ITI Examination System

By:

Ahmed Mostafa

Amany Saeed

Amir El Sayed

Bassant Samy

Mohamed Fatouh

Content

1. Summary

The ITI Examination System is designed to manage all aspects of student enrollment, coursework, assessment, and graduation at the Information Technology Institute (ITI) across its 21 branches in Egypt. The system supports two program types—the 9-month Professional Training Program (PTP) and the 4-month Intensive Code Camp (ICC)—each following its own batch schedule. Students enroll in tracks composed of multiple courses, take branch-specific exams, and must fulfill requirements including freelancing engagements, certifications, and a capstone project to graduate. Instructors deliver courses, supervise tracks, and set exams; branch managers oversee operations. The application will cover the end-to-end lifecycle from user management through reporting.

2. Conceptual Schema

Below is a high-level ER diagram description capturing entities, key attributes, and relationships:

Entities

- **Manager** (*ID*, SSN, Name, DOB, Phone, Email, Gender)
- **Branch** (*ID*, Name, Location, Email, Phone)
- **Program** (*ID*, Name, Type)
- **Batch** (*ID*, Name, StartDate, EndDate)
- **Track** (*ID*, Name, Description)
- **Course** (*ID*, Name, StartDate, EndDate)
- **Topic** (*ID*, Name, Description)
- **Instructor** (*ID*, SSN, FirstName, LastName, Email, DOB, Gender, Phone, Address, Degree, Salary)
- **Student** (*ID*, SSN, FirstName, LastName, DOB, Gender, Phone, Address, Email, Faculty, Status, LinkedInURL)
- **Company** (*ID*, Name, Type, Location)
- **Freelance** (*ID*, Field, Platform, Income, Date)
- **Certificate** (*ID*, Name, Field, Source, Cost, Date)
- **GraduationProject** (*ID*, Name, Score)
- **Exam** (*ID*, Name, StartDate, EndDate)
- **Question** (*ID*, Text, Type, Level, Mark)
- **Choice** (*ID*, Text, IsCorrect)

Key Relationships

1. **Manage**: Manager ↔ Branch (1:1, both total, attribute: HiredDate)
2. **Has**: Branch ↔ Program (M:N, both total)
3. **In**: Branch ↔ Student (1:M, both total)
4. **Has**: Branch ↔ Exam (1:M, total on Exam)
5. **Has**: Program ↔ Batch (1:M, both total)
6. **Has**: Batch ↔ Student (1:M, both total)
7. **Has**: Program ↔ Track (1:M, both total)
8. **Supervise**: Track ↔ Instructor (1:1, total on Track)
9. **Has**: Track ↔ Student (1:M, both total)
10. **Hired**: Company ↔ Student (1:M, partial, attributes: HiredDate, Salary, Title)
11. **Work**: Company ↔ Instructor (1:M, partial, attributes: HiredDate, Salary, Title)
12. **Teach**: Instructor ↔ Student (M:N, both total)
13. **Got**: Student ↔ Freelance (M:N, both total)
14. **Got**: Student ↔ Certificate (M:N, both total)
15. **WorkOn**: Student ↔ GraduationProject (1:M, both total)
16. **Lead**: Student ↔ Student (Unary 1:M, one side total)
17. **Learn**: Student ↔ Course (M:N, both total)
18. **Instruct**: Instructor ↔ Course (M:N, both total)
19. **Include**: Course ↔ Topic (M:1, both total)
20. **Include**: Course ↔ Question (1:M, both total)
21. **Contain**: Course ↔ Exam (M:N, total on Exam)
22. **Has**: Question ↔ Choice (1:M, both total)

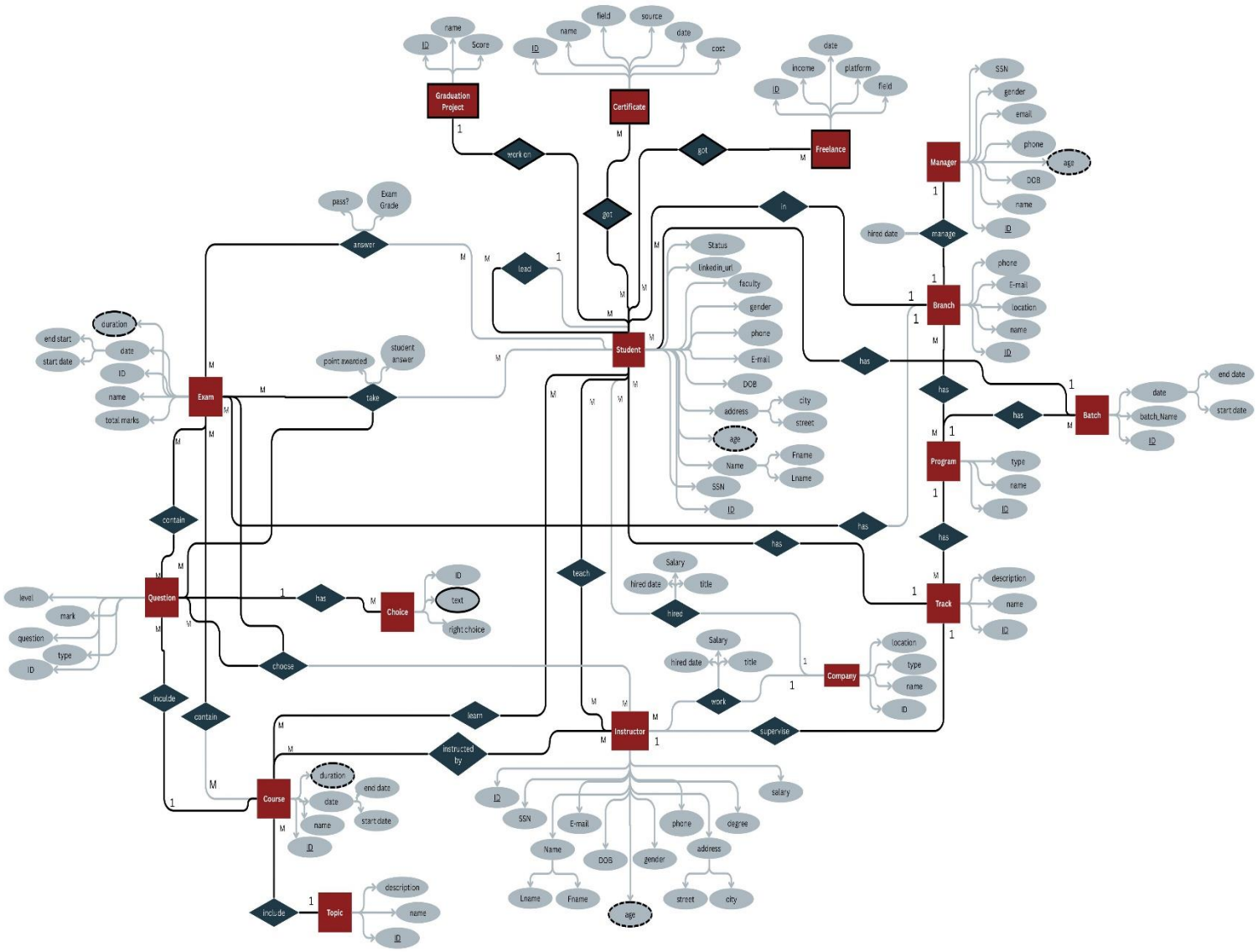
23. **Contain**: Question ↔ Exam (M:N, both total)

24. **Answer**: Student ↔ Exam (M:N, total on Exam, attributes: Grade, PassFlag)

25. **Take**: Student ↔ Exam ↔ Question (Ternary M:N, total on Exam & Question)

26. **Choose**: Instructor ↔ Exam ↔ Question (Ternary M:N, total on Exam & Question)

The Entity Relationship Diagram:



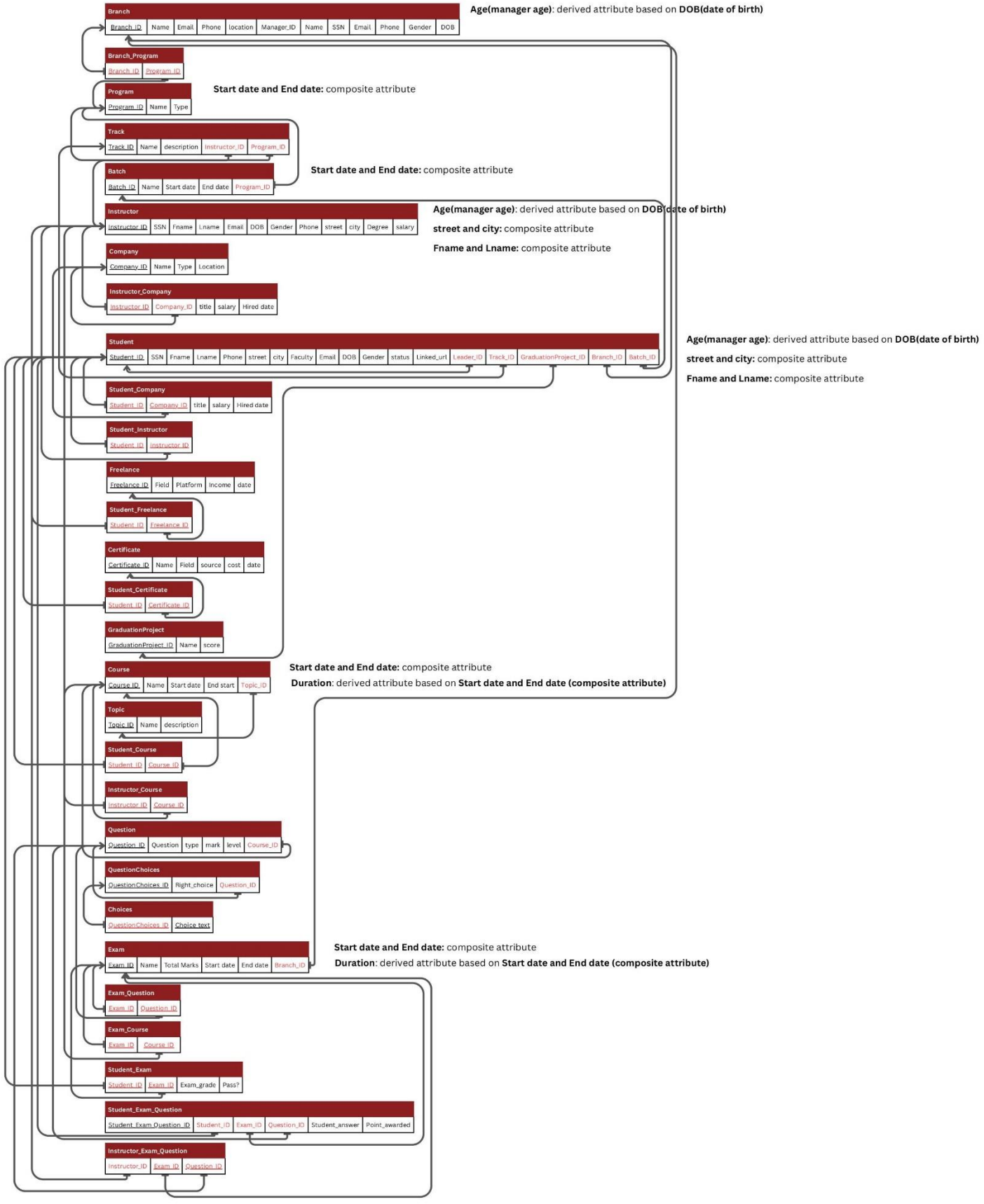
3. Mapping

This section maps conceptual entities and relationships to relational tables, primary keys (PK), foreign keys (FK), and associative tables for M:N relationships.

Table	Columns	PK	FK
Manager	ManagerID, SSN, Name, DOB, Phone, Email, Gender	ManagerID	
Branch	BranchID, Name, Location, Email, Phone	BranchID	
BranchManager	BranchID, ManagerID, HiredDate	(BranchID, ManagerID)	FK BranchID → Branch, FK ManagerID → Manager
Program	ProgramID, Name, Type	ProgramID	
BranchProgram	BranchID, ProgramID	(BranchID, ProgramID)	FK BranchID → Branch, FK ProgramID → Program
Batch	BatchID, ProgramID, Name, StartDate, EndDate	BatchID	FK ProgramID → Program
Student	StudentID, SSN, FirstName, LastName, DOB, Gender, Phone, Address, Email, Faculty, Status, LinkedInURL, BranchID, BatchID, LeaderID, InstructorID	StudentID	FK BranchID → Branch; FK BatchID → Batch; FK LeaderID → Student; FK InstructorID→Instructor
Instructor	InstructorID, SSN, FirstName, LastName, Email, DOB, Gender, Phone, Address, Degree, Salary	InstructorID	
Track	TrackID, ProgramID, Name, Description, SupervisorID	TrackID	FK ProgramID → Program FK SupervisorID → Instructor
TrackStudent	TrackID, StudentID	(TrackID, StudentID)	FK TrackID → Track FK StudentID → Student
Company	CompanyID, Name, Type, Location	CompanyID	
StudentCompany	StudentID, CompanyID, HiredDate, Salary, Title	(StudentID, CompanyID)	FKs → Student, Company
InstructorCompany	InstructorID, CompanyID, HiredDate, Salary, Title	(InstructorID, CompanyID)	FKs → Instructor, Company

StudentInstructor	StudentID, InstructorID	(StudentID, InstructorID)	FKs → Student, Instructor
Freelance	FreelanceID, Field, Platform, Income, Date	FreelanceID	
StudentFreelance	StudentID, FreelanceID	(StudentID, FreelanceID)	FKs → Student, Freelance
Certificate	CertificateID, Name, Field, Source, Cost, Date	CertificateID	
StudentCertificate	StudentID, CertificateID	(StudentID, CertificateID)	FKs → Student, Certificate
GraduationProject	ProjectID, Name, Score, StudentID	ProjectID	FK StudentID → Student
Course	CourseID, Name, StartDate, EndDate	CourseID	
CourseTopic	CourseID, TopicID	(CourseID, TopicID)	FKs → Course, Topic
Topic	TopicID, Name, Description	TopicID	
CourseQuestion	CourseID, QuestionID	(CourseID, QuestionID)	FKs → Course, Question
Question	QuestionID, Text, Type, Level, Mark	QuestionID	
Choice	ChoiceID, QuestionID, Text, IsCorrect	ChoiceID	FK QuestionID → Question
Exam	ExamID, Name, StartDate, EndDate	ExamID	
ExamCourse	ExamID, CourseID	(ExamID, CourseID)	FKs → Exam, Course
ExamQuestion	ExamID, QuestionID	(ExamID, QuestionID)	FKs → Exam, Question
StudentExam	StudentID, ExamID, Grade, PassFlag	(StudentID, ExamID)	FKs → Student, Exam
StudentExamQuestion	StudentID, ExamID, QuestionID, AnswerDetail	(StudentID, ExamID, QuestionID)	
InstructorExamQuestion	InstructorID, ExamID, QuestionID	(InstructorID, ExamID, QuestionID)	

The Mapping Diagram:



4. Physical Schema

Below are the SQL Data Definition Language (DDL) statements for implementing the database schema:

- **Branch**: Stores information about ITI branches and their associated manager details.

```sql
CREATE TABLE Branch (
    Branch_ID INT PRIMARY KEY,
    Branch_Name NVARCHAR(50) NOT NULL UNIQUE,
    Branch_Email NVARCHAR(100),
    Branch_phone BIGINT,
    Branch_location NVARCHAR(100),
    Manager_ID INT,
    Manager_Name NVARCHAR(50),
    Manager_SSN BIGINT,
    Manager_Email NVARCHAR(100),
    Manager_Phone BIGINT,
    Manager_Gender NVARCHAR(20),
    Manager_DOB DATE
);
```

- **Program**: Contains program identifiers, names, and types (e.g., PTP, ICC).

```sql
CREATE TABLE Program (
    Program_ID INT PRIMARY KEY,
    P_Name NVARCHAR(50) NOT NULL,
    P_Type NVARCHAR(100) Not null,
);
```

- **Branch_Program**: A many-to-many relationship linking branches with the programs they offer.

```sql
CREATE TABLE Branch_Program (
    Branch_ID INT,
    Program_ID INT,
    CONSTRAINT Branch_Program_PK PRIMARY KEY (Branch_ID, Program_ID),
    CONSTRAINT FK_Branch_Program_Branch FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID) on update cascade,
    CONSTRAINT FK_Branch_Program_Program FOREIGN KEY (Program_ID) REFERENCES Program(Program_ID) on update cascade
);
```

- **Batch**: Represents program cycles, with start and end dates, linked to a specific program.

```sql
CREATE TABLE Batch (
    batch_ID INT PRIMARY KEY,
    Batch_Name NVARCHAR(50) NOT NULL UNIQUE,
    P_Start_Date DATE,
    P_End_Date DATE,
    Program_ID INT NOT NULL,
    CONSTRAINT FK_Program_Batch FOREIGN KEY (Program_ID) REFERENCES Program(Program_ID) on update cascade
);
```

- **Instructor**: Stores instructor details, qualifications, contact info, and salary.

```sql
CREATE TABLE Instructor (
    Instructor_ID INT PRIMARY KEY,
    SSN BIGINT NOT NULL UNIQUE,
    Fname NVARCHAR(30) NOT NULL,
    Lname NVARCHAR(30) NOT NULL,
    Email NVARCHAR(100),
    DOB DATE,
    Gender NVARCHAR(20),
    phone BIGINT,
    street NVARCHAR(100),
    city NVARCHAR(50),
    Degree nvarchar(100),
    ITI_salary BIGINT
);
```

- **Track**: Defines tracks under each program and links to a supervising instructor.

```sql
CREATE TABLE Track (
    Track_ID INT PRIMARY KEY,
    Track_Name NVARCHAR(100) NOT NULL UNIQUE,
    Track_description NVARCHAR(200),
    Program_ID INT Not null,
    Instructor_ID INT Not null,
    CONSTRAINT FK_Track_Program FOREIGN KEY (Program_ID) REFERENCES Program(Program_ID),
    CONSTRAINT FK_Track_Instructor FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID) on update cascade
);
```

- **Company**: Holds company data involved in hiring students or instructors.

```sql
Create Table Company (
    Company_ID INT PRIMARY KEY,
    Name nvarchar(50) NOT NULL,
    Type nvarchar(50),
    Location nvarchar(100)
);
```

- **Instructor_Company**: Captures companies instructors work for, including job title and salary.

```sql
Create Table Instructor_Company (
    Instructor_ID INT PRIMARY KEY,
    Company_ID INT NOT NULL,
    title NVARCHAR(30) NOT NULL,
    salary BIGINT,
    Hired_date DATE,
    CONSTRAINT FK_Instructor_Company_Instructor FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID),
    CONSTRAINT FK_Instructor_Company_Company FOREIGN KEY (Company_ID) REFERENCES Company(Company_ID) on update cascade
);
```

- **GraduationProject**: Details about graduation projects students must complete.

```
Create Table GraduationProject (
    GraduationProject_ID INT PRIMARY KEY,
    GraduationProject_Name nvarchar(200) NOT NULL,
    GraduationProject_Score INT NOT NULL
);
```

- **Student**: Central table for students, linking to tracks, branches, batches, and graduation projects.

```
Create Table Student (
    Student_ID INT PRIMARY KEY,
    SSN BIGINT NOT NULL UNIQUE,
    Fname nvarchar(50) NOT NULL,
    Lname nvarchar(50) NOT NULL,
    Email nvarchar(100),
    DOB Date,
    Gender nvarchar(20),
    phone BIGINT,
    street nvarchar(100),
    city nvarchar(50),
    Faculty nvarchar(100),
    status nvarchar(100),
    Linked_In_URL nvarchar(100),
    Leader_ID Int,
    Track_ID INT NOT NULL,
    GraduationProject_ID INT,
    Branch_ID INT NOT NULL,
    Batch_ID INT NOT NULL,
    CONSTRAINT FK_Student_Leader FOREIGN KEY (Leader_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Track FOREIGN KEY (Track_ID) REFERENCES Track(Track_ID),
    CONSTRAINT FK_Student_GraduationProject FOREIGN KEY (GraduationProject_ID) REFERENCES GraduationProject(GraduationProject_ID),
    CONSTRAINT FK_Student_Branch FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID) on update cascade,
    CONSTRAINT FK_Student_Batch FOREIGN KEY (Batch_ID) REFERENCES Batch(Batch_ID) on update cascade
);
```

- **Student_Company**: Tracks companies that hire students along with employment details.

```
Create Table Student_Company (
    Student_ID INT PRIMARY KEY,
    Company_ID Int NOT NULL,
    title nvarchar(50) NOT NULL,
    salary bigInt,
    Hired_date date,
    CONSTRAINT FK_Student_Company_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Company_Company FOREIGN KEY (Company_ID) REFERENCES Company(Company_ID) on update cascade
);
```

- **Student_Instructor**: Maps which instructors are teaching which students.

```
Create Table Student_Instructor (
    Student_ID Int,
    Instructor_ID INT,
    CONSTRAINT Student_Instructor_PK PRIMARY KEY (Student_ID, Instructor_ID),
    CONSTRAINT FK_Student_Instructor_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Instructor_Instructor FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID) on update cascade
);
```

- **Freelance**: Contains data about freelancing work students undertake.

```
Create Table Freelance (
    Freelance_ID INT PRIMARY KEY,
    Freelance_Field nvarchar(100) NOT NULL,
    Freelance_Platform nvarchar(50) NOT NULL,
    Freelance_Income BIGINT,
    Freelance_date date
);
```

- **Student_Freelance**: Associates students with their freelance work.

```
Create Table Student_Freelance (
    Student_ID Int,
    Freelance_ID INT,
    CONSTRAINT Student_Freelance_PK PRIMARY KEY(Student_ID,Freelance_ID),
    CONSTRAINT FK_Student_Freelance_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Freelance_Freelance FOREIGN KEY (Freelance_ID) REFERENCES Freelance(Freelance_ID) on update cascade
);
```

- **Certificate**: Information on certificates earned by students.

```
Create Table Certificate (
    Certificate_ID INT PRIMARY KEY,
    Certificate_Name nvarchar(100) NOT NULL,
    Certificate_Field nvarchar(100) NOT NULL,
    Certificate_source nvarchar(50) NOT NULL,
    Certificate_cost Int,
    Certificate_date date
);
```

- **Student_Certificate**: Links students with the certificates they hold.

```
Create Table Student_Certificate (
    Student_ID Int,
    Certificate_ID INT,
    CONSTRAINT Student_Certificate_PK PRIMARY KEY(Student_ID,Certificate_ID),
    CONSTRAINT FK_Student_Certificate_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Certificate_Certificate FOREIGN KEY (Certificate_ID) REFERENCES Certificate(Certificate_ID) on update cascade
);
```

- **Topic**: Contains academic or technical topics for courses.

```
Create Table Topic (
    Topic_ID INT PRIMARY KEY,
    Topic_Name nvarchar(100) NOT NULL UNIQUE,
    Topic_description nvarchar(250)
);
```

- **Course**: Describes courses with duration and the related topic.

```
CREATE TABLE Course (
    Course_ID INT PRIMARY KEY,
    Course_Name NVARCHAR(150) NOT NULL UNIQUE,
    Course_Start_date DATETIME,
    Course_end_date DATETIME,
    Topic_ID INT,
    CONSTRAINT FK_Course_Topic FOREIGN KEY (Topic_ID) REFERENCES Topic(Topic_ID) on update cascade
);
```

- **Student_Course**: Tracks course enrollments of students.

```
Create Table Student_Course (
    Student_ID Int,
    Course_ID INT,
    CONSTRAINT Student_Course_PK PRIMARY KEY(Student_ID,Course_ID),
    CONSTRAINT FK_Student_Course_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Course_Course FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID) on update cascade
);
```

- **Instructor_Course**: Tracks which instructors teach which courses.

```
Create Table Instructor_Course (
    Instructor_ID Int,
    Course_ID INT,
    CONSTRAINT Instructor_Course_PK PRIMARY KEY(Instructor_ID,Course_ID),
    CONSTRAINT FK_Instructor_Course_Instructor FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID),
    CONSTRAINT FK_Instructor_Course_Course FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID) on update cascade
);
```

- **Question**: Questions for exams, categorized by type and difficulty.

```
Create Table Question (
    Question_ID INT PRIMARY KEY,
    Question_Text NVARCHAR(500) NOT NULL UNIQUE,
    type NVARCHAR(30) NOT NULL,
    Question_Mark Int NOT NULL,
    level NVARCHAR(20),
    Course_ID INT NOT NULL,
    CONSTRAINT FK_Question_Course FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID) on update cascade
);
```

- **Choices**: Lists correct answers and options linked to questions.

```
Create Table Choices (
    Choices_ID INT PRIMARY KEY,
    right_choice NVARCHAR(100) NOT NULL,
    Question_ID INT NOT NULL,
    CONSTRAINT FK_Choices_Question FOREIGN KEY (Question_ID) REFERENCES Question(Question_ID) on update cascade
);
```

- **Choice_Text**: Contains the multiple choices for each question.

```
Create Table Choice_Text (
    Choices_ID INT,
    Choice_Text NVARCHAR(100),
    CONSTRAINT Choice_Text_PK PRIMARY KEY(Choices_ID, Choice_Text),
    CONSTRAINT FK_Choice_Text_Choices FOREIGN KEY (Choices_ID) REFERENCES Choices(Choices_ID) on update cascade
);
```

- **Exam**: Represents exam events with time and location details.

```
Create Table Exam (
    Exam_ID INT PRIMARY KEY,
    Exam_Name nvarchar(100) NOT NULL,
    Exam_Total_Marks Int NOT NULL,
    Exam_Start_date DATETIME NOT NULL,
    Exam_End_date DATETIME NOT NULL,
    Branch_ID INT NOT NULL,
    CONSTRAINT FK_Exam_Branch FOREIGN KEY (Branch_ID) REFERENCES Branch(Branch_ID) on update cascade
);
```

- **Exam_Question**: Maps questions to the exams they appear in.

```
Create Table Exam_Question (
    Exam_ID INT,
    Question_ID INT,
    CONSTRAINT Exam_Question_PK PRIMARY KEY(Exam_ID, Question_ID),
    CONSTRAINT FK_Exam_Question_Exam FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID) on update cascade,
    CONSTRAINT FK_Exam_Question_Question FOREIGN KEY (Question_ID) REFERENCES Question(Question_ID) on update cascade
);
```

- **Exam_Course**: Maps exams to the related courses.

```
Create Table Exam_Course (
    Exam_ID INT,
    Course_ID INT,
    CONSTRAINT Exam_Course_PK PRIMARY KEY (Exam_ID, Course_ID),
    CONSTRAINT FK_Exam_Course_Exam FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID) on update cascade,
    CONSTRAINT FK_Exam_Course_Course FOREIGN KEY (Course_ID) REFERENCES Course(Course_ID) on update cascade
);
```

- **Student_Exam**: Stores students' grades and pass status for each exam.

```
Create Table Student_Exam (
    Student_ID INT,
    Exam_ID INT,
    IS_Pass NVARCHAR(10) NOT NULL,
    Exam_Grade INT NOT NULL,
    CONSTRAINT Student_Exam_PK PRIMARY KEY (Student_ID, Exam_ID),
    CONSTRAINT FK_Student_Exam_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Exam_Exam FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID)
);
```

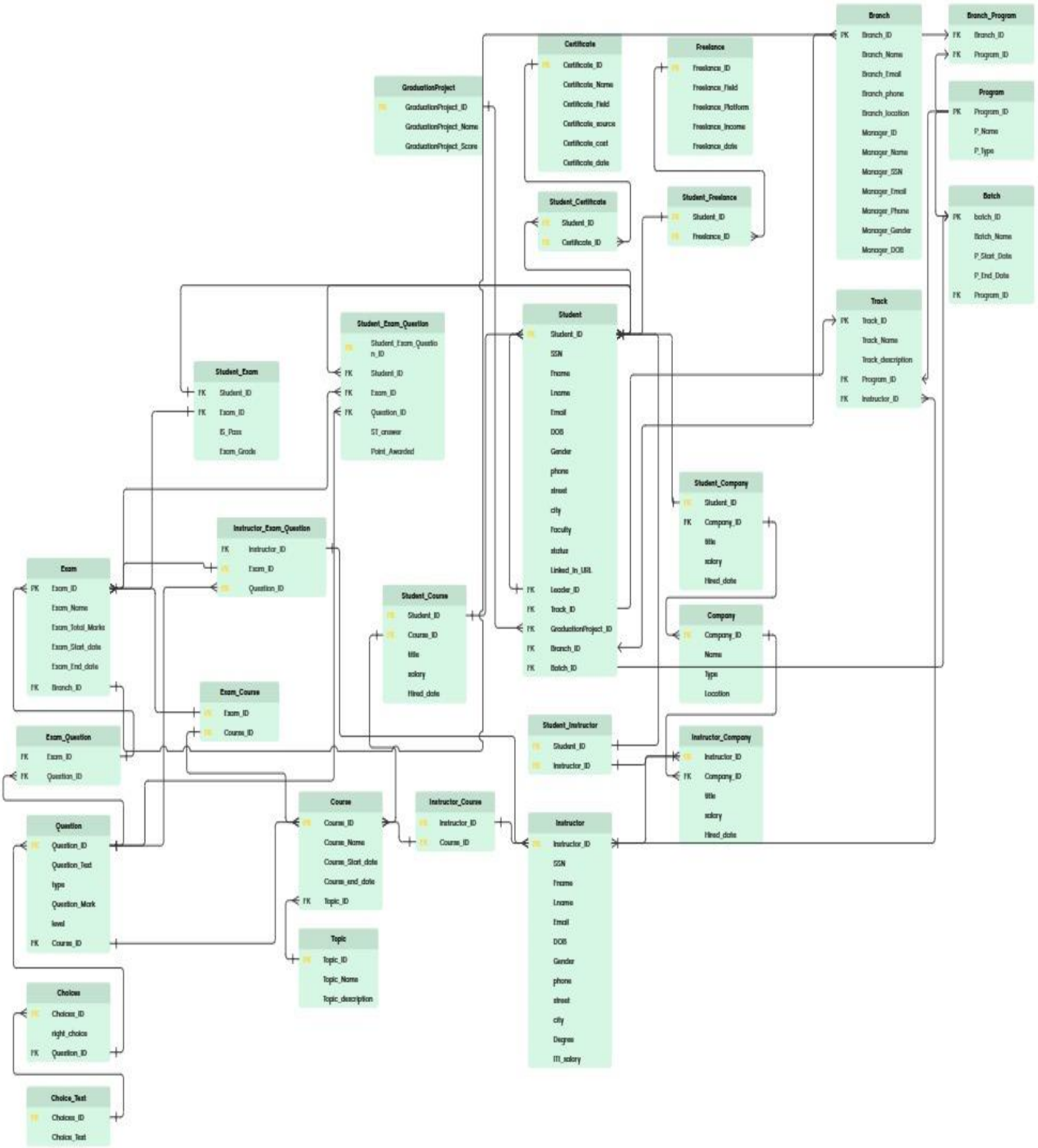
- **Student_Exam_Question**: Captures student answers and points per question.

```
Create Table Student_Exam_Question (
    Student_Exam_Question_ID INT PRIMARY KEY,
    Student_ID INT NOT NULL,
    Exam_ID INT NOT NULL,
    Question_ID INT NOT NULL,
    ST_answer NVARCHAR(50) NOT NULL,
    Point_Awarded INT NOT NULL,
    CONSTRAINT FK_Student_Exam_Question_Student FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
    CONSTRAINT FK_Student_Exam_Question_Exam FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID),
    CONSTRAINT FK_Student_Exam_Question_Question FOREIGN KEY (Question_ID) REFERENCES Question(Question_ID)
);
```

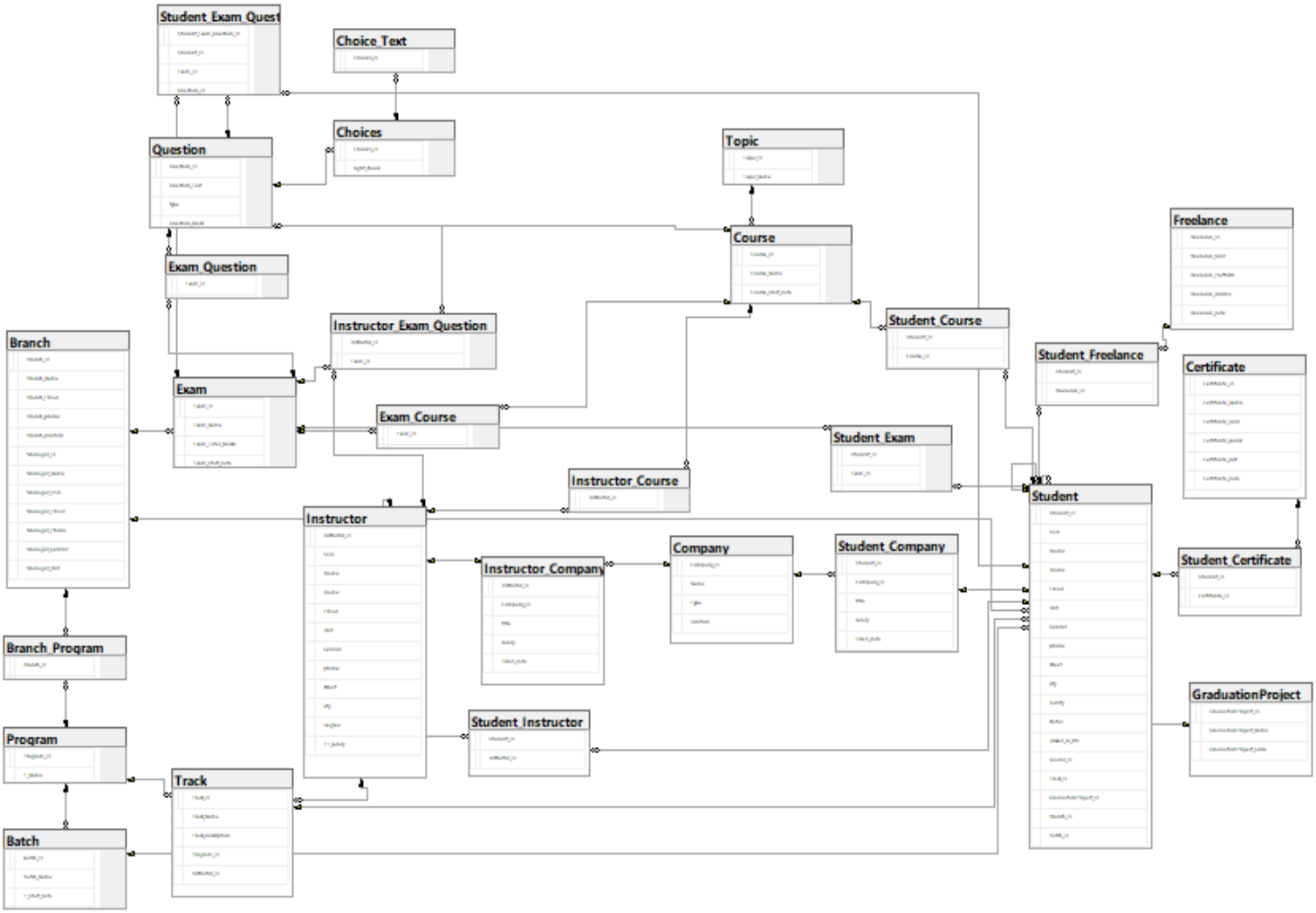
- **Instructor_Exam_Question**: Indicates which instructors assigned which questions to which exams.

```
Create Table Instructor_Exam_Question (
    Instructor_ID INT NOT NULL,
    Exam_ID INT,
    Question_ID INT,
    CONSTRAINT Instructor_Exam_Question_PK PRIMARY KEY (Exam_ID, Question_ID),
    CONSTRAINT FK_Instructor_Exam_Question_Instructor FOREIGN KEY (Instructor_ID) REFERENCES Instructor(Instructor_ID) on update cascade,
    CONSTRAINT FK_Instructor_Exam_Question_Exam FOREIGN KEY (Exam_ID) REFERENCES Exam(Exam_ID) on update cascade,
    CONSTRAINT FK_Instructor_Exam_Question_Question FOREIGN KEY (Question_ID) REFERENCES Question(Question_ID) on update cascade
);
```

The Diagram:


Branch
Branch_ID
Branch_Name
Branch_Email
Branch_phone
Branch_location
Manager_ID
Manager_Name
Manager_SSN
Manager_Email
Manager_Phone
Manager_Gender
Manager_DOB
Branch_Program
Branch_ID
Program_ID
Program
Program_ID
P_Name
P_Type
Batch
batch_ID
Batch_Name
P_Start_Date
P_End_Date
Program_ID
Certificate
Certificate_ID
Certificate_Name
Certificate_Field
Certificate_source
Certificate_cost
Certificate_date
Freelance
Freelance_ID
Freelance_Field
Freelance_Platform
Freelance_Income
Freelance_date
GraduationProject
GraduationProject_ID
GraduationProject_Name
GraduationProject_Score
Student_Certificate
Student_ID
Certificate_ID
Student_Freelance
Student_ID
Freelance_ID
Track
Track_ID
Track_Name
Track_description
Program_ID
Instructor_ID
Student
Student_ID
SSN
Fname
Lname
Email
DOB
Gender
phone
street
city
Faculty
status
Linked_In_URL
Leader_ID
Track_ID
GraduationProject_ID
Branch_ID
Batch_ID
Student_Exam_Question
Student_Exam_Question_ID
Student_ID
Exam_ID
Question_ID
ST_answer
Point_Awarded
Student_Exam
Student_ID
Exam_ID
IS_Pass
Exam_Grade
Student_Company
Student_ID
Company_ID
title
salary
Hired_date
Instructor_Exam_Question
Instructor_ID
Exam_ID
Question_ID
Company
Company_ID
Name
Type
Location
Exam
Exam_ID
Exam_Name
Exam_Total_Marks
Exam_Start_date
Exam_End_date
Branch_ID
Student_Course
Student_ID
Course_ID
title
salary
Hired_date
Exam_Course
Exam_ID
Course_ID
Student_Instructor
Student_ID
Instructor_ID
Instructor_Company
Instructor_ID
Company_ID
title
salary
Hired_date
Exam_Question
Exam_ID
Question_ID
Course
Course_ID
Course_Name
Course_Start_date
Course_end_date
Topic_ID
Instructor_Course
Instructor_ID
Course_ID
Instructor
Instructor_ID
SSN
Fname
Lname
Email
DOB
Gender
phone
street
city
Degree
ITI_salary
Question
Question_ID
Question_Text
type
Question_Mark
level
Course_ID
Topic
Topic_ID
Topic_Name
Topic_description
Choices
Choices_ID
right_choice
Question_ID
Choice_Text
Choices_ID
Choice_Text

The Diagram in SQL Server Management System (SSMS):



5. Data Generation

The generation script was got from ChatGpt and insert data at the tables like (student, Instructor, branch, track, course and exam ...)

For example: the insertion script for branch table

```
INSERT INTO Branch (Branch_ID, Branch_Name, Branch_Email, Branch_phone, Branch_location, Manager_ID, Manager_Name, Manager_SSN, Manager_Email, Manager_Phone, Manager_Gender, Manager_DOB) VALUES
(1, 'Cairo University', 'cairo@iti.org.eg', 01112345678, 'Giza', 1, 'Adel Hassan', 29010101400123, 'adel.hassan@iti.org.eg', 01099887766, 'male', '1975-03-15'),
(2, 'Alexandria', 'alex@iti.org.eg', 01122334455, 'Alexandria', 2, 'Samir Saleh', 28510101400456, 'samir.saleh@iti.org.eg', 01234567891, 'male', '1978-05-20'),
(3, 'Assiut', 'assiut@iti.org.eg', 01099887711, 'Assiut', 3, 'Nadia Kamel', 29022201400888, 'nadia.kamel@iti.org.eg', 01522334411, 'female', '1980-10-10'),
(4, 'Aswan', 'aswan@iti.org.eg', 01055667722, 'Aswan', 4, 'Ahmed Tarek', 28830301400333, 'ahmed.tarek@iti.org.eg', 01199887722, 'male', '1982-07-09'),
(5, 'Beni Suef', 'beni@iti.org.eg', 01233445566, 'Beni Suef', 5, 'Maha Said', 28940401400222, 'maha.said@iti.org.eg', 01033445511, 'female', '1979-12-01'),
(6, 'Fayoum', 'fayoum@iti.org.eg', 01144556677, 'Fayoum', 6, 'Ehab Youssef', 28750501400555, 'ehab.youssef@iti.org.eg', 01222334477, 'male', '1981-01-11'),
(7, 'Ismailia', 'ismailia@iti.org.eg', 01055667733, 'Ismailia', 7, 'Laila Mostafa', 28660601400777, 'laila.mostafa@iti.org.eg', 01144556633, 'female', '1983-09-25'),
(8, 'Mansoura', 'mansoura@iti.org.eg', 01566778899, 'Mansoura', 8, 'Yasser Fathy', 28570701400331, 'yasser.fathy@iti.org.eg', 01066778811, 'male', '1980-06-06'),
(9, 'Menofia', 'menofia@iti.org.eg', 01299887766, 'Menofia', 9, 'Salma Farid', 28480801400666, 'salma.farid@iti.org.eg', 01122334477, 'female', '1976-08-08'),
(10, 'Minya', 'minya@iti.org.eg', 01177665544, 'Minya', 10, 'Omar Khaled', 28390901400544, 'omar.khaled@iti.org.eg', 01255667799, 'male', '1977-11-11'),
(11, 'Qena', 'qena@iti.org.eg', 01044556677, 'Qena', 11, 'Rania Nabil', 28301011400339, 'rania.nabil@iti.org.eg', 01177889911, 'female', '1981-03-21'),
(12, 'Sohag', 'sohag@iti.org.eg', 01266778899, 'Sohag', 12, 'Tamer Hossam', 28211111400123, 'tamer.hossam@iti.org.eg', 01566778877, 'male', '1975-12-17'),
(13, 'Tanta', 'tanta@iti.org.eg', 01133445566, 'Tanta', 13, 'Abeer Adel', 28521211400654, 'abeer.adel@iti.org.eg', 01033445599, 'female', '1980-04-30'),
(14, 'Zagazig', 'zagazig@iti.org.eg', 01522334455, 'Zagazig', 14, 'Ayman Lotfy', 28131311400987, 'ayman.lotft@iti.org.eg', 01299887755, 'male', '1982-02-02'),
(15, 'New Valley', 'newvalley@iti.org.eg', 01255667788, 'New Valley', 15, 'Mona Atef', 28741411400222, 'mona.atef@iti.org.eg', 01122334422, 'female', '1983-05-14'),
(16, 'Damanhour', 'damanhour@iti.org.eg', 01177889900, 'Damanhour', 16, 'Sherif Younes', 28551511400444, 'sherif.younes@iti.org.eg', 01566778800, 'male', '1979-06-19'),
(17, 'Al Arish', 'arish@iti.org.eg', 01533445566, 'Al Arish', 17, 'Reham Fathy', 28661611400777, 'reham.fathy@iti.org.eg', 01266778833, 'female', '1984-08-08'),
(18, 'Banha', 'banha@iti.org.eg', 01066778899, 'Banha', 18, 'Khaled El Sayed', 28471711400233, 'khaled.elsayed@iti.org.eg', 01133445588, 'male', '1980-01-01'),
(19, 'Port Said', 'portsaid@iti.org.eg', 01111223344, 'Port Said', 19, 'Nourhan Yehia', 28581811400555, 'nourhan.yehia@iti.org.eg', 01577889900, 'female', '1977-10-10'),
(20, 'New Capital', 'newcapital@iti.org.eg', 01233445599, 'New Capital', 20, 'Mohamed Shaker', 28391911400444, 'mohamed.shaker@iti.org.eg', 01055667799, 'male', '1978-09-01'),
(21, 'Smart Village', 'smartvillage@iti.org.eg', 01055667788, 'Smart Village', 21, 'Ahmed El Sayed', 28202011400333, 'ahmed.elsayed@iti.org.eg', 01233445566, 'male', '1981-07-07');
GO
```

6. Important SPs for the System

This chapter includes key stored procedures that automate and facilitate common database operations in the system.
Additionally, full CRUD (Create, Read, Update, Delete) operations have been implemented for every single table in the system. However, for brevity, only the CRUD procedures for the Branch table are documented below as a representative example.

- **GenerateExam:** Automatically creates an exam by randomly selecting a specified number of MCQ and True/False questions from a course. It calculates the total exam mark, stores the exam, and maps it to selected questions.

```
CREATE or alter PROCEDURE GenerateExam
  @ExamName VARCHAR(50),
  @Course VARCHAR(30),
  @NumOfMCQ INT,
  @NumOfTF INT,
  @StartDate DATETIME,
  @EndDate DATETIME
AS
BEGIN
  DECLARE @id INT;

  WITH first_CTE AS (
    SELECT TOP(@NumOfMCQ) q.Question_ID, q.Question_Mark
    FROM Question q INNER JOIN Course c ON q.Course_ID = c.Course_ID
    WHERE c.Course_Name = @Course AND q.type = 'MCQ'
    ORDER BY NEWID()
    UNION ALL
    SELECT TOP(@NumOfTF) q.Question_ID, q.Question_Mark
    FROM Question q INNER JOIN Course c ON q.Course_ID = c.Course_ID
    WHERE c.Course_Name = @Course AND q.type = 'True/False'
    ORDER BY NEWID()
  )
  SELECT * INTO #TempQuestions FROM first_CTE;

  INSERT INTO Exam (Exam_Name, Exam_Total_Marks, Exam_Start_date, Exam_End_date)
  VALUES (@ExamName, (SELECT SUM(CAST(Question_Mark AS INT)) FROM #TempQuestions), @StartDate, @EndDate);

  SELECT @id = SCOPE_IDENTITY();

  INSERT INTO Exam_Question (Exam_ID, Question_ID)
  SELECT @id, Question_ID FROM #TempQuestions;
  DROP TABLE #TempQuestions;
  SELECT 'Exam Generated Successfully'
END
```

- **ExamAnswer:** Records a student's answer to a specific exam question. It compares the submitted answer with the correct one and awards the appropriate points.

```
CREATE or alter PROCEDURE ExamAnswer
  @Exam_ID INT,
  @Student_ID INT,
  @Question_ID INT,
  @Student_Answer NVARCHAR(50)
AS
BEGIN
  DECLARE @ChoicePoint INT;

  IF @Student_Answer = (
    SELECT MAX(c.right_choice)
    FROM Question q INNER JOIN Choices c ON q.Question_ID = c.Question_ID
    WHERE q.Question_ID = @Question_ID)
    SET @ChoicePoint = (SELECT MAX(Question_Mark) FROM Question WHERE Question_ID = @Question_ID)
  ELSE
    SET @ChoicePoint = 0;

  INSERT INTO Student_Exam_Question (Student_Exam_Question_ID, Exam_ID, Student_ID, Question_ID, ST_answer, Point_Awarded)
  VALUES ((SELECT MAX(Student_Exam_Question_ID)+1 FROM Student_Exam_Question), @Student_ID, @Exam_ID, @Question_ID, @Student_Answer, @ChoicePoint);
END
```

- **ExamCorrection:** Grades a student's exam and determines whether the student passed or failed based on a 60% threshold. If the student didn't attend, it returns a message instead.

```
CREATE or alter PROCEDURE ExamCorrection
  @Student_ID INT,
  @Exam_ID INT
AS
BEGIN
  DECLARE @ISPass VARCHAR(30);

  IF NOT EXISTS (
    SELECT * FROM Student_Exam_Question
    WHERE Student_ID = @Student_ID AND Exam_ID = @Exam_ID)
    SELECT 'This student didnt attend the exam';
  ELSE
  BEGIN
    WITH ExamGrade_CTE AS (
      SELECT SUM(Point_Awarded) AS ExamGrade
      FROM Student_Exam_Question
      WHERE Student_ID = @Student_ID AND Exam_ID = @Exam_ID
    )
    SELECT * INTO #TempExamGrade FROM ExamGrade_CTE;

    IF (SELECT SUM(ExamGrade) FROM #TempExamGrade) / (SELECT SUM(Exam_Total_Marks) FROM Exam WHERE Exam_ID = @Exam_ID) >= 0.6
      SET @ISPass = 'Pass';
    ELSE
      SET @ISPass = 'Fail';

    INSERT INTO Student_Exam(Student_ID, Exam_ID, IS_Pass, Exam_Grade)
    VALUES(@Student_ID, @Exam_ID, @ISPass, (SELECT SUM(ExamGrade) FROM #TempExamGrade));

    DROP TABLE #TempExamGrade;
  END
END
```

- **isCourseListedForAllCoursesAllStudents:** Updates student status to 'Course Listed' for students who failed at least two exams from the same course.

```
CREATE OR ALTER PROCEDURE isCourseListedForAllCoursesAllStudents
AS
BEGIN
  UPDATE s
  SET [status] = 'Course Listed'
  FROM Student s
  WHERE [status] = 'student'
  AND EXISTS (
    SELECT 1
    FROM (
      SELECT c.Course_ID, se.Student_ID
      FROM Student_Exam se
      INNER JOIN Exam e ON se.Exam_ID = e.Exam_ID
      INNER JOIN Exam_Course ec ON e.Exam_ID = ec.Exam_ID
      INNER JOIN Course c ON ec.Course_ID = c.Course_ID
      WHERE se.Student_ID = s.Student_ID
      GROUP BY c.Course_ID, se.Student_ID
      HAVING COUNT(e.Exam_ID) >= 2 AND SUM(CASE WHEN se.IS_Pass = 'Fail' THEN 1 ELSE 0 END) >= 2
    ) t
    WHERE t.Student_ID = s.Student_ID
  );
END
```

- **Branch CRUD Operations:** To demonstrate standardized database operations, the following procedures are implemented only for the Branch table.

    1. Select a Branch by ID

```
CREATE OR ALTER PROCEDURE sp_Select_Branch @Branch_ID INT
AS
BEGIN TRY
  SELECT * FROM Branch WHERE Branch_ID = @Branch_ID;
END TRY
BEGIN CATCH
  SELECT 'Branch Is Not Exists' AS ErrorMessage;
END CATCH;
```

2. Insert a new Branch

```
CREATE OR ALTER PROCEDURE sp_Insert_Branch
  @Branch_Name NVARCHAR(50),
  @Branch_Email NVARCHAR(100) = NULL,
  @Branch_phone BIGINT = NULL,
  @Branch_location NVARCHAR(100) = NULL,
  @Manager_ID INT = NULL,
  @Manager_Name NVARCHAR(50) = NULL,
  @Manager_SSN BIGINT = NULL,
  @Manager_Email NVARCHAR(100) = NULL,
  @Manager_Phone BIGINT = NULL,
  @Manager_Gender NVARCHAR(20) = NULL,
  @Manager_DOB DATE = NULL
AS
BEGIN TRY
  INSERT INTO Branch (Branch_ID, Branch_Name, Branch_Email, Branch_phone, Branch_location,
            Manager_ID, Manager_Name, Manager_SSN, Manager_Email, Manager_Phone, Manager_Gender, Manager_DOB)
  VALUES (((SELECT MAX(Branch_ID) FROM Branch)+1),
            @Branch_Name, @Branch_Email, @Branch_phone, @Branch_location,
            @Manager_ID, @Manager_Name, @Manager_SSN, @Manager_Email, @Manager_Phone, @Manager_Gender, @Manager_DOB);
END TRY
BEGIN CATCH
  SELECT 'Branch Is Already Exists' AS ErrorMessage;
END CATCH;
```

3. Update an existing Branch

```
CREATE OR ALTER PROCEDURE sp_Update_Branch
  @Branch_ID INT,
  @Branch_Name NVARCHAR(50),
  @Branch_Email NVARCHAR(100) = NULL,
  @Branch_phone BIGINT = NULL,
  @Branch_location NVARCHAR(100) = NULL,
  @Manager_ID INT = NULL,
  @Manager_Name NVARCHAR(50) = NULL,
  @Manager_SSN BIGINT = NULL,
  @Manager_Email NVARCHAR(100) = NULL,
  @Manager_Phone BIGINT = NULL,
  @Manager_Gender NVARCHAR(20) = NULL,
  @Manager_DOB DATE = NULL
AS
BEGIN TRY
  UPDATE Branch
  SET Branch_Name = @Branch_Name,
      Branch_Email = @Branch_Email,
      Branch_phone = @Branch_phone,
      Branch_location = @Branch_location,
      Manager_ID = @Manager_ID,
      Manager_Name = @Manager_Name,
      Manager_SSN = @Manager_SSN,
      Manager_Email = @Manager_Email,
      Manager_Phone = @Manager_Phone,
      Manager_Gender = @Manager_Gender,
      Manager_DOB = @Manager_DOB
  WHERE Branch_ID = @Branch_ID;
END TRY
BEGIN CATCH
  SELECT 'Branch Is Not Exists' AS ErrorMessage;
END CATCH;
```

4. Delete a Branch by ID

```sql
CREATE OR ALTER PROCEDURE sp_Delete_Branch @Branch_ID INT
AS
BEGIN TRY
  DELETE FROM Branch WHERE Branch_ID = @Branch_ID;
END TRY
BEGIN CATCH
  SELECT 'Branch Is Not Exists' AS ErrorMessage;
END CATCH;
```

7. Data Warehouse Modeling

Below are the SQL Data Definition Language (DDL) statements for implementing the data warehouse schema:

- **Dim_Student:** Stores student-related information, including project and freelance data.

```
CREATE TABLE Dim_Student (
    Student_SK INT PRIMARY KEY IDENTITY(1,1),
    Student_ID_BK INT,
    SSN BIGINT,
    Fname NVARCHAR(50),
    Lname NVARCHAR(50),
    Email NVARCHAR(100),
    DOB DATE,
    Gender NVARCHAR(20),
    phone BIGINT,
    street NVARCHAR(100),
    city NVARCHAR(50),
    Faculty NVARCHAR(100),
    status NVARCHAR(100),
    Linked_In_URL NVARCHAR(100),
    Leader_ID INT,
    GraduationProject_ID_BK INT,
    GraduationProject_Name NVARCHAR(200),
    GraduationProject_Score INT,
    Freelance_ID_BK INT,
    Freelance_Field NVARCHAR(100),
    Freelance_Platform NVARCHAR(50),
    Freelance_Income BIGINT,
    Freelance_date DATE,
    ssc TINYINT,
    iscurrent TINYINT,
    StartDate DATETIME,
    EndDate DATETIME
);
```

- **Dim_Instructor:** Holds personal and professional details of instructors.

```
CREATE TABLE Dim_instructor (
    instructor_id_sk INT PRIMARY KEY IDENTITY(1,1),
    instructor_id_bk INT,
    ssn BIGINT,
    fname NVARCHAR(30),
    lname NVARCHAR(30),
    email NVARCHAR(100),
    DOB DATE,
    Gender NVARCHAR(20),
    phone BIGINT,
    street NVARCHAR(100),
    city NVARCHAR(50),
    degree NVARCHAR(100),
    iti_salary BIGINT,
    ssc TINYINT,
    iscurrent TINYINT,
    startdate DATETIME,
    enddate DATETIME
);
```

- **Dim_Branch:** Contains data about ITI branches and their managers.

```
CREATE TABLE Dim_Branch (
    Branch_id_sk INT PRIMARY KEY IDENTITY(1,1),
    Branch_id_bk INT,
    Branch_Name NVARCHAR(50),
    Branch_email NVARCHAR(100),
    Branch_phone BIGINT,
    Branch_Location NVARCHAR(100),
    Manager_id_bk INT,
    Manager_name NVARCHAR(50),
    Manager_ssn BIGINT,
    Manager_email NVARCHAR(100),
    Manager_phone BIGINT,
    Manager_gender NVARCHAR(20),
    Manager_DOB DATE,
    SSC TINYINT,
    iscurrent TINYINT,
    startdate DATETIME,
    enddate DATETIME
);
```

- **Dim_Company:** Includes company details and employment info for both students and instructors.

```
CREATE TABLE Dim_company (
    company_sk INT PRIMARY KEY IDENTITY(1,1),
    company_id_bk INT,
    name NVARCHAR(50),
    type NVARCHAR(50),
    location NVARCHAR(100),
    instructor_id_bk INT,
    student_id_bk INT,
    st_title NVARCHAR(50),
    st_salary BIGINT,
    st_hired_date DATE,
    ins_title NVARCHAR(30),
    ins_salary BIGINT,
    ins_hired_date DATE,
    ssc TINYINT,
    iscurrent TINYINT,
    startdate DATETIME,
    enddate DATETIME
);
```

- **Dim_Course:** Describes course data and associated topic information.

```
CREATE TABLE Dim_Course (
    course_sk INT PRIMARY KEY IDENTITY(1,1),
    course_id_bk INT,
    course_name NVARCHAR(150),
    course_start_date DATETIME,
    course_end_date DATETIME,
    topic_id_bk INT,
    topic_name NVARCHAR(100),
    topic_description NVARCHAR(250),
    ssc TINYINT,
    iscurrent TINYINT,
    startdate DATETIME,
    enddate DATETIME
);
```

- **Dim_Exam:** Captures exam details, including related questions and choices.

```
CREATE TABLE Dim_Exam (
    Exam_SK INT PRIMARY KEY IDENTITY(1,1),
    Exam_ID_BK INT,
    Exam_Name NVARCHAR(100),
    Exam_Total_Marks INT,
    Exam_Start_date DATETIME,
    Exam_End_date DATETIME,
    Question_ID_BK INT,
    Question_Text NVARCHAR(500),
    type NVARCHAR(30),
    Question_Mark INT,
    level NVARCHAR(20),
    Choices_ID_BK INT,
    right_choice NVARCHAR(100),
    ssc TINYINT,
    iscurrent TINYINT,
    StartDate DATETIME,
    EndDate DATETIME
);
```

- **Dim_Track:** Defines track information along with associated program and batch data.

```
CREATE TABLE Dim_Track (
    Track_id_sk INT PRIMARY KEY IDENTITY(1,1),
    Track_id_bk INT,
    Track_Name NVARCHAR(100),
    Track_description NVARCHAR(200),
    program_id_bk INT,
    P_name NVARCHAR(50),
    P_Type NVARCHAR(100),
    Batch_id_bk INT,
    Batch_Name NVARCHAR(50),
    Batch_startdate DATE,
    Batch_enddate DATE,
    ssc TINYINT,
    iscurrent TINYINT,
    startdate DATETIME,
    enddate DATETIME
);
```

- **DimDate:** A standard date dimension for time-based analytics.

```
CREATE TABLE DimDate (
    DateSK INT PRIMARY KEY,
    Date DATETIME NOT NULL,
    Day CHAR(2) NOT NULL,
    DaySuffix VARCHAR(4) NOT NULL,
    DayOfWeek VARCHAR(9) NOT NULL,
    DOWInMonth TINYINT NOT NULL,
    DayOfYear INT NOT NULL,
    WeekOfYear TINYINT NOT NULL,
    WeekOfMonth TINYINT NOT NULL,
    Month CHAR(2) NOT NULL,
    MonthName VARCHAR(9) NOT NULL,
    Quarter TINYINT NOT NULL,
    QuarterName VARCHAR(6) NOT NULL,
    Year CHAR(4) NOT NULL,
    StandardDate VARCHAR(10),
    HolidayText VARCHAR(50),
    FiscalDay CHAR(2),
    FiscalMonth CHAR(2),
    FiscalMonthName VARCHAR(9),
    FiscalQuarter TINYINT,
    FiscalQuarterName VARCHAR(6),
    FiscalYear CHAR(4)
);
```

- **DimTime:** Provides breakdown of time (hour, minute, second) for event timestamping.

```
CREATE TABLE DimTime (
    TimeSK INT PRIMARY KEY IDENTITY(1,1),
    Time DATETIME NOT NULL,
    Hour CHAR(2) NOT NULL,
    MilitaryHour CHAR(2) NOT NULL,
    Minute CHAR(2) NOT NULL,
    Second CHAR(2) NOT NULL,
    AmPm CHAR(2) NOT NULL,
    StandardTime CHAR(11)
);
```

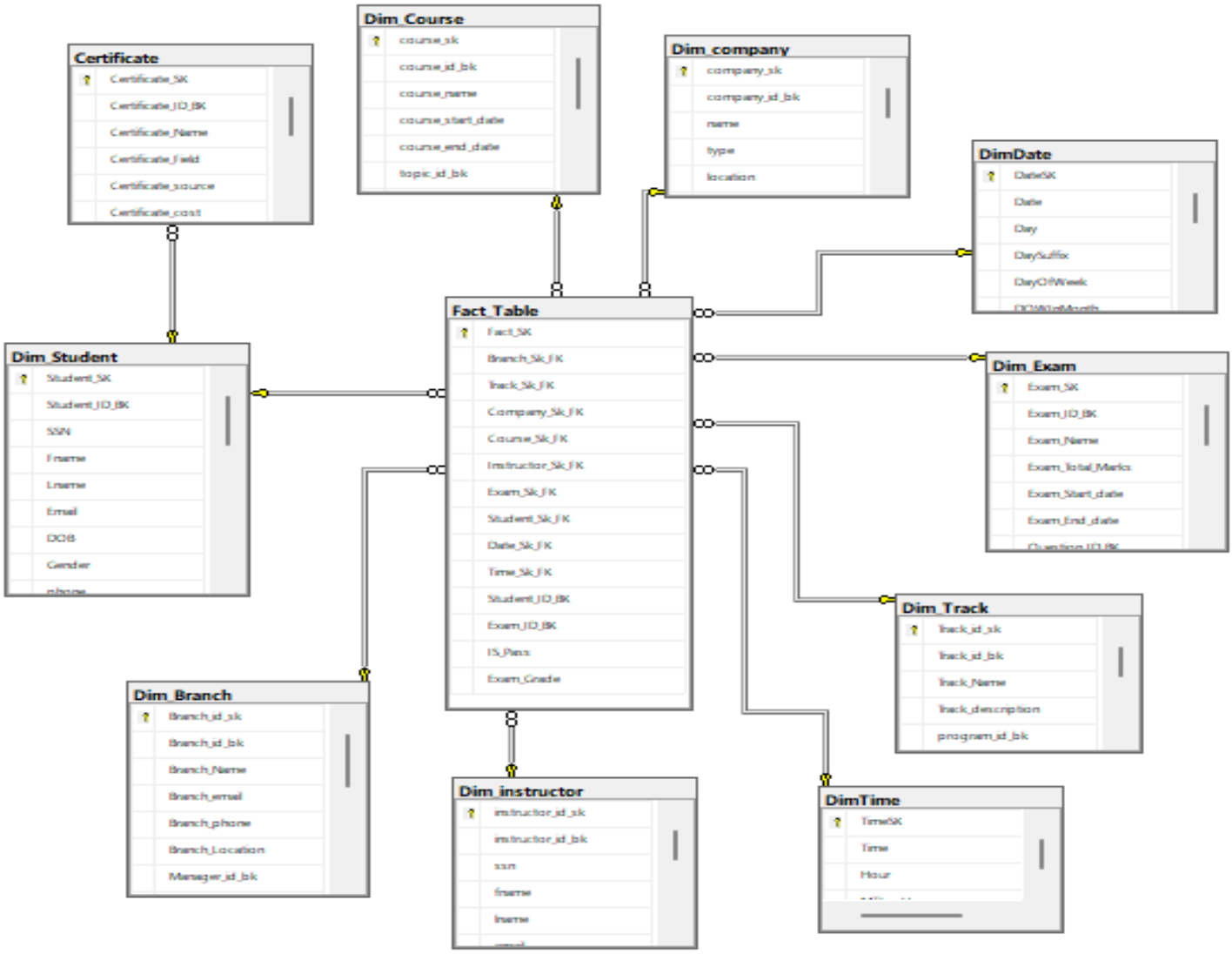
- **Certificate:** Fact-like dimension that links students to certificates with validity tracking.

```sql
CREATE TABLE Certificate (
    Certificate_SK INT PRIMARY KEY IDENTITY(1,1),
    Certificate_ID_BK INT,
    Certificate_Name NVARCHAR(100),
    Certificate_Field NVARCHAR(100),
    Certificate_source NVARCHAR(50),
    Certificate_cost INT,
    Certificate_date DATE,
    Student_Sk_FK INT,
    ssc TINYINT,
    iscurrent TINYINT,
    StartDate DATETIME,
    EndDate DATETIME
);
```

- **Fact_Table:** Central fact table capturing exam performance and connections to all dimensions.

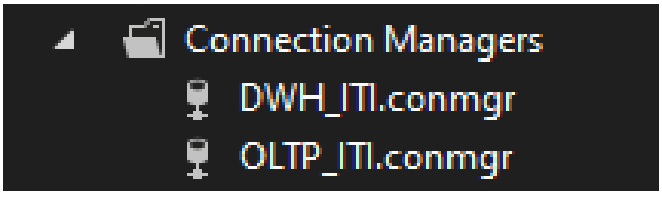
```sql
CREATE TABLE Fact_Table (
    Fact_SK INT PRIMARY KEY IDENTITY(1,1),
    Branch_Sk_FK INT,
    Track_Sk_FK INT,
    Company_Sk_FK INT,
    Course_Sk_FK INT,
    Instructor_Sk_FK INT,
    Exam_Sk_FK INT,
    Student_Sk_FK INT,
    Date_Sk_FK INT,
    Time_Sk_FK INT,
    Student_ID_BK INT,
    Exam_ID_BK INT,
    IS_Pass NVARCHAR(10),
    Exam_Grade INT
);
```

The Diagram in SQL Server Management System (SSMS):



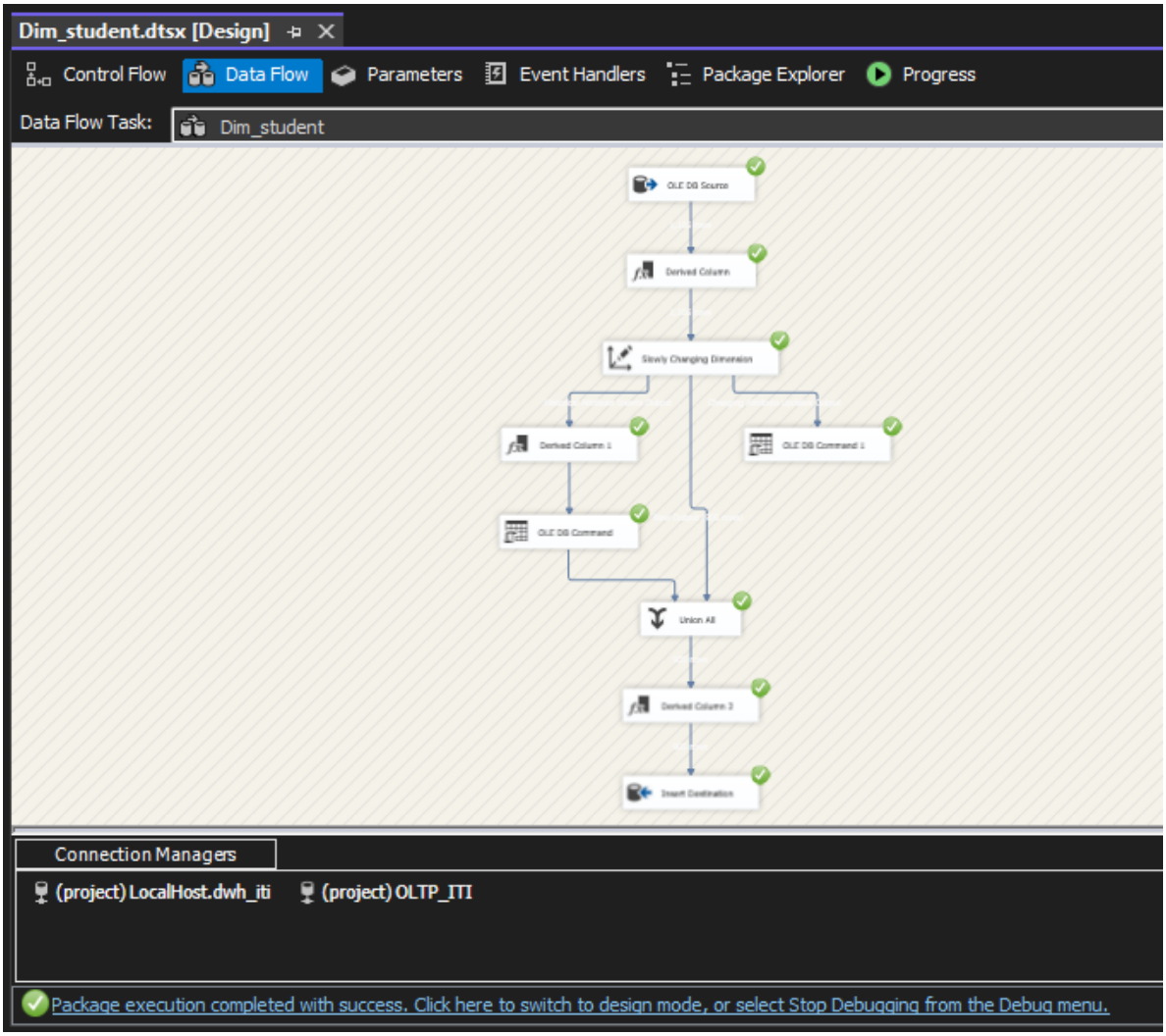
8. ETL: from Database to Data warehouse

Starts with adding Source and Destination connections, the source is database, and the destination is data warehouse:

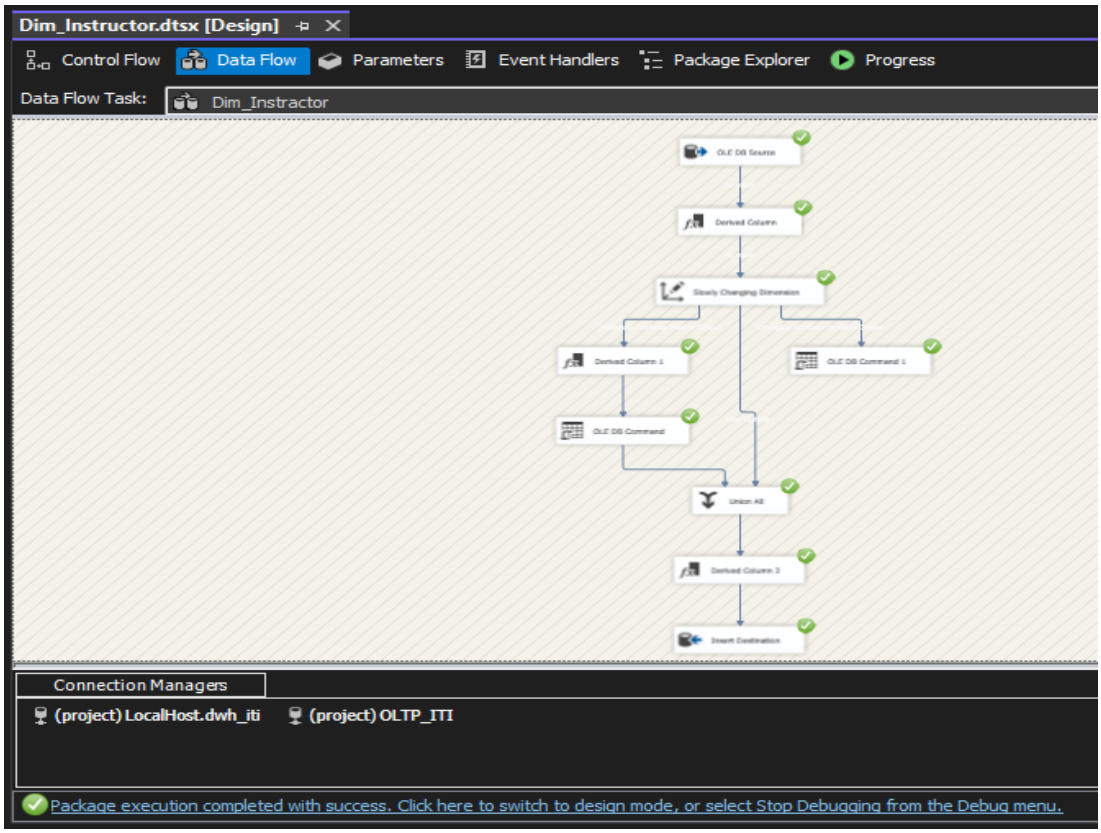


Then creating SSIS packages for each dimension and fact tables and deploy them.

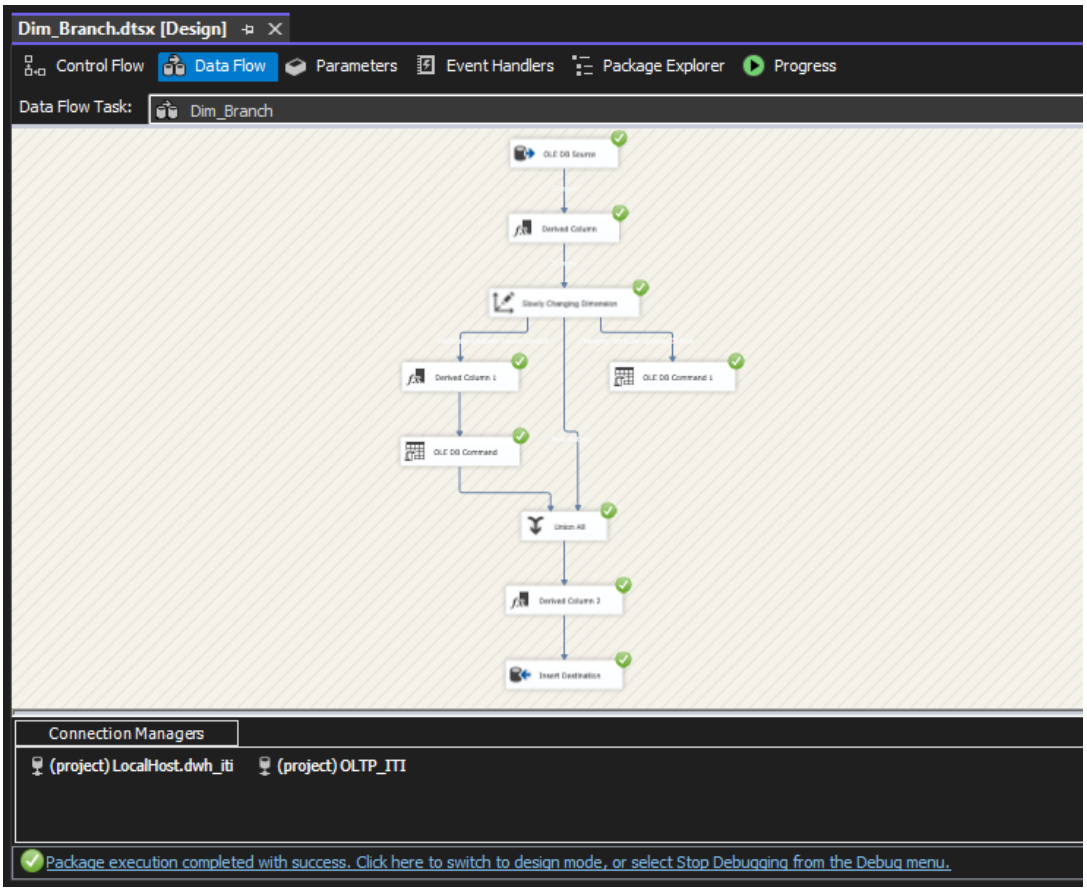
Dim_Student:


Dim_student.dtsx [Design]
Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Dim_student
Connection Managers
(project) LocalHost.dwh_iti
(project) OLTP_ITI
Package execution completed with success. Click here to switch to design mode, or select Stop Debugging from the Debug menu.

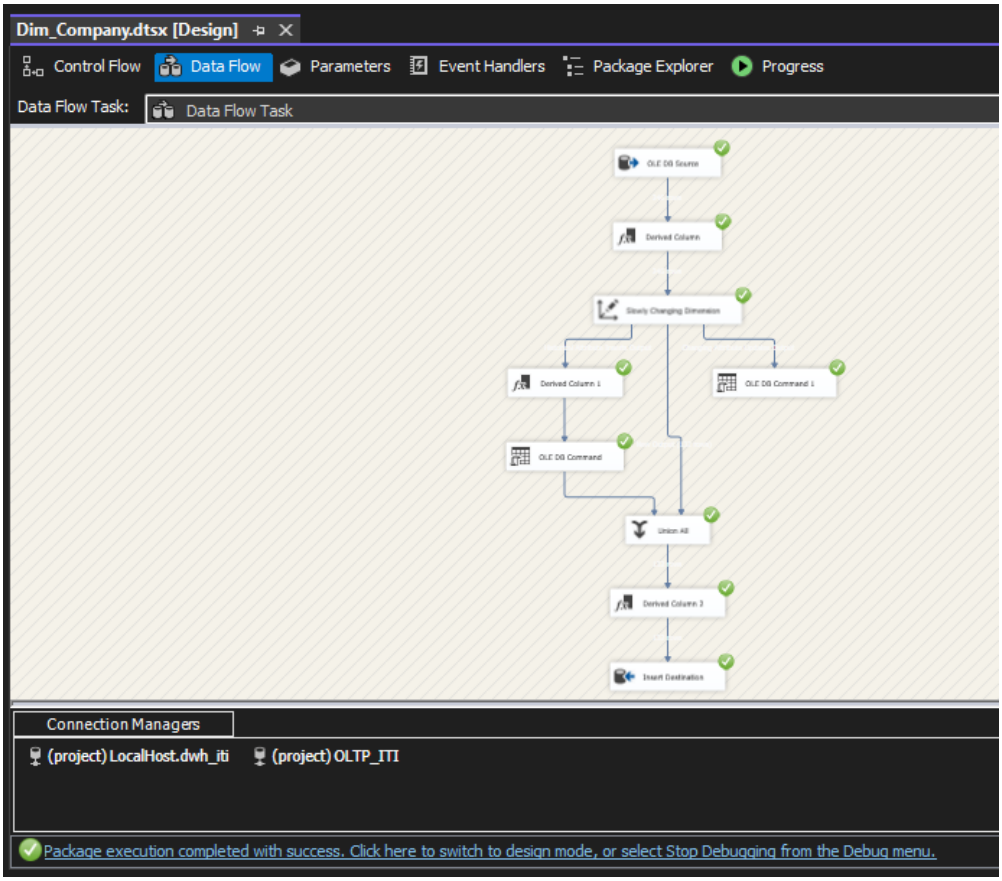
Dim_Instructor:



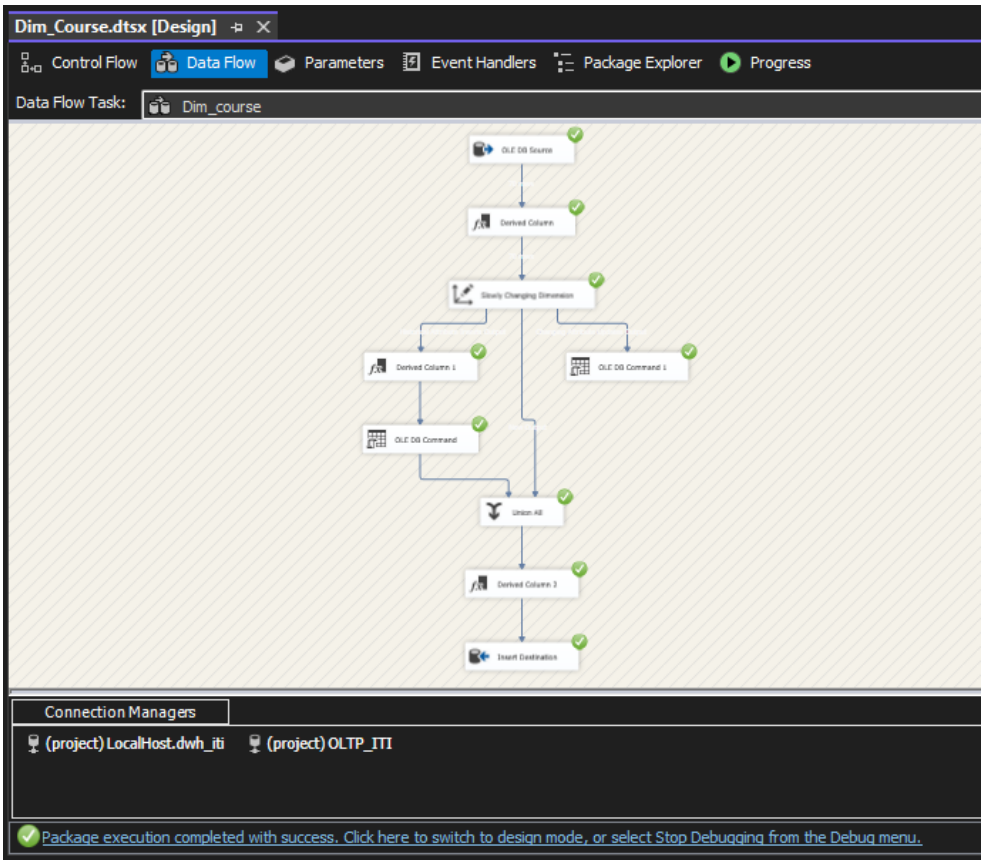
Dim_Branch:



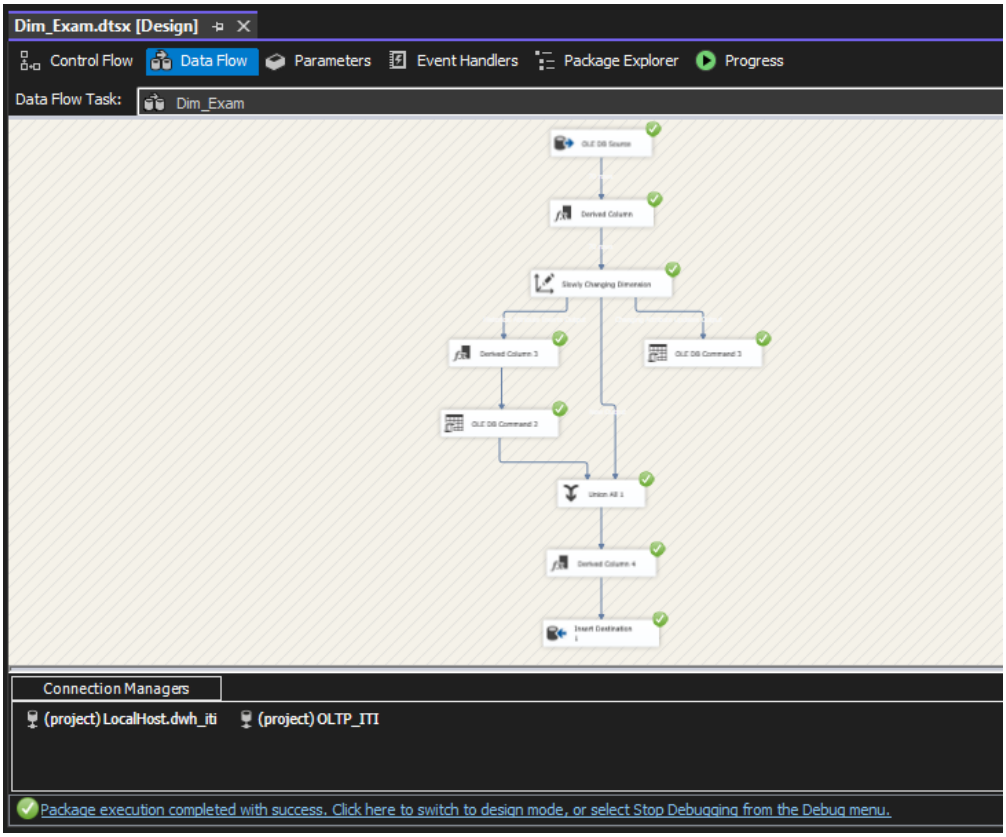
Dim_Company:



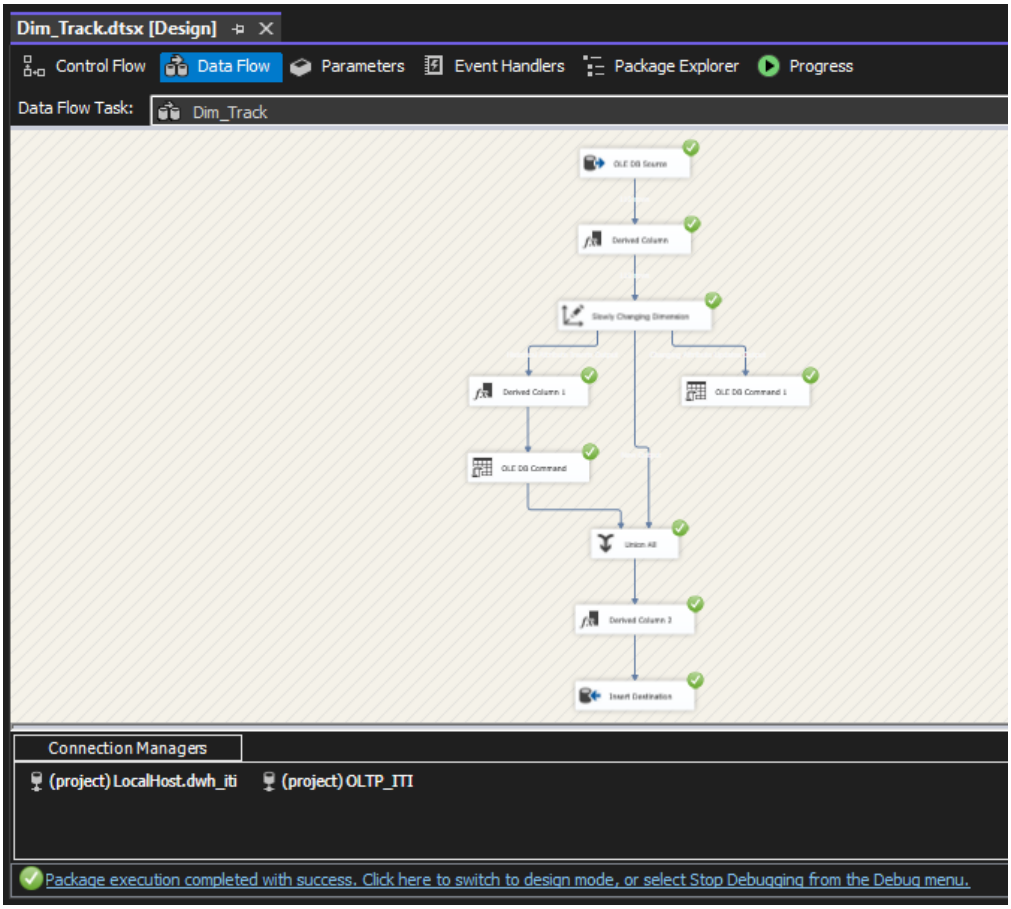
Dim_Course:



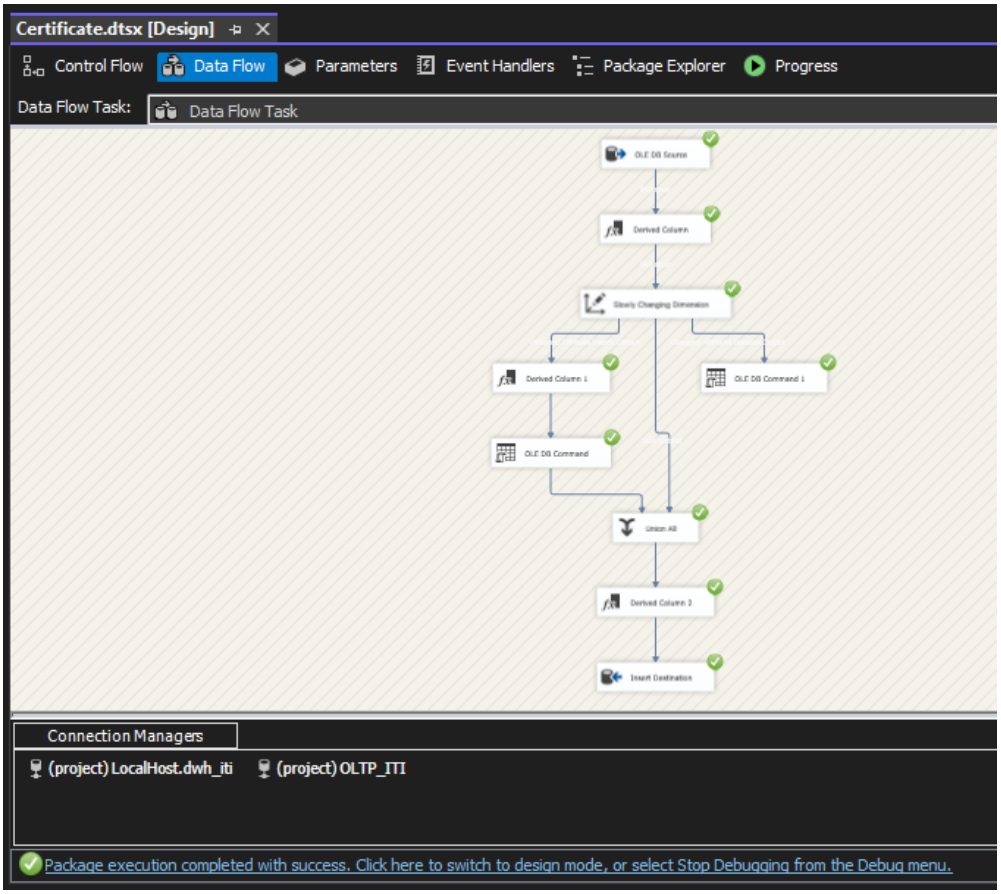
Dim_Exam:



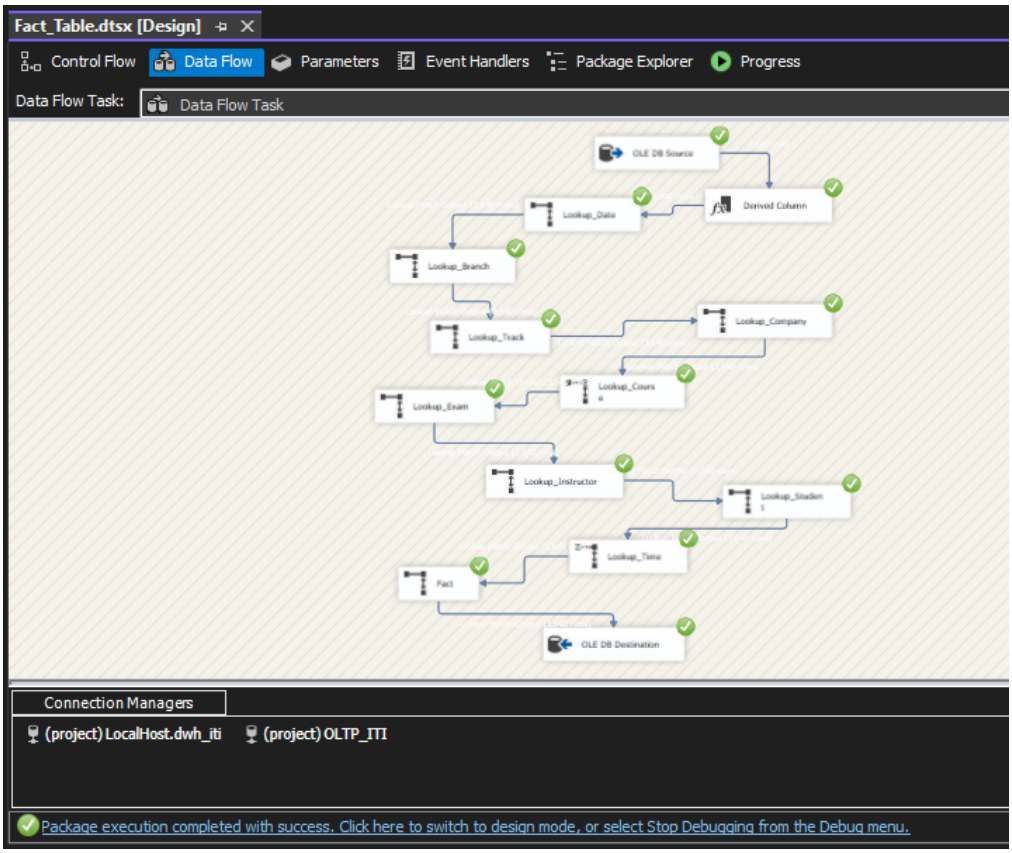
Dim_Track:



Certificate:



Fact_Table:

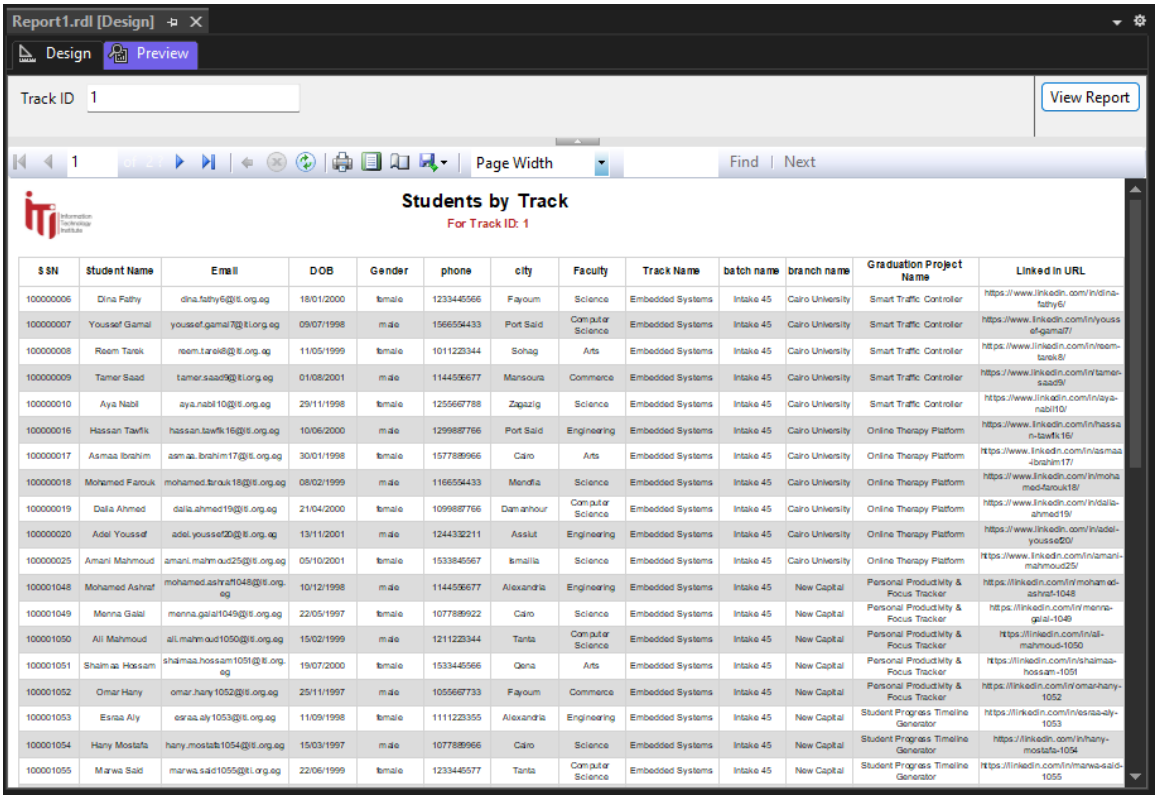


9. The Reports in SQL Server Reporting Service (SSRS):

Below are the Stored Procedures for reporting and Reports:

- **Report 1:** Report that returns the students information according to Department No parameter.

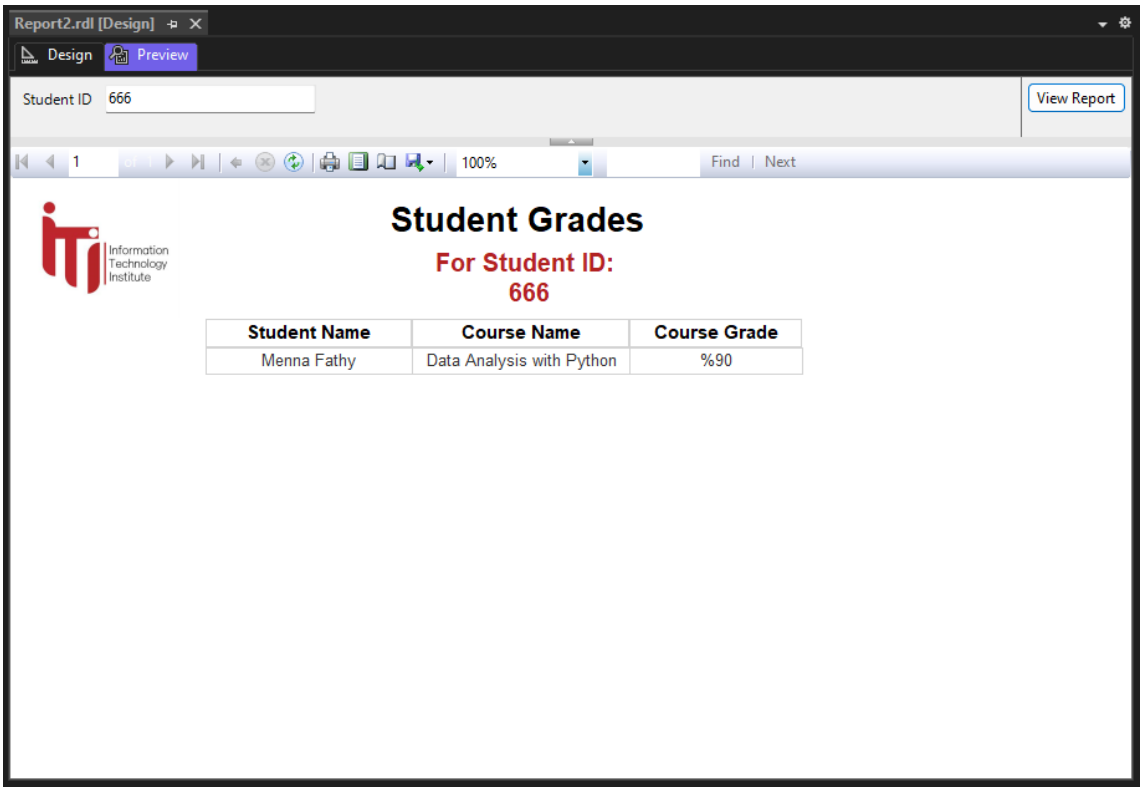
```
CREATE OR ALTER PROCEDURE [dbo].[report_1] @TrackID INT AS BEGIN
SELECT s.*,
       t.Track_Name,
       b.batch_name,
       br.branch_name,
       gp.GraduationProject_Name
FROM Student s
INNER JOIN Track t ON s.Track_ID = t.Track_ID
INNER JOIN Batch b ON s.Batch_ID = b.Batch_ID
INNER JOIN Branch br ON s.Branch_ID = br.Branch_ID
LEFT JOIN GraduationProject gp ON s.GraduationProject_ID = gp.GraduationProject_ID
WHERE s.Track_ID = @TrackID
  AND s.status = 'student' END
```



SSN	Student Name	Email	DOB	Gender	phone	city	Faculty	Track Name	batch name	branch name	Graduation Project Name	Linked In URL
100000006	Dina Fathy	dina.fathy6@iti.org.eg	18/01/2000	female	1233445566	Fayoum	Science	Embedded Systems	Intake 45	Cairo University	Smart Traffic Controller	https://www.linkedin.com/in/dina-fathy6/
100000007	Youssef Gamal	youssef.gamal7@iti.org.eg	09/07/1998	male	1566554433	Port Said	Computer Science	Embedded Systems	Intake 45	Cairo University	Smart Traffic Controller	https://www.linkedin.com/in/youssef-gamal7/
100000008	Reem Tarek	reem.tarek8@iti.org.eg	11/05/1999	female	1011223344	Sohag	Arts	Embedded Systems	Intake 45	Cairo University	Smart Traffic Controller	https://www.linkedin.com/in/reem-tarek8/
100000009	Tamer Saad	tamer.saad9@iti.org.eg	01/08/2001	male	1144556677	Mansoura	Commerce	Embedded Systems	Intake 45	Cairo University	Smart Traffic Controller	https://www.linkedin.com/in/tamer-saad9/
100000010	Aya Nabil	aya.nabil10@iti.org.eg	29/11/1998	female	1255667788	Zagazig	Science	Embedded Systems	Intake 45	Cairo University	Smart Traffic Controller	https://www.linkedin.com/in/aya-nabil10/
100000016	Hassan Tawfik	hassan.tawfik16@iti.org.eg	10/06/2000	male	1299887766	Port Said	Engineering	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/hassan-tawfik16/
100000017	Asmaa Ibrahim	asmaa.ibrahim17@iti.org.eg	30/01/1998	female	1577889966	Cairo	Arts	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/asmaa-ibrahim17/
100000018	Mohamed Farouk	mohamed.farouk18@iti.org.eg	08/02/1999	male	1166554433	Mendfa	Science	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/mohamed-farouk18/
100000019	Dalia Ahmed	dalia.ahmed19@iti.org.eg	21/04/2000	female	1099887766	Damanhour	Computer Science	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/dalia-ahmed19/
100000020	Adel Youssef	adel.youssef20@iti.org.eg	13/11/2001	male	1244332211	Assiut	Engineering	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/adel-youssef20/
100000025	Amani Mahmoud	amani.mahmoud25@iti.org.eg	05/10/2001	female	1533845667	Ismailia	Science	Embedded Systems	Intake 45	Cairo University	Online Therapy Platform	https://www.linkedin.com/in/amani-mahmoud25/
100001048	Mohamed Ashraf	mohamed.ashraf1048@iti.org.eg	10/12/1998	male	1144556677	Alexandria	Engineering	Embedded Systems	Intake 45	New Capital	Personal Productivity & Focus Tracker	https://linkedin.com/in/mohamed-ashraf-1048
100001049	Menna Galal	menna.galal1049@iti.org.eg	22/05/1997	female	1077889922	Cairo	Science	Embedded Systems	Intake 45	New Capital	Personal Productivity & Focus Tracker	https://linkedin.com/in/menna-galal-1049
100001050	Ali Mahmoud	ali.mahmoud1050@iti.org.eg	15/02/1999	male	1211223344	Tanta	Computer Science	Embedded Systems	Intake 45	New Capital	Personal Productivity & Focus Tracker	https://linkedin.com/in/ali-mahmoud-1050
100001051	Shaimaa Hossam	shaimaa.hossam1051@iti.org.eg	19/07/2000	female	1533445566	Qena	Arts	Embedded Systems	Intake 45	New Capital	Personal Productivity & Focus Tracker	https://linkedin.com/in/shaimaa-hossam-1051
100001052	Omar Hany	omar.hany1052@iti.org.eg	25/11/1997	male	1055667733	Fayoum	Commerce	Embedded Systems	Intake 45	New Capital	Personal Productivity & Focus Tracker	https://linkedin.com/in/omar-hany-1052
100001053	Esraa Aly	esraa.aly1053@iti.org.eg	11/09/1998	female	1111223355	Alexandria	Engineering	Embedded Systems	Intake 45	New Capital	Student Progress Timeline Generator	https://linkedin.com/in/esraa-aly-1053
100001054	Hany Mostafa	hany.mostafa1054@iti.org.eg	15/03/1997	male	1077889966	Cairo	Science	Embedded Systems	Intake 45	New Capital	Student Progress Timeline Generator	https://linkedin.com/in/hany-mostafa-1054
100001055	Marwa Said	marwa.said1055@iti.org.eg	22/06/1999	female	1233445577	Tanta	Computer Science	Embedded Systems	Intake 45	New Capital	Student Progress Timeline Generator	https://linkedin.com/in/marwa-said-1055

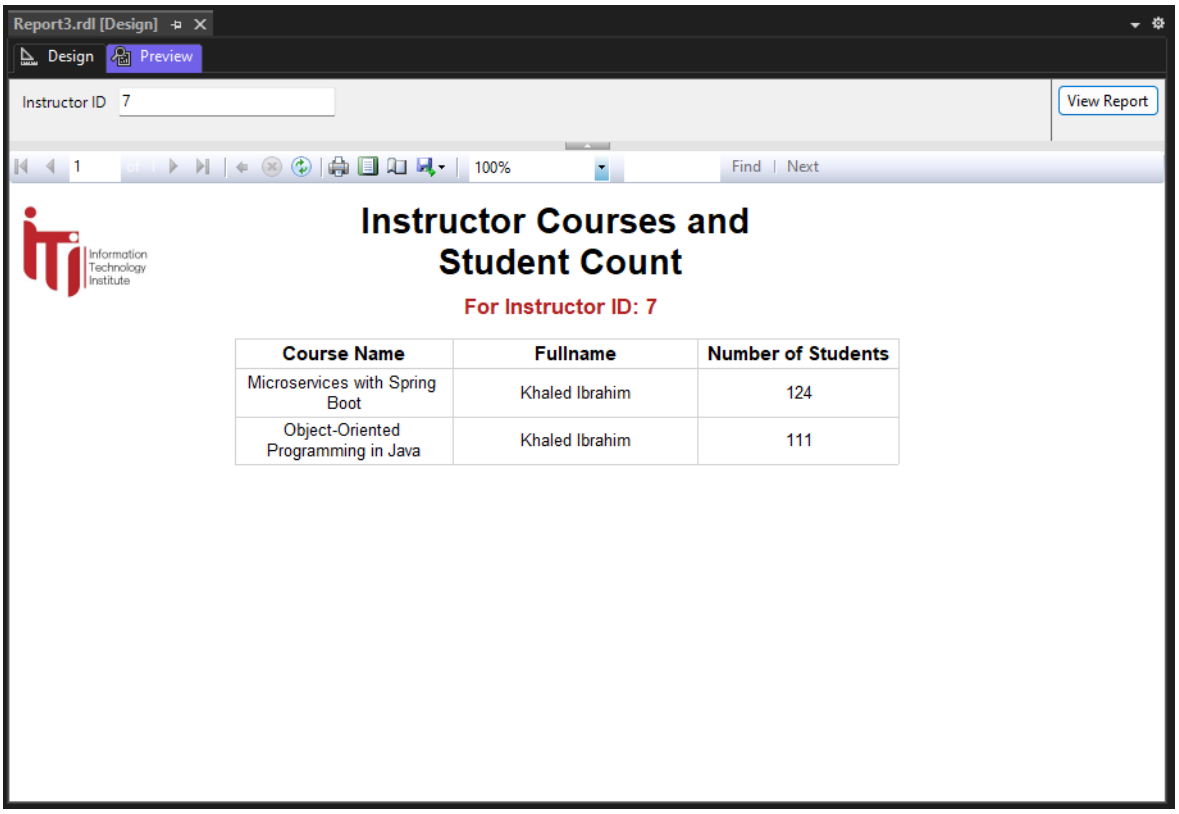
- **Report 2:** Report that takes the student ID and returns the grades of the student in all courses. %

```
CREATE OR ALTER PROCEDURE [dbo].[report_2] @Student_ID INT AS BEGIN
SELECT s.Fname+' '+s.Lname AS Fulname,
       c.Course_Name AS [Course Name],
       concat('%', cast(se.Exam_Grade AS float)/cast(e.Exam_Total_Marks AS float)*100) AS [Course Grade]
FROM Student s
INNER JOIN Student_Exam se ON s.Student_ID = se.Student_ID
INNER JOIN Exam e ON se.Exam_ID = e.Exam_ID
INNER JOIN Exam_Course ec ON e.Exam_ID = ec.Exam_ID
INNER JOIN Course c ON ec.Course_ID = c.Course_ID
WHERE s.Student_ID = @Student_ID END
```



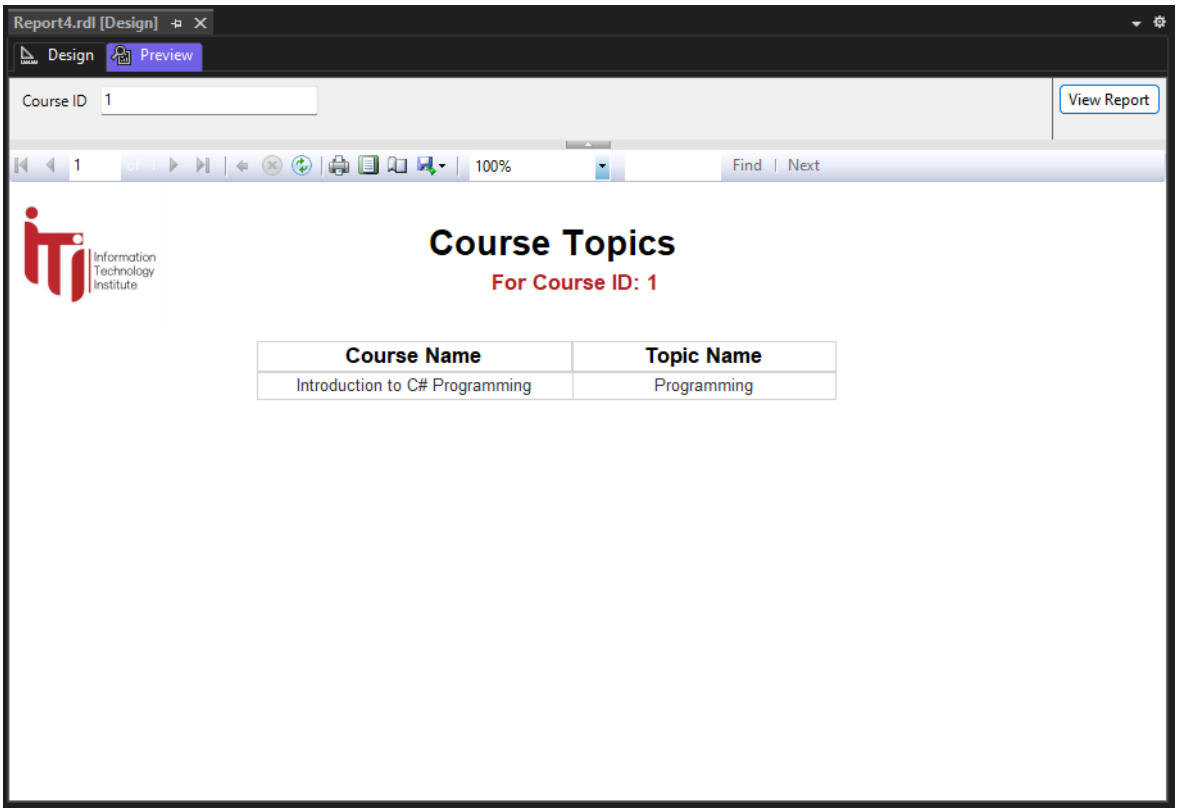
- **Report 3:** Report that takes the instructor ID and returns the name of the courses that he teaches and the number of students per course.

```
CREATE OR ALTER PROCEDURE [dbo].[report_3] @Instructor_ID INT AS BEGIN
SELECT c.Course_Name AS [Course Name],
       i.Fname+' '+i.Lname AS Fullname,
       COUNT(s.Student_ID) AS [Number of Students]
FROM Student s
INNER JOIN Student_Course sc ON sc.Student_ID=s.Student_ID
INNER JOIN Course c ON c.Course_ID = sc.Course_ID
INNER JOIN Instructor_Course ic ON c.Course_ID = ic.Course_ID
INNER JOIN Instructor i ON i.Instructor_ID = ic.Instructor_ID
WHERE i.Instructor_ID = @Instructor_ID
GROUP BY c.Course_Name,
         i.Fname+' '+i.Lname END
```



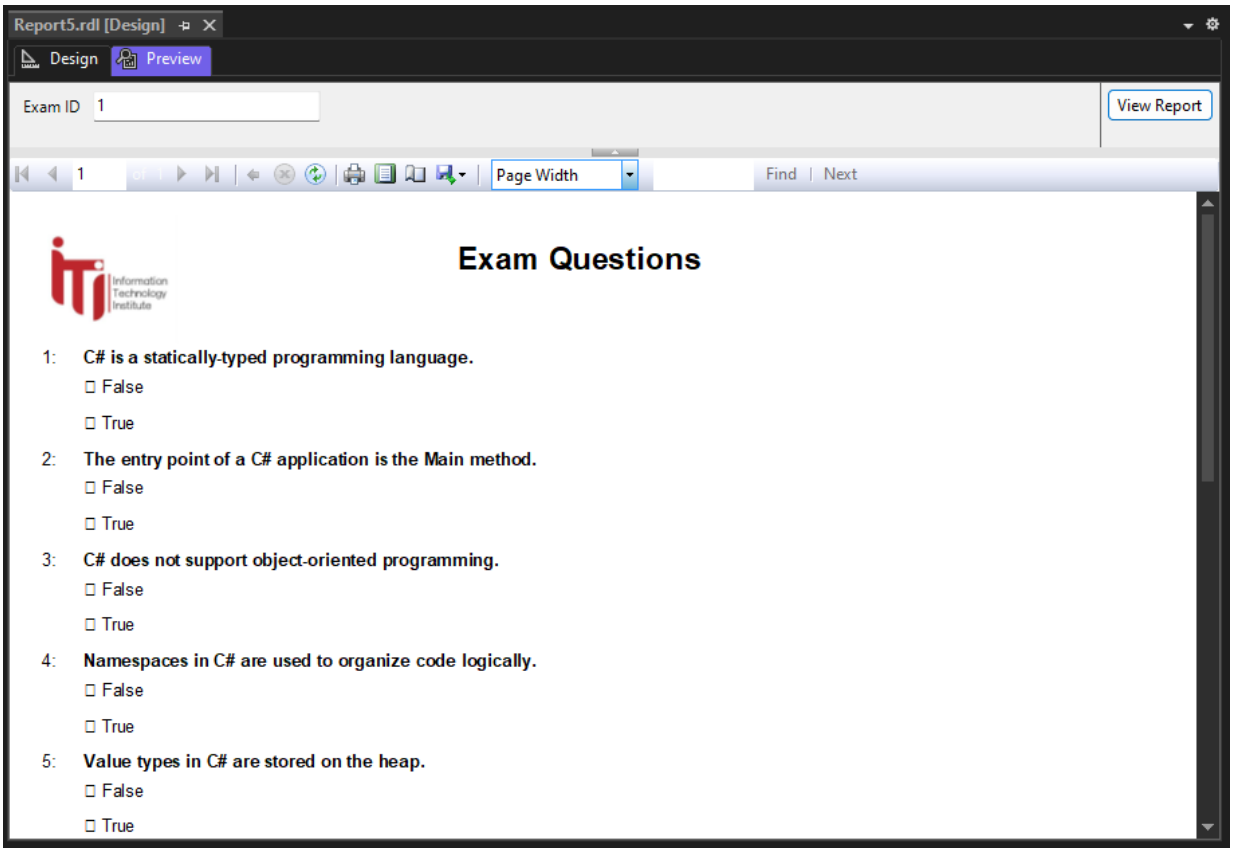
- **Report 4:** Report that takes course ID and returns its topics.

```
CREATE OR ALTER PROCEDURE [dbo].[report_4] @Course_ID INT AS BEGIN
SELECT c.Course_Name,
       t.Topic_Name
FROM Course c
INNER JOIN Topic t ON c.Topic_ID=t.Topic_ID
WHERE c.Course_ID=@Course_ID END
```



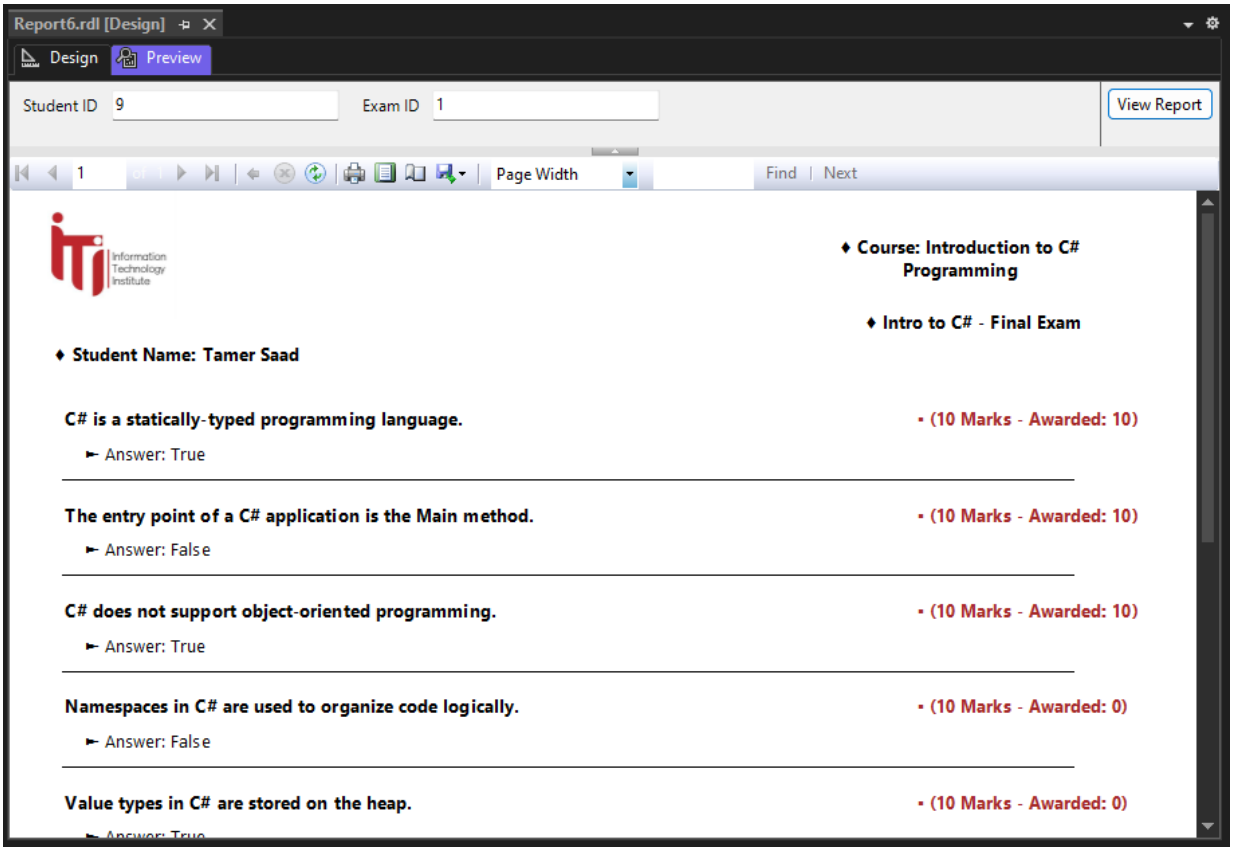
- **Report 5:** Report that takes exam number and returns the Questions in it and choices.

```
CREATE OR ALTER PROCEDURE [dbo].[report_5] @Exam_ID INT AS BEGIN
SELECT q.Question_ID,
       q.Question_Text,
       ct.Choice_Text
FROM Exam e
INNER JOIN Exam_Question eq ON e.Exam_ID=eq.Exam_ID
INNER JOIN Question q ON eq.Question_ID=q.Question_ID
INNER JOIN Choices c ON c.Question_ID=q.Question_ID
INNER JOIN Choice_Text ct ON c.Choices_ID=ct.Choices_ID
WHERE e.Exam_ID=@Exam_ID END
```



- **Report 6:** Report that takes exam number and the student ID then returns the Questions in this exam with the student answers.

```sql
CREATE OR ALTER PROCEDURE [dbo].[report_6] @Student_ID INT, @Exam_ID INT AS BEGIN
SELECT s.Student_ID,
       s.Fname + ' ' + s.Lname AS FullName,
       e.Exam_ID,
       e.Exam_Name,
       q.Question_ID,
       q.Question_Text,
       q.Question_Mark,
       seq.Point_Awarded,
       c.course_name,
       seq.ST_answer AS StudentAnswer
FROM Student_Exam_Question seq
INNER JOIN Question q ON seq.Question_ID = q.Question_ID
INNER JOIN Student s ON s.Student_ID = seq.Student_ID
INNER JOIN Exam e ON seq.Exam_ID = e.Exam_ID
INNER JOIN Exam_Course ec ON ec.Exam_ID = e.Exam_ID
INNER JOIN Course c ON c.Course_ID = ec.Course_ID
WHERE seq.Student_ID = @Student_ID
  AND seq.Exam_ID = @Exam_ID
ORDER BY q.Question_ID END
```

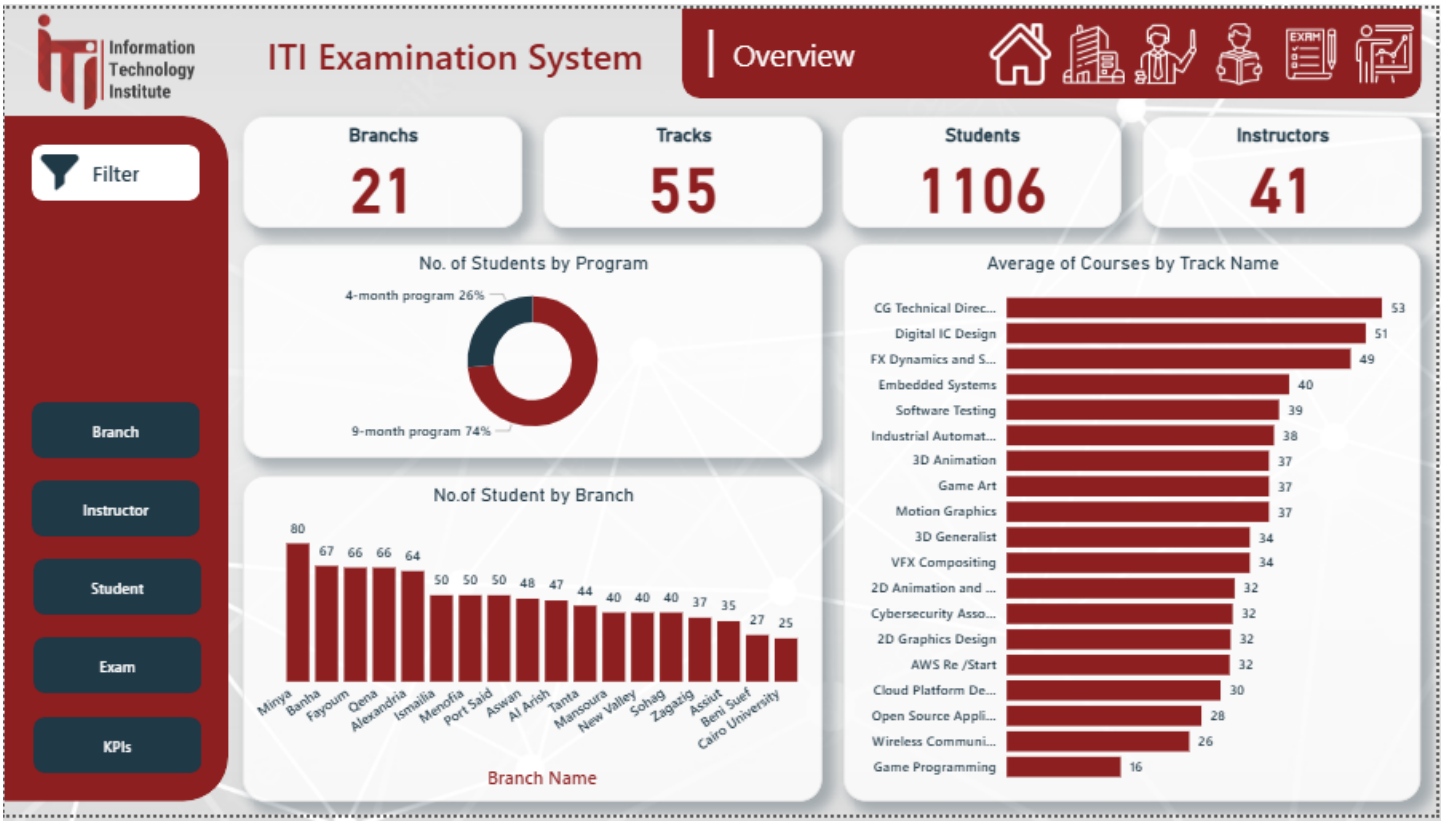


10. Insights & visualization using Power BI

As part of the project, 20 interactive dashboards were developed using Power BI to provide meaningful insights into various aspects of the ITI Examination System.
These dashboards cover key areas such as students, instructors, exams, courses, branches, partnering companies, certificates, freelance activities, and graduation projects.
Each dashboard was designed to support data-driven decision making, offering visual summaries, trends, and performance indicators.
The use of filters and interactive elements allows users to explore the data dynamically and focus on specific segments or metrics as needed.
The main objective of these visualizations was to identify patterns, track performance, and improve the overall efficiency of academic and administrative processes.
These insights empowered stakeholders to make informed decisions based on real-time data and comprehensive analytics.

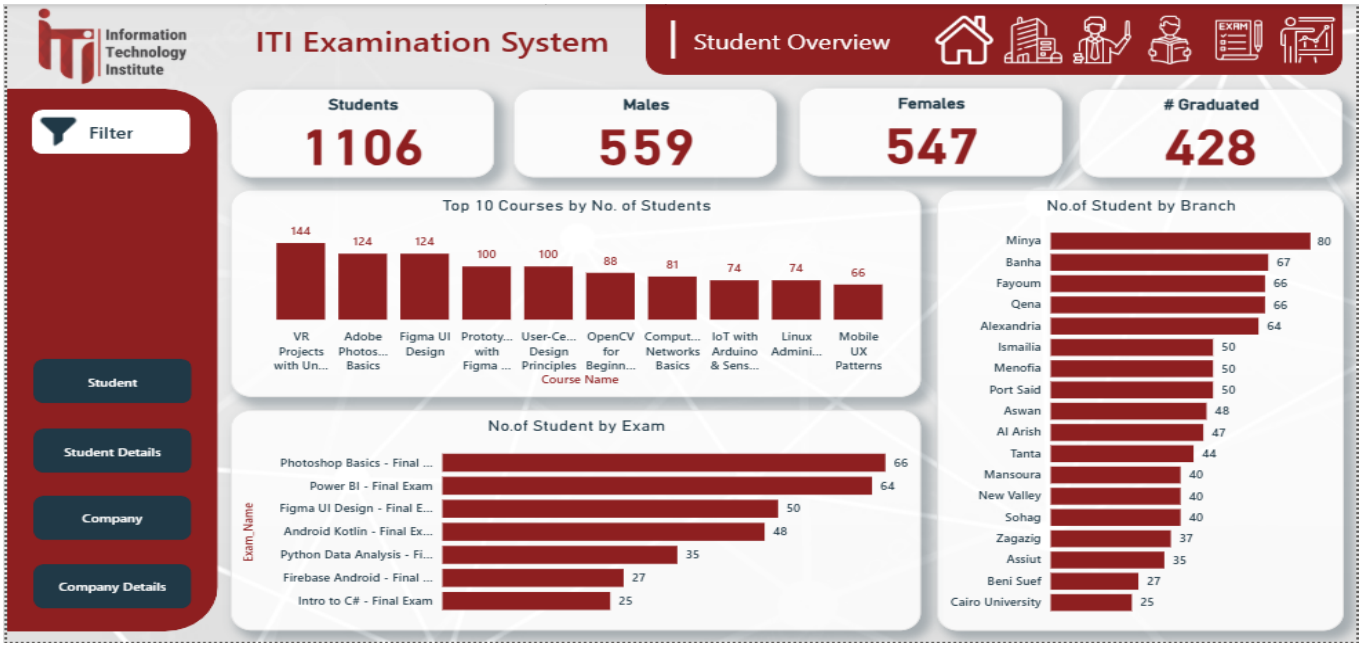
- **Overview dashboard**

The dashboard offers a high-level summary of the ITI Examination System, presenting essential statistics such as the total number of branches, tracks, students, and instructors. It also includes visual insights into student distribution by program and branch, as well as the average number of courses per track.



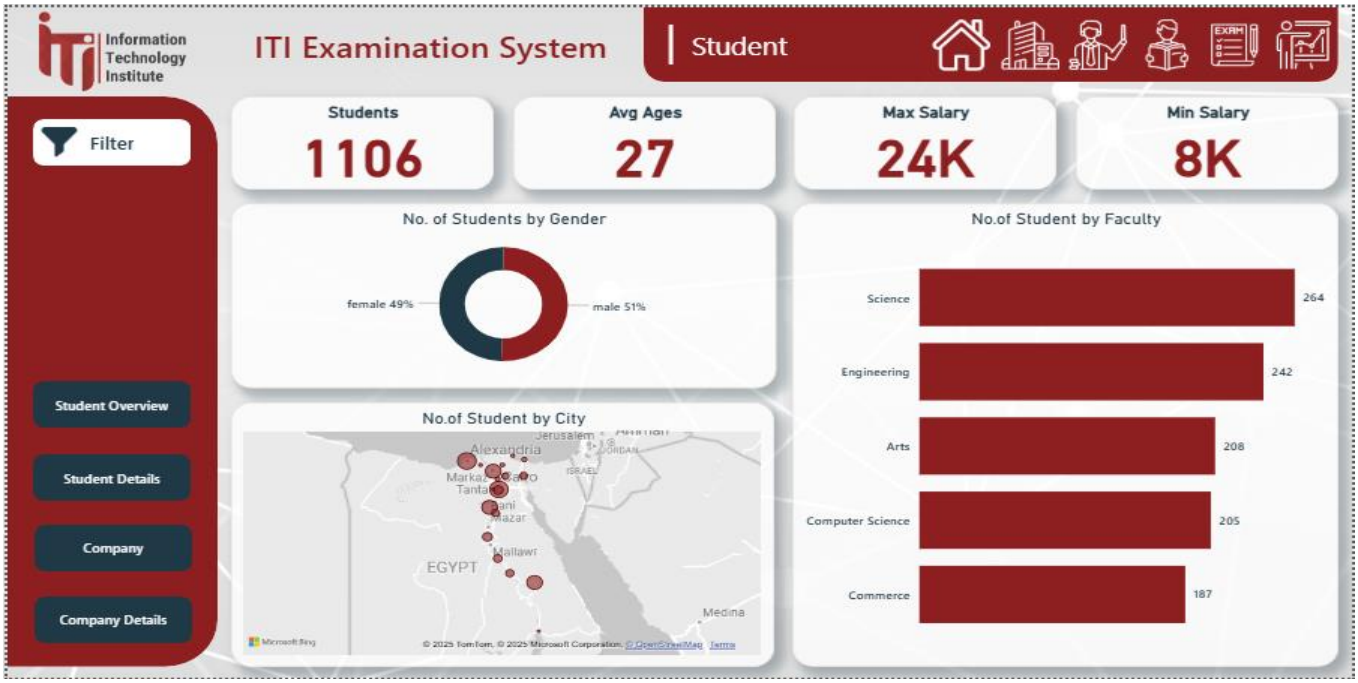
- **Student overview dashboard**

This dashboard summarizes key student metrics, including total count, gender distribution, and graduation numbers.
It highlights the most attended courses, exam participation, and student distribution across ITI branches.



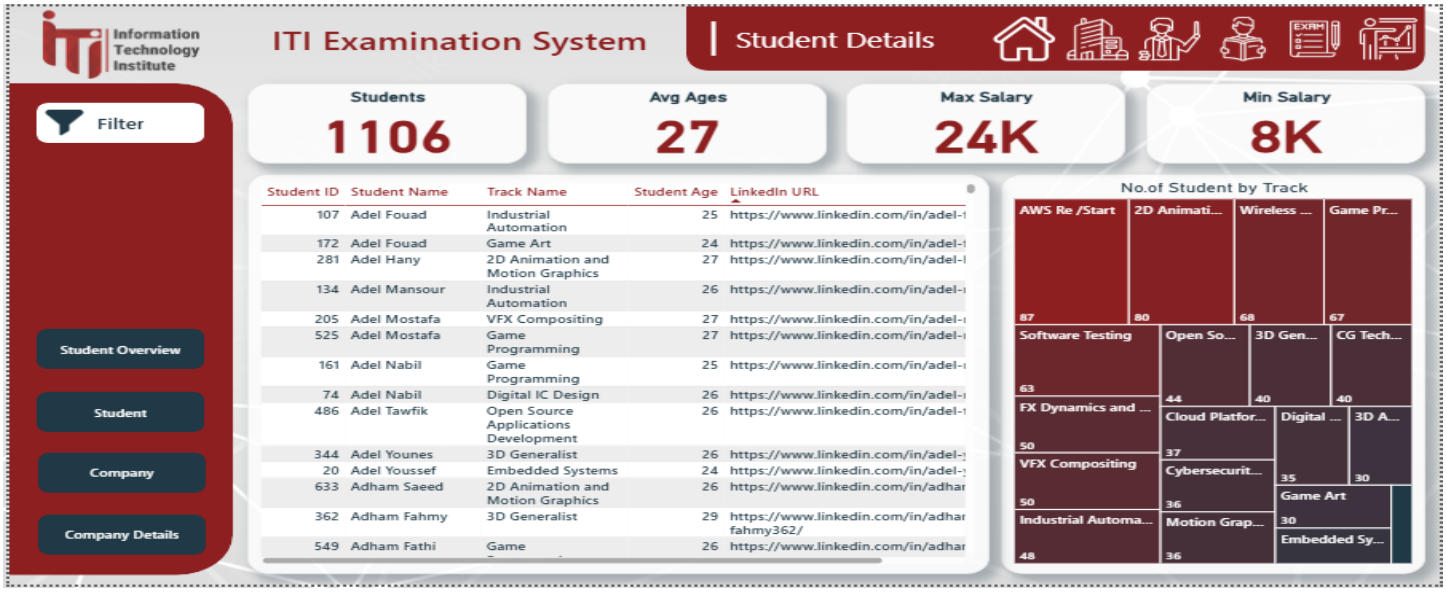
- **Student dashboard**

This dashboard presents demographic and academic data about students, including average age, gender distribution, and salary range.
It also visualizes student distribution by city and faculty, supporting targeted analysis of educational backgrounds and regional trends.



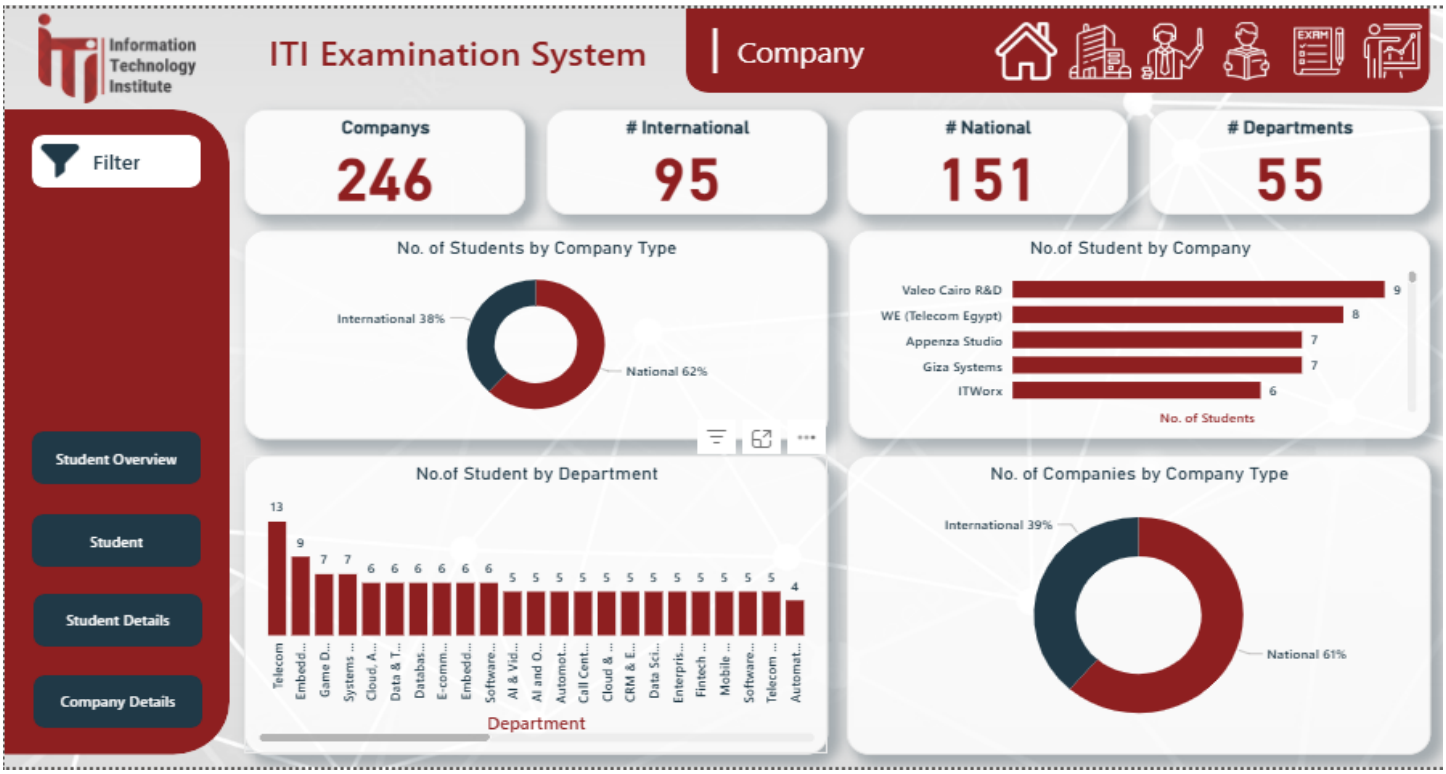
- **Student Details dashboard**

This dashboard provides a comprehensive overview of student details, including the total number of students, average age, and salary range.
It also highlights student distribution across various technical tracks, enabling insights into specialization trends and training focus areas



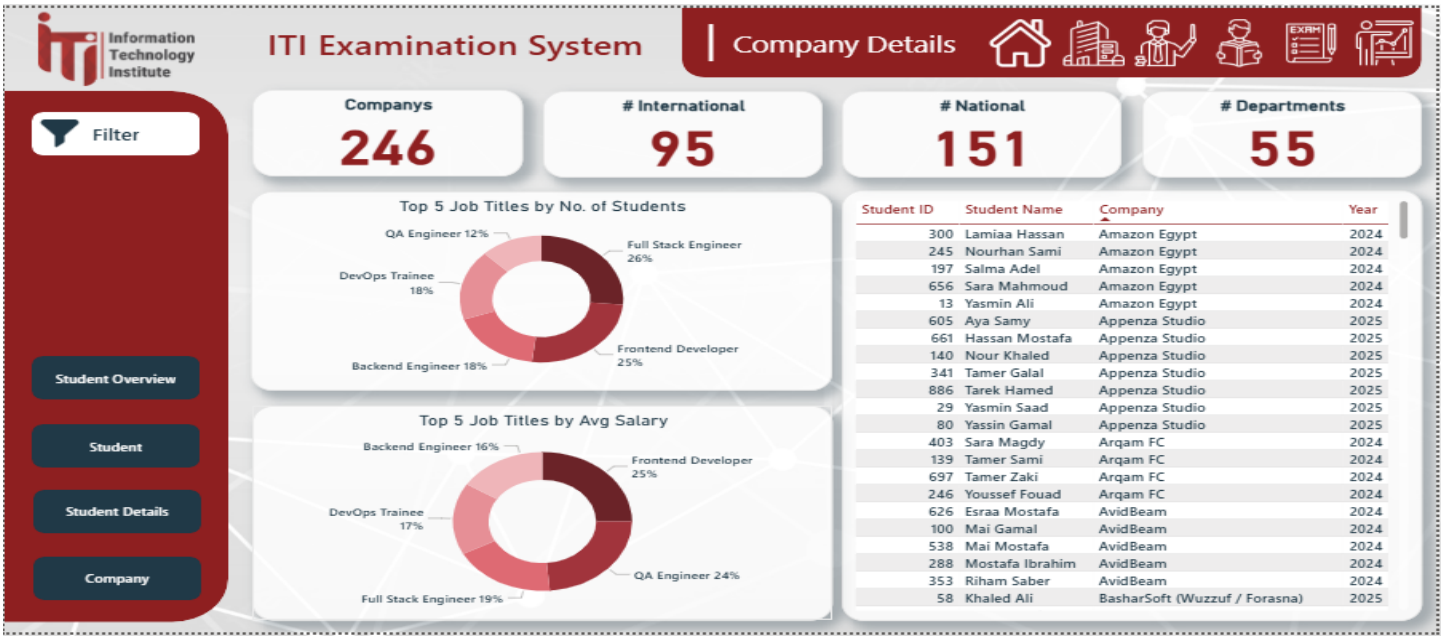
- **Company dashboard**

This dashboard summarizes student-company relationships, showing the number of partnering companies, their national or international status, and student distribution across departments.
It also visualizes key company engagement metrics, enabling analysis of collaboration trends and departmental placement patterns.



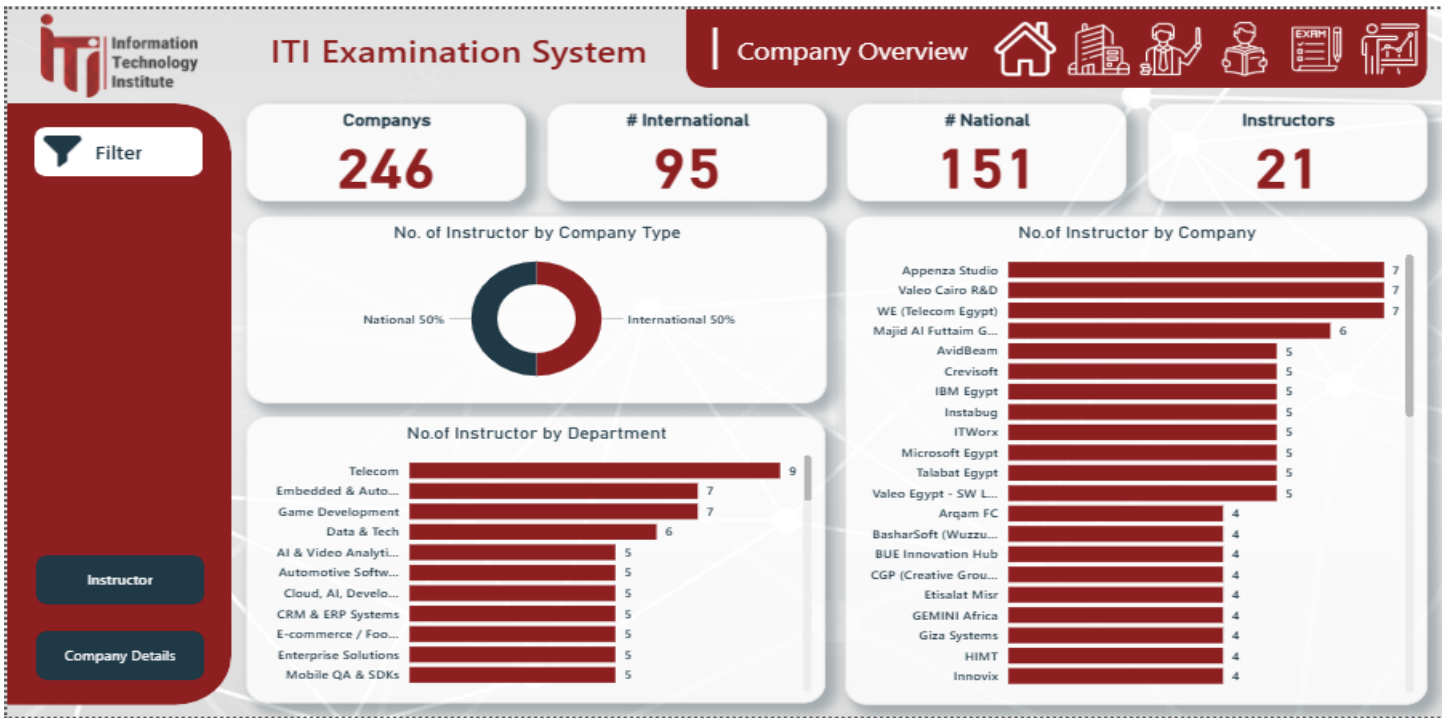
- **Company Details dashboard**

This dashboard provides insights into the top job titles secured by students, both by count and average salary, highlighting career trends and industry demand.
It also presents a detailed mapping of student placements across companies and graduation years, supporting evaluation of hiring patterns and institutional partnerships.



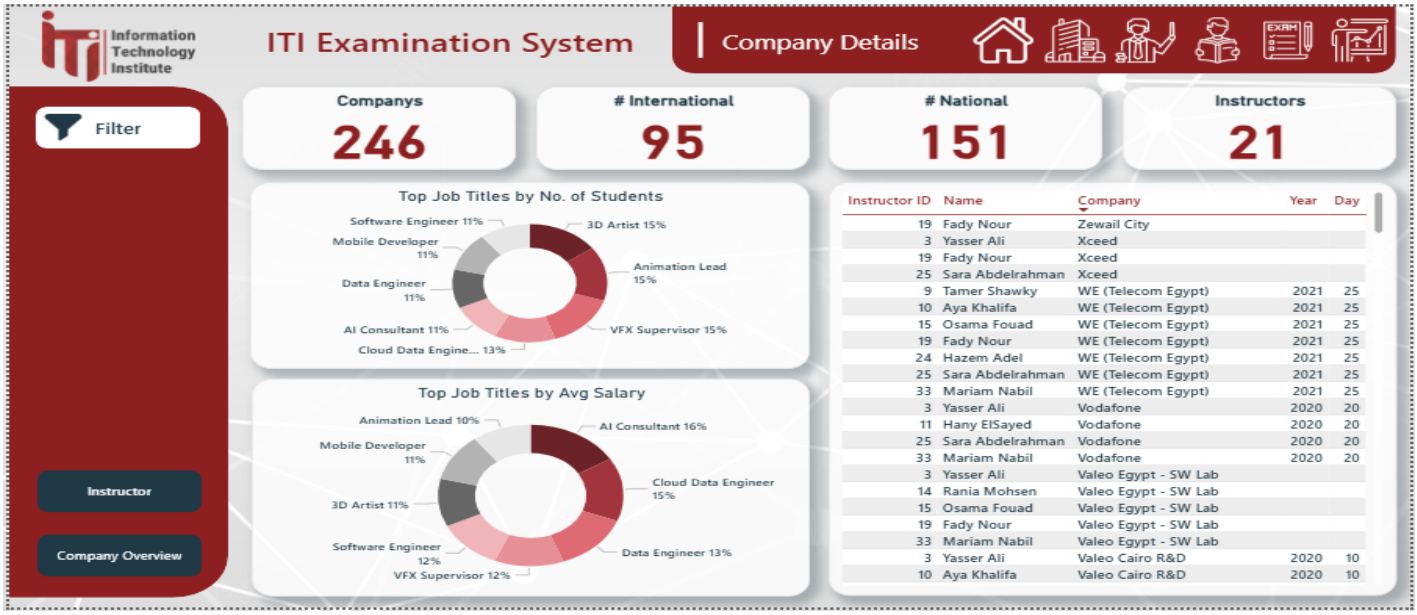
- **Company Overview dashboard**

This dashboard provides a comprehensive overview of student details, including the total number of students, average age, and salary range.
It also highlights student distribution across various technical tracks, enabling insights into specialization trends and training focus areas.



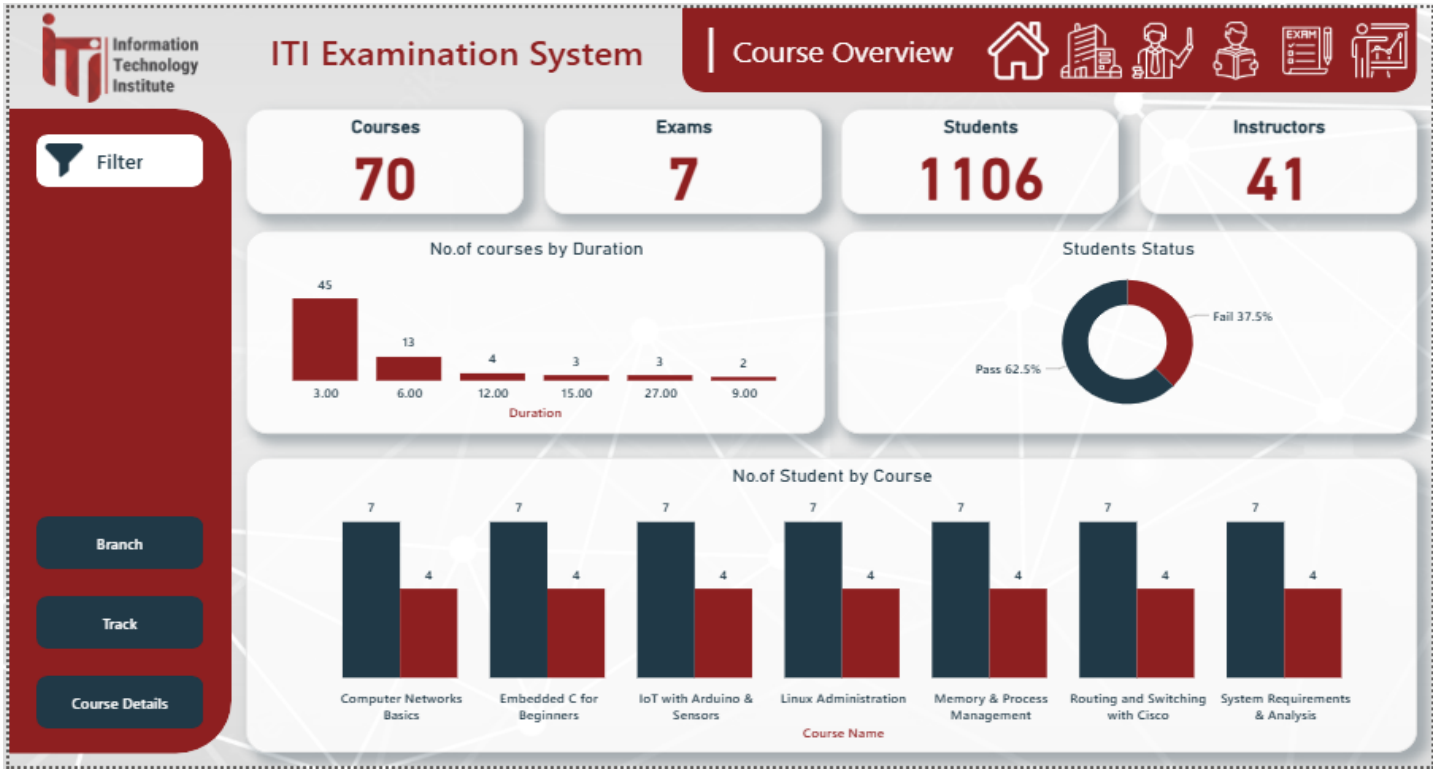
- **Company Details dashboard**

This dashboard provides insights into the top job titles secured by students, both by count and average salary, highlighting career trends and industry demand.
It also presents a detailed mapping of student placements across companies and graduation years, supporting evaluation of hiring patterns and institutional partnerships.



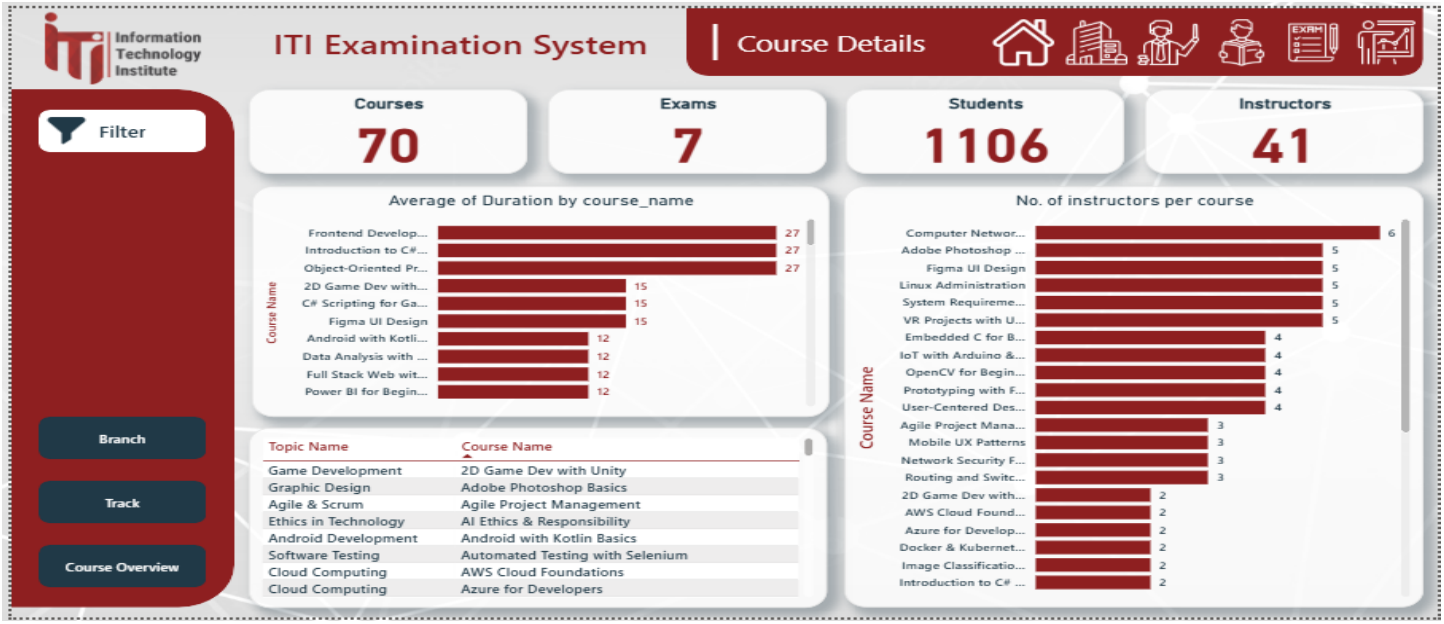
- **Course Overview dashboard**

This dashboard summarizes course performance and participation at ITI, including total courses, exams, students, and instructors.
It highlights course durations, student success rates, and enrollment distribution across key technical subjects



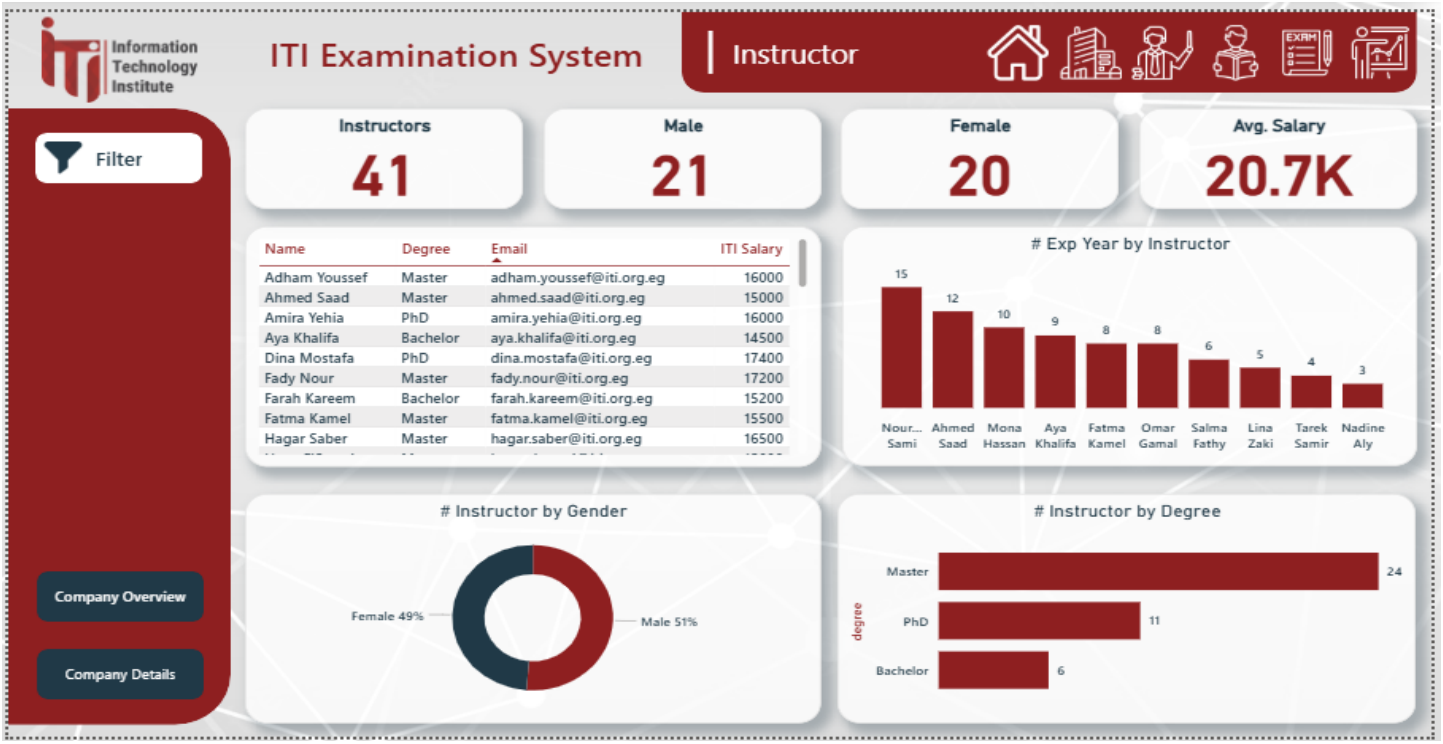
- **Course Details dashboard**

This dashboard offers detailed insights into individual courses, showcasing their average durations and the number of instructors assigned per course.
It also maps courses to their corresponding topics, supporting deeper analysis of instructor allocation and training scope.



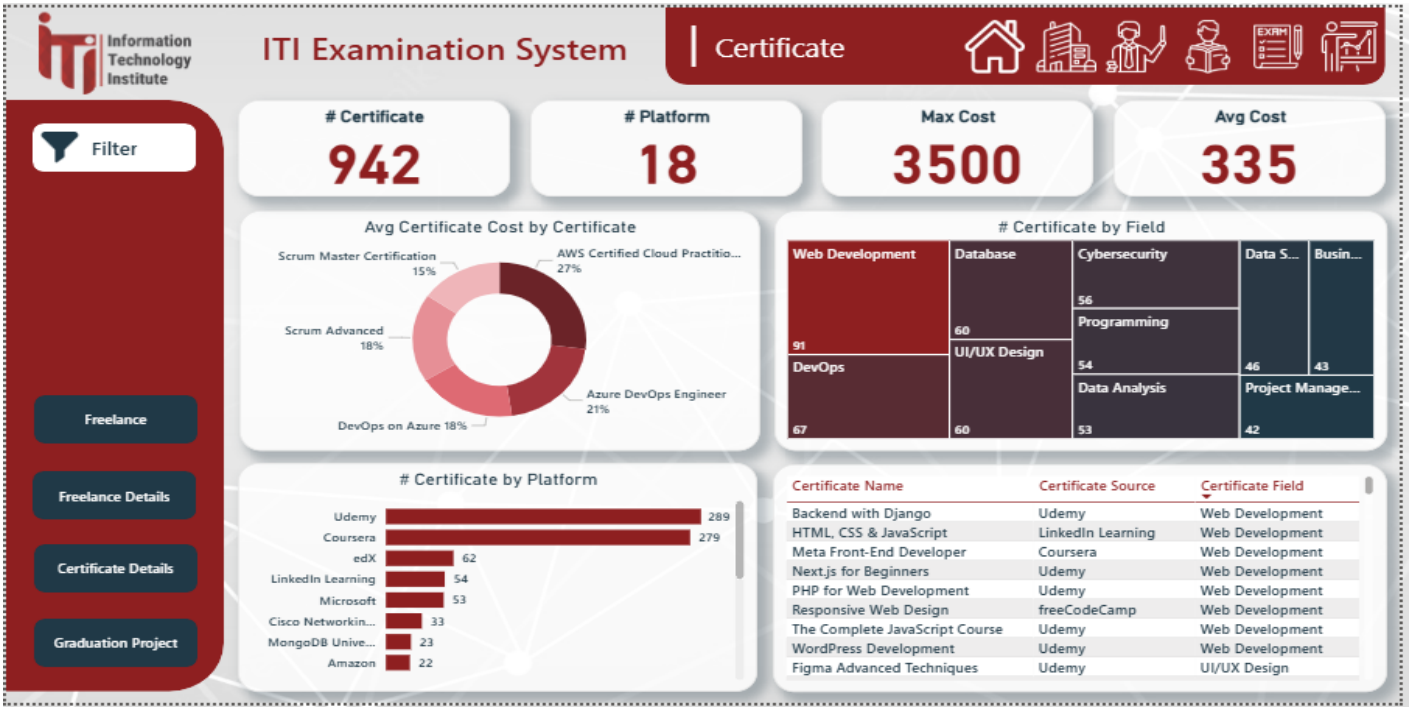
- **Instructor dashboard**

This dashboard provides comprehensive insights into instructor demographics and academic qualifications, displaying key metrics such as gender distribution, average salaries, and degree attainment levels.
The visualization also highlights instructor experience years and geographic distribution, enabling data-driven analysis of workforce patterns and regional educational trends.



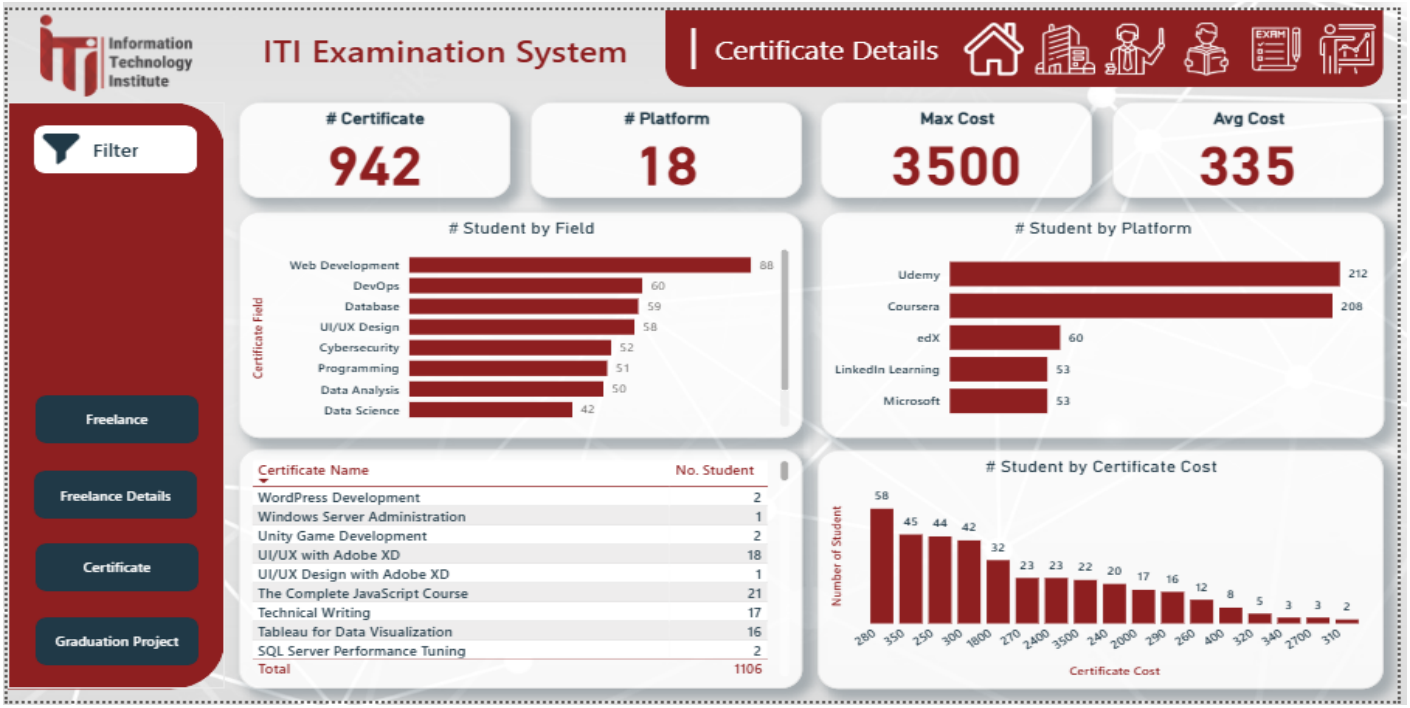
- **Certificate dashboard**

This dashboard tracks certification costs, platforms, and fields, highlighting trends like cloud and Scrum certifications. It helps analyze skill demand and training focus across web development, DevOps, and UI/UX design.



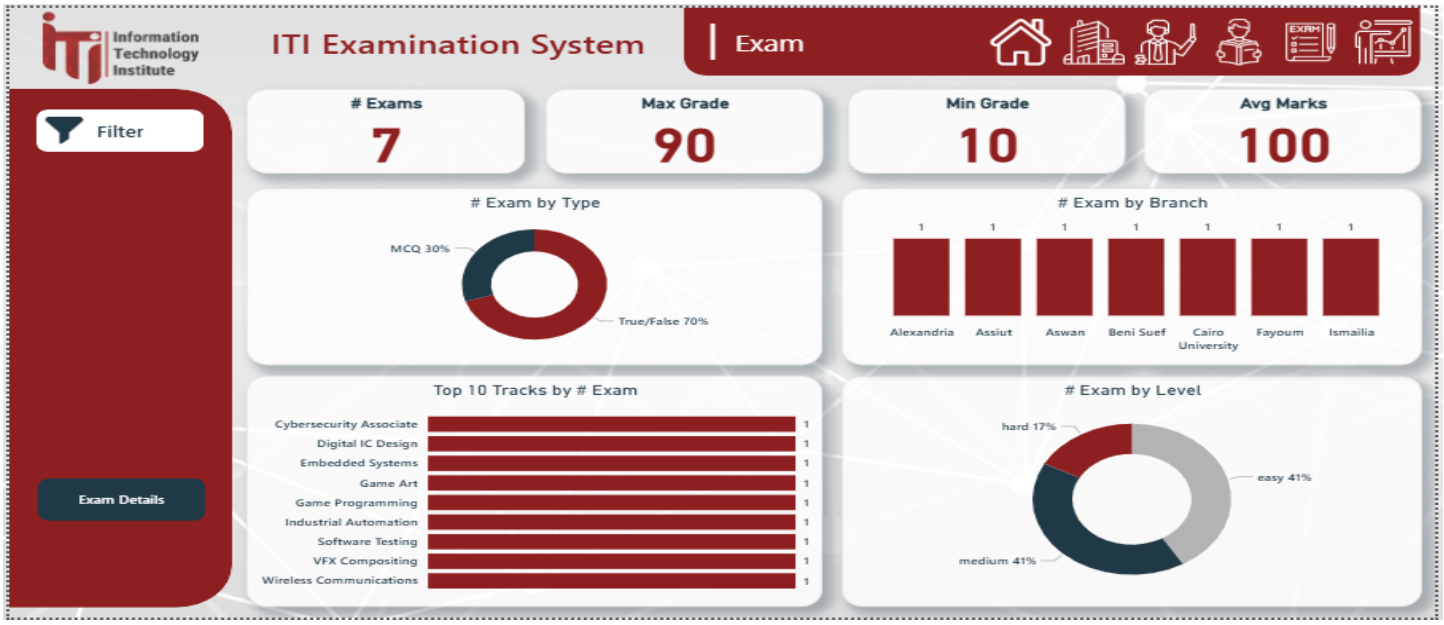
- **Certificate Details dashboard**

This dashboard provides detailed insights into student certification data, including the number of certificates, platforms used, and average costs.
It also highlights student distribution across various fields, platforms, and certificate costs, enabling effective analysis of learning trends and preferences.



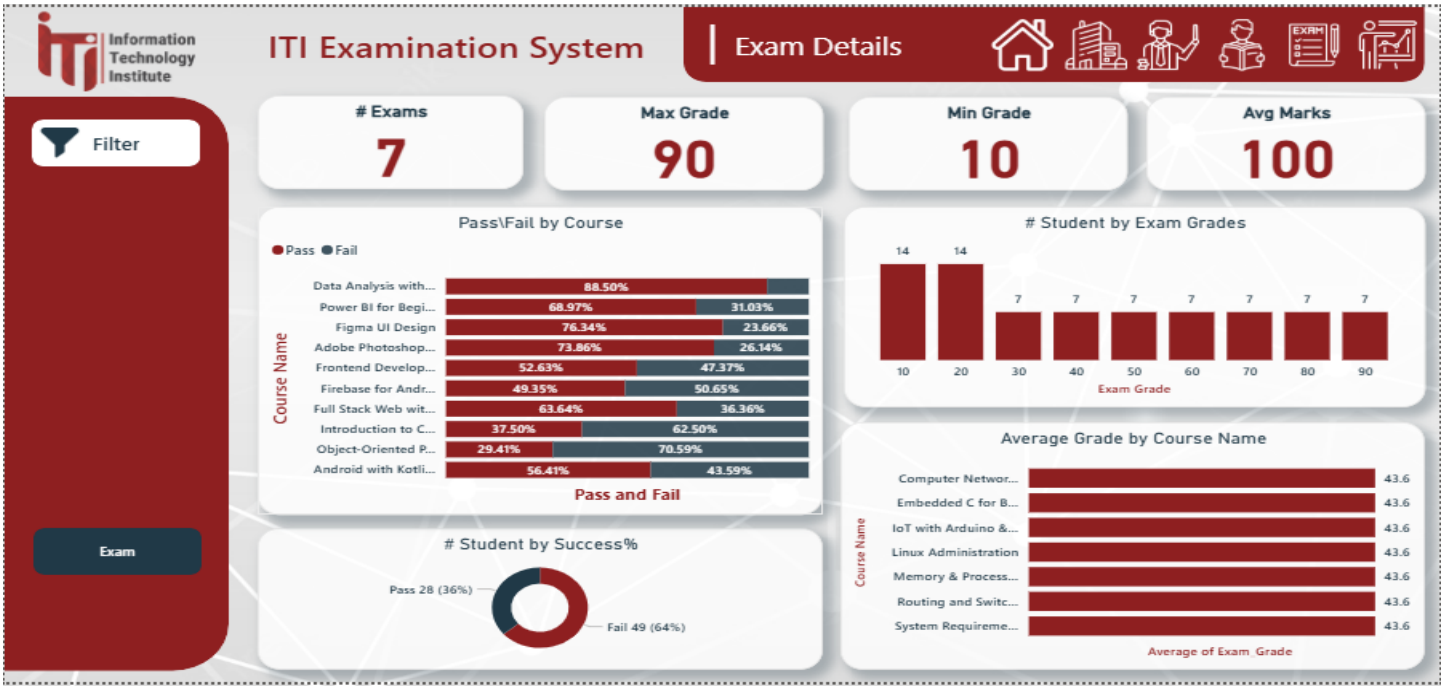
- **Exam dashboard**

This dashboard illustrates key metrics related to student exams, including the number of exams, grade distribution, and average marks.
It also visualizes exams by type, level, track, and branch, supporting a comprehensive evaluation of academic performance and exam difficulty.



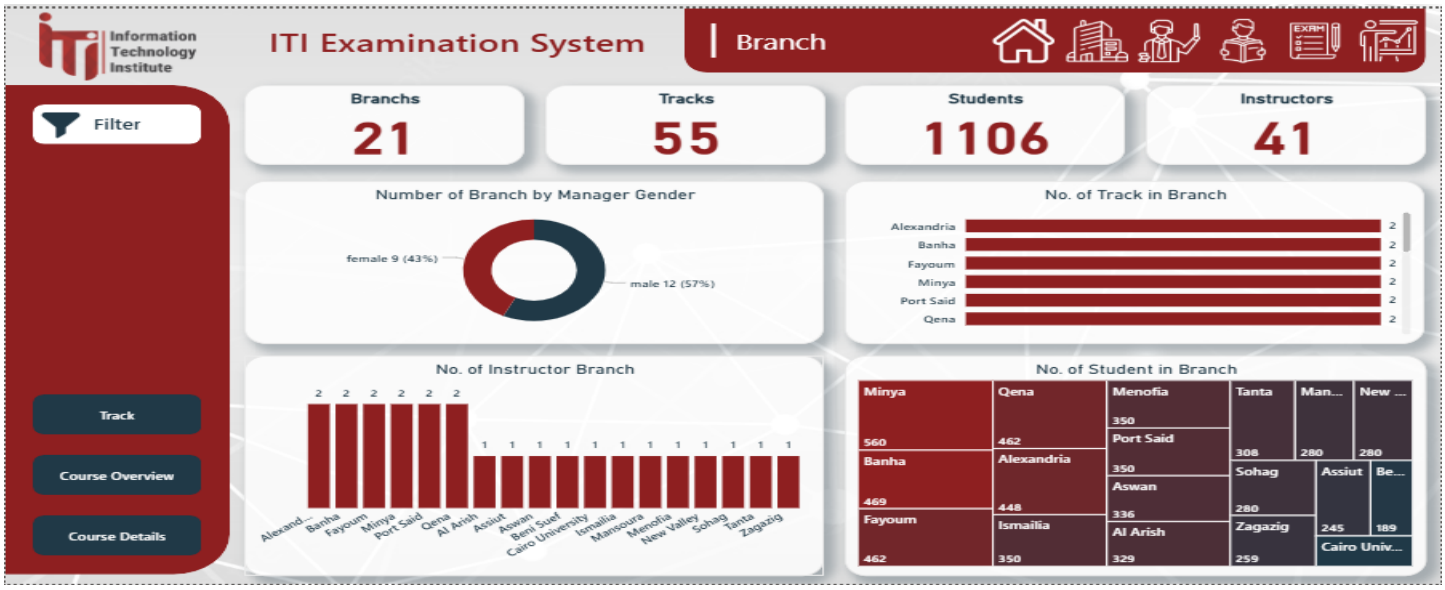
- **Exam Details dashboard**

This dashboard visualizes pass/fail rates and average grades by course, highlighting trends like the 36% overall pass rate and top-performing subjects (e.g., Computer Networking at 4.8/5). It enables quick identification of challenging courses (e.g., Object-Oriented Programming with 70.56% fail rate) for targeted academic improvements.



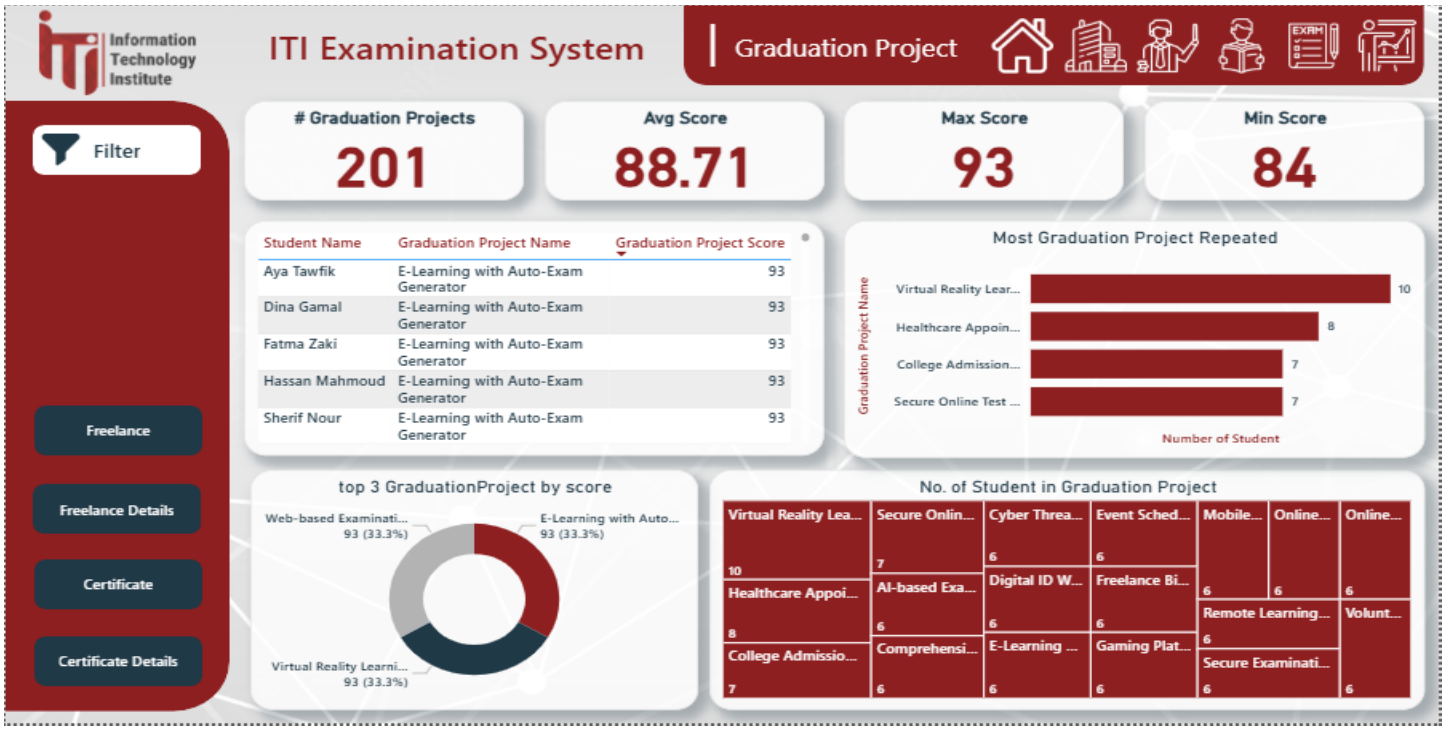
- **Branch dashboard**

This dashboard provides an overview of branch-level statistics, including the number of branches, tracks, students, and instructors.
It also visualizes data such as student and instructor distribution across branches, and gender representation among branch managers.



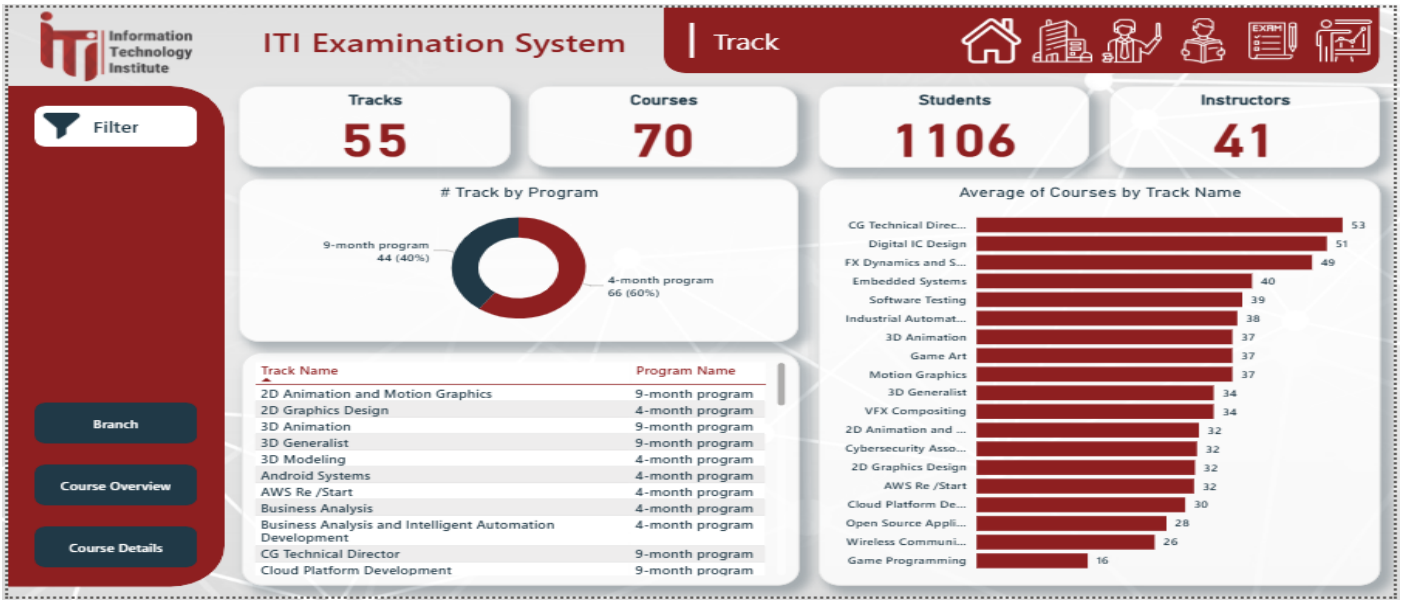
- **Graduation Project dashboard**

This dashboard tracks 201 graduation projects, highlighting top-scoring initiatives like *Web-based Examination System* and *E-Learning with Auto-Exam Generator* (both scoring 93/100). It reveals trends in project topics, student participation, and performance metrics to evaluate academic outcomes and innovation focus areas.



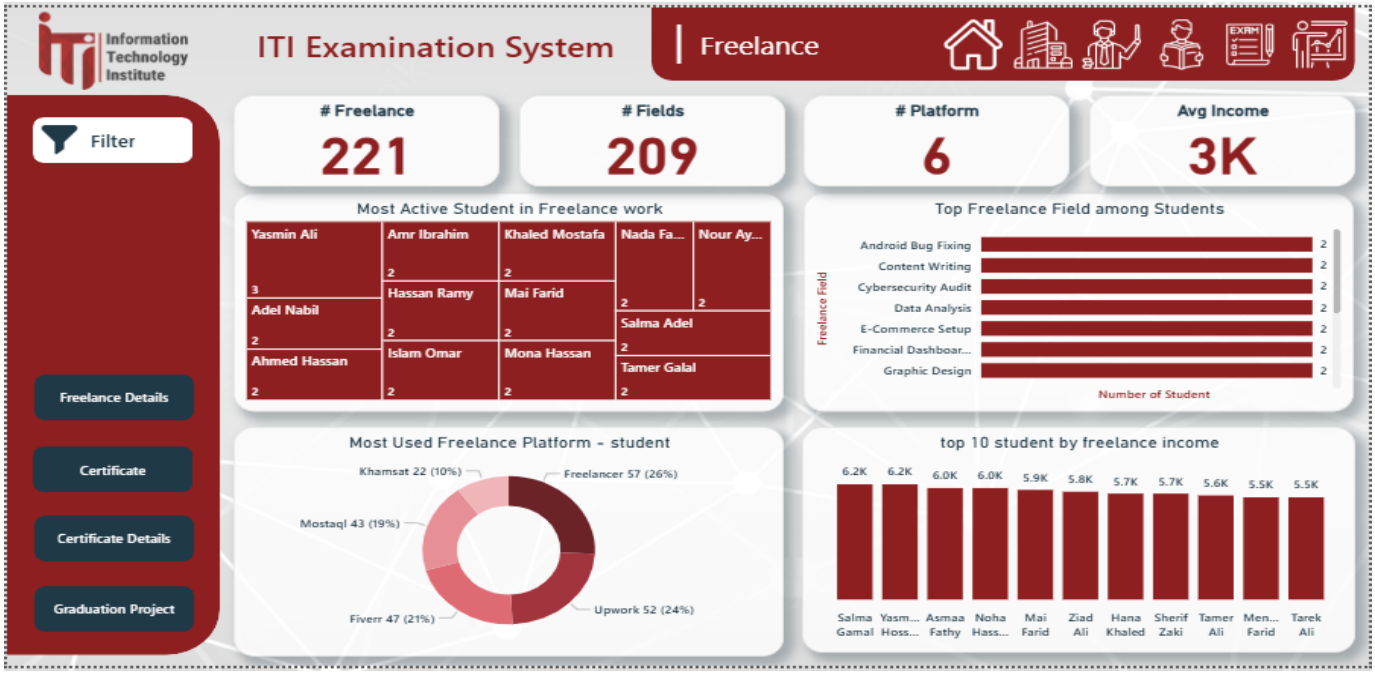
- **Track dashboard**

This dashboard displays detailed information about the tracks offered, including the number of tracks, courses, students, and instructors.
It also compares track types by program duration and visualizes the average number of courses per track, supporting academic planning and program analysis.



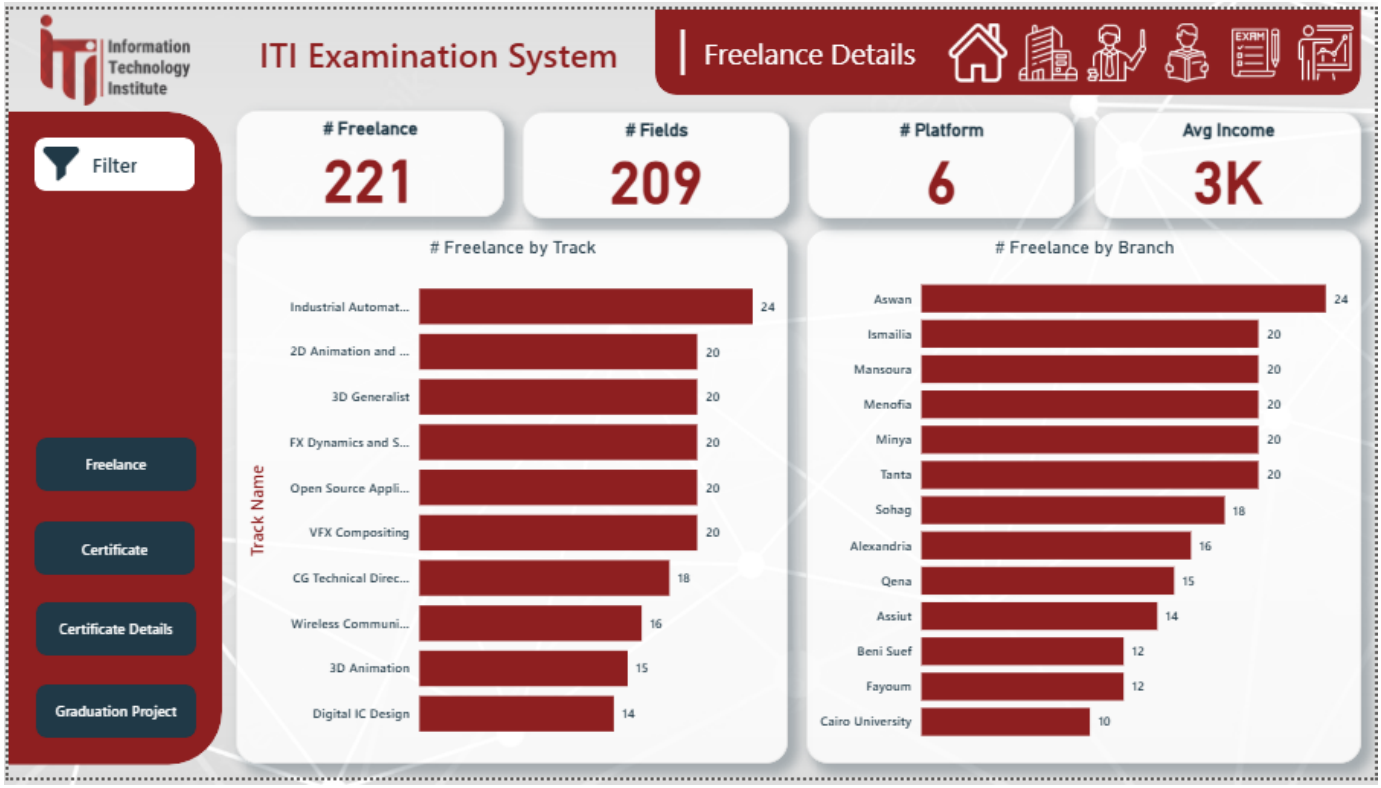
- **Freelance dashboard**

This dashboard presents key data about students' freelance activities, including the number of freelancers, fields, platforms, and average income.
It also highlights the most active students, most used freelance platforms, and top-earning students, supporting analysis of freelance trends and student engagement.



- **Freelance details dashboard**

This dashboard analyzes 221 freelance engagements across 6 platforms, showing top fields like *Industrial Automation* (24 projects) and *3D Animation* (15 projects), with an average income of 3K. It also tracks regional distribution by branch to identify geographic trends in freelance activity

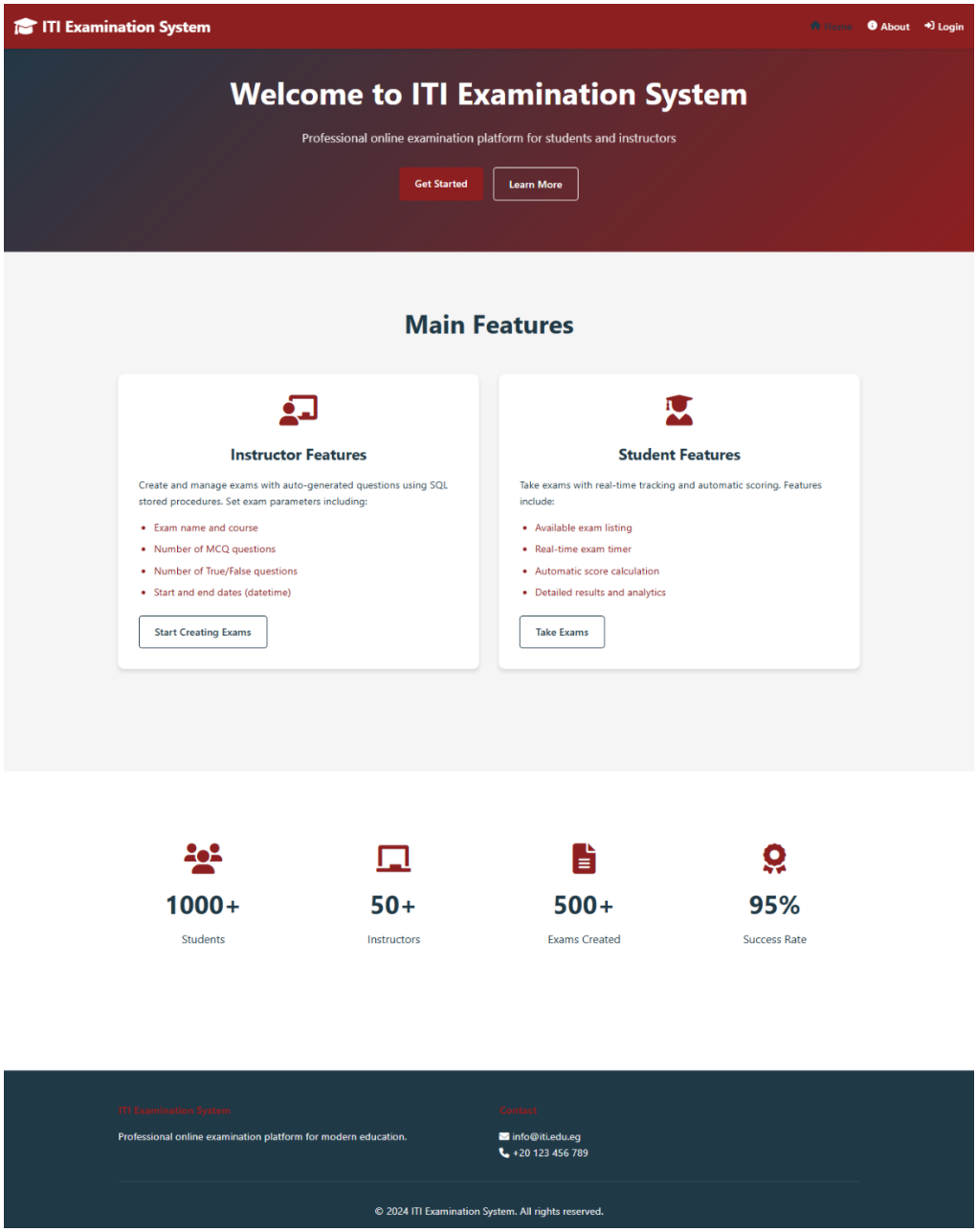


11. Examination System Website

This chapter describes the main user interface components of the Examination System Website. It outlines the core pages available for both instructors and students, along with the functionalities offered on each page. Screenshots are available for each of these pages.

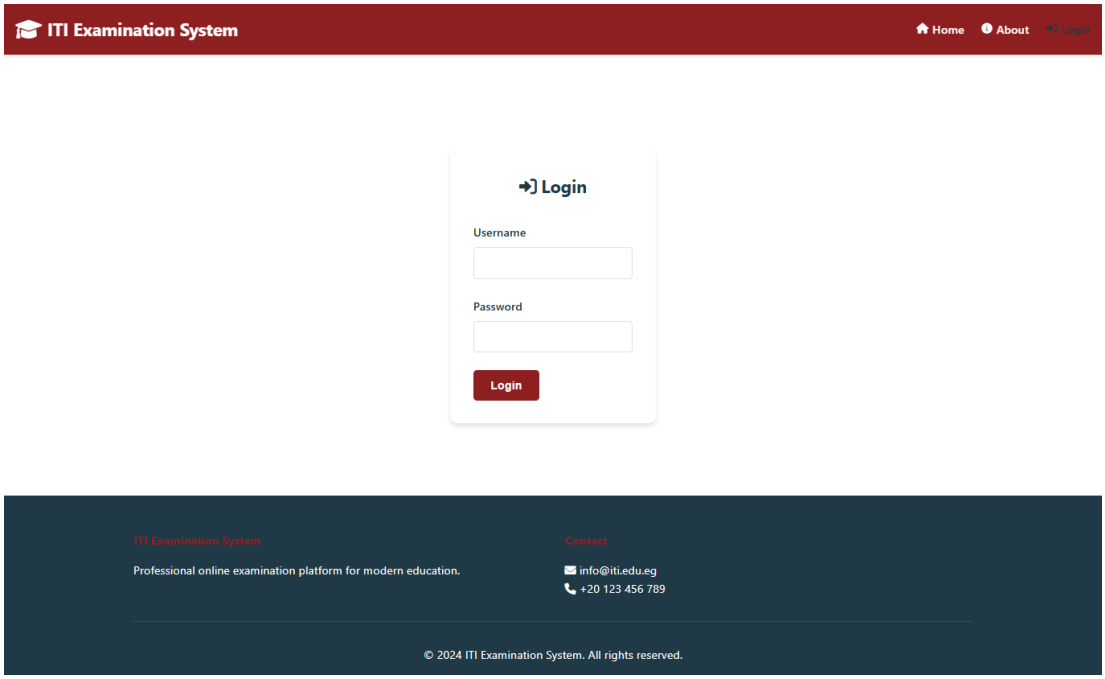
- **Home Page**

The landing page that welcomes users to the system. It typically contains branding, a brief overview of the platform, and navigation to login or further information.



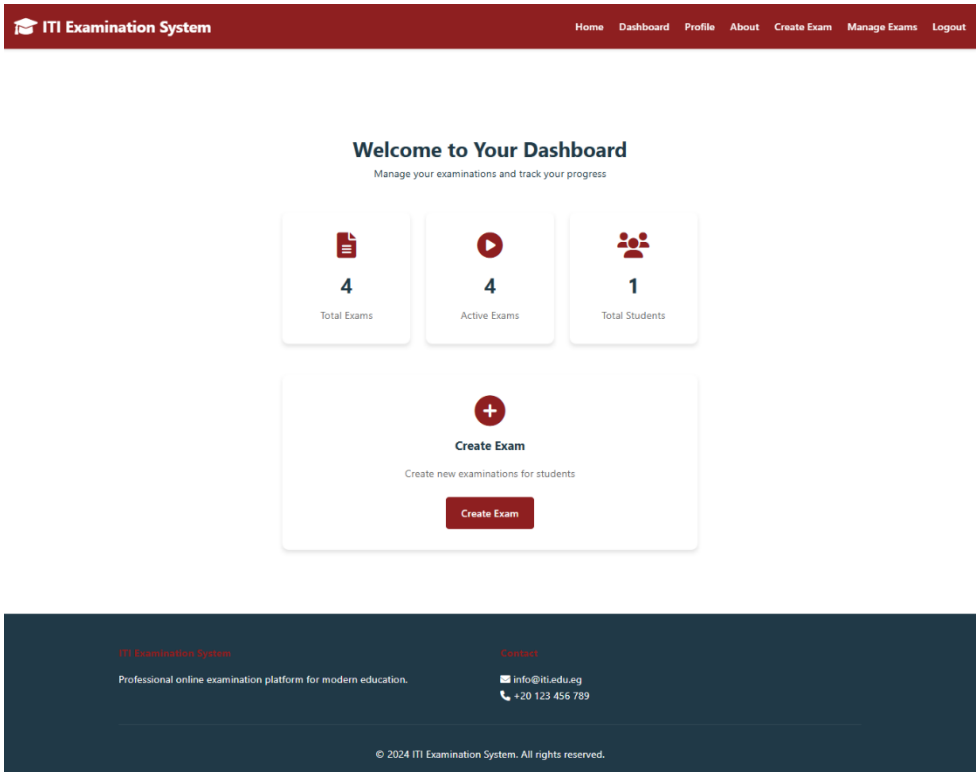
- Login Page

Provides authentication for both instructors and students. Upon successful login, users are redirected to their respective dashboards.



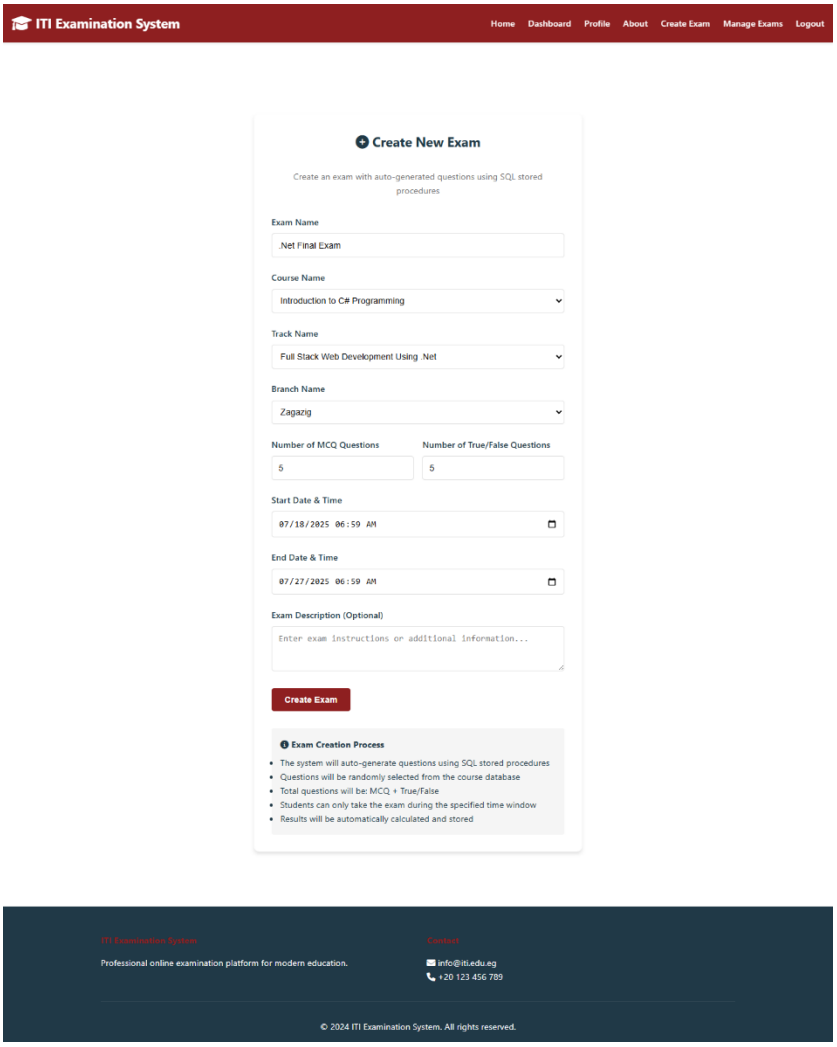
- Instructor Dashboard

A personalized dashboard where instructors can manage exams, track progress, and navigate to other functionalities such as creating or editing exams.



- Create Exam Page

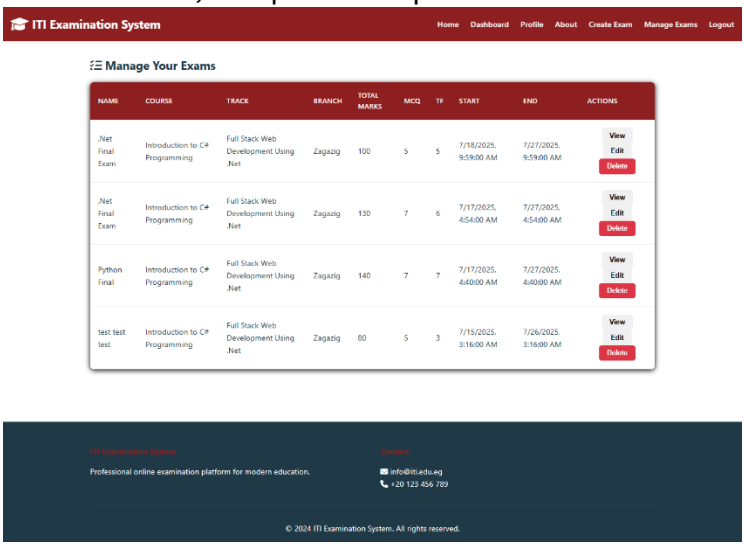
Enables instructors to generate a new exam by selecting course-related questions, defining start/end times, and assigning the exam to a track.



- Manage Exams Page

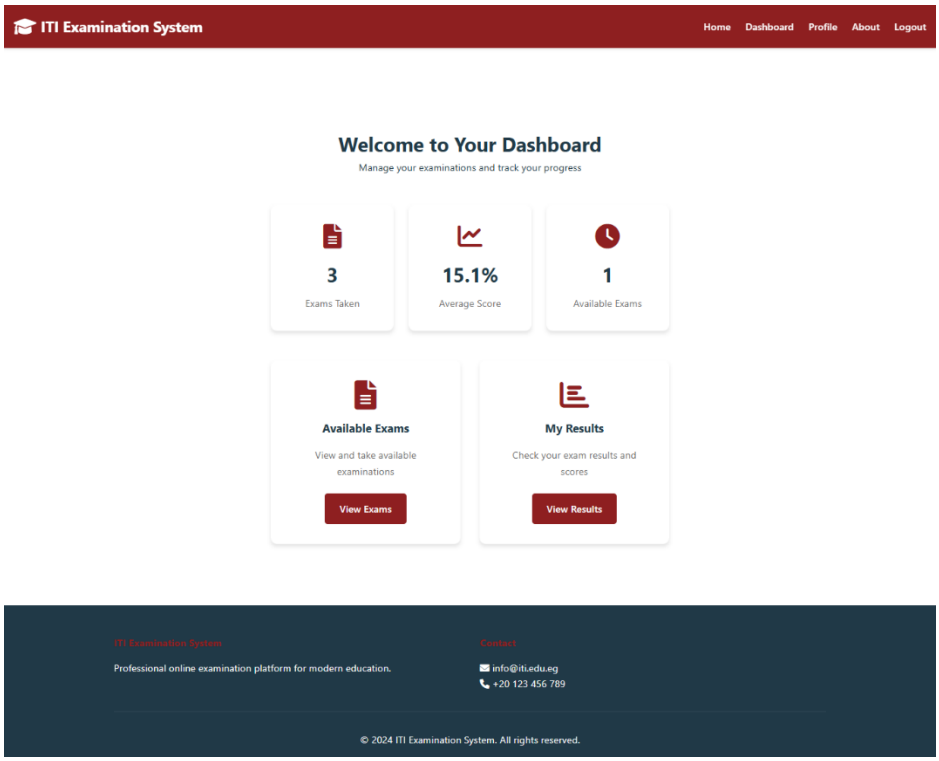
Allows instructors to view all their created exams, and provides options to:

1. View exam details
2. Edit existing exams
3. Delete exams



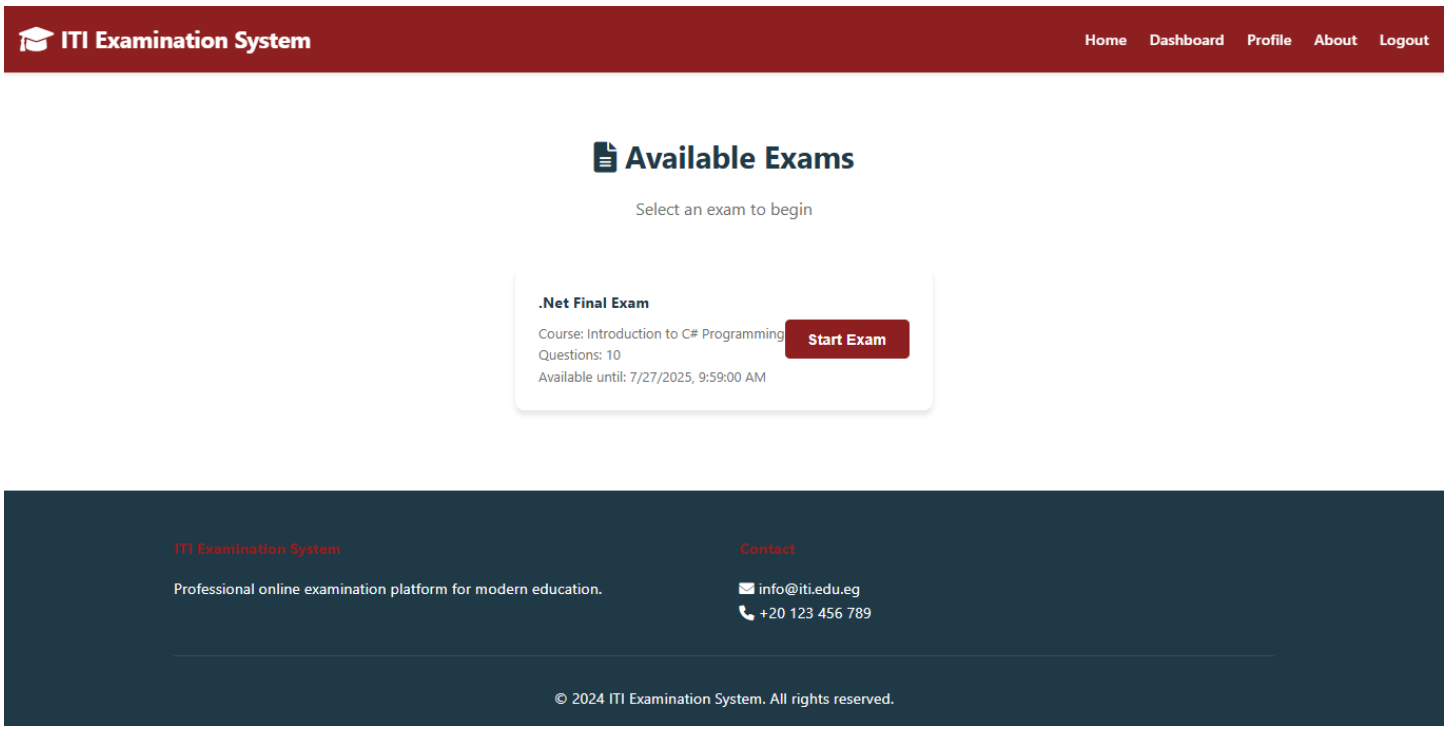
- Student Dashboard

A centralized page showing relevant course information, upcoming exams, and past results for the logged-in student.



- Available Exams Page (Student)

Lists all exams currently open and available for the student to take. It displays the course name, duration, and deadlines.



- Exam Page (Take the Exam)

Presents the actual exam interface. The student answers the listed questions (MCQ/True-False) and submits responses for evaluation.