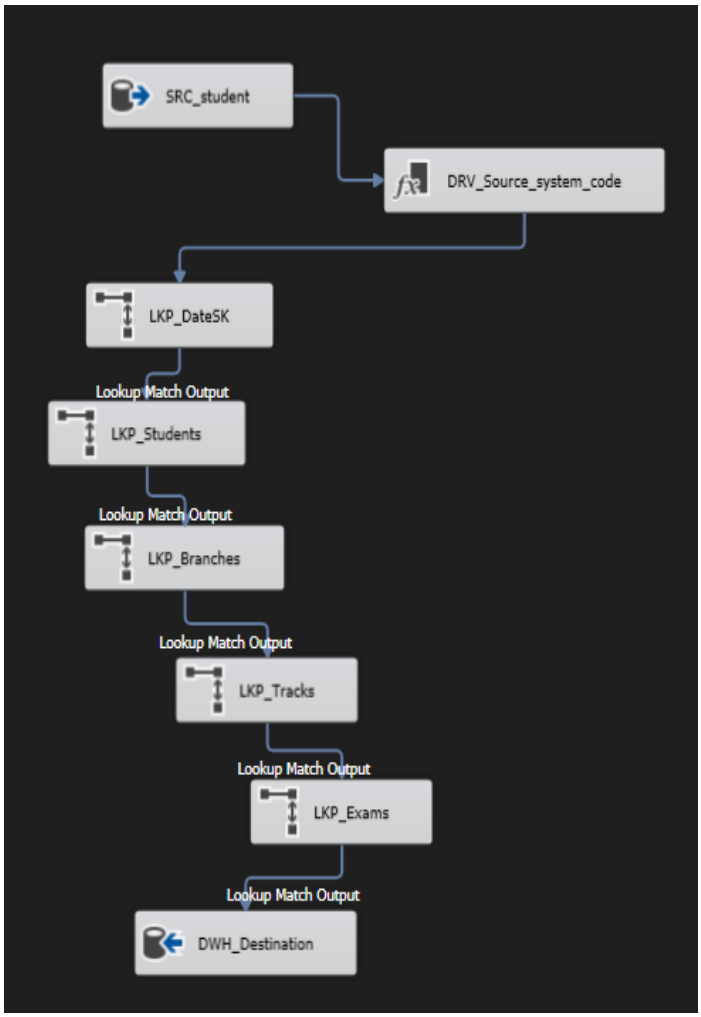
9. Dim_Questions:



Dimension Columns	Change Type
Choice	Historical attribute
CorrectAnswer	Historical attribute
Course_ID	Changing attribute
Exam_ID_FK	Fixed attribute
Question_Level	Historical attribute
Question_Text	Historical attribute
Question_Type	Historical attribute
source_system_code	Fixed attribute

10. Fact_Students:

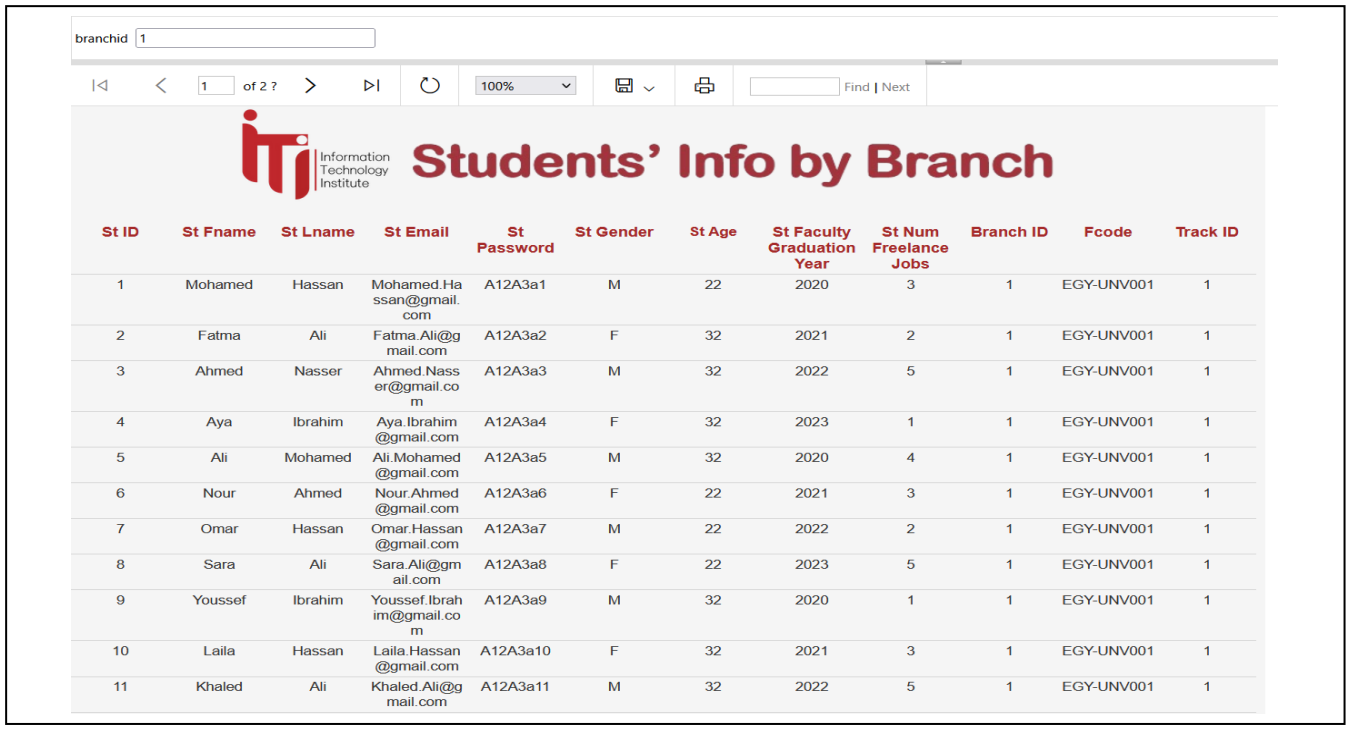




10. SSRS Reports

10.1 Report that returns the students information according to Branch_ID Paramter.

This Report uses the studsinfobybranchid stored procedure.


branchid 1

1 of 2 ? 100% Find | Next

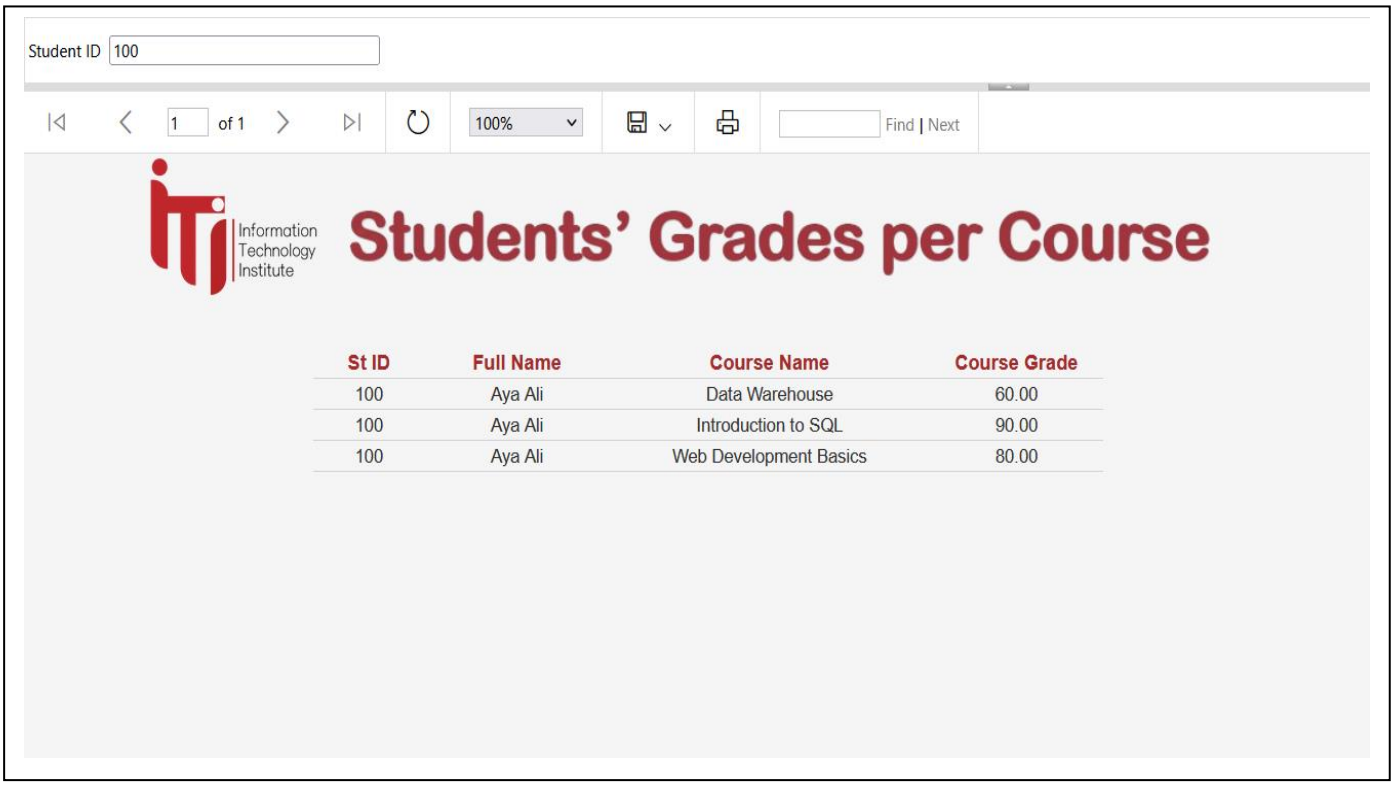
Information Technology Institute

Students' Info by Branch

St ID	St Fname	St Lname	St Email	St Password	St Gender	St Age	St Faculty Graduation Year	St Num Freelance Jobs	Branch ID	Fcode	Track ID
1	Mohamed	Hassan	Mohamed.Hassan@gmail.com	A12A3a1	M	22	2020	3	1	EGY-UNV001	1
2	Fatma	Ali	Fatma.Ali@gmail.com	A12A3a2	F	32	2021	2	1	EGY-UNV001	1
3	Ahmed	Nasser	Ahmed.Nasser@gmail.com	A12A3a3	M	32	2022	5	1	EGY-UNV001	1
4	Aya	Ibrahim	Aya.Ibrahim@gmail.com	A12A3a4	F	32	2023	1	1	EGY-UNV001	1
5	Ali	Mohamed	Ali.Mohamed@gmail.com	A12A3a5	M	32	2020	4	1	EGY-UNV001	1
6	Nour	Ahmed	Nour.Ahmed@gmail.com	A12A3a6	F	22	2021	3	1	EGY-UNV001	1
7	Omar	Hassan	Omar.Hassan@gmail.com	A12A3a7	M	22	2022	2	1	EGY-UNV001	1
8	Sara	Ali	Sara.Ali@gmail.com	A12A3a8	F	22	2023	5	1	EGY-UNV001	1
9	Youssef	Ibrahim	Youssef.Ibrahim@gmail.com	A12A3a9	M	32	2020	1	1	EGY-UNV001	1
10	Laila	Hassan	Laila.Hassan@gmail.com	A12A3a10	F	32	2021	3	1	EGY-UNV001	1
11	Khaled	Ali	Khaled.Ali@gmail.com	A12A3a11	M	32	2022	5	1	EGY-UNV001	1

10.2 Report that takes the student ID and returns the grades of the student in all courses.

This Report uses GetStudentGrade Stored Procedure.


Student ID 100

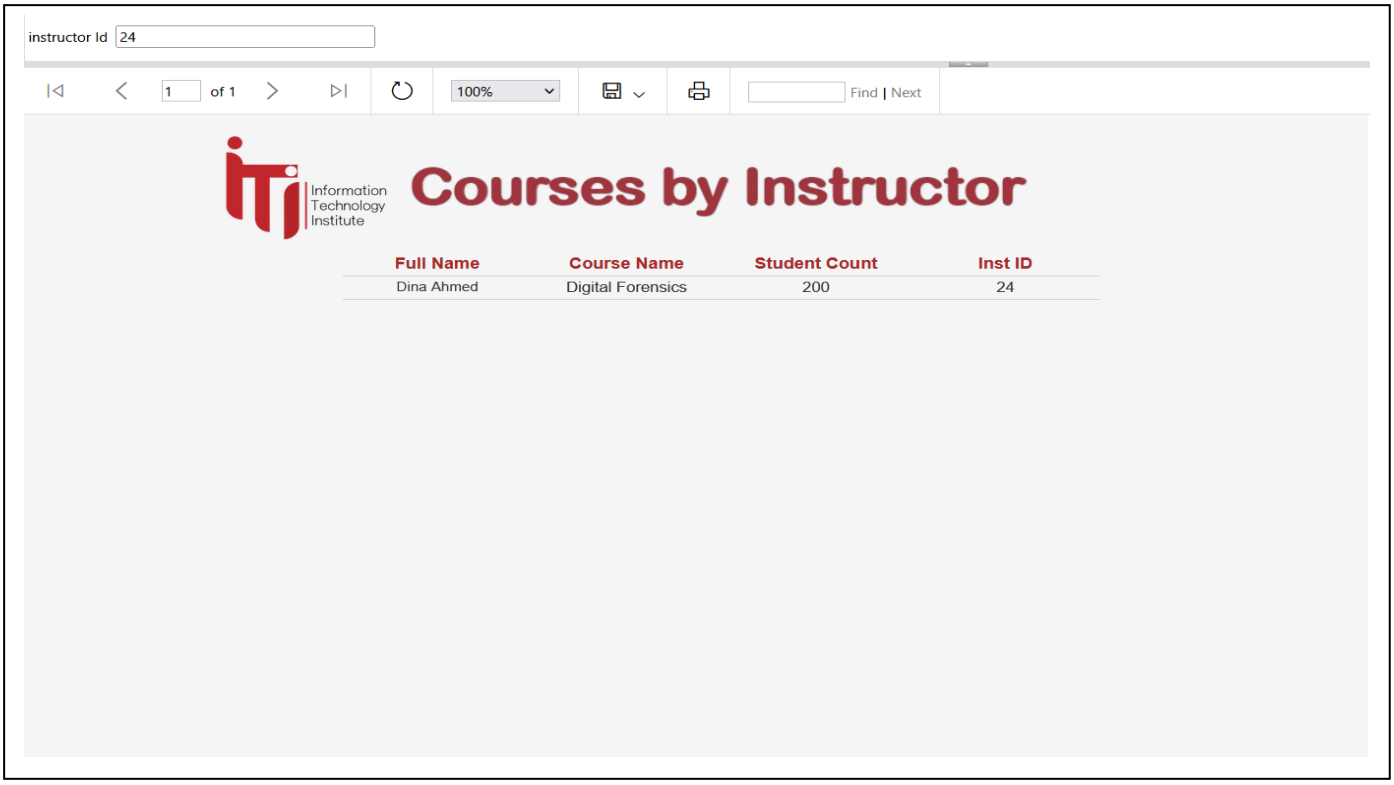
1 of 1 100% Find | Next

Information Technology Institute

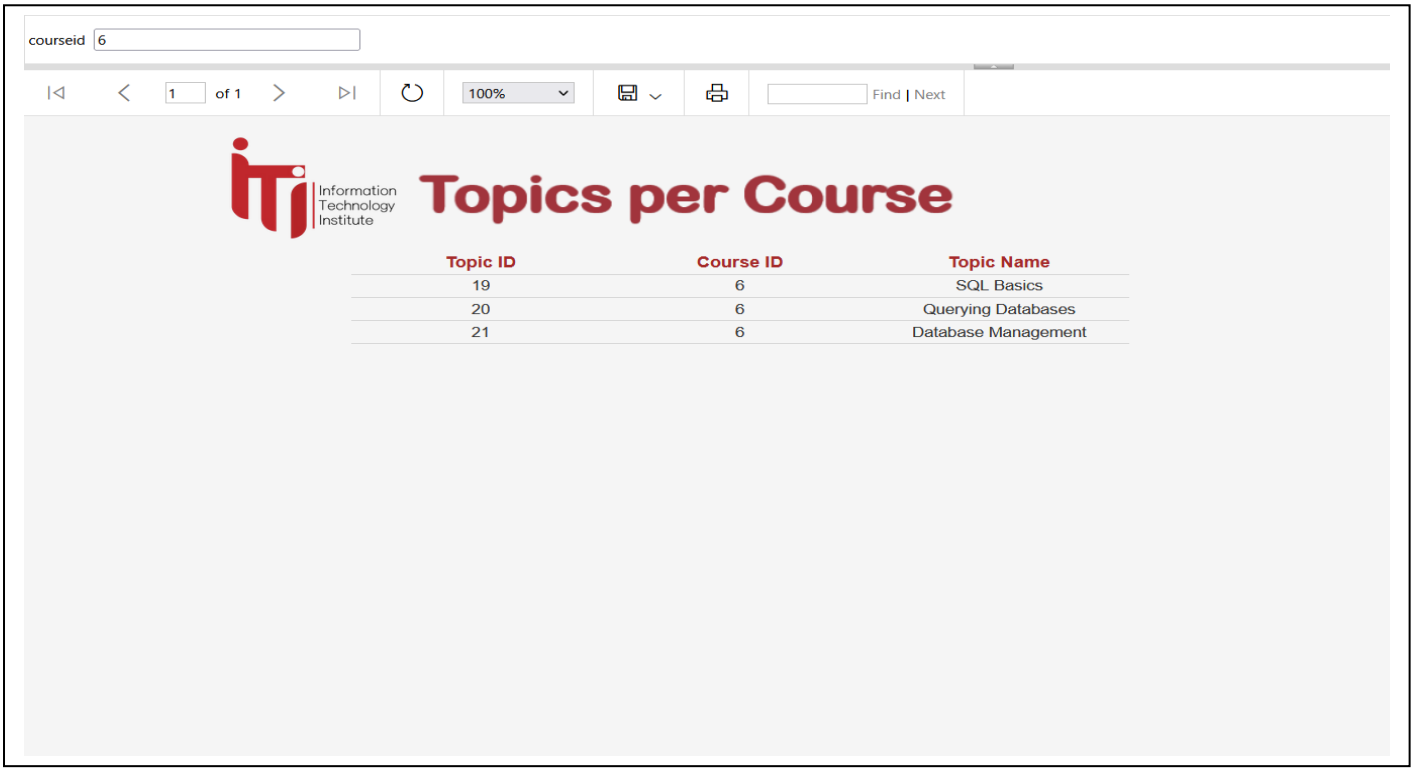
Students' Grades per Course

St ID	Full Name	Course Name	Course Grade
100	Aya Ali	Data Warehouse	60.00
100	Aya Ali	Introduction to SQL	90.00
100	Aya Ali	Web Development Basics	80.00

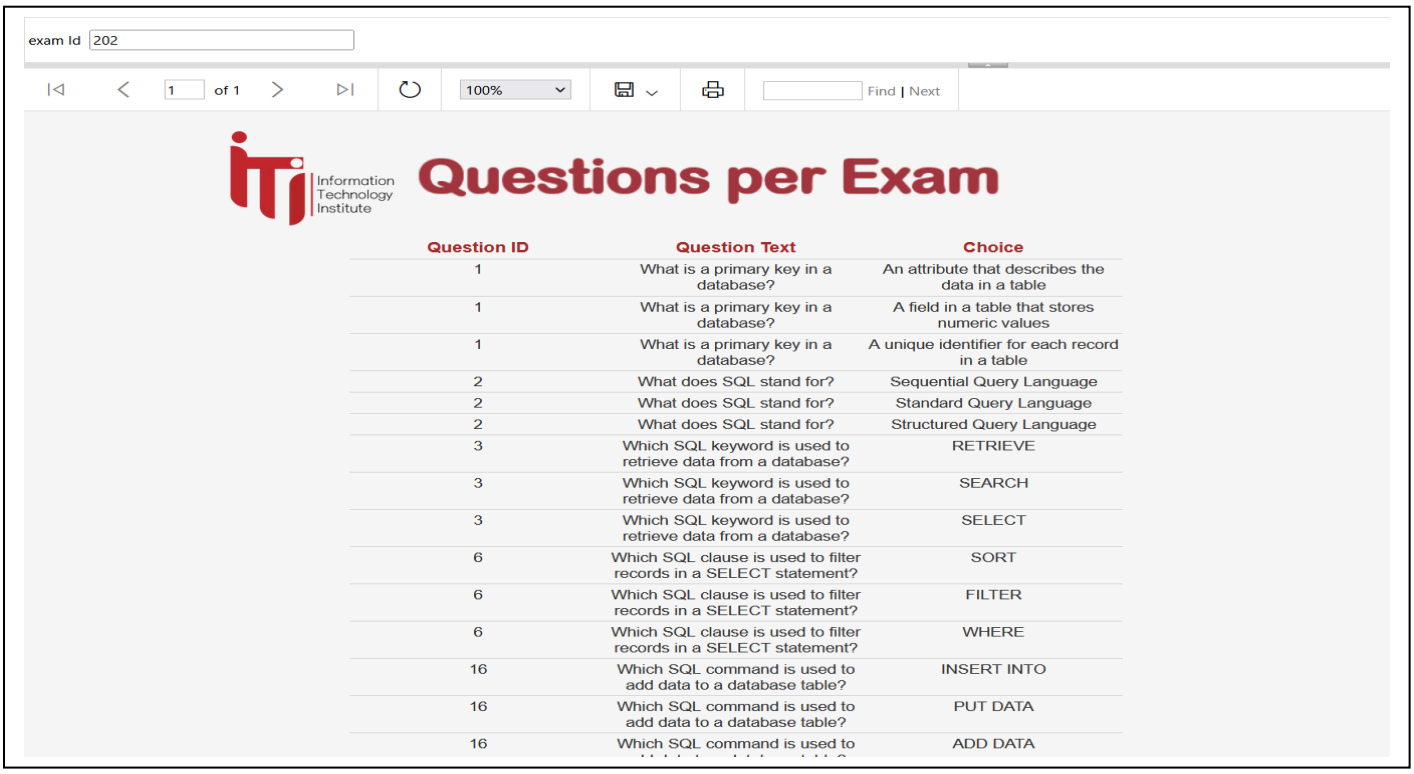
10.3 Report that takes the Instructor ID and returns the course he/she teaches and the number of students studying that course.
This Report uses GetInstructorCourses Stored Procedure.



10.4 Report that takes the Course ID and returns the topics related to this course.
This Report uses topicsbycourseid Stored Procedure.



10.5 Report that takes the Exam ID and returns the Questions in this exam with their choices.
This Report uses GetExamQS Stored Procedure.


exam Id 202

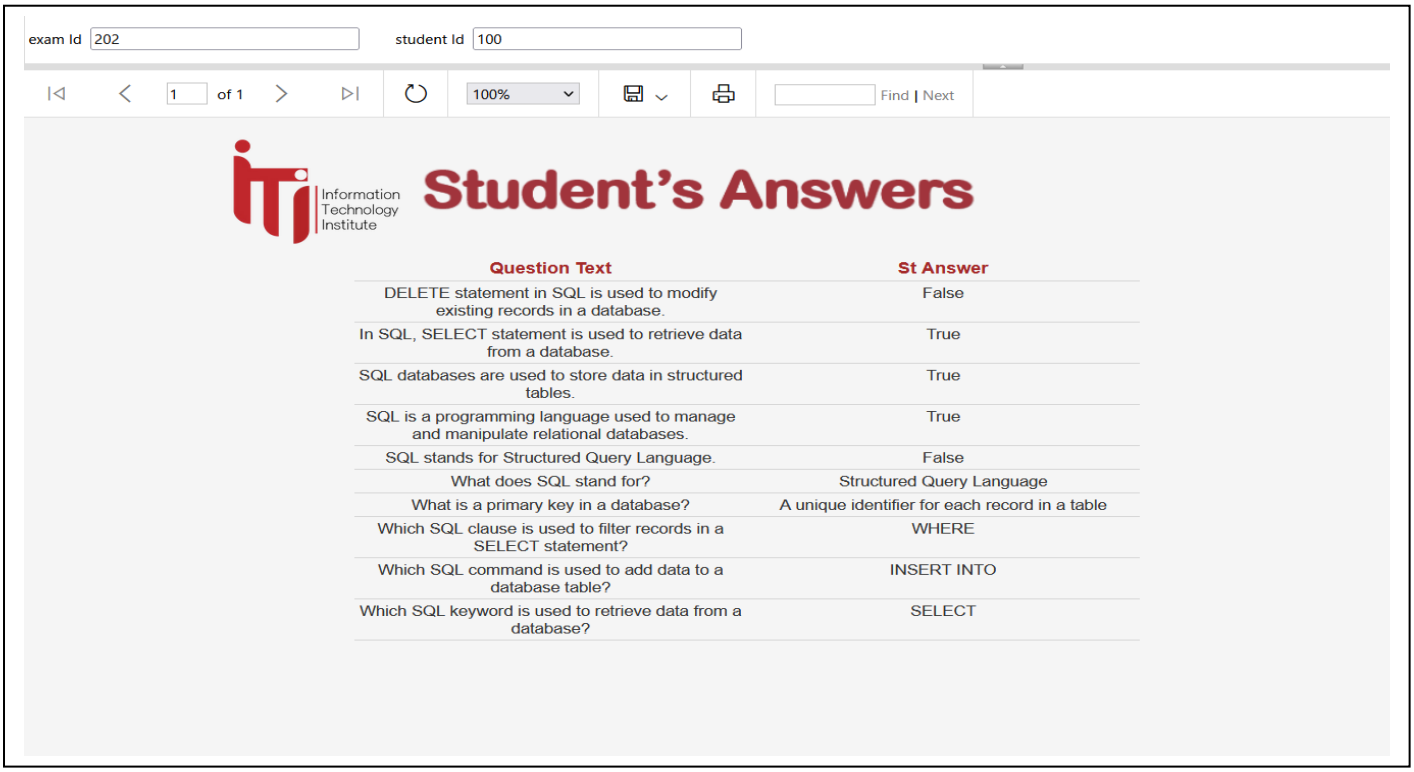
1 of 1 100% Find | Next

Information Technology Institute

Questions per Exam

Question ID	Question Text	Choice
1	What is a primary key in a database?	An attribute that describes the data in a table
1	What is a primary key in a database?	A field in a table that stores numeric values
1	What is a primary key in a database?	A unique identifier for each record in a table
2	What does SQL stand for?	Sequential Query Language
2	What does SQL stand for?	Standard Query Language
2	What does SQL stand for?	Structured Query Language
3	Which SQL keyword is used to retrieve data from a database?	RETRIEVE
3	Which SQL keyword is used to retrieve data from a database?	SEARCH
3	Which SQL keyword is used to retrieve data from a database?	SELECT
6	Which SQL clause is used to filter records in a SELECT statement?	SORT
6	Which SQL clause is used to filter records in a SELECT statement?	FILTER
6	Which SQL clause is used to filter records in a SELECT statement?	WHERE
16	Which SQL command is used to add data to a database table?	INSERT INTO
16	Which SQL command is used to add data to a database table?	PUT DATA
16	Which SQL command is used to	ADD DATA

10.6 Report that takes the Exam ID and ID of the Student who took the exam and returns the student answers.
This Report uses GetStudentAns Stored Procedure.


exam Id 202 student Id 100

1 of 1 100% Find | Next

Information Technology Institute

Student's Answers

Question Text	St Answer
DELETE statement in SQL is used to modify existing records in a database.	False
In SQL, SELECT statement is used to retrieve data from a database.	True
SQL databases are used to store data in structured tables.	True
SQL is a programming language used to manage and manipulate relational databases.	True
SQL stands for Structured Query Language.	False
What does SQL stand for?	Structured Query Language
What is a primary key in a database?	A unique identifier for each record in a table
Which SQL clause is used to filter records in a SELECT statement?	WHERE
Which SQL command is used to add data to a database table?	INSERT INTO
Which SQL keyword is used to retrieve data from a database?	SELECT

11. Website

ITI Online Exam System with ASP.NET Core 6 MVC

In this project, I created an online exam system where students can log in with their email and password provided by ITI and saved in the database before. After logging in, the exam instructions page will appear with all details related to the exam. Then, the student will be redirected to the exam page with a timer set to 30 minutes and 10 questions. After submitting the exam, the student will be directed to the grade page with the student's full name, grade percentage, and exam status. If the student does not answer any questions and the exam time expires, the exam will be submitted automatically, and the student will be directed to the grade page with a grade of 0.0% and a status of failed.

Frontend Used Tools:

Html, CSS, JavaScript.

Backend Used Tools:

ASP.NET Core 6.
Entity Framework Core version 6.0.2 package.
Entity Framework Core.SqlServer Version 6.0.26 package.
EntityFrameworkCore.Tools Version 6.0.26 package.
EF Core Power Tools.

Development Environment and tools used

ASP.NET Core 6 and SQL Server as the database are used during project development, and a database backup named “Exam_System_OLTP” needs to be restored. Necessary Entity Framework Core version 6.0.26, Entity Framework Core.SqlServer Version 6.0.26 and EntityFrameworkCore.Tools Version 6.0.26 packages are used to communicate with the database. During development, a database-first approach is embraced.
As a result, ASP.NET Core 6 is required to run this project.

Software Architecture (Screens Code):

Home Page

Figure 1: Home Page Screen (Student)

Source Code:

```
using ITIOnlineExaminationSystem.Model;
using Microsoft.AspNetCore.Mvc;
using System.Diagnostics;

namespace ITIOnlineExaminationSystem.Controllers
{
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
        }
    }
}
```

```
@{
    ViewData["Title"] = "Home Page";
    Layout = "_Layout";
}

<div class="intro-container">
    <div class="intro">
        <div class="intro-text">
            <h6 class="welcome">WELCOME TO</h6>
            <h2 class="intro-text-description">
                ITI Online Examination system
            </h2>

            <div class="intro-line"></div>
            <input type="button" value="Get Started" class="btn" id="loginBtn" onclick="openexamstartPage()" />
        </div>
        <div class="main-img-container">
            <img src="~/images/Main.svg" class="main-img" />
        </div>
    </div>

    <div class="main-img-container">
        <img src="~/images/Main.svg.svg" class="main-img" />
    </div>
    <script>
        function openexamstartPage() {
            window.open("Account/LogIn", "_self");
        }
    </script>
</div>
```

This is the Home Page that greets the student once he/she enters the website. The student can navigate to the login page by clicking on the 'Get Started' button.

Login Page

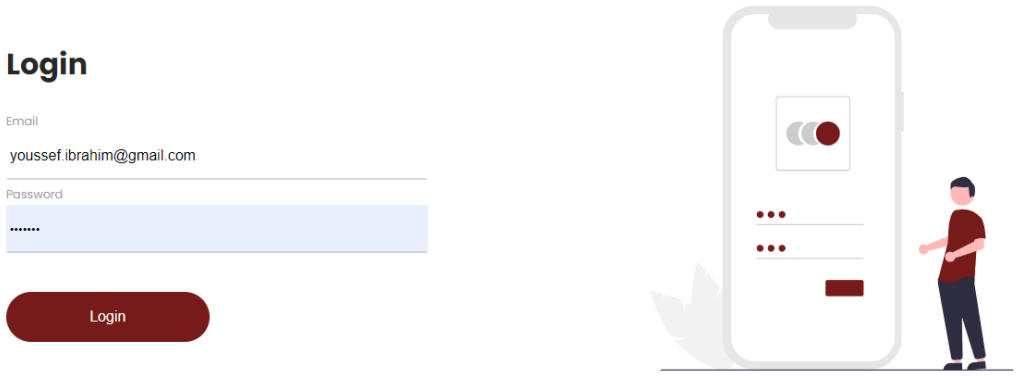


Figure 2: Login Screen (Student)

Source Code:

```
namespace ITIOnlineExaminationSystem.Controllers
{
    public class AccountController : Controller
    {

        private readonly ITIOnlineExamSystemContext _dbContext;

        public AccountController(ITIOnlineExamSystemContext dbContext)
        {
            _dbContext = dbContext;
        }
        [HttpGet]
        public IActionResult LogIn()
        {
            return View();
        }

        [HttpPost]

        public IActionResult LogIn(LoginViewModel model)
        {

            if (ModelState.IsValid)
            {
                if (model.StEmail is null)
                {

                    ModelState.AddModelError(string.Empty, "Invalid email or password");
                    return View(model);

                }
                // Query the database to find a user with the provided email
                var user = _dbContext.StudentsInfo.FirstOrDefault(u => u.St_Email == model.StEmail);
```

```
@model ITIOnlineExaminationSystem.Model.Account.LoginViewModel
@{
    ViewData["Title"] = "Log In Page";
    Layout = "_AuthLayout";
}

<div class="signUp-container">
    <div class="form-container">
        <h1 class="signUp-text">Login</h1>
        <form asp-action="LogIn" asp-controller="Account" method="post" >
            <div class="input-label">
                <label for="StEmail">Email</label>
                <br />
                <input type="email" class="text email" id="StEmail" asp-for="StEmail" name="StEmail" class="form-control" placeholder="Example@gmail.com" required=""/>
                <span asp-validation-for="@Model.StEmail"></span>
            </div>
            <div class="input-label">
                <label for="StPassword">Password</label>
                <br />
                <input type="password" class="text" id=" StPassword" asp-for="StPassword" name="StPassword" class="form-control" placeholder="Password" required=""/>
                <span asp-validation-for="@Model.StPassword"></span>
            </div>
            <div class="btns-container">

             <input type="submit" value="Login" class="btn" id="loginBtn" />

            </div>
        </form>
    </div>
    <div class="signUp-img-container">
        <img src="~/images/SignIn.svg" class="signUp-img" />
```

- The student can easily log in to his account from this page to start the exam if he has an email and password provided by ITI and saved in the database. Upon successful login operation, the student is redirected to the exam instructions page.

Exam Instructions Page

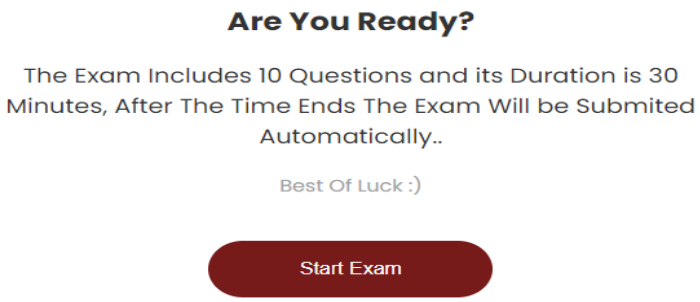


Figure 3: Exam Instructions Screen

Source Code:

```
namespace ITIOnlineExaminationSystem.Controllers
{
        //Exam Controller
        public class ExamController : Controller
        {
                private readonly ITIOnlineExamSystemContext _context;

                public ExamController(ITIOnlineExamSystemContext context)
                {
                        _context = context;
                }
                //Action of Exam instruction that return view of exam instructions
                [HttpGet]
                public IActionResult ExamInstructions()
                {
                        return View();

                }
```

```
@{
    ViewData["Title"] = "Ready Page";
}

<link rel="stylesheet" href="css/loading.css">
<div class="ready-contanier">

    <div class="ready-text-btn-container">
        <div class="ready-text-container">
            <h2 class="ready-tittle">Are You Ready?</h2>
            <p class="ready-pragraph">
                The Exam Includes 10 Questions and its Duration is 30 Minutes, After The
                Time Ends The Exam Will be Submited Automatically..
            </p>
            <span class="best-text">Best Of Luck :)</span>
        </div>
        <input type="submit" value="Start Exam" class="btn" onclick="startExamBtn()"/>
    </div>

    <script>
        function startExamBtn() {
            window.open("/Exam/Exsamination", "_self");
        }
    </script>
    </div>
```

- The Exam instructions screen shows the exam instructions for student who will take the exam.

Exam Page

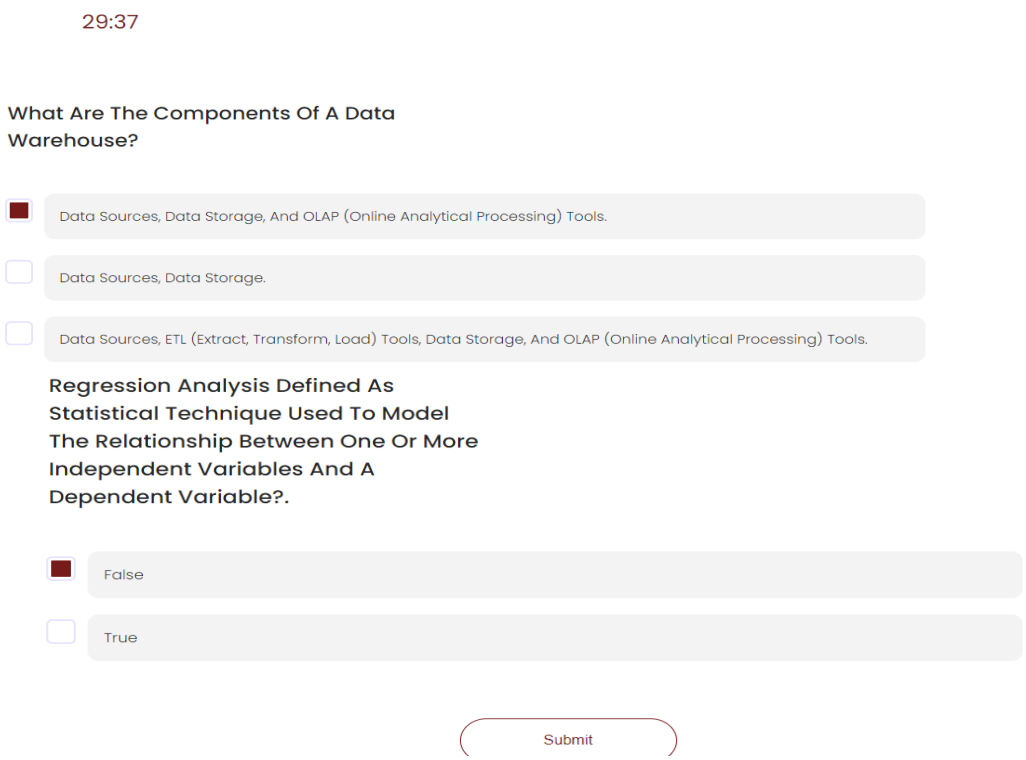


Figure 4: Exam with timer Screen

Source Code:

```
public class ExamController : Controller
        public IActionResult ExamInstructions()
        }
        //Action of Exsamination return view of examination
        [HttpGet]
        public async Task<IActionResult> Exsamination()
        {
                // take needed premeters of sp of exam generation to call it
                SPExamGenerationViewModel sPExamGenerationViewModel = new SPExamGenerationViewModel();
                //get exam id
                int Exam_ID = _context.Exam_Quest_Generation.OrderByDescending(e => e.Exam_ID).FirstOrDefault().Exam_ID;
                var course = _context.Courses.Where(c => c.Course_ID == 200).FirstOrDefault();
    //get st id from session after st logging in
                int? St_ID = HttpContext.Session.GetInt32("St_ID");

    sPExamGenerationViewModel.Exam_ID = Exam_ID + 1;
                sPExamGenerationViewModel.Course_ID = course.Course_ID;
                sPExamGenerationViewModel.Exam_Level = "Intermediate";
                sPExamGenerationViewModel.Exam_Type = "Online";
                sPExamGenerationViewModel.Exam_Duration = "30 min";
                sPExamGenerationViewModel.Exam_FullMark = 50;
                sPExamGenerationViewModel.Exam_Name = course.Course_Name;
                sPExamGenerationViewModel.Num_TF_Questions = 5;
                sPExamGenerationViewModel.Num_MCQ_Questions = 5;
                //calling stored procedure of Exam_Generation
                var storedGenResult =
                        await _context.Procedures.Exam_GenerationAsync(
                         sPExamGenerationViewModel.Exam_ID,
                         sPExamGenerationViewModel.Course_ID,
                         sPExamGenerationViewModel.Exam_Level,
                         sPExamGenerationViewModel.Exam_Type,
                         sPExamGenerationViewModel.Exam_Duration,
                         sPExamGenerationViewModel.Exam_FullMark,
```

```
                sPExamGenerationViewModel.Exam_Name,
                sPExamGenerationViewModel.Num_TF_Questions = 5,
                sPExamGenerationViewModel.Num_MCQ_Questions = 5
            );

            Exams exam = _context.Exams
                    .Include(c => c.Exam_Quest_Generation)
                    .ThenInclude(e => e.Question)
                    .ThenInclude(c => c.Question_Choices)
                    .Where(c => c.Exam_ID == sPExamGenerationViewModel.Exam_ID)
                    .FirstOrDefault();

            List<ExamAndQuestions> qAndA = exam.Exam_Quest_Generation.Select(c => new ExamAndQuestions
            {
                    ExamID = c.Exam_ID,
                    QuestionId = c.Question_ID,
                    QuestionText = c.Question.Question_Text,
                    Choices = c.Question.Question_Choices.ToList(),
                    QuestionType = c.Question.Question_Type
            }).OrderBy(c => c.QuestionId).ToList();


             ExamViewModel result = new ExamViewModel()
             {
                    Questions = qAndA,
                    Exam_Duration = exam.Exam_Duration,
                    Exam_FullMark = exam.Exam_FullMark,
                    Exam_Name = exam.Exam_Name,
                    Exam_Date = exam.Exam_Date,
                    Exam_Type = exam.Exam_Type,
                    Exam_Level = exam.Exam_Level,
                    Exam_ID = exam.Exam_ID,
                    Course = exam.Course,
                    Course_ID = exam.Course_ID
             };

             // Pass the exam to the view for display
             return View(result);
        }
        //Action that take student Answer with question id and send it to storde proc of exam answer
        [HttpPost]
        public async Task<IActionResult> Exsamination(ExamViewModel examViewModel)
        {
                // Get the student ID from the session
                int? St_ID = HttpContext.Session.GetInt32("St_ID");
                Dictionary<int, string> QId_ChoiceTxtDictionary = new Dictionary<int, string>();
    //to solve problem of if student dont answer any exam and exam time expire to => take questions id from exam and make st answer =""
                //and split the studend answer as key=> Question id, value => St Answer
                if (examViewModel.sPExam != null && examViewModel.sPExam.Count() > 0)
                {
        QId_ChoiceTxtDictionary = examViewModel.sPExam.ToDictionary(x => int.Parse(x.Split('_')[0]), x => x.Split('_')[1]);
    }
    else
}
else
{
                    QId_ChoiceTxtDictionary.
                            Add((int)_context.Exam_Quest_Generation
                            .Where(c => c.Exam_ID == examViewModel.Exam_ID)
                            .FirstOrDefault()?
                            .Question_ID
                            , "");
}
foreach (var q_and_A in QId_ChoiceTxtDictionary)
    {
        await _context.Procedures.exam_answerAsync(
                    St_ID, examViewModel.Exam_ID,
                    q_and_A.Key, q_and_A.Value);
    }
    await _context.Procedures.Exam_CorrectionAsync(examViewModel.Exam_ID,
        St_ID);

            return RedirectToAction("Grade", "Exam", new { ExamId=examViewModel.Exam_ID}); // Redirect to a different action after saving answers
    }
```

```
@model ITIOnlineExaminationSystem.Model.ViewModels.ExamViewModel
@{
    ViewData["Title"] = "Exam";
    int mcqCount = 0; // Counter for MCQ questions
    int tfCount = 0; // Counter for TF questions
    Layout = "_Layout";
}
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="https://kit.fontawesome.com/3304859e00.js" crossorigin="anonymous"></script>
<!-- script -->
<script src="../js/qustionsLibrary.js"></script>
<script src="../js/cokies.js"></script>

<div class="exam-container">
    <div class="test-container">
        <div class="timer-container">
            <div id="timer"></div>
        </div>

        <div class="content-container">
            @using (Html.BeginForm("Exsamination", "Exam", FormMethod.Post))
            {
                @Html.HiddenFor(model => model.Exam_ID)
                @Html.HiddenFor(model => model.Questions)
                @foreach (var question in Model.Questions)
                {
                    // Check if the question is an MCQ question and display only 5 MCQ questions
                    if (question.QuestionType == "MCQ" && mcqCount < 5)
                    {
                        <div id="ques-container">
```

```
    // Check if the question is an MCQ question and display only 5 MCQ questions
    if (question.QuestionType == "MCQ" && mcqCount < 5)
    {
        <div id="ques-container">
            <div class="que-mark-container">
                <div id="quesHeader">@question.QuestionText</div>
            </div>
            @for (int j = 0; j < question.Choices.Count; j++)
            {
                @Html.RadioButtonFor(m => m.sPExam[Model.Questions.ToList().IndexOf(question)], question.QuestionId + "_" + question.Choices[j].Choice,
                        new { @id = question.QuestionId })
                <label for="@($"{question.QuestionId}")" id="@($"{question.Choices[j].Question_ID}")" class="answers-lables">@($"{question.Choices[j].Choice}") </label>
                <br>
            }
        </div>
        mcqCount++; // Increment the counter for MCQ questions
    }
}

@foreach (var question in Model.Questions)
{
    // Check if the question is a TF question and display only 5 TF questions
    if (question.QuestionType == "T or F" && tfCount < 5)
    {
        <div id="ques-container">
            <div class="que-mark-container">
                <div id="quesHeader">@question.QuestionText</div>
            </div>
            @for (int j = 0; j < question.Choices.Count; j++)
            {
```

```
                    @Html.RadioButtonFor(m => m.sPExam[Model.Questions.ToList().IndexOf(question)], question.QuestionId + "_" + question.Choices[j].Choice,
                            new { @id = question.QuestionId })

                    <label for="@($"{question.Choices}")" id="@($"{question.Choices[j].Question_ID}")" class="answers-lables">@($"{question.Choices[j].Choice}") </label>

                    <br>
                }
            </div>
            tfCount++; // Increment the counter for TF questions
        }
    }


    <div class="nav-submit-container">
        <input type="submit" value="Submit" class="btn border-btn btn-exam " id="click">
    </div>
}
</div>
</div>
</div>
<script type="text/javascript">
    function startExamBtn() {
        window.open("/Exam/Exsamination", "_self");
    }
</script>
```

```
    }
</script>
<script>
    function startTimer(duration, display) {
        var timer = duration, minutes, seconds;
        var intervalId = setInterval(function () {
            minutes = parseInt(timer / 60, 10);
            seconds = parseInt(timer % 60, 10);

            minutes = minutes < 10 ? "0" + minutes : minutes;
            seconds = seconds < 10 ? "0" + seconds : seconds;

            display.textContent = minutes + ":" + seconds;
            if (--timer < 0) {
                clearInterval(intervalId); // Stop the timer
                // Add code here to submit the exam
                document.getElementById("click").click() // Submit the form

            }
        }, 1000);
    }

    window.onload = function () {
        var duration = 60 * 30; // 30 minutes
        var display = document.querySelector('#timer');
        startTimer(duration, display);
    };

</script>
<script src="../js/main.js"></script>
```

- The exam screen displays a 30-minute timer and includes 10 random questions (true/false and multiple-choice) for the student taking the exam.

- Each student can take one exam in each course, but a student can take many exams across multiple courses.

- If the exam time expires, it will be submitted automatically.

Grade Page

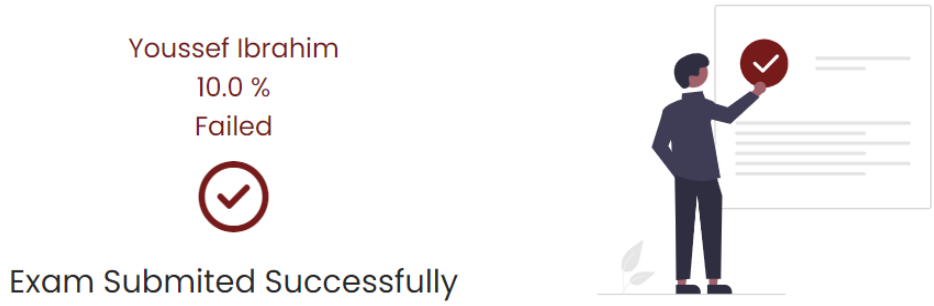


Figure 5: Grade Screen

Source Code:

```
        //Action that will return grade for student
        [HttpGet]
public async Task<IActionResult> Grade(int ExamId)
        {
                int? St_ID = HttpContext.Session.GetInt32("St_ID");
                var ExamResultGrade =  await _context.Student_Takes_Exam.Where(c => c.Exam_ID == ExamId && c.St_ID == St_ID).FirstOrDefaultAsync();
                StudentsInfo studentsInfo = _context.StudentsInfo.Where(c => c.St_ID == St_ID).FirstOrDefault();
                StudentResult studentResult = new StudentResult()
                {
                        St_ID = (int)St_ID,
                        Grade = (decimal)ExamResultGrade.Exam_Percentage,
                        St_Name = studentsInfo.St_Fname + " " + studentsInfo.St_Lname,
                        St_Status = ExamResultGrade.St_Status
    };
                return View(studentResult);
        }
}
```

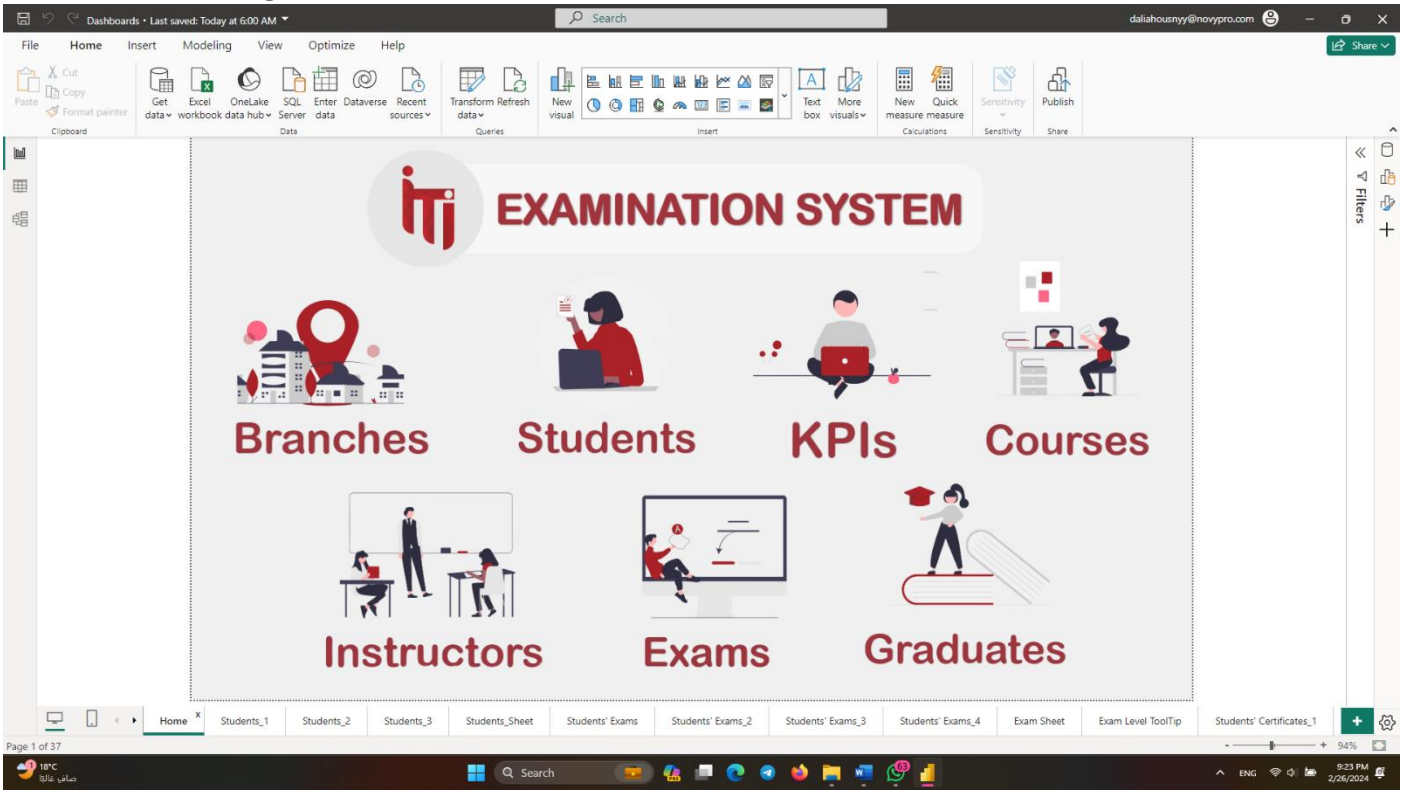
```
        <div class="grade-image">
            <img src="../images/Done.svg" class="grade-image" />
        </div>
        <div id="grade-page-container">

            <div class="g-username"> @Model.St_Name </div>
            <div class="g-grade"> @Model.Grade %</div>
            <div class="g-grade"> @Model.St_Status</div>

            <div class="done-icon"><i class="far fa-check-circle"></i></div>
            <div class="grade-page-title">Exam Submited Successfully</div>


        </div>
```

- The grade screen displays the student's full name, exam grade percentage, and status (Failed or Passed).

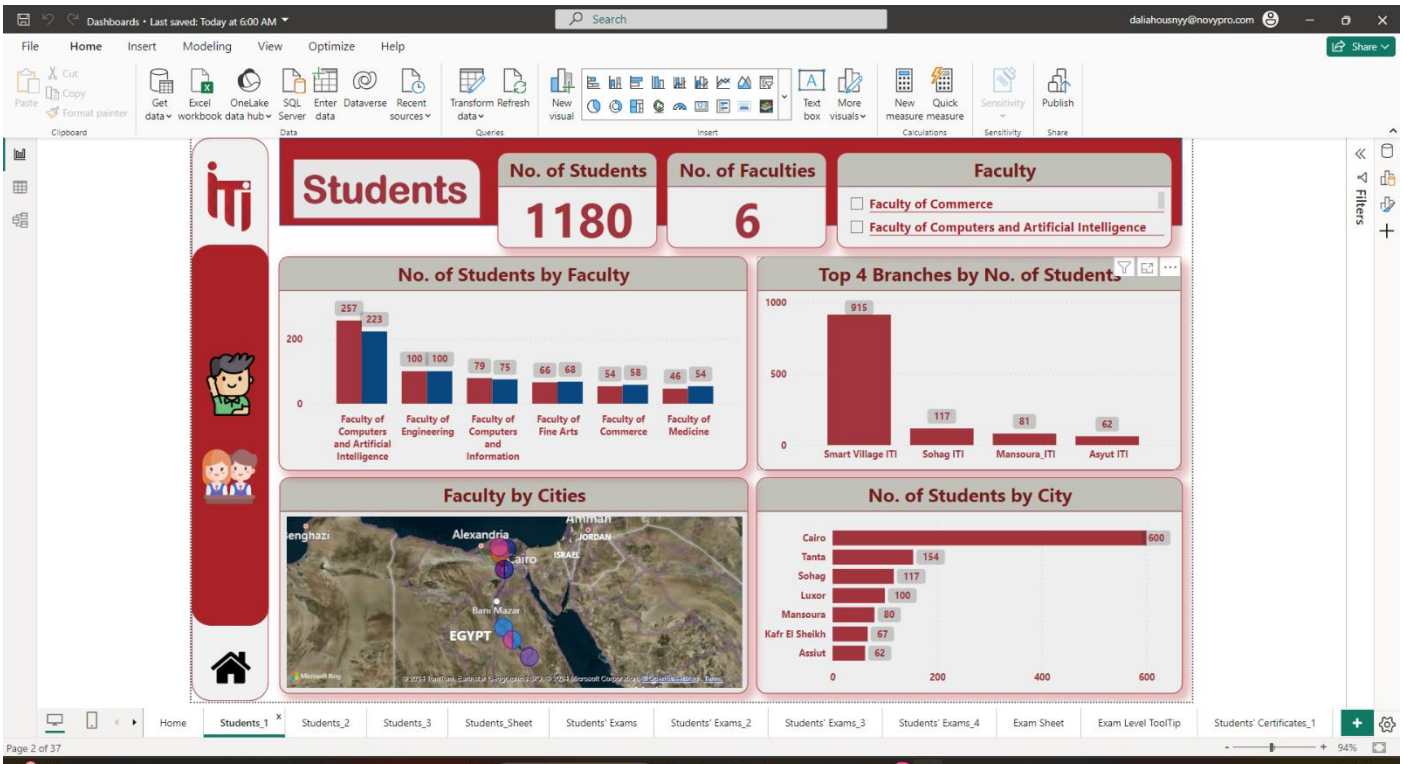
- If the exam time expires and the student has not answered any questions, the exam will be submitted automatically with a grade of 0.0% and a status of 'failed'.

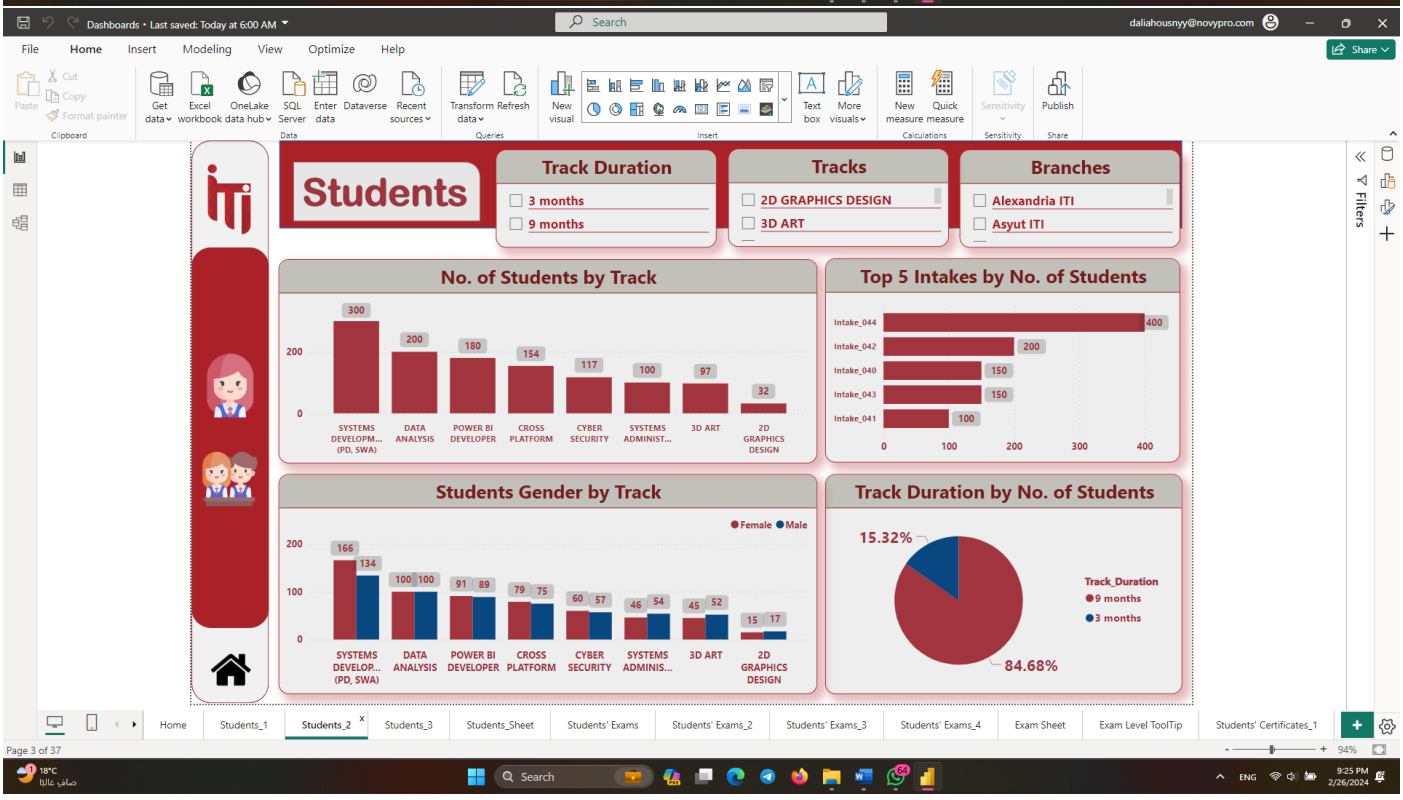
12. Dashboards

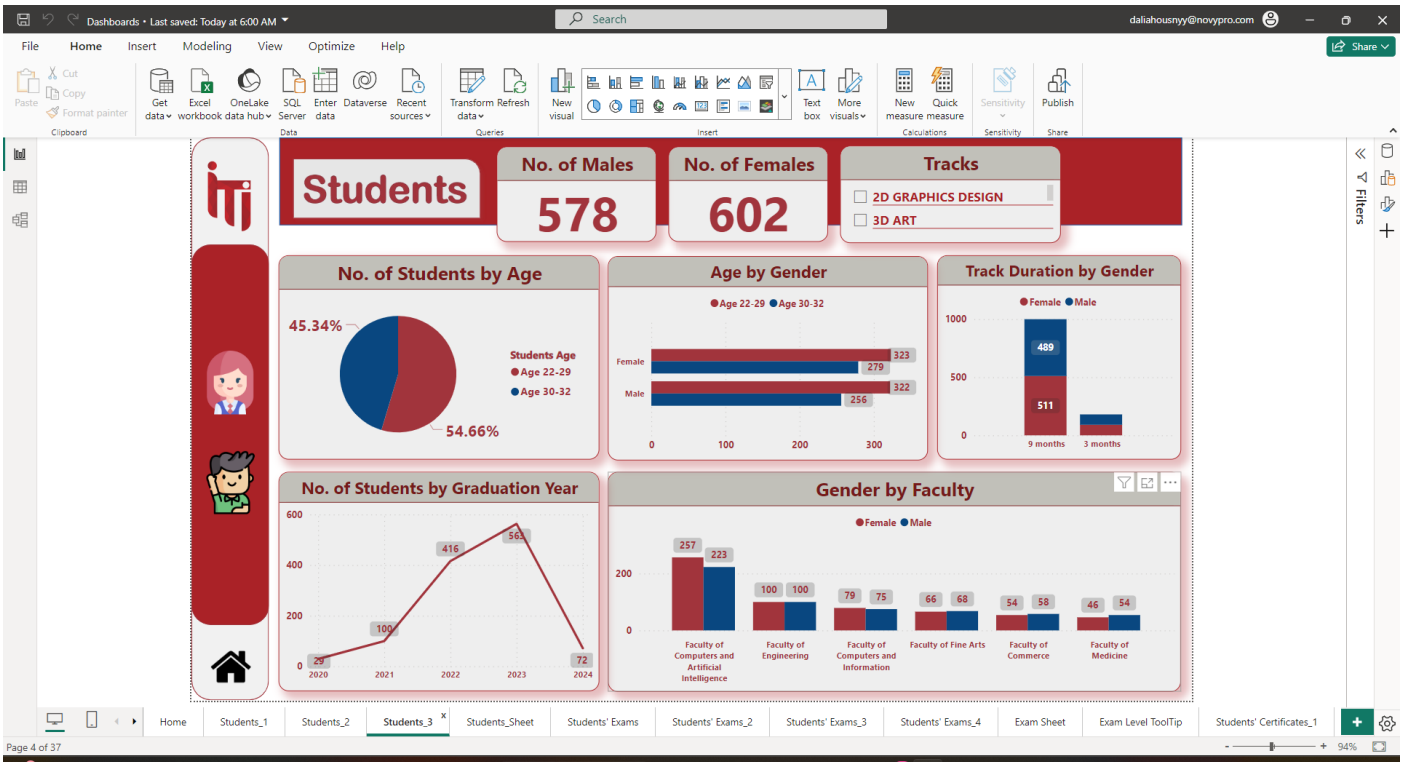
1. Home Page

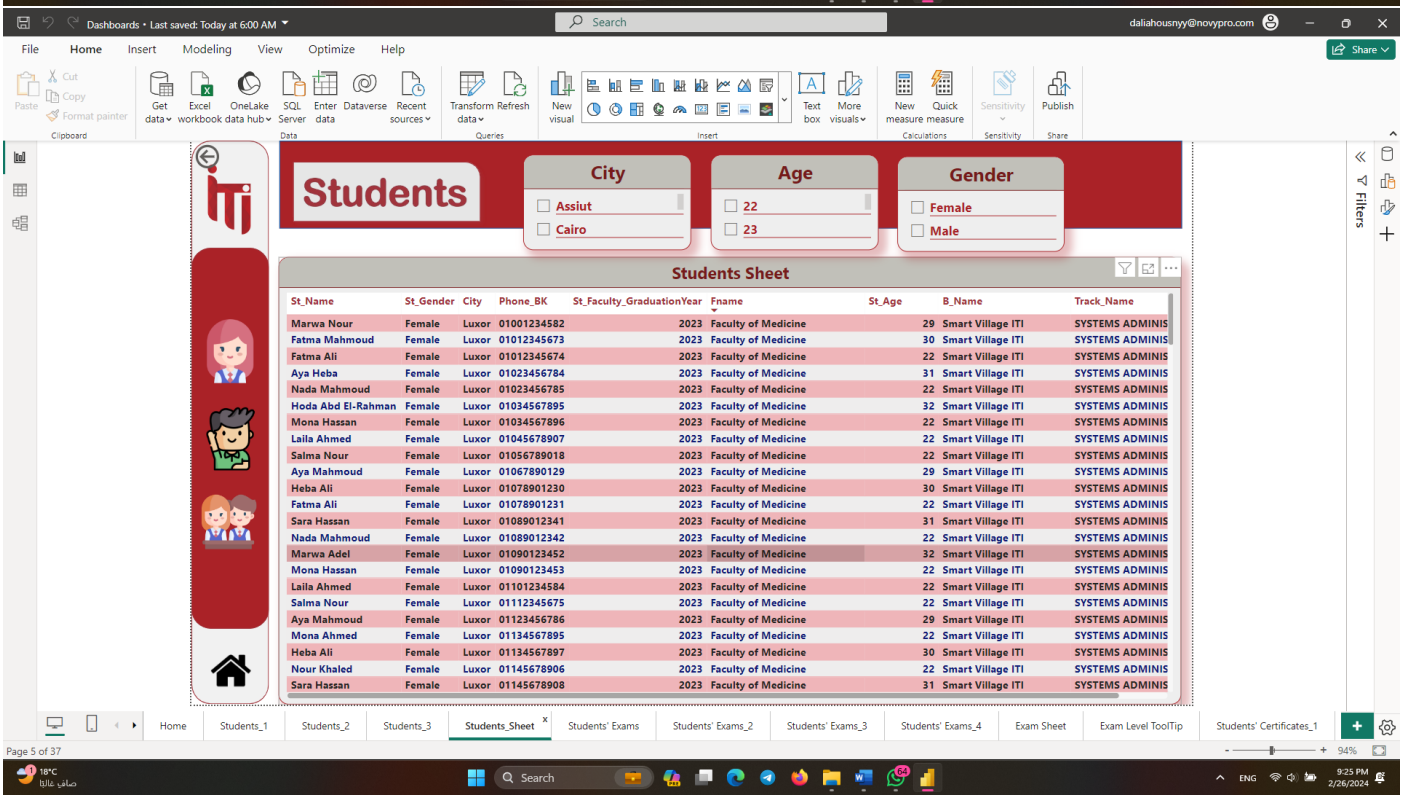


2. Students


Students
No. of Students
1180
No. of Faculties
6
Faculty
Faculty of Commerce
Faculty of Computers and Artificial Intelligence
No. of Students by Faculty
Faculty of Computers and Artificial Intelligence
Faculty of Engineering
Faculty of Computers and Information
Faculty of Fine Arts
Faculty of Commerce
Faculty of Medicine
Top 4 Branches by No. of Students
Smart Village ITI
Sohag ITI
Mansoura_ITI
Asyut ITI
Faculty by Cities
No. of Students by City
Cairo
Tanta
Sohag
Luxor
Mansoura
Kafr El Sheikh
Assiut

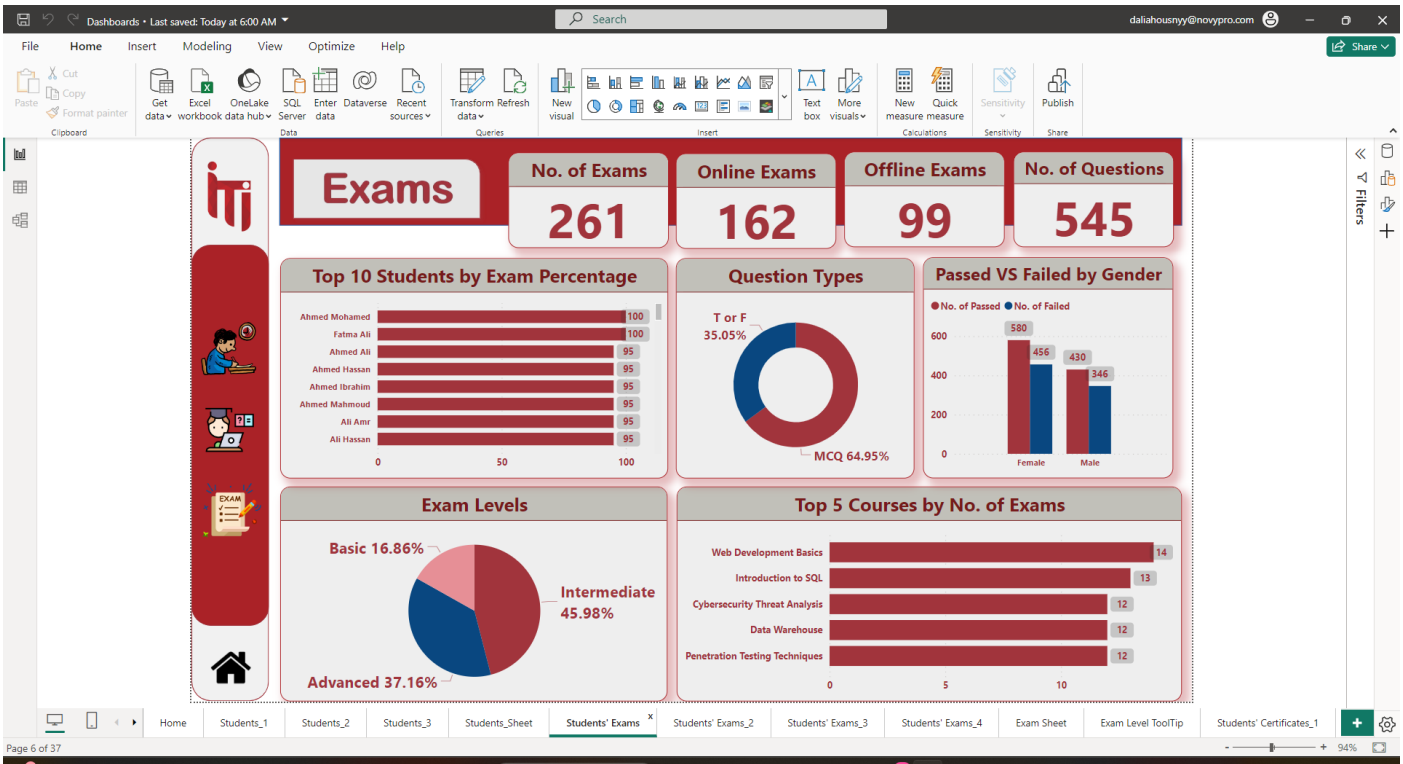

Students
Track Duration
3 months
9 months
Tracks
2D GRAPHICS DESIGN
3D ART
Branches
Alexandria ITI
Asyut ITI
No. of Students by Track
Top 5 Intakes by No. of Students
Students Gender by Track
Female
Male
Track Duration by No. of Students
15.32%
84.68%
Track_Duration
9 months
3 months

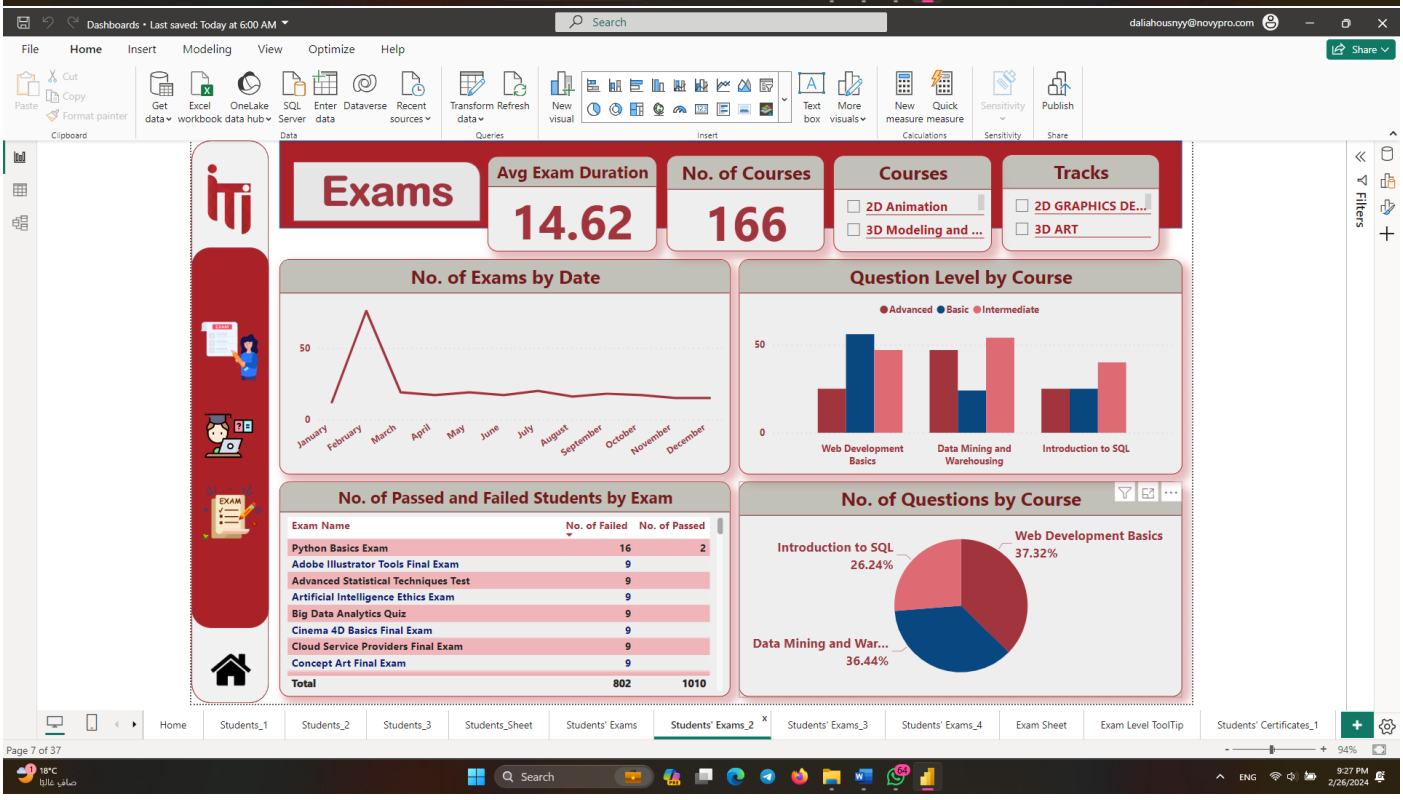


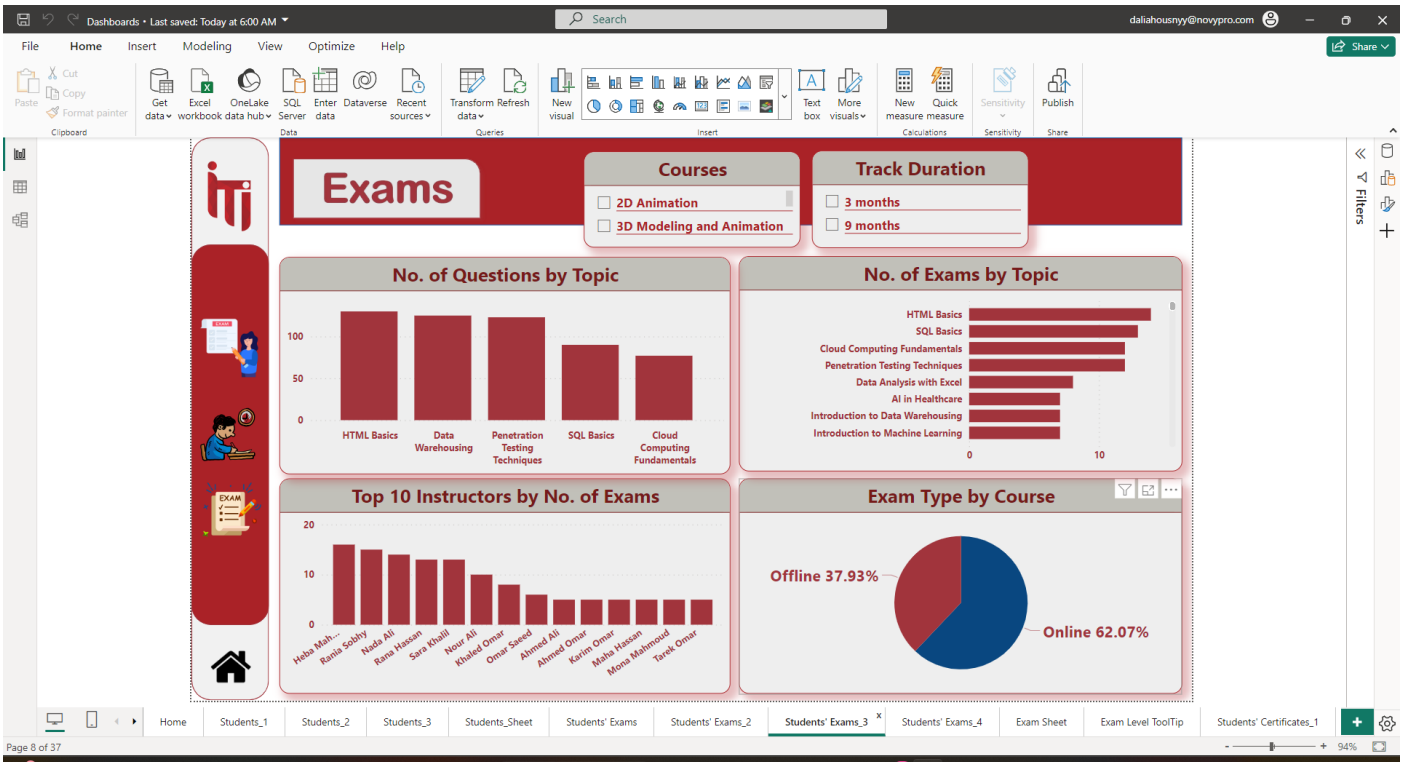


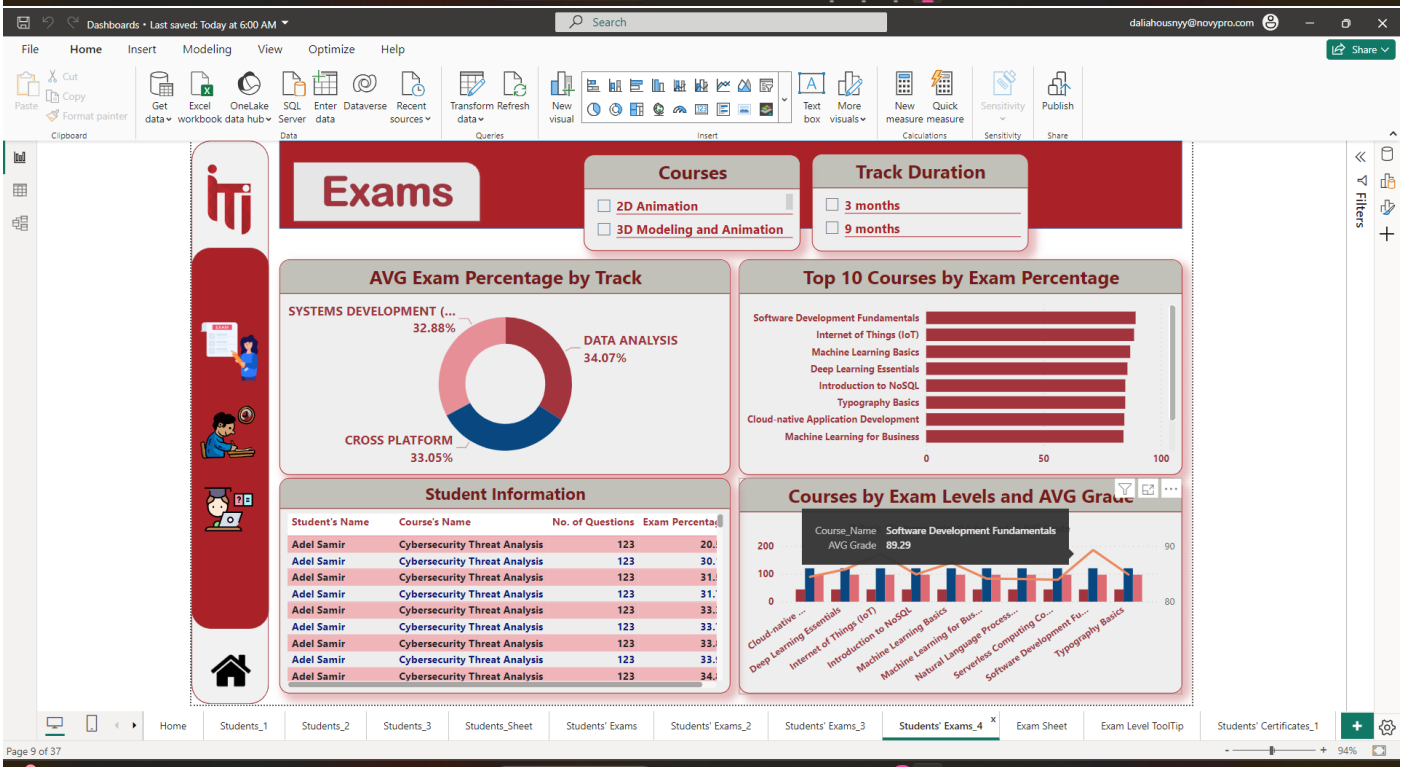
St_Name	St_Gender	City	Phone_BK	St_Faculty_GraduationYear	Fname	St_Age	B_Name	Track_Name
Marwa Nour	Female	Luxor	01001234582	2023	Faculty of Medicine	29	Smart Village ITI	SYSTEMS ADMINIS
Fatma Mahmoud	Female	Luxor	01012345673	2023	Faculty of Medicine	30	Smart Village ITI	SYSTEMS ADMINIS
Fatma Ali	Female	Luxor	01012345674	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Aya Heba	Female	Luxor	01023456784	2023	Faculty of Medicine	31	Smart Village ITI	SYSTEMS ADMINIS
Nada Mahmoud	Female	Luxor	01023456785	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Hoda Abd El-Rahman	Female	Luxor	01034567895	2023	Faculty of Medicine	32	Smart Village ITI	SYSTEMS ADMINIS
Mona Hassan	Female	Luxor	01034567896	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Laila Ahmed	Female	Luxor	01045678907	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Salma Nour	Female	Luxor	01056789018	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Aya Mahmoud	Female	Luxor	01067890129	2023	Faculty of Medicine	29	Smart Village ITI	SYSTEMS ADMINIS
Heba Ali	Female	Luxor	01078901230	2023	Faculty of Medicine	30	Smart Village ITI	SYSTEMS ADMINIS
Fatma Ali	Female	Luxor	01078901231	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Sara Hassan	Female	Luxor	01089012341	2023	Faculty of Medicine	31	Smart Village ITI	SYSTEMS ADMINIS
Nada Mahmoud	Female	Luxor	01089012342	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Marwa Adel	Female	Luxor	01090123452	2023	Faculty of Medicine	32	Smart Village ITI	SYSTEMS ADMINIS
Mona Hassan	Female	Luxor	01090123453	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Laila Ahmed	Female	Luxor	01101234584	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Salma Nour	Female	Luxor	01112345675	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Aya Mahmoud	Female	Luxor	01123456786	2023	Faculty of Medicine	29	Smart Village ITI	SYSTEMS ADMINIS
Mona Ahmed	Female	Luxor	01134567895	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Heba Ali	Female	Luxor	01134567897	2023	Faculty of Medicine	30	Smart Village ITI	SYSTEMS ADMINIS
Nour Khaled	Female	Luxor	01145678906	2023	Faculty of Medicine	22	Smart Village ITI	SYSTEMS ADMINIS
Sara Hassan	Female	Luxor	01145678908	2023	Faculty of Medicine	31	Smart Village ITI	SYSTEMS ADMINIS

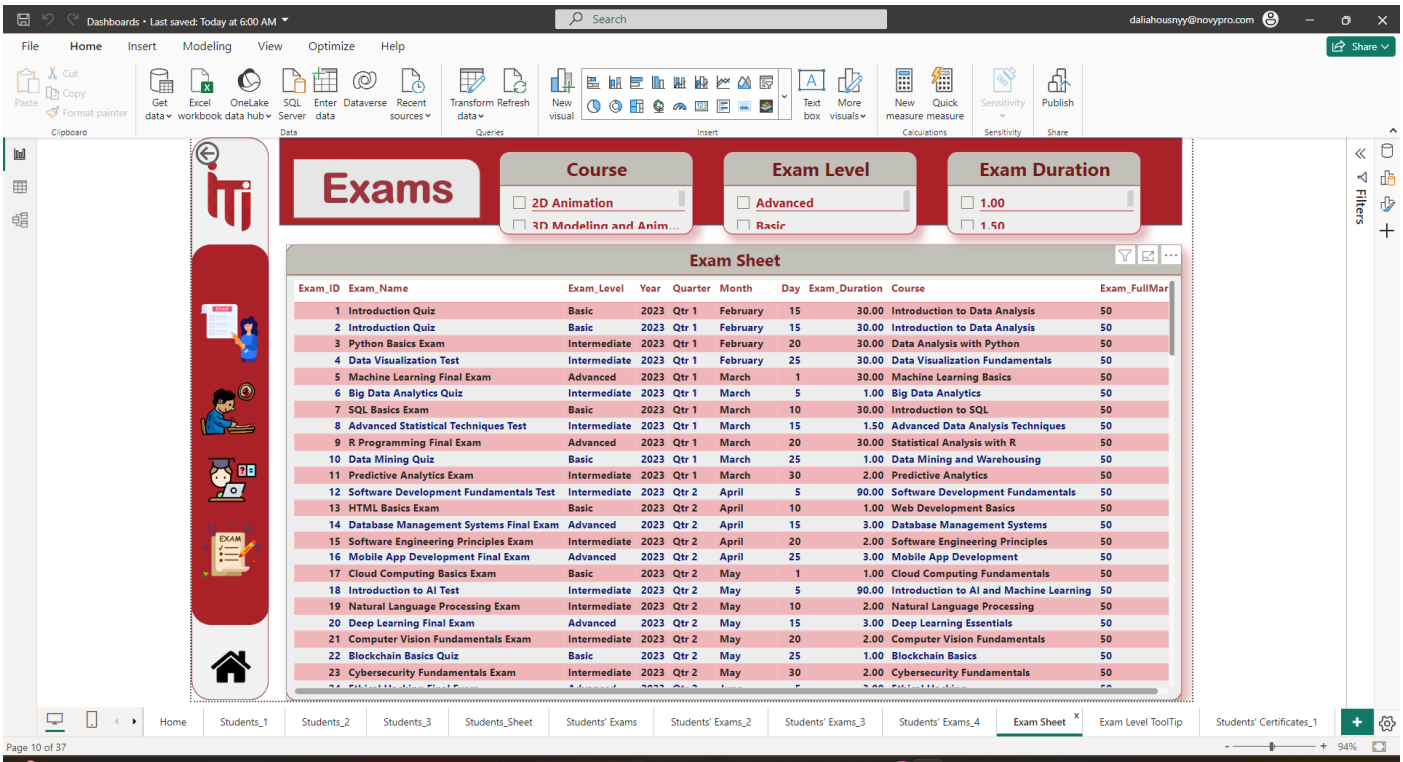
3. Exam

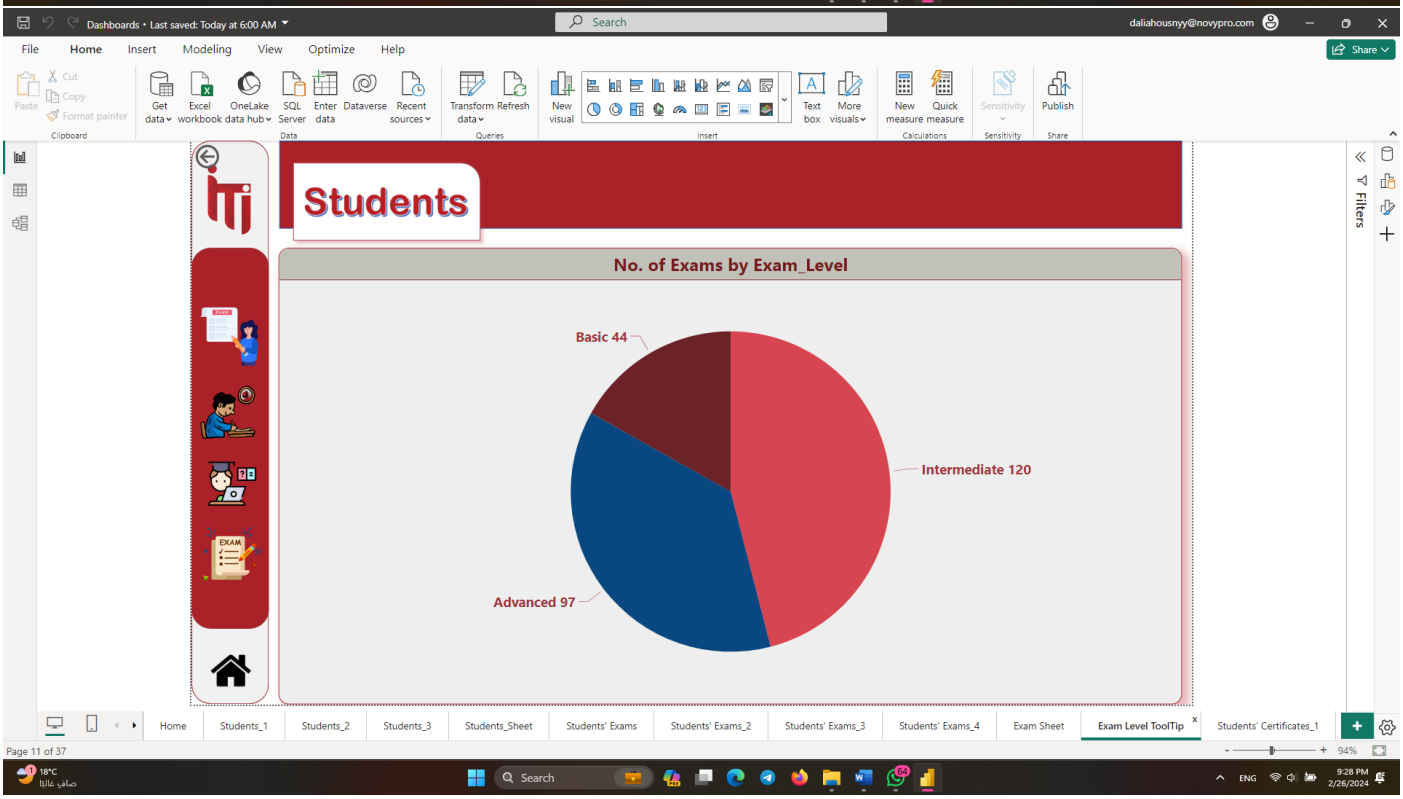

Exams
No. of Exams
261
Online Exams
162
Offline Exams
99
No. of Questions
545
Top 10 Students by Exam Percentage
Ahmed Mohamed 100
Fatma Ali 100
Ahmed Ali 95
Ahmed Hassan 95
Ahmed Ibrahim 95
Ahmed Mahmoud 95
Ali Amr 95
Ali Hassan 95
Question Types
T or F 35.05%
MCQ 64.95%
Passed VS Failed by Gender
No. of Passed
No. of Failed
580
456
430
346
Female
Male
Exam Levels
Basic 16.86%
Intermediate 45.98%
Advanced 37.16%
Top 5 Courses by No. of Exams
Web Development Basics 14
Introduction to SQL 13
Cybersecurity Threat Analysis 12
Data Warehouse 12
Penetration Testing Techniques 12


Exams
Avg Exam Duration
14.62
No. of Courses
166
Courses
2D Animation
3D Modeling and ...
Tracks
2D GRAPHICS DE...
3D ART
No. of Exams by Date
January
February
March
April
May
June
July
August
September
October
November
December
Question Level by Course
Advanced
Basic
Intermediate
Web Development Basics
Data Mining and Warehousing
Introduction to SQL
No. of Passed and Failed Students by Exam
Exam Name
No. of Failed
No. of Passed
Python Basics Exam 16 2
Adobe Illustrator Tools Final Exam 9
Advanced Statistical Techniques Test 9
Artificial Intelligence Ethics Exam 9
Big Data Analytics Quiz 9
Cinema 4D Basics Final Exam 9
Cloud Service Providers Final Exam 9
Concept Art Final Exam 9
Total 802 1010
No. of Questions by Course
Introduction to SQL 26.24%
Web Development Basics 37.32%
Data Mining and War... 36.44%

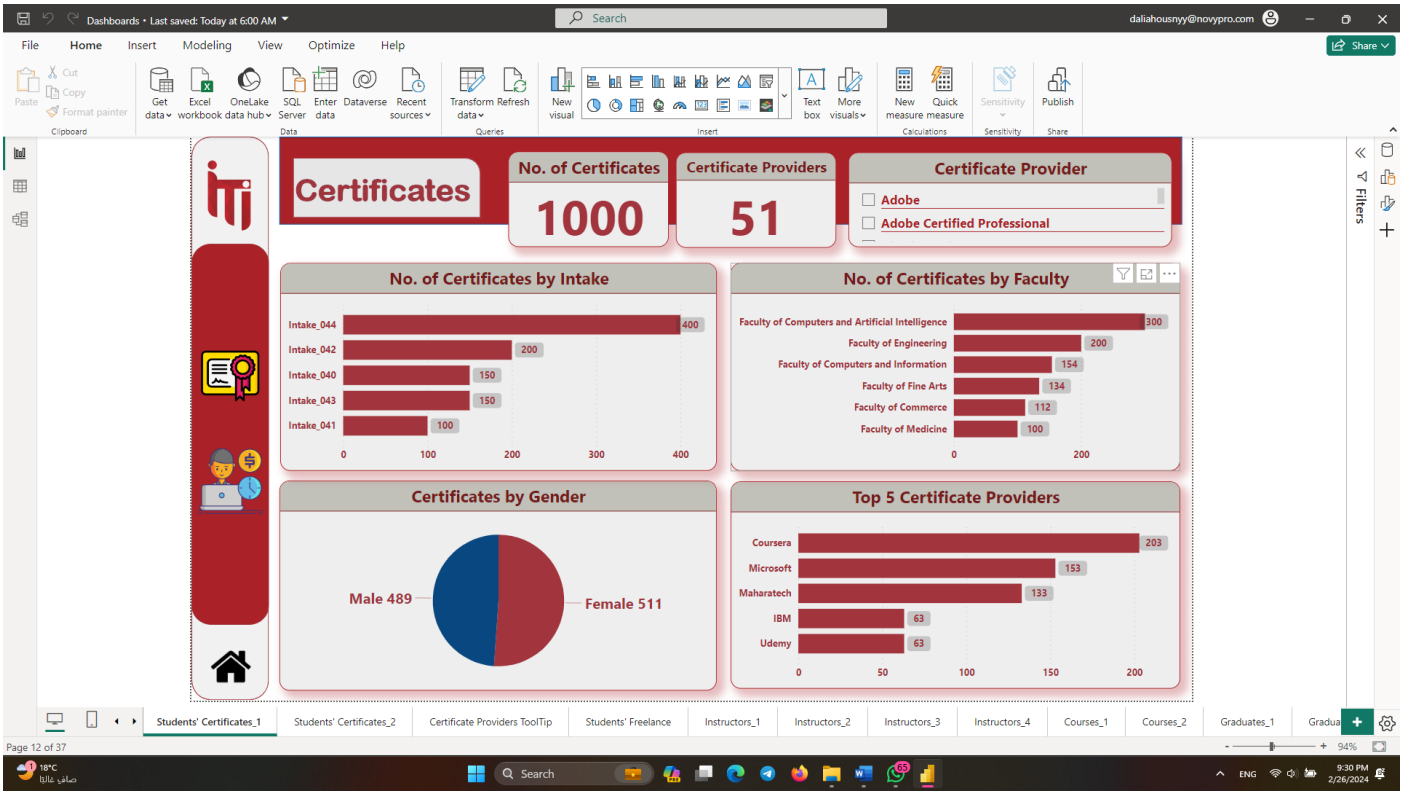

Exams
Courses
2D Animation
3D Modeling and Animation
Track Duration
3 months
9 months
No. of Questions by Topic
HTML Basics
Data Warehousing
Penetration Testing Techniques
SQL Basics
Cloud Computing Fundamentals
No. of Exams by Topic
HTML Basics
SQL Basics
Cloud Computing Fundamentals
Penetration Testing Techniques
Data Analysis with Excel
AI in Healthcare
Introduction to Data Warehousing
Introduction to Machine Learning
Top 10 Instructors by No. of Exams
Exam Type by Course
Offline 37.93%
Online 62.07%

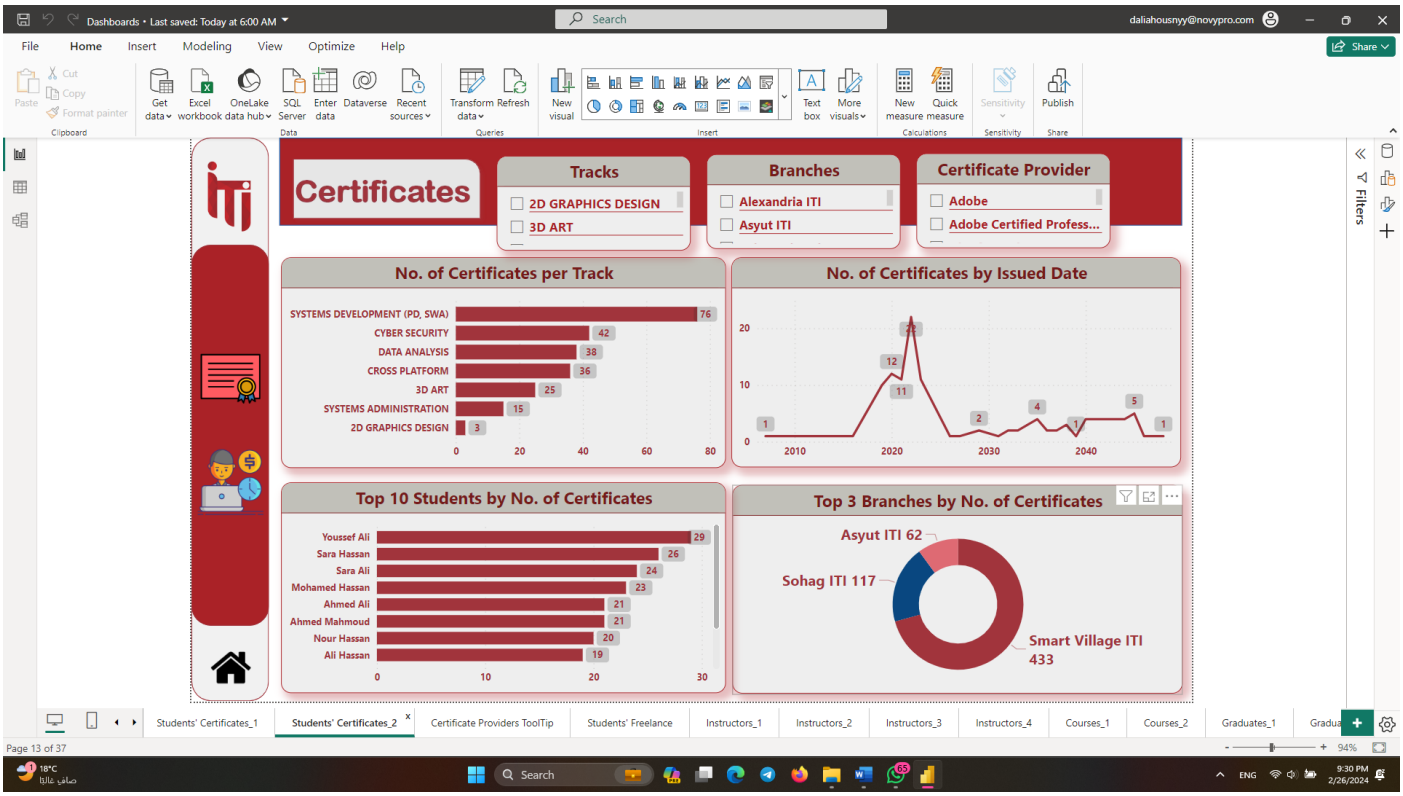

Exams
Courses
2D Animation
3D Modeling and Animation
Track Duration
3 months
9 months
AVG Exam Percentage by Track
SYSTEMS DEVELOPMENT (...
32.88%
DATA ANALYSIS
34.07%
CROSS PLATFORM
33.05%
Top 10 Courses by Exam Percentage
Software Development Fundamentals
Internet of Things (IoT)
Machine Learning Basics
Deep Learning Essentials
Introduction to NoSQL
Typography Basics
Cloud-native Application Development
Machine Learning for Business
Student Information
Courses by Exam Levels and AVG Grade

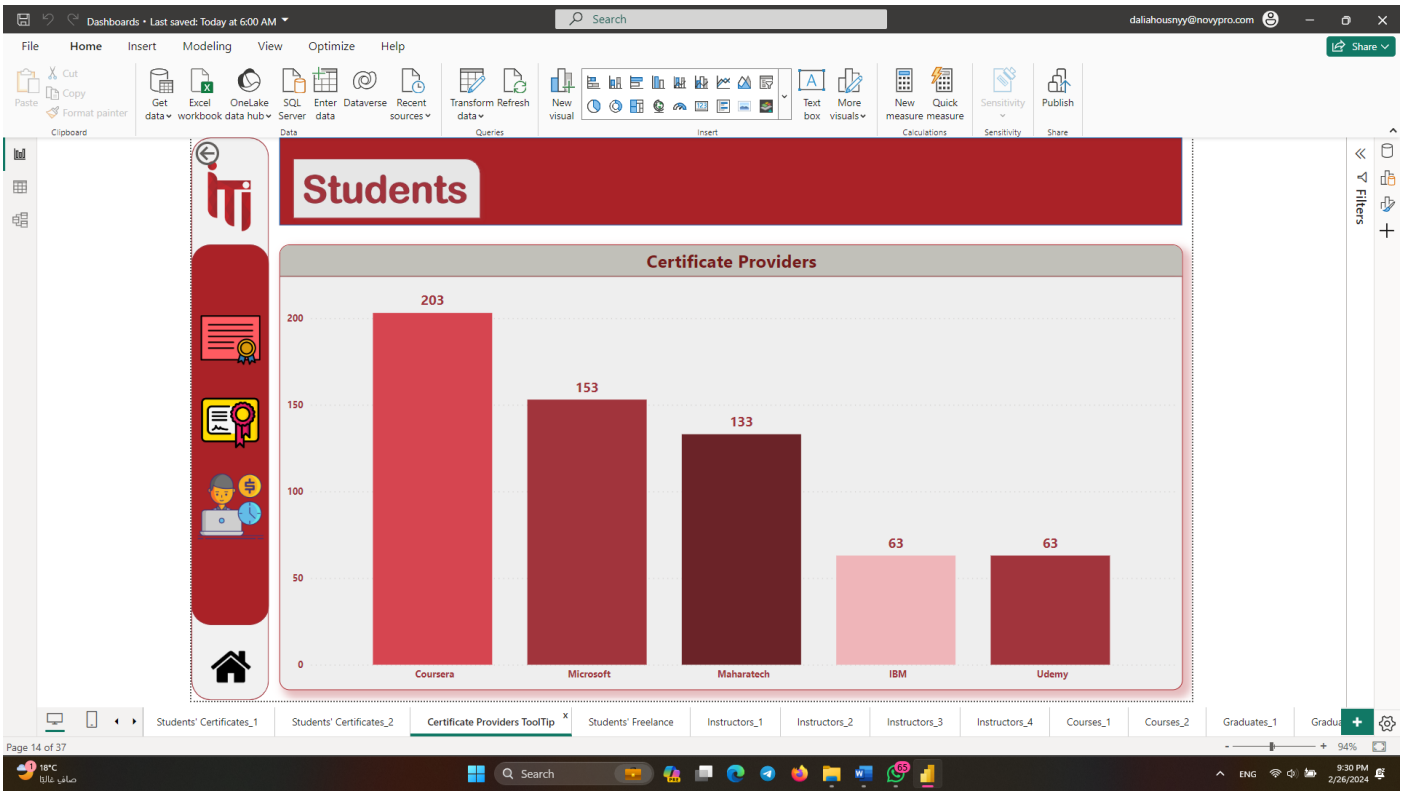

Exams
Course
2D Animation
Exam Level
Advanced
Exam Duration
1.00
Exam Sheet
Exam_ID Exam_Name Exam_Level Year Quarter Month Day Exam_Duration Course Exam_FullMar
1 Introduction Quiz Basic 2023 Qtr 1 February 15 30.00 Introduction to Data Analysis 50
2 Introduction Quiz Basic 2023 Qtr 1 February 15 30.00 Introduction to Data Analysis 50
3 Python Basics Exam Intermediate 2023 Qtr 1 February 20 30.00 Data Analysis with Python 50
4 Data Visualization Test Intermediate 2023 Qtr 1 February 25 30.00 Data Visualization Fundamentals 50
5 Machine Learning Final Exam Advanced 2023 Qtr 1 March 1 30.00 Machine Learning Basics 50
6 Big Data Analytics Quiz Intermediate 2023 Qtr 1 March 5 1.00 Big Data Analytics 50
7 SQL Basics Exam Basic 2023 Qtr 1 March 10 30.00 Introduction to SQL 50
8 Advanced Statistical Techniques Test Intermediate 2023 Qtr 1 March 15 1.50 Advanced Data Analysis Techniques 50
9 R Programming Final Exam Advanced 2023 Qtr 1 March 20 30.00 Statistical Analysis with R 50
10 Data Mining Quiz Basic 2023 Qtr 1 March 25 1.00 Data Mining and Warehousing 50
11 Predictive Analytics Exam Intermediate 2023 Qtr 1 March 30 2.00 Predictive Analytics 50
12 Software Development Fundamentals Test Intermediate 2023 Qtr 2 April 5 90.00 Software Development Fundamentals 50
13 HTML Basics Exam Basic 2023 Qtr 2 April 10 1.00 Web Development Basics 50
14 Database Management Systems Final Exam Advanced 2023 Qtr 2 April 15 3.00 Database Management Systems 50
15 Software Engineering Principles Exam Intermediate 2023 Qtr 2 April 20 2.00 Software Engineering Principles 50
16 Mobile App Development Final Exam Advanced 2023 Qtr 2 April 25 3.00 Mobile App Development 50
17 Cloud Computing Basics Exam Basic 2023 Qtr 2 May 1 1.00 Cloud Computing Fundamentals 50
18 Introduction to AI Test Intermediate 2023 Qtr 2 May 5 90.00 Introduction to AI and Machine Learning 50
19 Natural Language Processing Exam Intermediate 2023 Qtr 2 May 10 2.00 Natural Language Processing 50
20 Deep Learning Final Exam Advanced 2023 Qtr 2 May 15 3.00 Deep Learning Essentials 50
21 Computer Vision Fundamentals Exam Intermediate 2023 Qtr 2 May 20 2.00 Computer Vision Fundamentals 50
22 Blockchain Basics Quiz Basic 2023 Qtr 2 May 25 1.00 Blockchain Basics 50
23 Cybersecurity Fundamentals Exam Intermediate 2023 Qtr 2 May 30 2.00 Cybersecurity Fundamentals 50


Students
No. of Exams by Exam_Level
Basic 44
Intermediate 120
Advanced 97

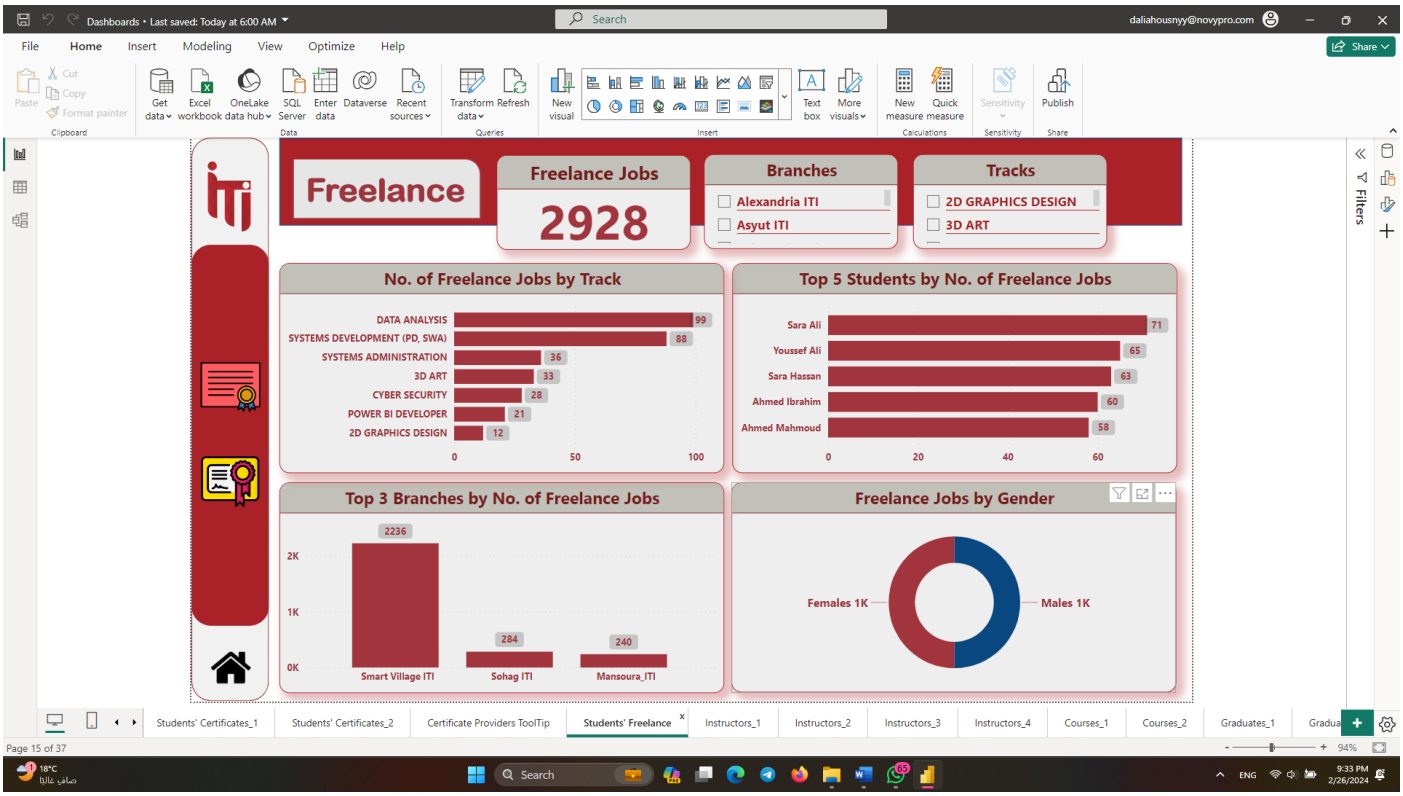
4. Students' Certificates



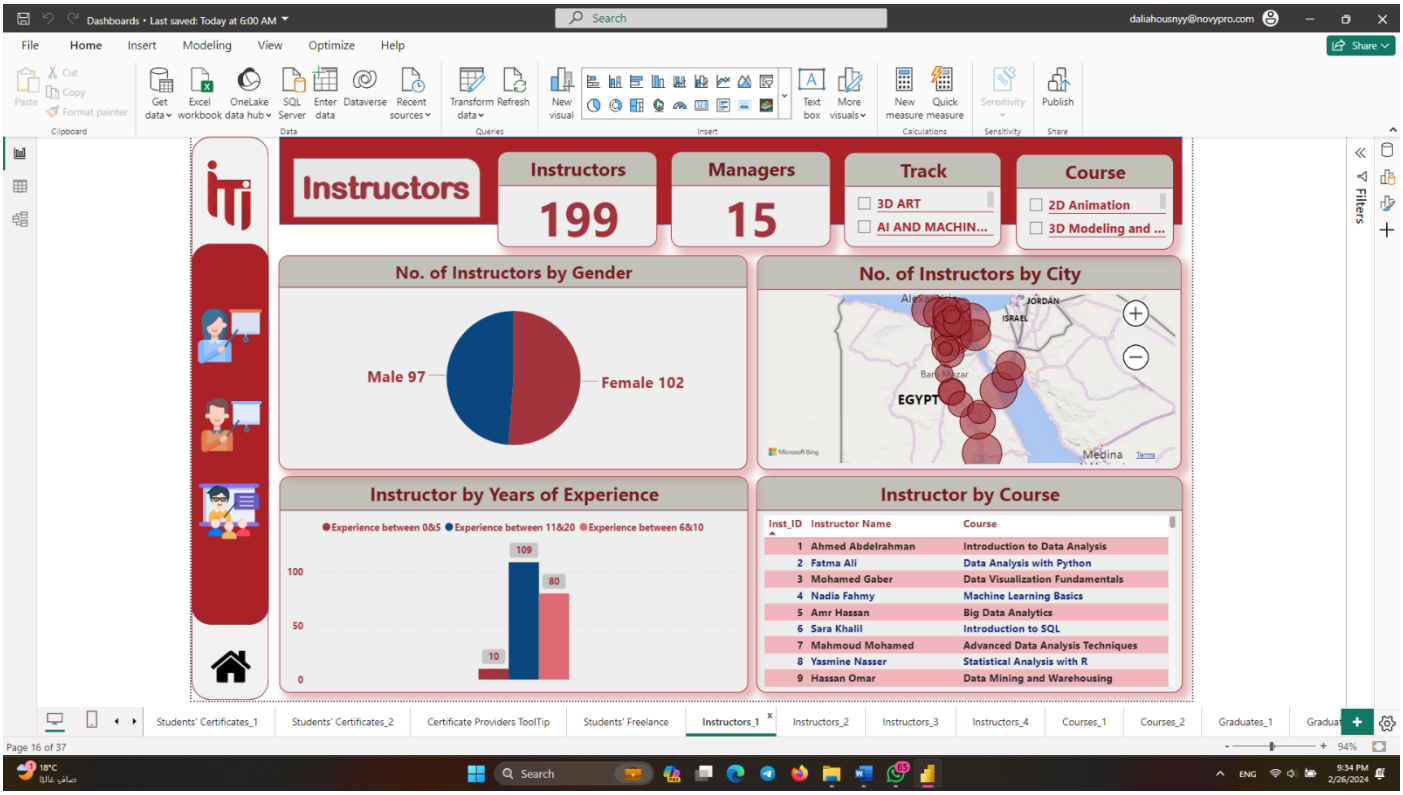


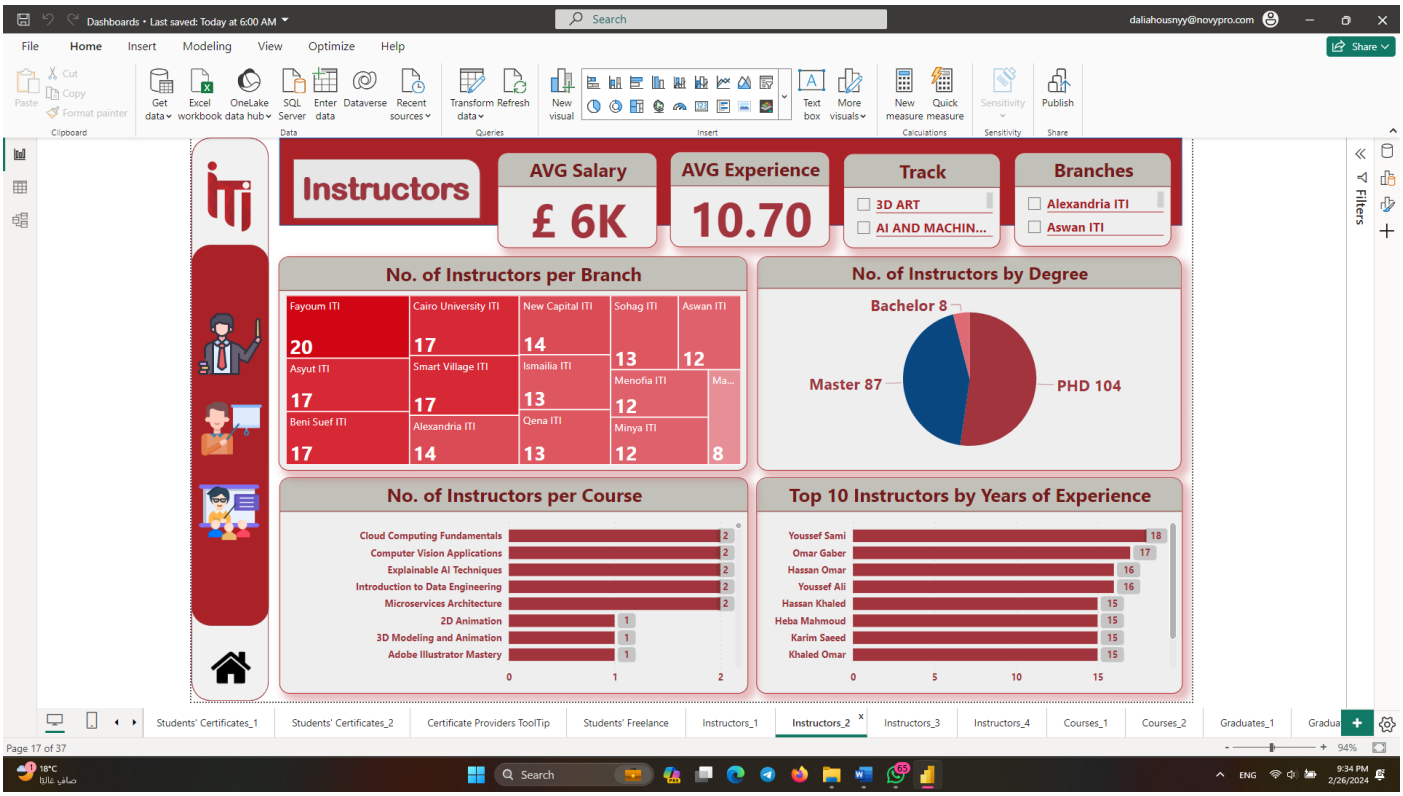


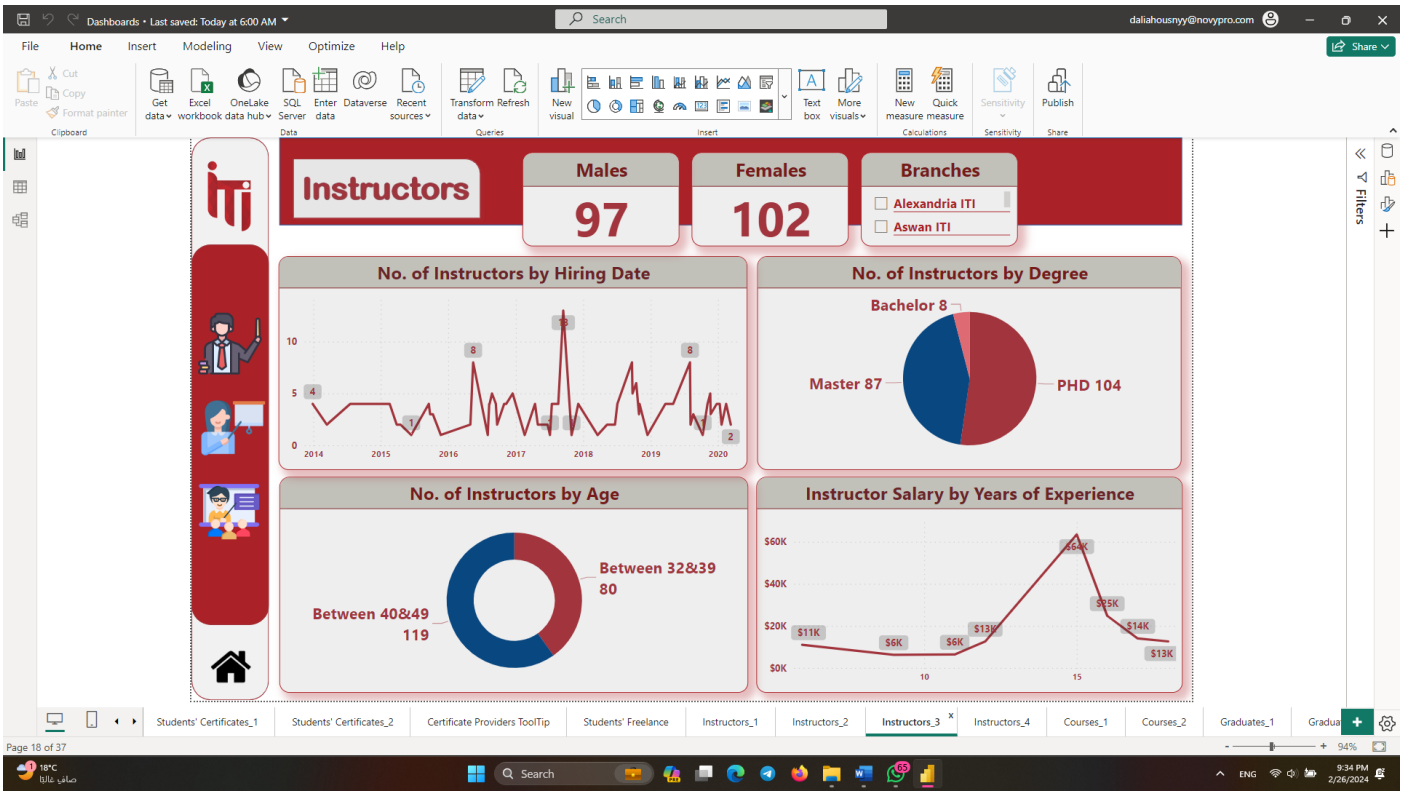
5. Students’ Freelance Jobs

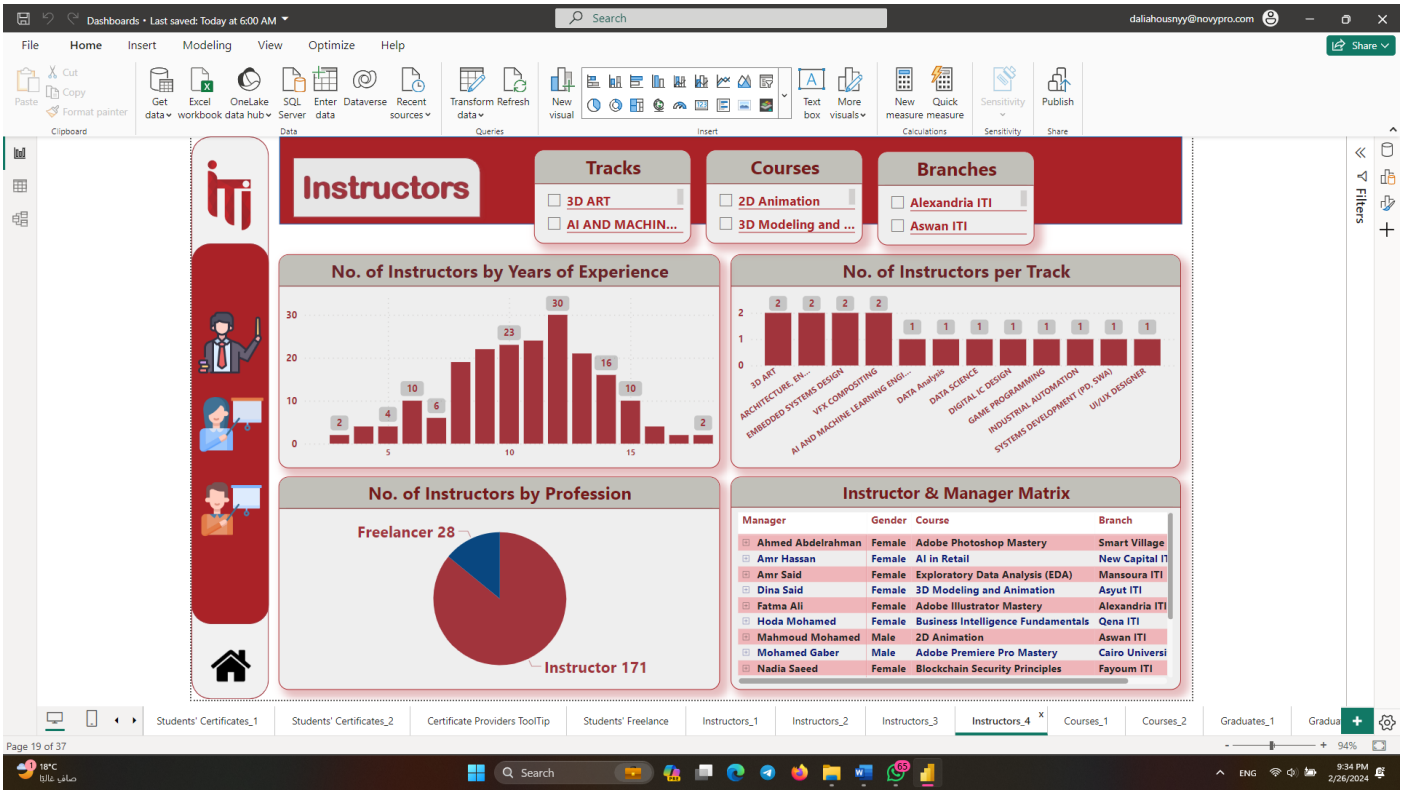


6. Instructors



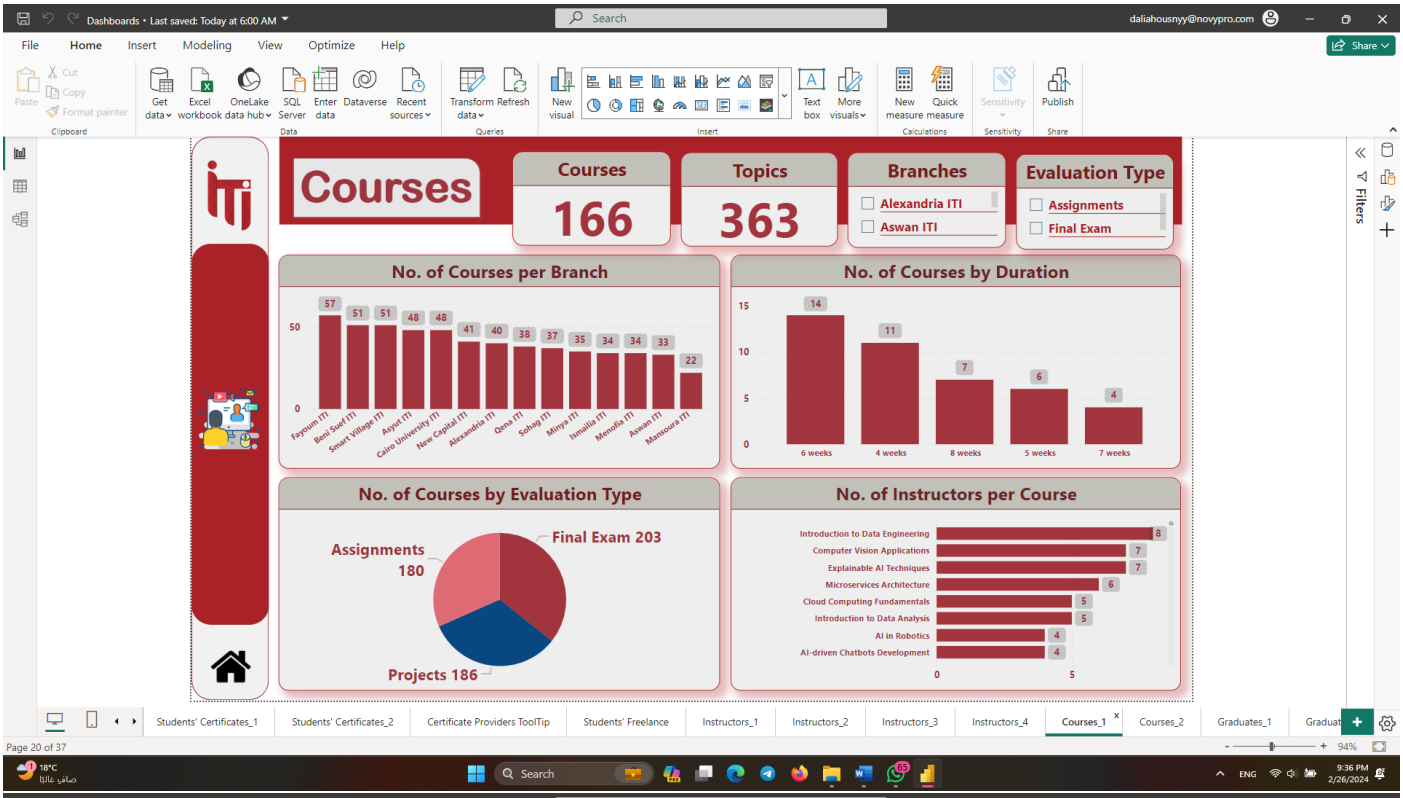


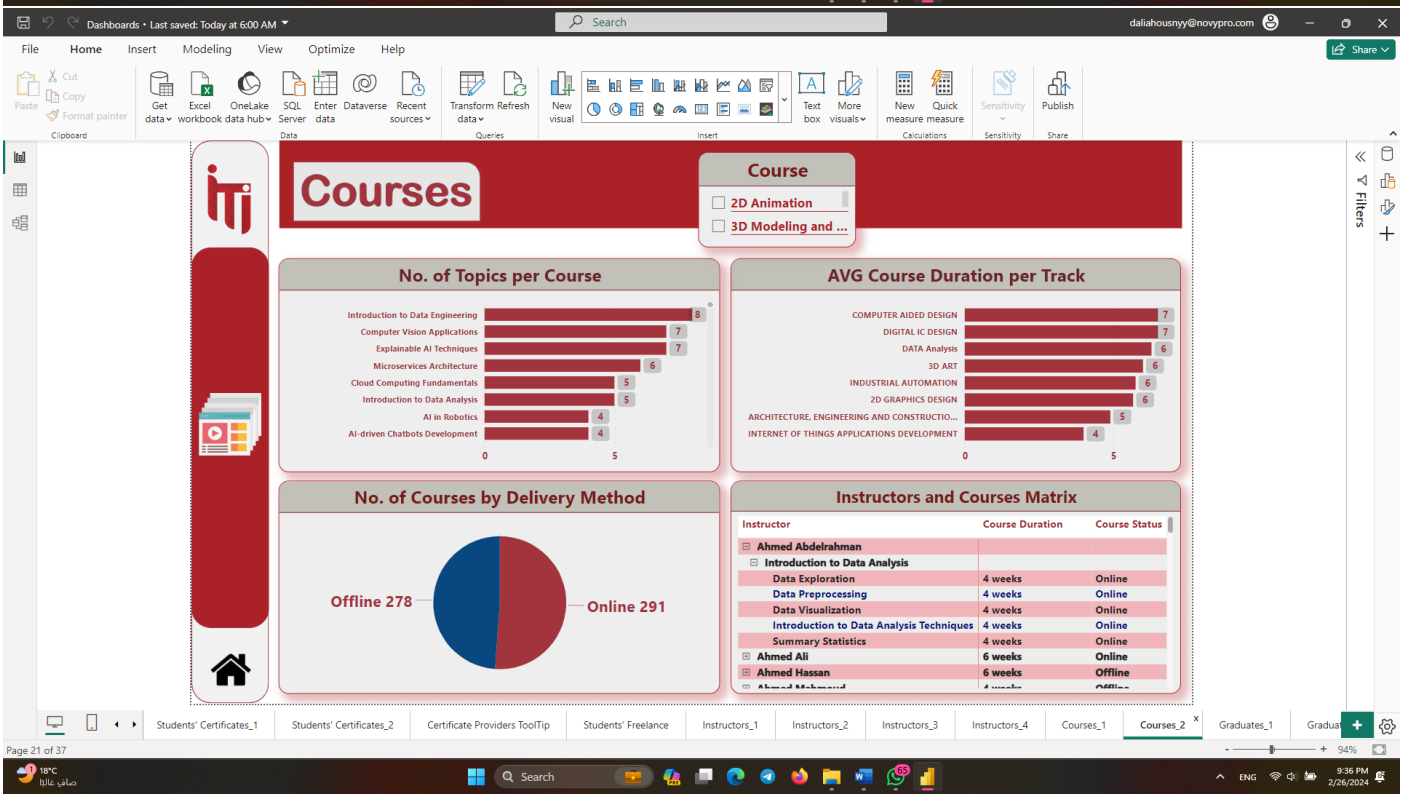




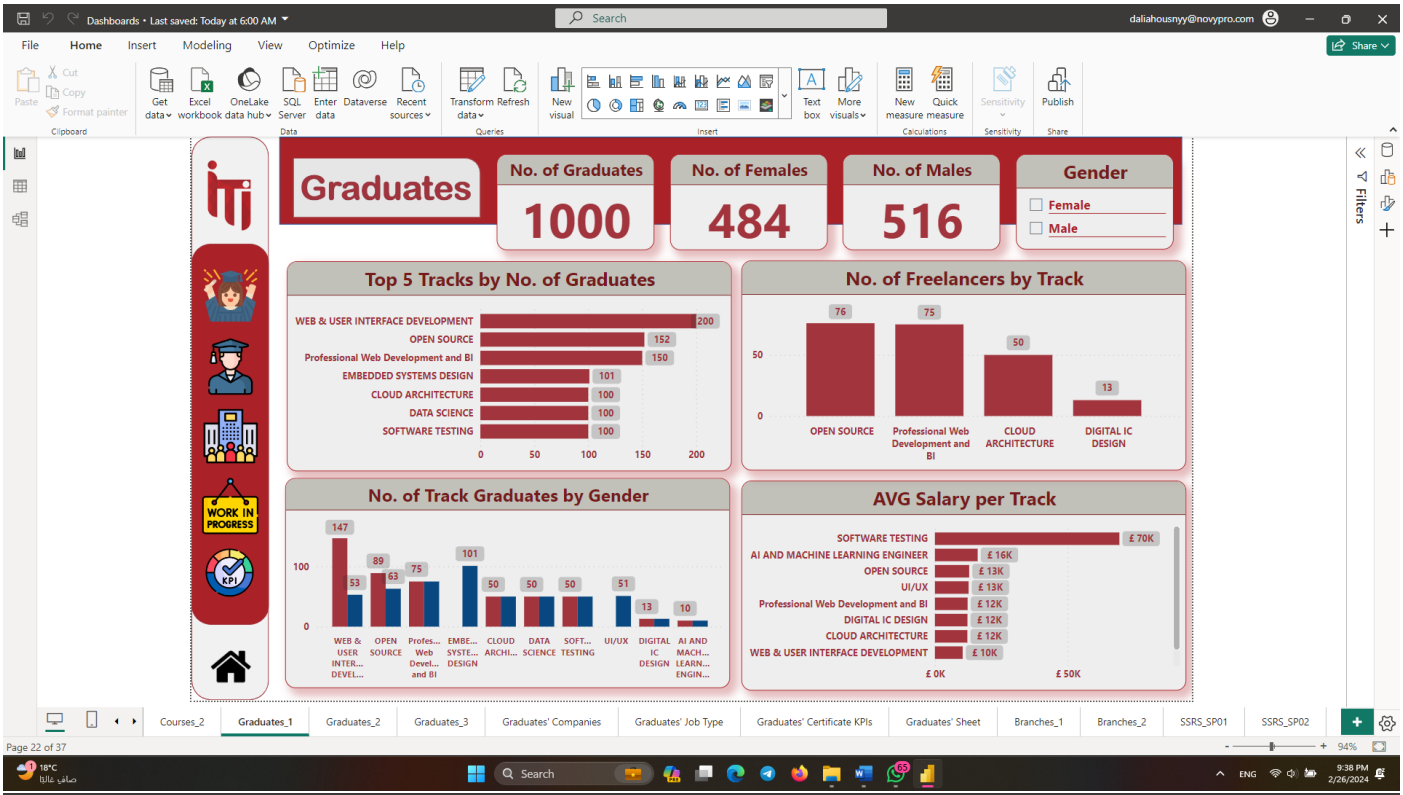
Manager	Gender	Course	Branch
Ahmed Abdelrahman	Female	Adobe Photoshop Mastery	Smart Village
Amr Hassan	Female	AI in Retail	New Capital I
Amr Said	Female	Exploratory Data Analysis (EDA)	Mansoura ITI
Dina Said	Female	3D Modeling and Animation	Asyut ITI
Fatma Ali	Female	Adobe Illustrator Mastery	Alexandria ITI
Hoda Mohamed	Female	Business Intelligence Fundamentals	Qena ITI
Mahmoud Mohamed	Male	2D Animation	Aswan ITI
Mohamed Gaber	Male	Adobe Premiere Pro Mastery	Cairo Universi
Nadia Saeed	Female	Blockchain Security Principles	Fayoum ITI

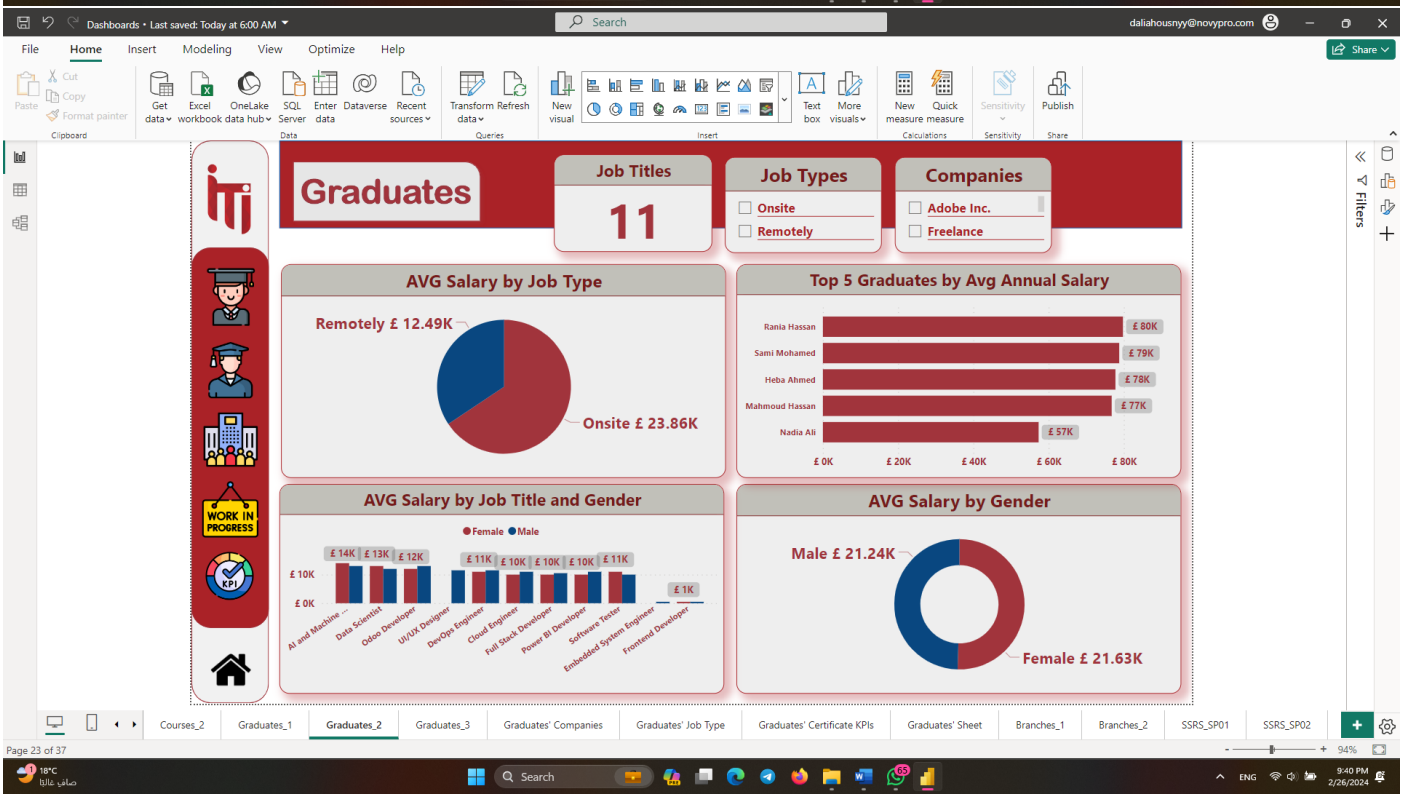
7. Courses

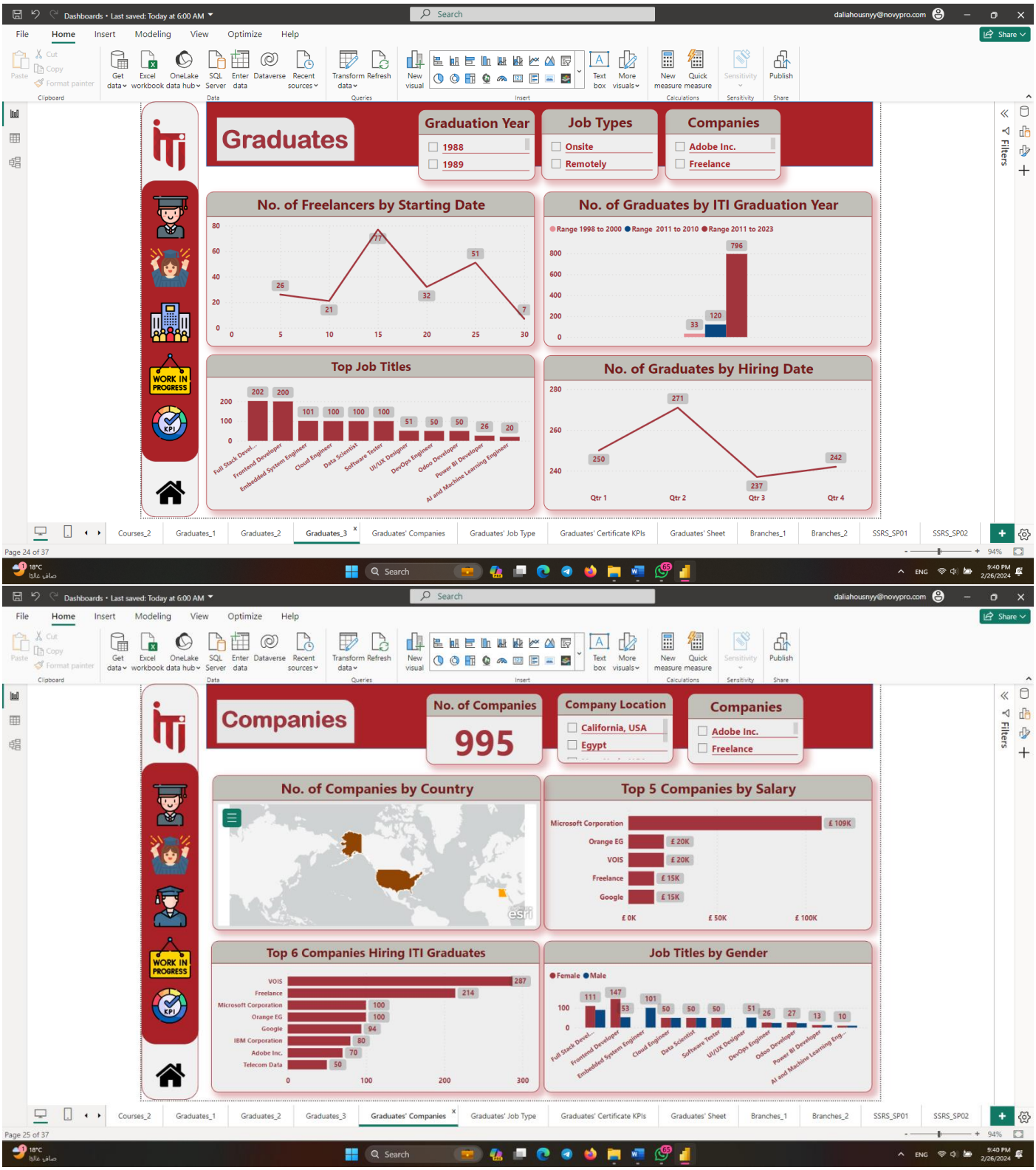

Courses
Courses
166
Topics
363
Branches
Alexandria ITI
Aswan ITI
Evaluation Type
Assignments
Final Exam
No. of Courses per Branch
57 51 51 48 48 41 40 38 37 35 34 34 33 22
Fayoum ITI
Beni Suef ITI
Smart Village ITI
Asyut ITI
Cairo University ITI
New Capital ITI
Alexandria ITI
Qena ITI
Sohag ITI
Minya ITI
Ismailia ITI
Menofia ITI
Aswan ITI
Mansoura ITI
No. of Courses by Duration
14 11 7 6 4
6 weeks
4 weeks
8 weeks
5 weeks
7 weeks
No. of Courses by Evaluation Type
Assignments
180
Final Exam 203
Projects 186
No. of Instructors per Course
Introduction to Data Engineering 8
Computer Vision Applications 7
Explainable AI Techniques 7
Microservices Architecture 6
Cloud Computing Fundamentals 5
Introduction to Data Analysis 5
AI in Robotics 4
AI-driven Chatbots Development 4
Filters
Students' Certificates_1
Students' Certificates_2
Certificate Providers ToolTip
Students' Freelance
Instructors_1
Instructors_2
Instructors_3
Instructors_4
Courses_1
Courses_2
Graduates_1
Page 20 of 37


Courses
Course
2D Animation
3D Modeling and ...
No. of Topics per Course
Introduction to Data Engineering 8
Computer Vision Applications 7
Explainable AI Techniques 7
Microservices Architecture 6
Cloud Computing Fundamentals 5
Introduction to Data Analysis 5
AI in Robotics 4
AI-driven Chatbots Development 4
AVG Course Duration per Track
COMPUTER AIDED DESIGN 7
DIGITAL IC DESIGN 7
DATA Analysis 6
3D ART 6
INDUSTRIAL AUTOMATION 6
2D GRAPHICS DESIGN 6
ARCHITECTURE, ENGINEERING AND CONSTRUCTIO... 5
INTERNET OF THINGS APPLICATIONS DEVELOPMENT 4
No. of Courses by Delivery Method
Offline 278
Online 291
Instructors and Courses Matrix
Instructor
Course Duration
Course Status
Ahmed Abdelrahman
Introduction to Data Analysis
Data Exploration 4 weeks Online
Data Preprocessing 4 weeks Online
Data Visualization 4 weeks Online
Introduction to Data Analysis Techniques 4 weeks Online
Summary Statistics 4 weeks Online
Ahmed Ali 6 weeks Online
Ahmed Hassan 6 weeks Offline
Filters
Students' Certificates_1
Students' Certificates_2
Certificate Providers ToolTip
Students' Freelance
Instructors_1
Instructors_2
Instructors_3
Instructors_4
Courses_1
Courses_2
Graduates_1
Page 21 of 37

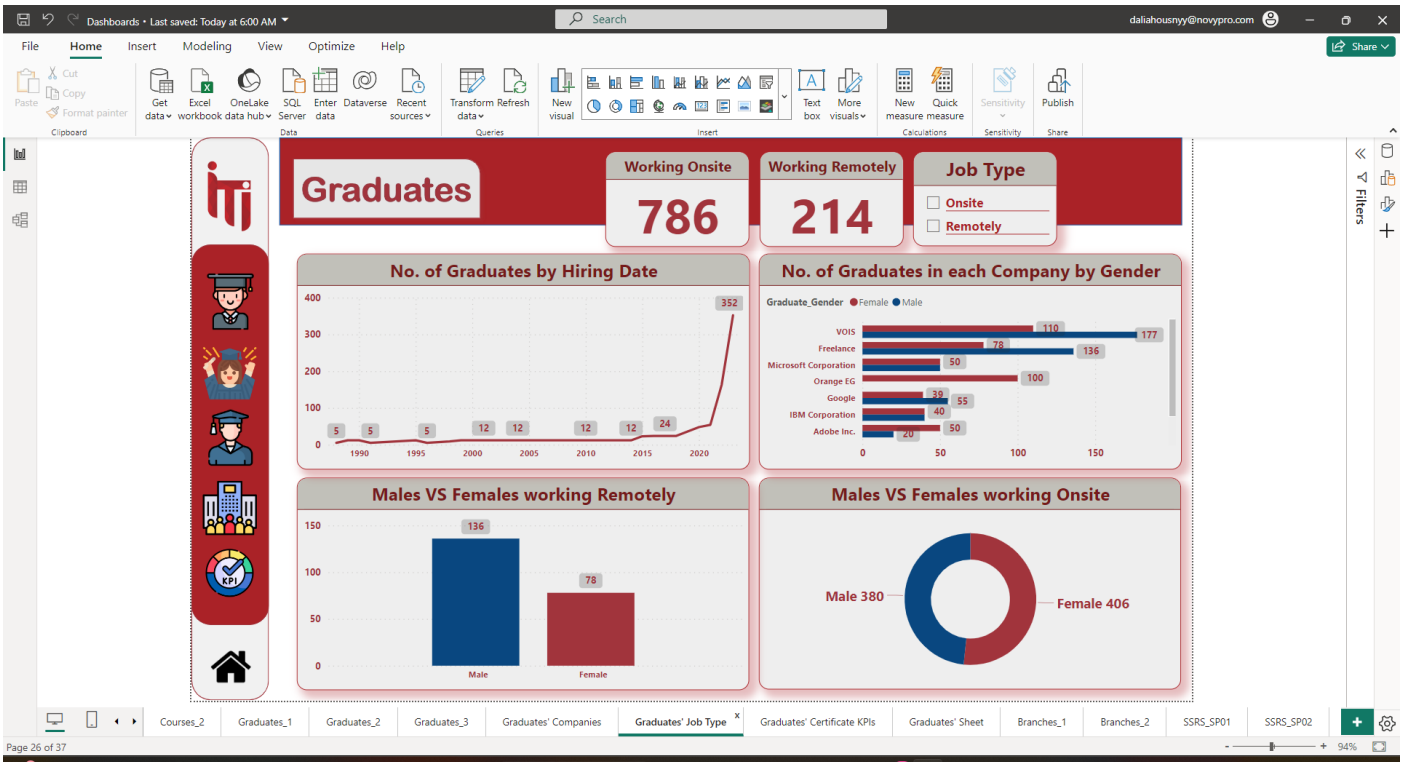
8. Graduates

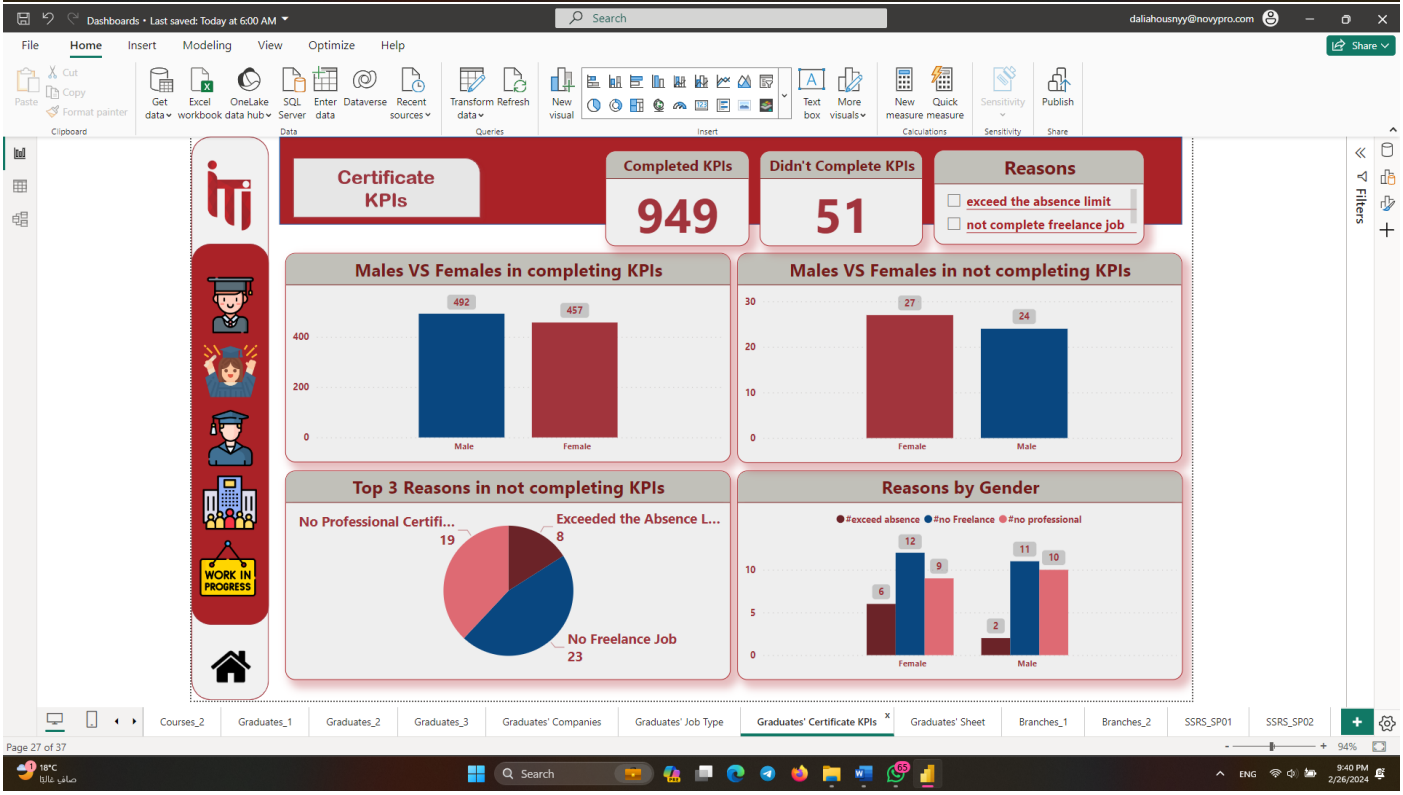

Graduates
No. of Graduates
1000
No. of Females
484
No. of Males
516
Gender
Female
Male
Top 5 Tracks by No. of Graduates
WEB & USER INTERFACE DEVELOPMENT 200
OPEN SOURCE 152
Professional Web Development and BI 150
EMBEDDED SYSTEMS DESIGN 101
CLOUD ARCHITECTURE 100
DATA SCIENCE 100
SOFTWARE TESTING 100
No. of Freelancers by Track
OPEN SOURCE 76
Professional Web Development and BI 75
CLOUD ARCHITECTURE 50
DIGITAL IC DESIGN 13
No. of Track Graduates by Gender
AVG Salary per Track
SOFTWARE TESTING £ 70K
AI AND MACHINE LEARNING ENGINEER £ 16K
OPEN SOURCE £ 13K
UI/UX £ 13K
Professional Web Development and BI £ 12K
DIGITAL IC DESIGN £ 12K
CLOUD ARCHITECTURE £ 12K
WEB & USER INTERFACE DEVELOPMENT £ 10K
Courses_2
Graduates_1
Graduates_2
Graduates_3
Graduates' Companies
Graduates' Job Type
Graduates' Certificate KPIs
Graduates' Sheet
Branches_1
Branches_2
SSRS_SP01
SSRS_SP02

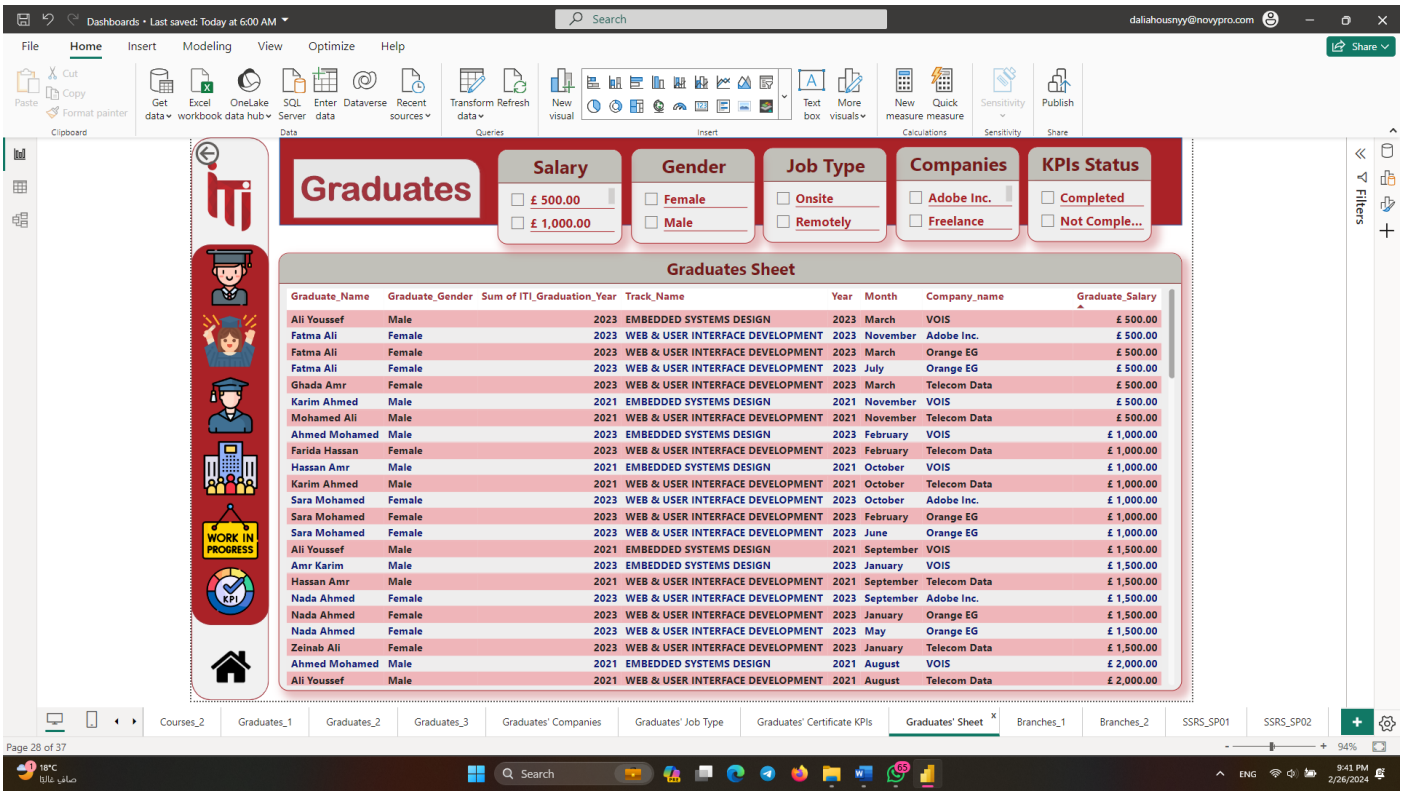

Graduates
Job Titles
11
Job Types
Onsite
Remotely
Companies
Adobe Inc.
Freelance
AVG Salary by Job Type
Remotely £ 12.49K
Onsite £ 23.86K
Top 5 Graduates by Avg Annual Salary
Rania Hassan £ 80K
Sami Mohamed £ 79K
Heba Ahmed £ 78K
Mahmoud Hassan £ 77K
Nadia Ali £ 57K
AVG Salary by Job Title and Gender
Female Male
AVG Salary by Gender
Male £ 21.24K
Female £ 21.63K
Courses_2
Graduates_1
Graduates_2
Graduates_3
Graduates' Companies
Graduates' Job Type
Graduates' Certificate KPIs
Graduates' Sheet
Branches_1
Branches_2
SSRS_SP01
SSRS_SP02


Graduates
Graduation Year
1988
1989
Job Types
Onsite
Remotely
Companies
Adobe Inc.
Freelance
No. of Freelancers by Starting Date
No. of Graduates by ITI Graduation Year
Range 1998 to 2000
Range 2011 to 2010
Range 2011 to 2023
Top Job Titles
No. of Graduates by Hiring Date
Qtr 1
Qtr 2
Qtr 3
Qtr 4
Courses_2
Graduates_1
Graduates_2
Graduates_3
Graduates' Companies
Graduates' Job Type
Graduates' Certificate KPIs
Graduates' Sheet
Branches_1
Branches_2
SSRS_SP01
SSRS_SP02
Page 24 of 37

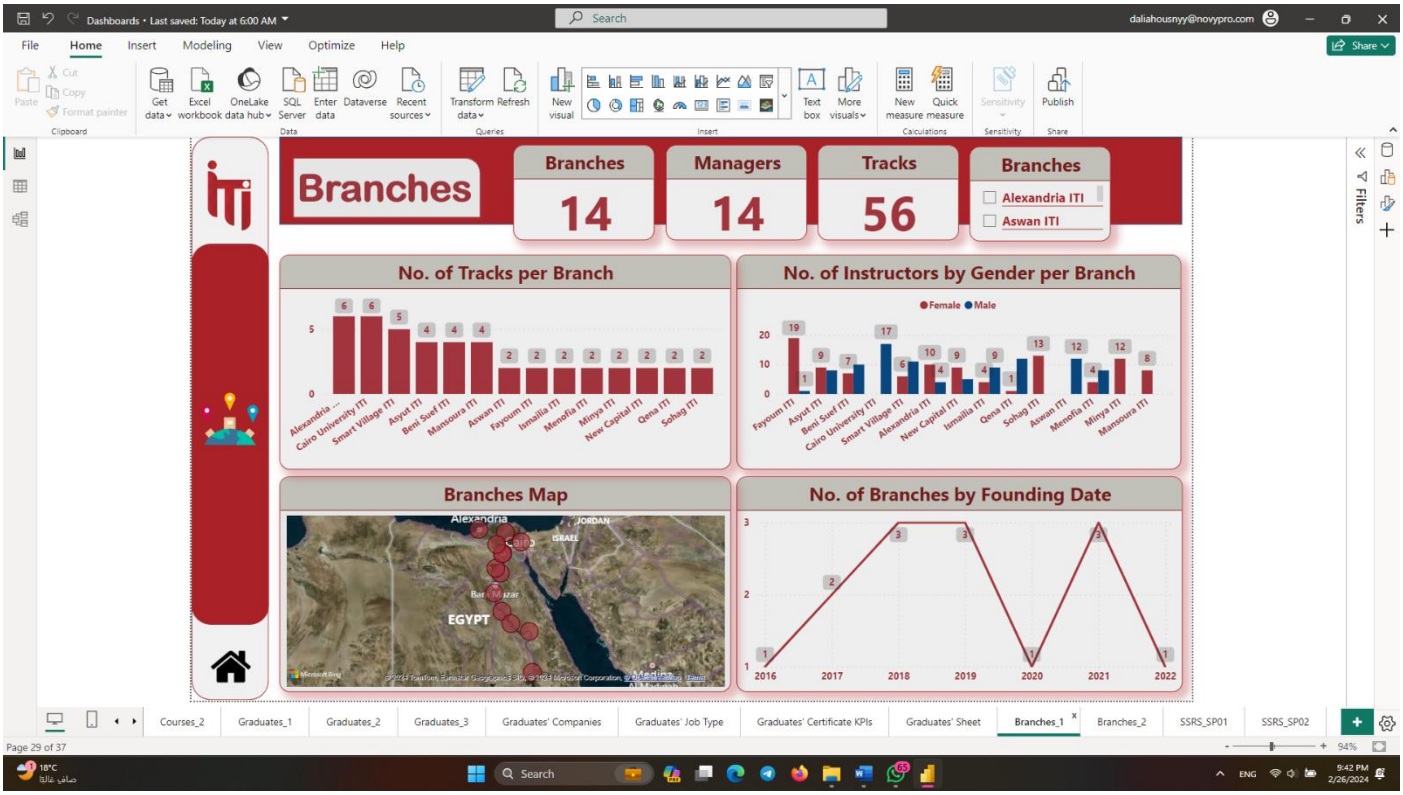
Companies
No. of Companies
995
Company Location
California, USA
Egypt
Companies
Adobe Inc.
Freelance
No. of Companies by Country
Top 5 Companies by Salary
Microsoft Corporation
Orange EG
VOIS
Freelance
Google
Top 6 Companies Hiring ITI Graduates
Job Titles by Gender
Female
Male
Courses_2
Graduates_1
Graduates_2
Graduates_3
Graduates' Companies
Graduates' Job Type
Graduates' Certificate KPIs
Graduates' Sheet
Branches_1
Branches_2
SSRS_SP01
SSRS_SP02
Page 25 of 37

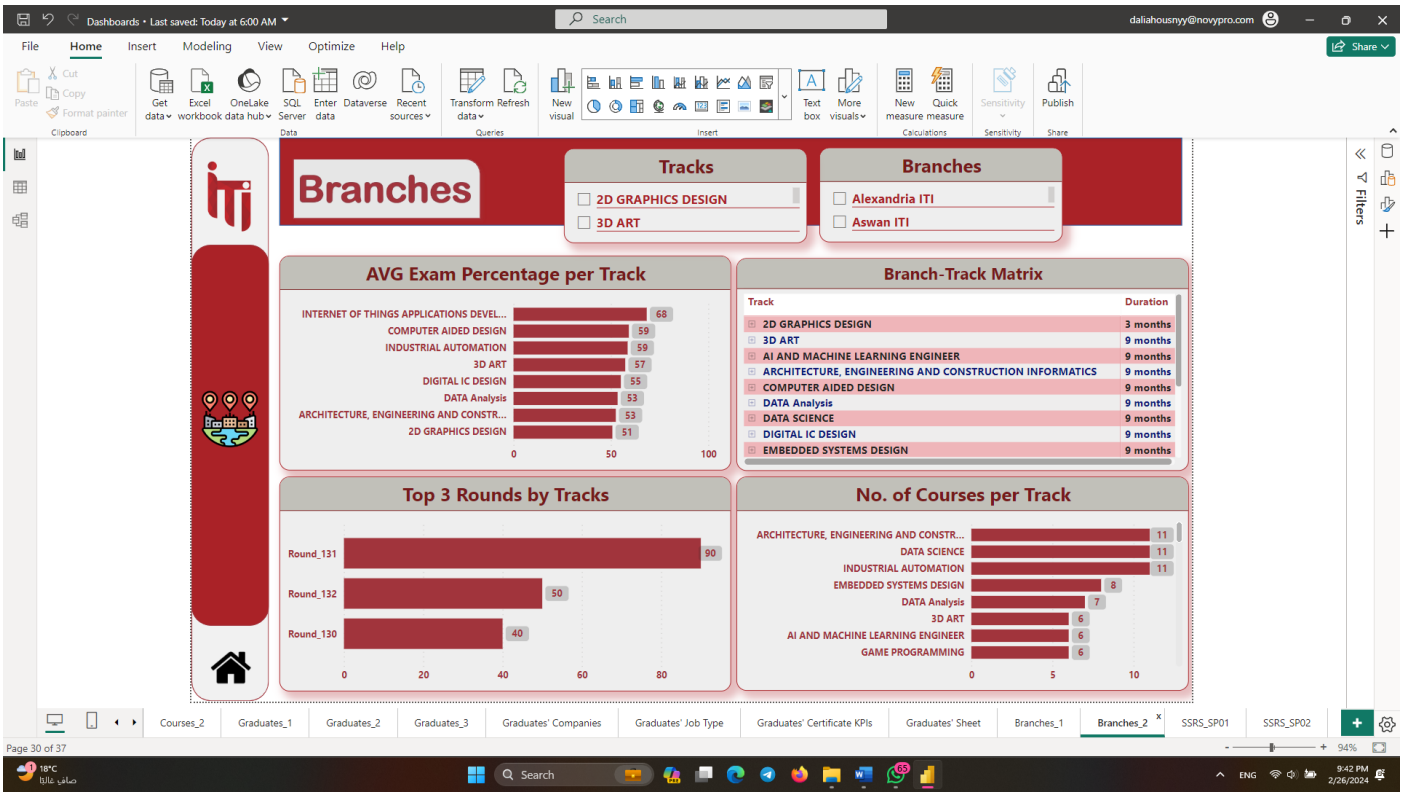

Graduates
Working Onsite
786
Working Remotely
214
Job Type
Onsite
Remotely
No. of Graduates by Hiring Date
No. of Graduates in each Company by Gender
Graduate_Gender Female Male
VOIS
Freelance
Microsoft Corporation
Orange EG
Google
IBM Corporation
Adobe Inc.
Males VS Females working Remotely
Male
Female
Males VS Females working Onsite
Male 380
Female 406


Certificate KPIs
Completed KPIs
949
Didn't Complete KPIs
51
Reasons
exceed the absence limit
not complete freelance job
Males VS Females in completing KPIs
Males VS Females in not completing KPIs
Top 3 Reasons in not completing KPIs
No Professional Certifi...
19
Exceeded the Absence L...
8
No Freelance Job
23
Reasons by Gender
#exceed absence
#no Freelance
#no professional
WORK IN PROGRESS

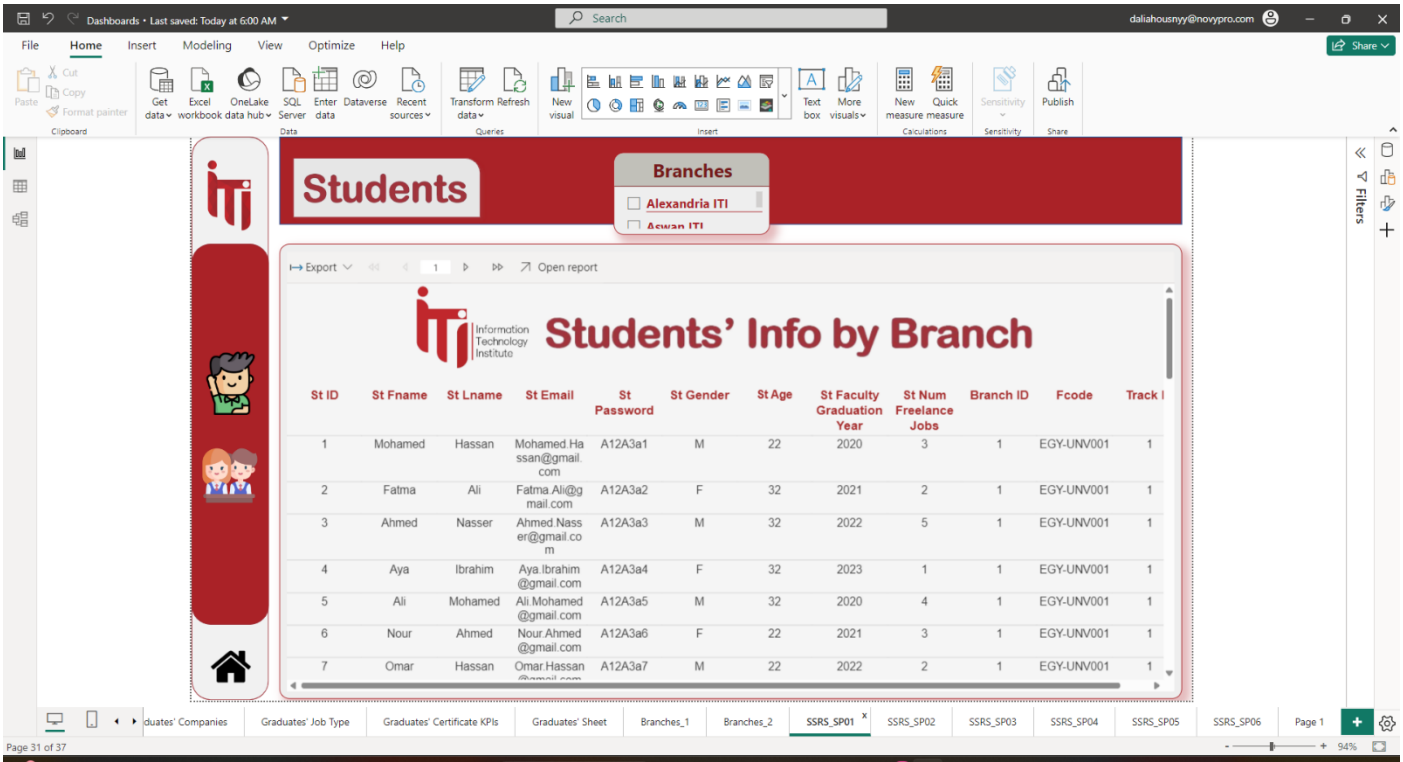

Graduates
Salary
£ 500.00
£ 1,000.00
Gender
Female
Male
Job Type
Onsite
Remotely
Companies
Adobe Inc.
Freelance
KPIs Status
Completed
Not Comple...
Graduates Sheet
Graduate_Name | Graduate_Gender | Sum of ITI_Graduation_Year | Track_Name | Year | Month | Company_name | Graduate_Salary
Ali Youssef | Male | 2023 | EMBEDDED SYSTEMS DESIGN | 2023 | March | VOIS | £ 500.00
Fatma Ali | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | November | Adobe Inc. | £ 500.00
Fatma Ali | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | March | Orange EG | £ 500.00
Fatma Ali | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | July | Orange EG | £ 500.00
Ghada Amr | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | March | Telecom Data | £ 500.00
Karim Ahmed | Male | 2021 | EMBEDDED SYSTEMS DESIGN | 2021 | November | VOIS | £ 500.00
Mohamed Ali | Male | 2021 | WEB & USER INTERFACE DEVELOPMENT | 2021 | November | Telecom Data | £ 500.00
Ahmed Mohamed | Male | 2023 | EMBEDDED SYSTEMS DESIGN | 2023 | February | VOIS | £ 1,000.00
Farida Hassan | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | February | Telecom Data | £ 1,000.00
Hassan Amr | Male | 2021 | EMBEDDED SYSTEMS DESIGN | 2021 | October | VOIS | £ 1,000.00
Karim Ahmed | Male | 2021 | WEB & USER INTERFACE DEVELOPMENT | 2021 | October | Telecom Data | £ 1,000.00
Sara Mohamed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | October | Adobe Inc. | £ 1,000.00
Sara Mohamed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | February | Orange EG | £ 1,000.00
Sara Mohamed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | June | Orange EG | £ 1,000.00
Ali Youssef | Male | 2021 | EMBEDDED SYSTEMS DESIGN | 2021 | September | VOIS | £ 1,500.00
Amr Karim | Male | 2023 | EMBEDDED SYSTEMS DESIGN | 2023 | January | VOIS | £ 1,500.00
Hassan Amr | Male | 2021 | WEB & USER INTERFACE DEVELOPMENT | 2021 | September | Telecom Data | £ 1,500.00
Nada Ahmed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | September | Adobe Inc. | £ 1,500.00
Nada Ahmed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | January | Orange EG | £ 1,500.00
Nada Ahmed | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | May | Orange EG | £ 1,500.00
Zeinab Ali | Female | 2023 | WEB & USER INTERFACE DEVELOPMENT | 2023 | January | Telecom Data | £ 1,500.00
Ahmed Mohamed | Male | 2021 | EMBEDDED SYSTEMS DESIGN | 2021 | August | VOIS | £ 2,000.00
Ali Youssef | Male | 2021 | WEB & USER INTERFACE DEVELOPMENT | 2021 | August | Telecom Data | £ 2,000.00
Courses_2
Graduates_1
Graduates_2
Graduates_3
Graduates' Companies
Graduates' Job Type
Graduates' Certificate KPIs
Graduates' Sheet
Branches_1
Branches_2
SSRS_SP01
SSRS_SP02
Page 28 of 37

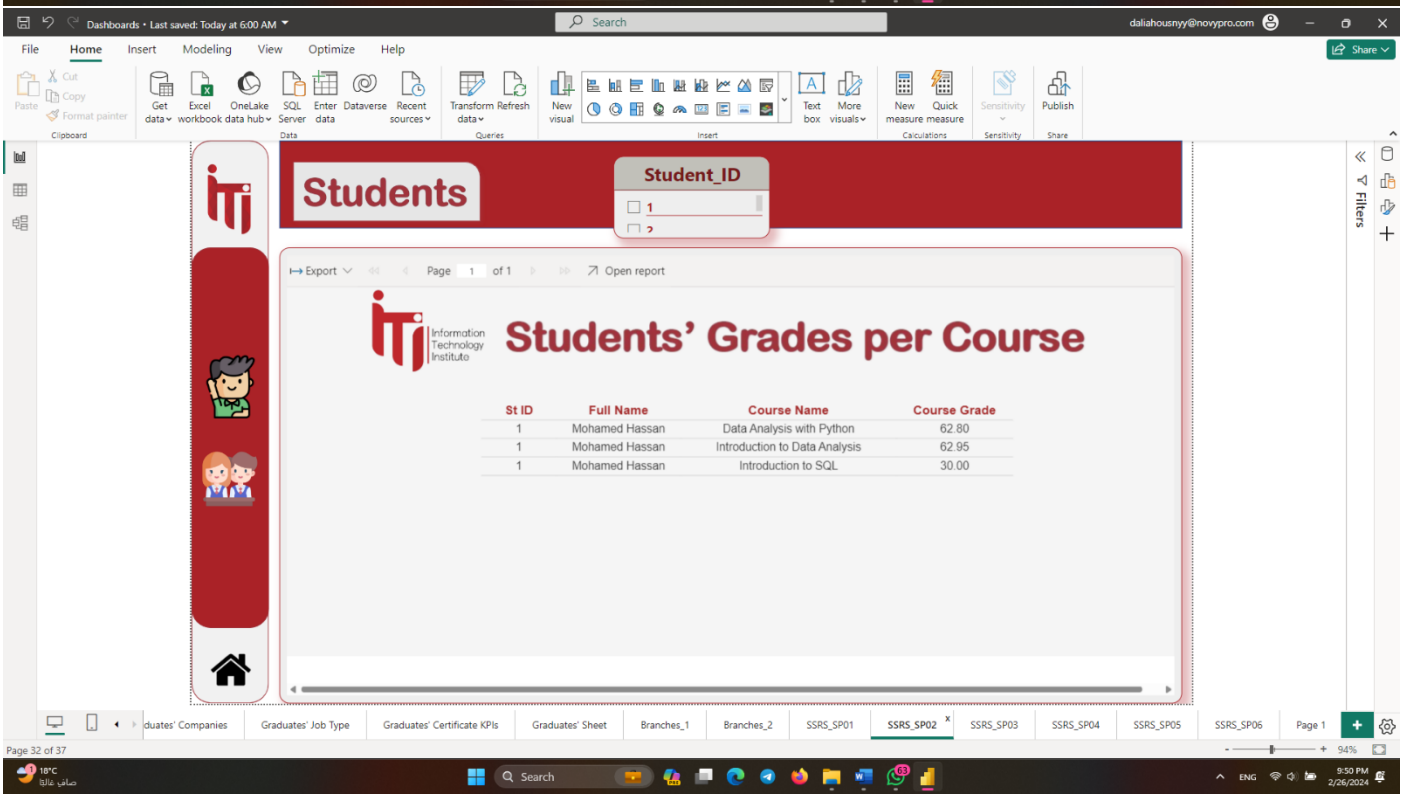
9. Branches

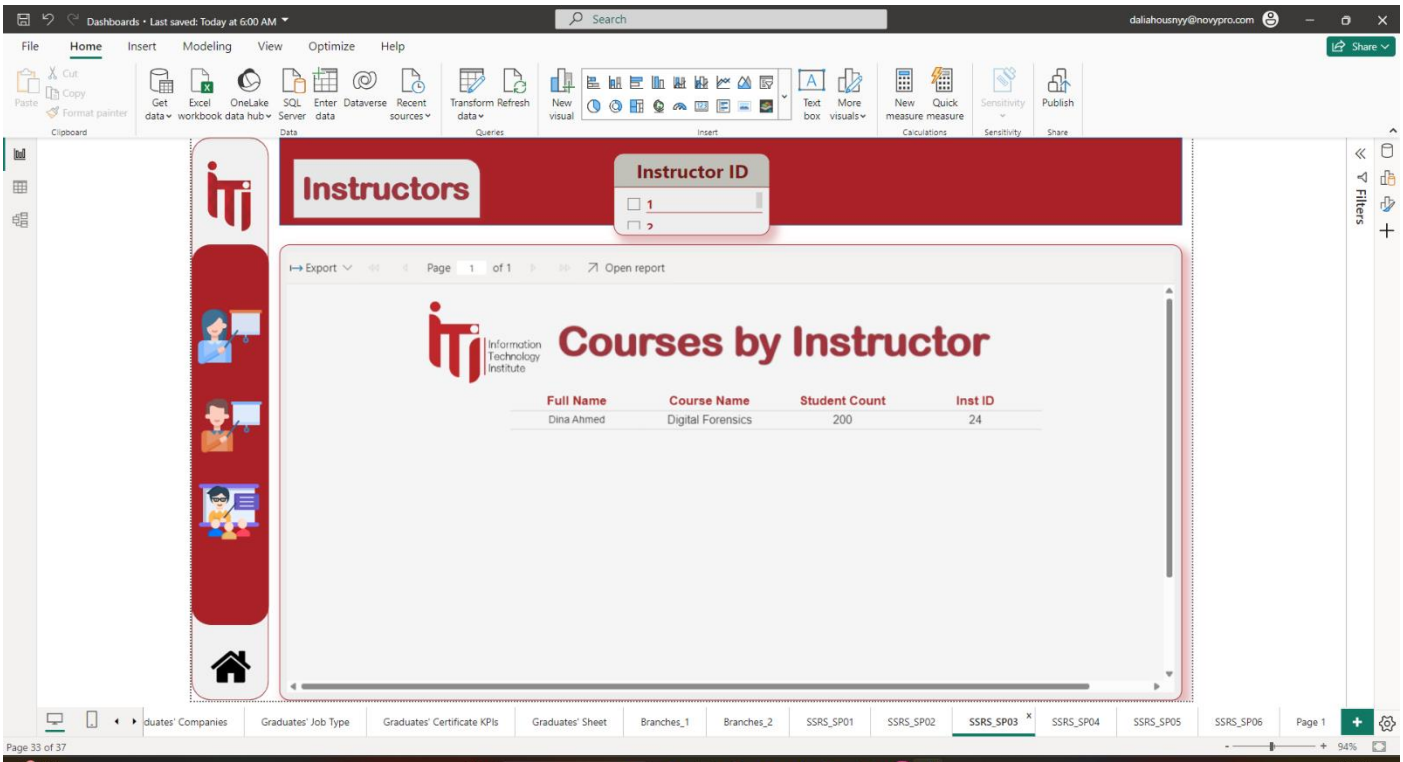

Branches
Branches
14
Managers
14
Tracks
56
Branches
Alexandria ITI
Aswan ITI
No. of Tracks per Branch
No. of Instructors by Gender per Branch
Female
Male
Branches Map
EGYPT
No. of Branches by Founding Date
2016
2017
2018
2019
2020
2021
2022

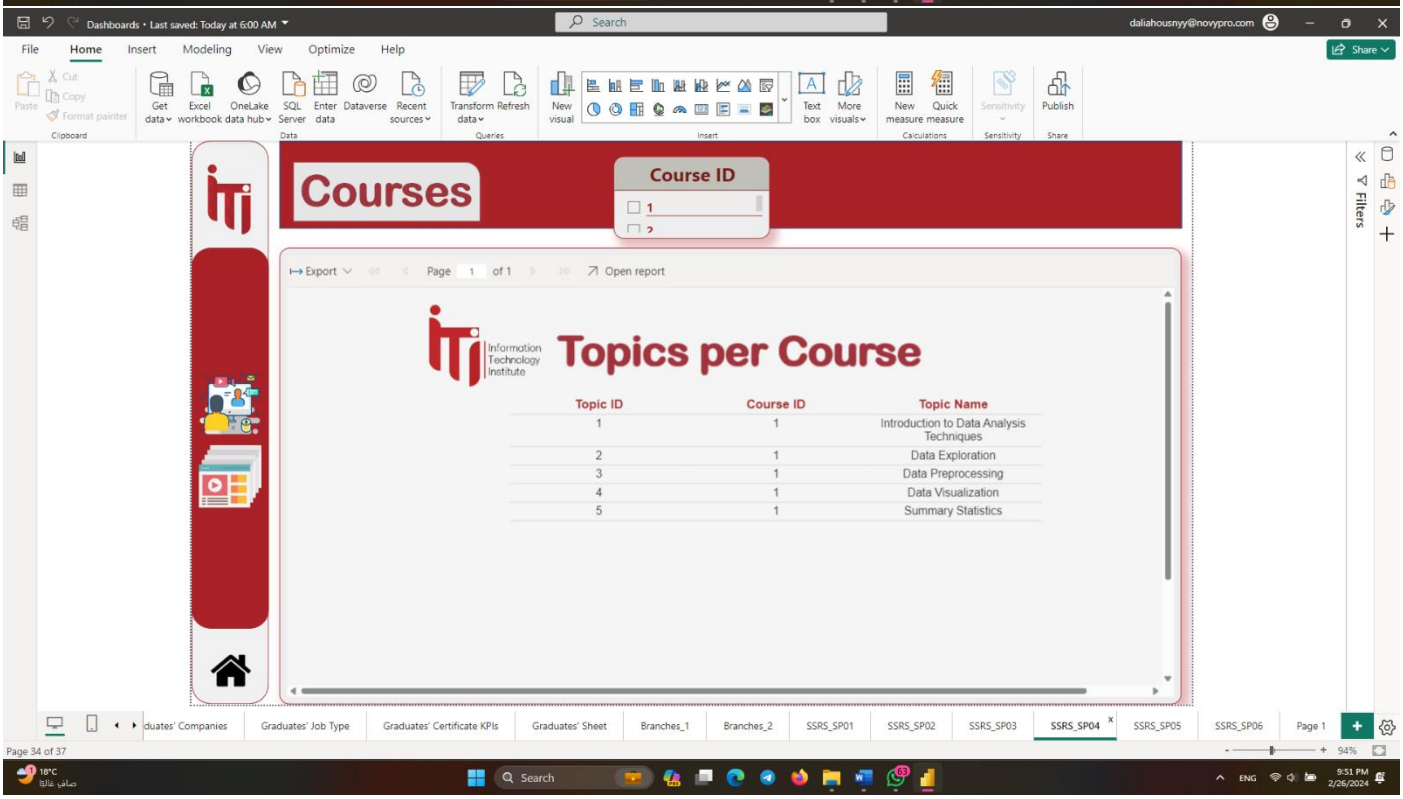

Branches
Tracks
2D GRAPHICS DESIGN
3D ART
Branches
Alexandria ITI
Aswan ITI
AVG Exam Percentage per Track
INTERNET OF THINGS APPLICATIONS DEVEL... 68
COMPUTER AIDED DESIGN 59
INDUSTRIAL AUTOMATION 59
3D ART 57
DIGITAL IC DESIGN 55
DATA Analysis 53
ARCHITECTURE, ENGINEERING AND CONSTR... 53
2D GRAPHICS DESIGN 51
Branch-Track Matrix
Track
Duration
2D GRAPHICS DESIGN 3 months
3D ART 9 months
AI AND MACHINE LEARNING ENGINEER 9 months
ARCHITECTURE, ENGINEERING AND CONSTRUCTION INFORMATICS 9 months
COMPUTER AIDED DESIGN 9 months
DATA Analysis 9 months
DATA SCIENCE 9 months
DIGITAL IC DESIGN 9 months
EMBEDDED SYSTEMS DESIGN 9 months
Top 3 Rounds by Tracks
Round_131 90
Round_132 50
Round_130 40
No. of Courses per Track
ARCHITECTURE, ENGINEERING AND CONSTR... 11
DATA SCIENCE 11
INDUSTRIAL AUTOMATION 11
EMBEDDED SYSTEMS DESIGN 8
DATA Analysis 7
3D ART 6
AI AND MACHINE LEARNING ENGINEER 6
GAME PROGRAMMING 6

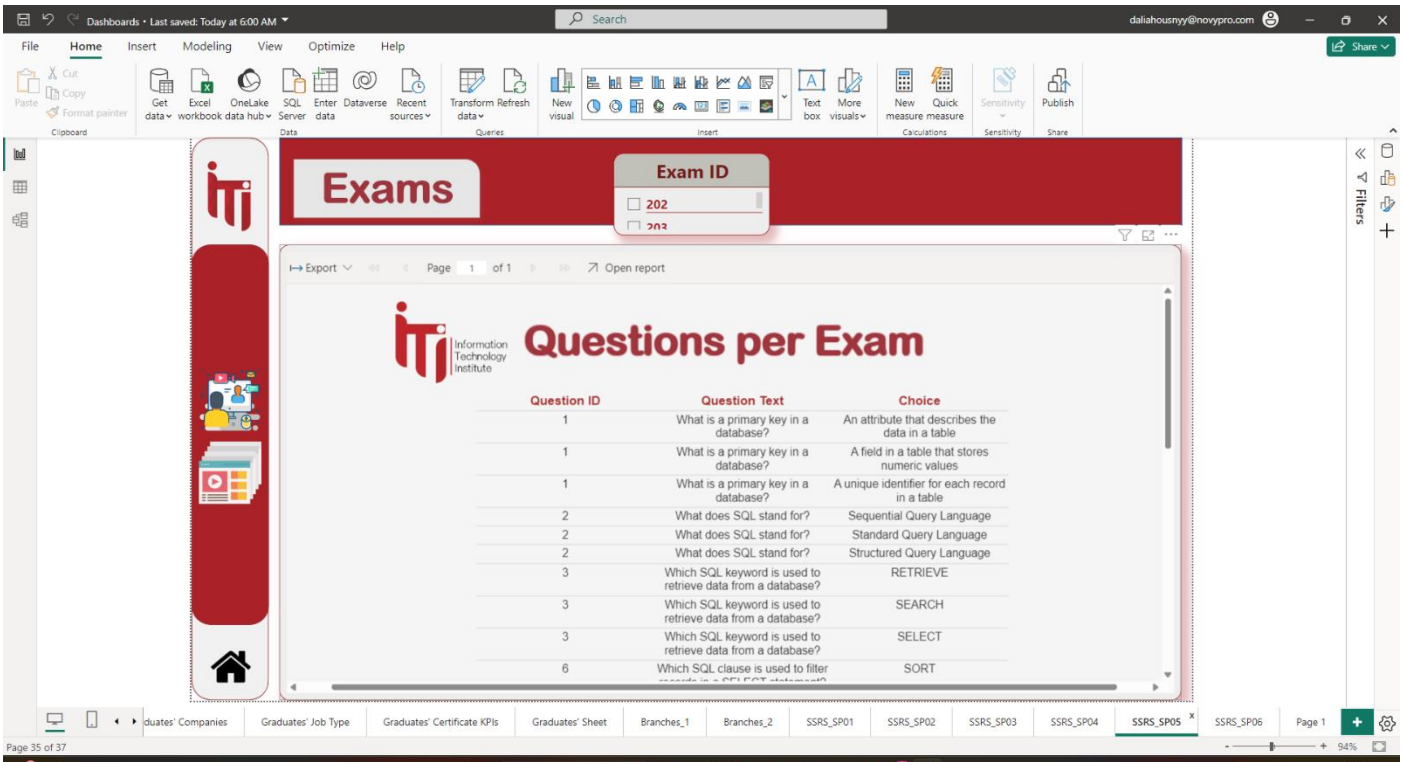
10. SSRS Paginated Reports

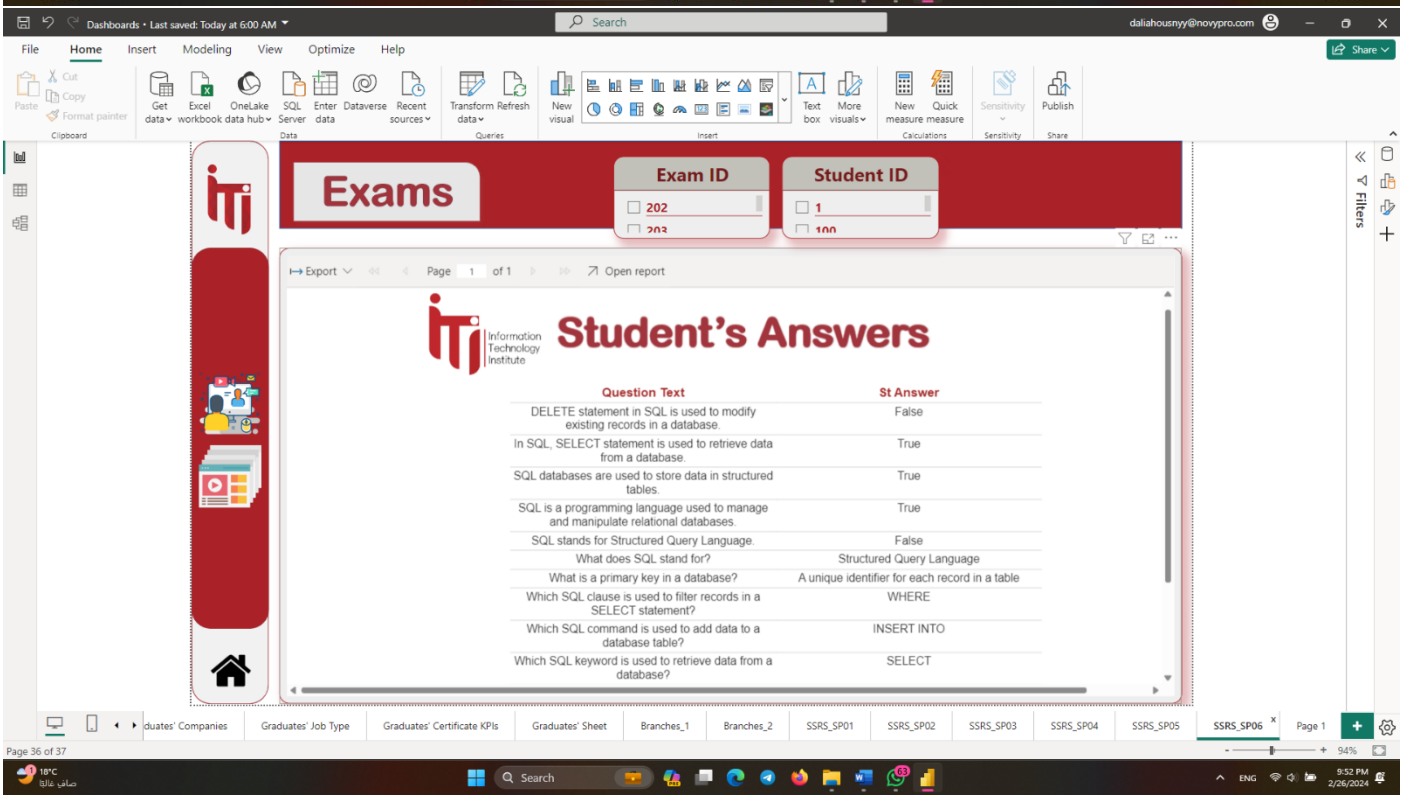




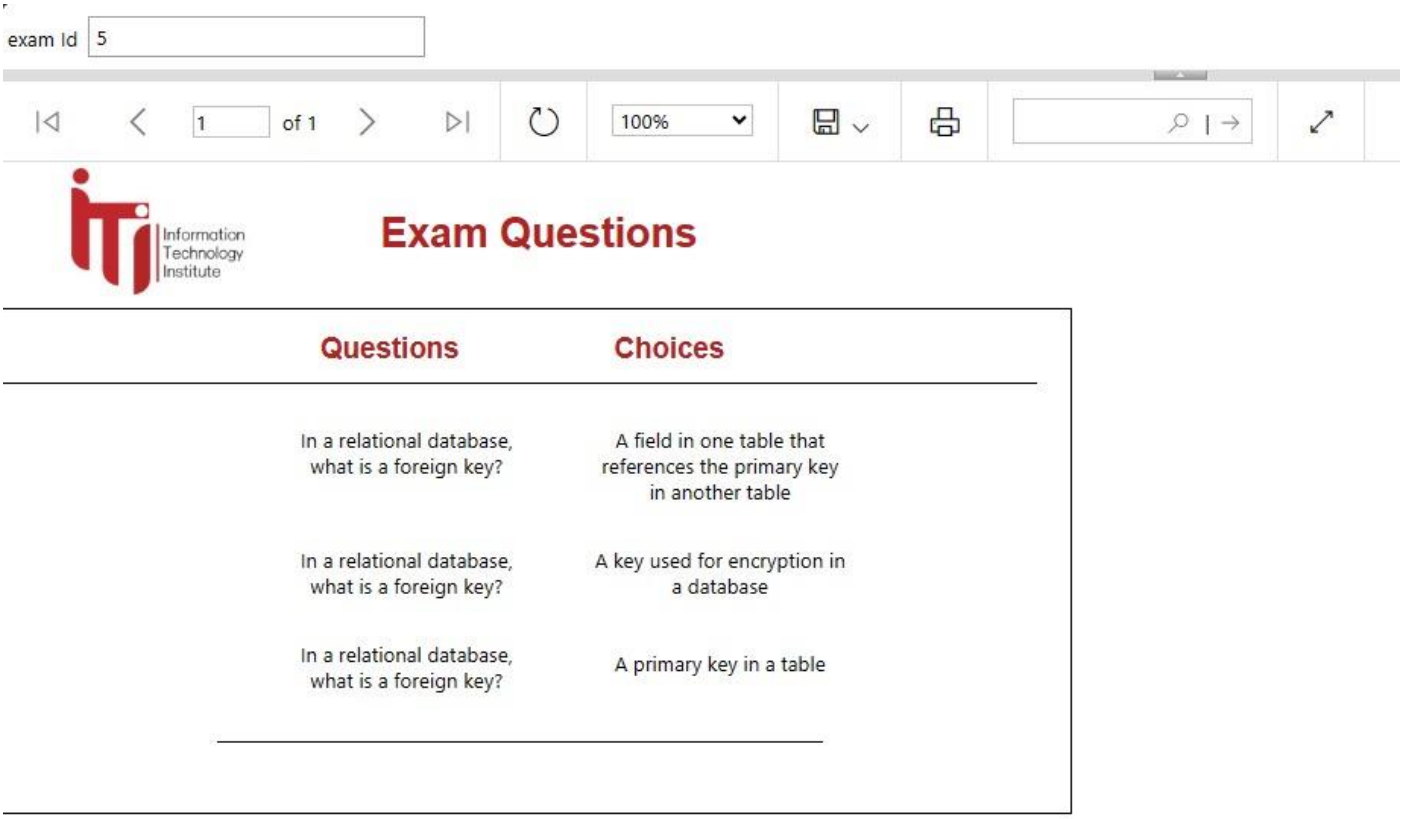

Instructors
Instructor ID
1
2
Export
Page 1 of 1
Open report
Information Technology Institute
Courses by Instructor
Full Name
Course Name
Student Count
Inst ID
Dina Ahmed
Digital Forensics
200
24
Filters
SSRS_SP03
Page 33 of 37


Courses
Course ID
1
2
Export
Page 1 of 1
Open report
Information Technology Institute
Topics per Course
Topic ID
Course ID
Topic Name
1
1
Introduction to Data Analysis Techniques
2
1
Data Exploration
3
1
Data Preprocessing
4
1
Data Visualization
5
1
Summary Statistics
Filters
SSRS_SP04
Page 34 of 37


Exams
Exam ID
202
Export
Page 1 of 1
Open report
Information Technology Institute
Questions per Exam
Question ID | Question Text | Choice
1 | What is a primary key in a database? | An attribute that describes the data in a table
1 | What is a primary key in a database? | A field in a table that stores numeric values
1 | What is a primary key in a database? | A unique identifier for each record in a table
2 | What does SQL stand for? | Sequential Query Language
2 | What does SQL stand for? | Standard Query Language
2 | What does SQL stand for? | Structured Query Language
3 | Which SQL keyword is used to retrieve data from a database? | RETRIEVE
3 | Which SQL keyword is used to retrieve data from a database? | SEARCH
3 | Which SQL keyword is used to retrieve data from a database? | SELECT
6 | Which SQL clause is used to filter | SORT


Exams
Exam ID
202
Student ID
1
Export
Page 1 of 1
Open report
Information Technology Institute
Student's Answers
Question Text | St Answer
DELETE statement in SQL is used to modify existing records in a database. | False
In SQL, SELECT statement is used to retrieve data from a database. | True
SQL databases are used to store data in structured tables. | True
SQL is a programming language used to manage and manipulate relational databases. | True
SQL stands for Structured Query Language. | False
What does SQL stand for? | Structured Query Language
What is a primary key in a database? | A unique identifier for each record in a table
Which SQL clause is used to filter records in a SELECT statement? | WHERE
Which SQL command is used to add data to a database table? | INSERT INTO
Which SQL keyword is used to retrieve data from a database? | SELECT

11. Freeform SSRS Report



12. Student Facebook Report