Examination System

Graduation project at ITI.

“Power BI Development” Track, 2024/2025

Project Team

Fatma Mohammed Hashem

Hamdy Magdy Hamdy

Mona Rizq El gammal

Mosaab Mahmoud Mohi

Mostafa Khaled Mostafa

Under Supervision of

ENG. Rami Nagi

Table of Contents

Contents

Abstract

The ITI Examination System database is designed to store and manage student enrollment records, academic performance, and examination results across multiple intakes and governorates. This project focuses on generating and analyzing structured data to extract meaningful insights regarding student progress, exam performance trends, and overall system efficiency. Since the data was synthetically generated, the analysis serves as a demonstration of how the system can be utilized for data-driven decision-making. Through interactive dashboards, key metrics such as student enrollment distribution, exam success rates, and employment tracking are visualized to assess the institution’s impact. The primary objective is to showcase the potential of data analysis in evaluating student outcomes, identifying areas for improvement, and supporting decision-making processes within ITI.

Chapter 1: Project Planning

1.1 Problem Definition

The ITI Examination System database is designed to store and manage student-related data, including enrollments, academic performance, and employment tracking. However, several challenges must be addressed to ensure effective data analysis and meaningful insights. The key challenges include:

- **Enrollment Data Management:** Managing student enrollments across multiple intakes and governorates requires an organized and structured database. Ensuring accurate data entry, efficient handling of multiple records, and minimizing inconsistencies are critical for reliable analysis.
- **Course and Exam Data Integration:** Maintaining a well-structured repository of courses and examination results is essential for evaluating student performance. The database must accurately link students to their respective courses, assessments, and results to provide a clear picture of academic progress.
- **Instructor and Student Interaction Analysis:** Analyzing instructor involvement and student engagement levels requires well-organized data. The system should enable tracking of instructor assignments, student participation, and overall academic performance trends.
- **Examination Performance Analysis:** Extracting meaningful insights from exam results is crucial for assessing student success rates. The database should allow efficient querying and visualization of performance trends, pass/fail rates, and overall exam effectiveness.
- **Data Security and Integrity:** A well-maintained student database must adhere to strict security protocols, ensuring the protection of sensitive information. Proper access controls, backup mechanisms, and compliance with data protection standards are necessary to maintain database reliability.
- **Employment Outcome Tracking:** Monitoring graduates' employment status is essential for evaluating ITI’s impact on job market readiness. The system should facilitate tracking of post-graduation employment trends, industry demand alignment, and areas requiring improvement in student training.
- **Dashboard Reporting and Decision Support:** The effectiveness of the database depends on how well data is visualized and interpreted. Interactive dashboards should enable real-time insights into student performance, enrollment trends, and employment outcomes, aiding ITI in making data-driven decisions.

Addressing these challenges will enhance the overall functionality and reliability of the ITI Examination System database. By improving data management, ensuring security, and leveraging analytical dashboards, the system can provide valuable insights into student success, institutional effectiveness, and areas for continuous improvement.

.

1.2 Project Features

The ITI Examination System database provides structured data storage and analytical insights to support decision-making. The key capabilities include:

- **Enrollment Data Management:** Organizes student enrollments across multiple intakes and governorates, enabling trend analysis.
- **Course and Study Material Insights:** Tracks course offerings and study materials to assess accessibility and relevance.
- **Instructor Assignment Analysis:** Evaluates instructor involvement and its impact on student performance.
- **Examination Performance Tracking:** Analyzes grading trends, pass/fail rates, and student success patterns.
- **Student Records Management:** Maintains a centralized database of student academic progress and exam results.
- **Employment Outcome Analysis:** Tracks post-graduation employment trends to measure ITI's impact on job placement.
- **Progress Monitoring and Insights:** Identifies students requiring additional support based on academic and employment data.

1.3 Literature Review

To better understand the ITI Examination System database and its analytical applications, we conducted a literature review focusing on data management in educational institutions, student performance analysis, and employment tracking. The key findings include:

- **Enrollment Data Analysis:** Studies have explored how institutions utilize enrollment data to track trends, predict student demand, and enhance administrative efficiency. Data-driven enrollment management improves decision-making by identifying patterns in student applications, intake distributions, and regional enrollment variations.
- **Student Performance Assessment:** Academic research highlights the importance of analyzing student performance data to identify success factors and areas for improvement. Methods such as statistical analysis of exam results, grade distributions, and historical trends help institutions refine assessment strategies and curriculum effectiveness.
- **Employment Outcome Tracking:** Monitoring post-graduation employment data provides valuable insights into the effectiveness of educational programs. Studies emphasize the role of data analytics in measuring job placement success, identifying skill gaps, and improving career readiness programs.
- **Data Security and Management:** Given the sensitive nature of student records, literature stresses the importance of secure data storage, reliable backup strategies, and compliance with data protection regulations. Strong authentication, access controls, and encryption techniques are critical to maintaining the integrity and confidentiality of academic data.

By leveraging these insights, educational institutions can enhance their data-driven decision-making processes, optimize academic performance tracking, and improve the alignment between educational outcomes and job market demands.

1.4 Database Structure Overview

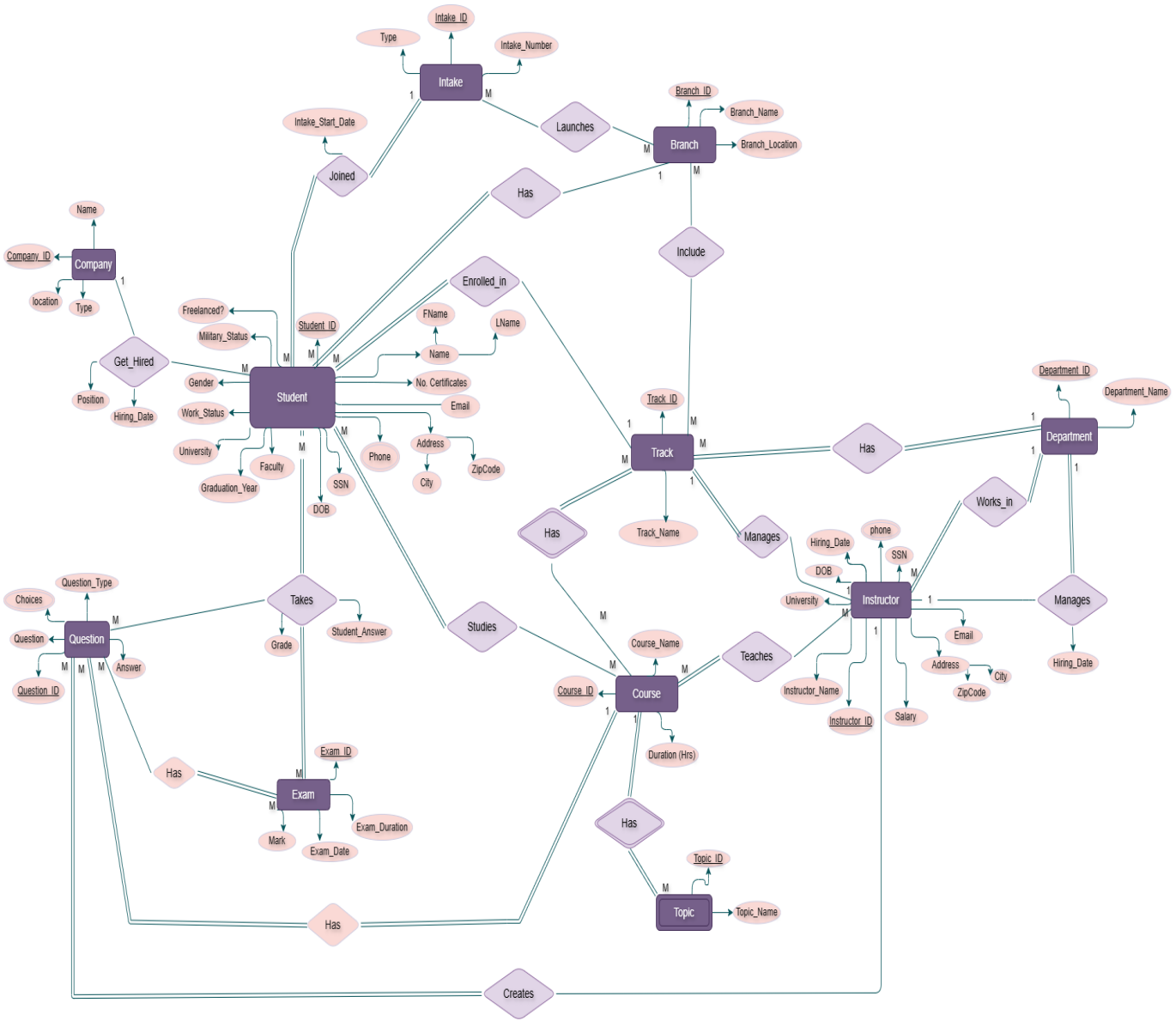
The ITI Examination System database is designed to store and manage structured data related to student enrollments, academic records, examinations, and post-graduation outcomes. The system ensures data accuracy, integrity, and accessibility to facilitate analytical reporting and decision-making.

The database follows a relational structure, organizing information efficiently while maintaining clear relationships between various entities. This structured approach allows for effective data retrieval, visualization, and analysis through dashboards and reporting tools.

The following chapters provide a detailed **Entity-Relationship Diagram (ERD)** and **Data Mapping**, illustrating how data is structured and interconnected within the system.

Chapter 2: ERD

2.1 ERD



Entity Descriptions

1. *Student*

Represents individuals enrolled in the ITI programs.

- Key Attributes: Student_ID, Name, Gender, DOB, University, Graduation_Year, Military_Status, Work_Status, Freelanced?
- Relationships:
  - Enrolled in an Intake
  - Takes Exams
  - Studies Courses
  - Can be Hired by a Company

2. Intake

Represents different enrollment periods or batches.

- Key Attributes: Intake_ID, Type, Intake_Start_Date, Intake_Number
- Relationships:
    - Students are Enrolled_in an Intake
    - Branches launch Intakes

3. Branch

Represents the different ITI locations.

- Key Attributes: Branch_ID, Branch_Name, Branch_Location
- Relationships:
    - Each Branch launches multiple Intakes
    - Tracks are included in Branches

4. Track

Represents specialized study areas or disciplines.

- Key Attributes: Track_ID, Track_Name
- Relationships:
    - Students Study in a Track
    - Each Track has multiple Courses
    - Managed by an Instructor

5. Course

Represents academic courses within a track.

- Key Attributes: Course_ID, Course_Name, Duration (Hours)
- Relationships:
    - Students Study a Course
    - Instructors teach Courses
    - Courses have multiple Topics

6. Instructor

Represents teachers responsible for delivering courses.

- Key Attributes: Instructor_ID, Instructor_Name, Email, Phone, University, Salary
- Relationships:
    - Teaches multiple Courses
    - Works in a Department
    - Manages a Track

7. Department

Represents academic or administrative divisions in ITI.

- Key Attributes: Department_ID, Department_Name
- Relationships:
  - Instructors work in a Department
  - Departments manage Instructors

8. Exam

Represents assessments taken by students.

- Key Attributes: Exam_ID, Exam_Date, Exam_Duration, Mark
- Relationships:
  - Students Take Exams
  - Exams contain multiple Questions

9. Question

Represents exam questions.

- Key Attributes: Question_ID, Question_Type, Choices, Answer
- Relationships:
  - An Exam has multiple Questions

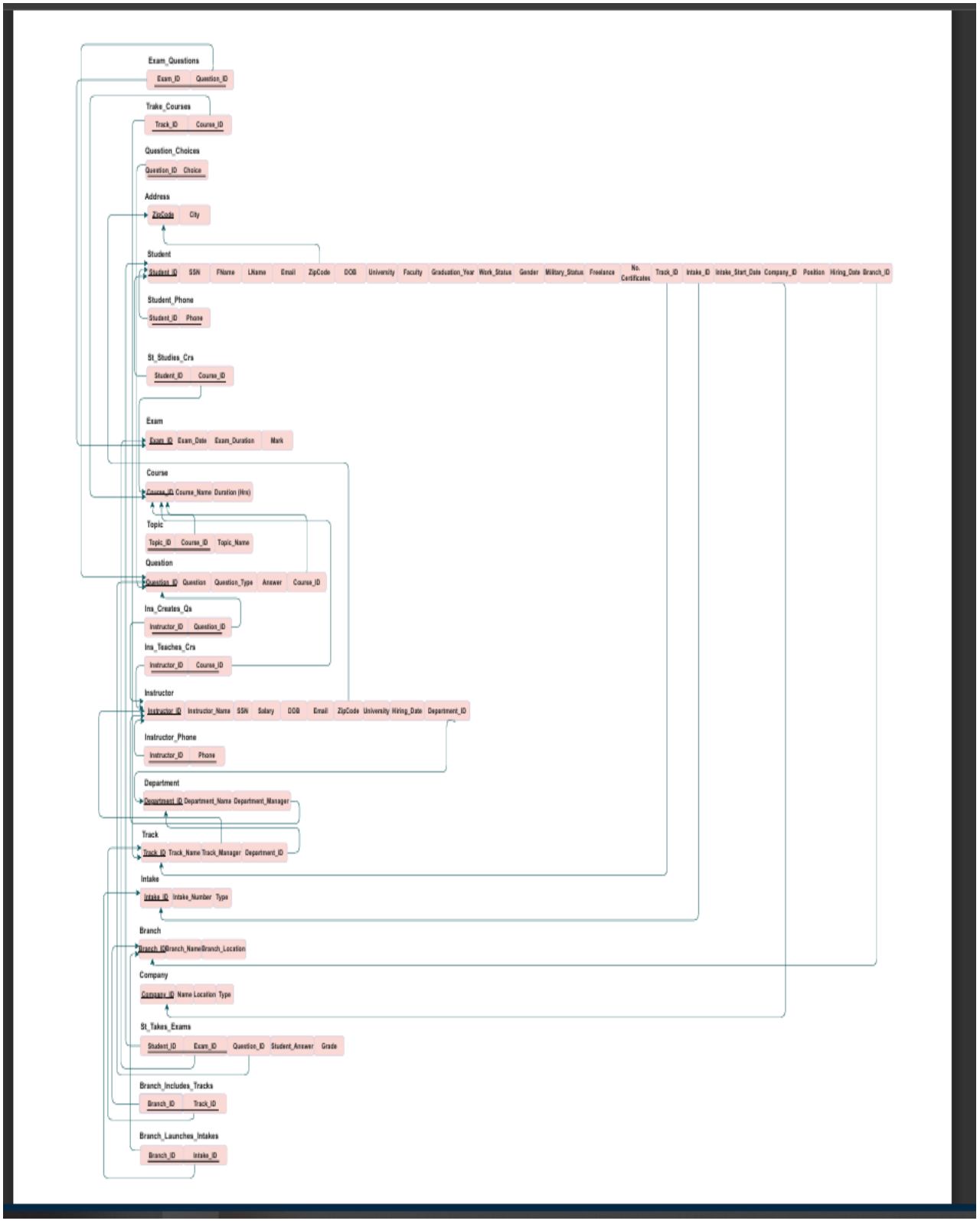
10. Company

Represents companies that hire students.

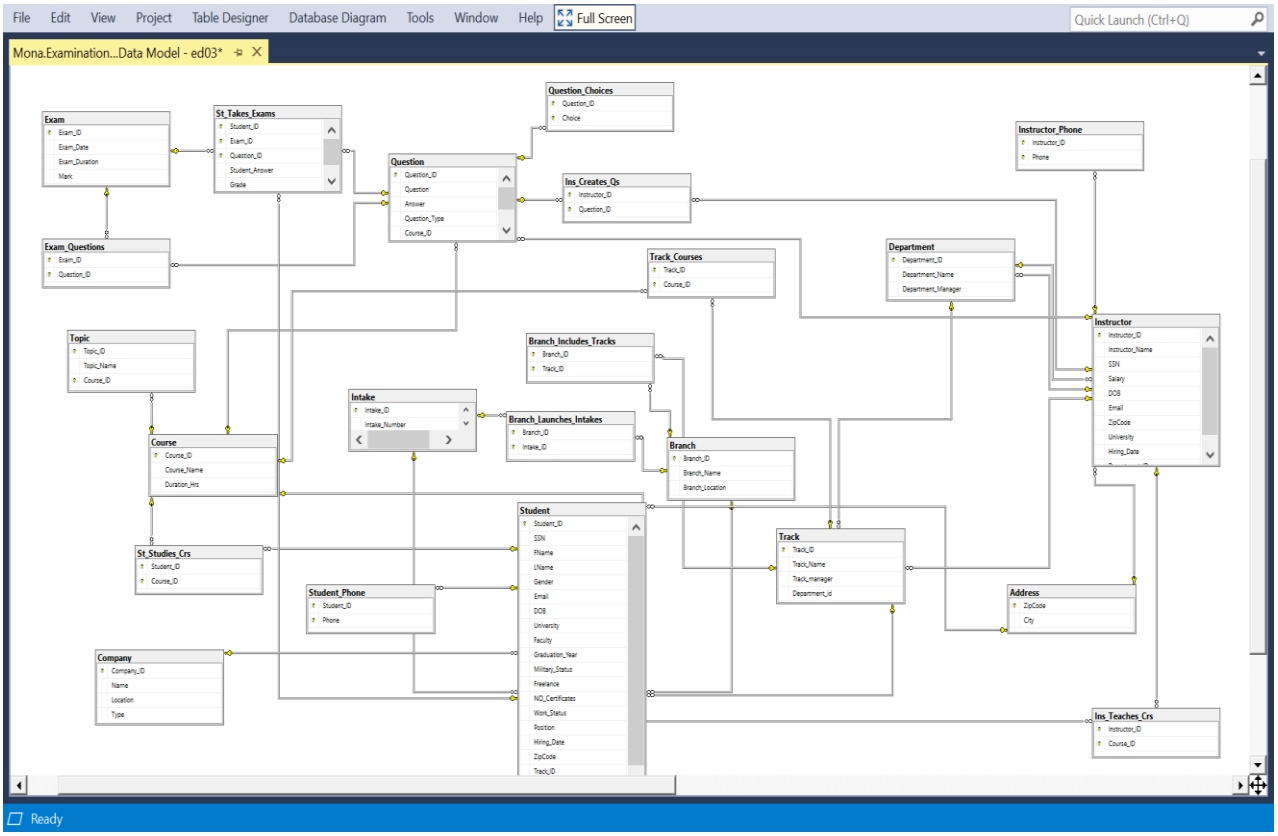
- Key Attributes: Company_ID, Name, Location, Type
- Relationships:
  - Students can Get_Hired by a Company

Chapter 3: Design

3.1 Mapping



3.2 DB Mapping



Main Entities

- **Student**: Stores student details such as (*Student_ID* PK, SSN, FName, LName, Gender, Email, DOB, University, Faculty, Graduation_Year, Military_Status, Freelance, No_Certificates, Work_Status, Position, Hiring_Date, ZipCode, *Track_ID* FK, *Intake_ID* FK, Intake_Start_Date, *Branch_ID* FK).
- **Instructor**: Stores instructor information, including (*Instructor_ID* PK, Instructor_Name, SSN, Salary, DOB, Email, University, ZipCode, Hiring_Date, *Department_ID* FK).
- **Course**: Represents courses with attributes like (*Course_ID* PK, Course_Name, Duration_Hrs).
- **Topic**: Stores topics related to courses (*Topic_ID* PK, Topic_Name, *Course_ID* FK).
- **Track**: Stores tracks (programs) with (*Track_ID* PK, Track_Name, Track_Manager, *Department_ID* FK).
- **Branch**: Represents branches where students enroll, with (*Branch_ID* PK, Branch_Name, Branch_Location).
- **Company**: Stores information about companies linked to students, with (*Company_ID* PK, Name, Location, Type).
- **Department**: Represents departments with (*Department_ID* PK, Department_Name, Department_Manager).
- **Exam**: Stores exam details like (*Exam_ID* PK, Exam_Date, Exam_Duration, Mark).
- **Question**: Stores exam questions with attributes like (*Question_ID* PK, Question, Answer, Question_Type, *Course_ID* FK).

Relationship Tables (Linking Entities)

- **St_Takes_Exams**: Links students to the exams they take, storing (*Student_ID* FK, *Exam_ID* FK, *Question_ID* FK, Student_Answer, Grade).
- **Exam_Questions**: Links exams to the questions included in them (*Exam_ID* FK, *Question_ID* FK).
- **St_Studies_Crs**: Links students to the courses they are studying (*Student_ID* FK, *Course_ID* FK).
- **Track_Courses**: Links tracks to the courses they include (*Track_ID* FK, *Course_ID* FK).
- **Ins_Teaches_Crs**: Links instructors to the courses they teach (*Instructor_ID* FK, *Course_ID* FK).
- **Ins_Creates_Qs**: Links instructors to the questions they create (*Instructor_ID* FK, *Question_ID* FK).
- **Branch_Includes_Tracks**: Defines which tracks belong to which branches (*Branch_ID* FK, *Track_ID* FK).
- **Branch_Launches_Intakes**: Links branches to the intakes they launch (*Branch_ID* FK, *Intake_ID* FK).

Supporting Tables

- **Question_Choices**: Stores multiple-choice answers for questions (*Question_ID* FK, Choice).
- **Instructor_Phone**: Stores phone numbers of instructors (*Instructor_ID* FK, Phone).
- **Student_Phone**: Stores phone numbers of students (*Student_ID* FK, Phone).
- **Address**: Links ZIP codes to cities (*ZipCode* PK, City).
- **Intake**: Stores intake information (*Intake_ID* PK, Intake_Number).

3.3 Database Dictionary

[dbo].[Student]

Columns

Column Name	Data Type	Allow Nulls
Student_ID	int	☐
SSN	varchar(14)	☐
FName	varchar(20)	☐
LName	varchar(20)	☐
Gender	varchar(1)	☑
Email	varchar(50)	☑
DOB	date	☑
University	varchar(100)	☑
Faculty	varchar(50)	☑
Graduation_Year	int	☑
Military_Status	varchar(15)	☑
Freelance	int	☑
NO_Certificates	int	☑
Work_Status	varchar(15)	☑
Position	varchar(50)	☑
Hiring_Date	date	☑
ZipCode	varchar(5)	☑
Track_ID	int	☑
Intake_ID	int	☑
Company_ID	int	☑
Intake_Start_Date	date	☑
Branch_id	int	☑
		☐

PK: Student_ID

FKs: Track_ID → **Track(Track_ID)**

Intake_ID → **Intake(Intake_ID)**

Branch_ID → **Branch(Branch_ID)**

ZipCode → **Address(ZipCode)**

Company_ID → **Company(Company_ID)**

[dbo].[Instructor]

Columns

Column Name	Data Type	Allow Nulls
Instructor_ID	int	☐
Instructor_Name	varchar(20)	☑
SSN	varchar(14)	☑
Salary	int	☑
DOB	date	☑
Email	varchar(100)	☑
ZipCode	varchar(5)	☑
University	varchar(100)	☑
Hiring_Date	date	☑
Department_ID	int	☑
		☐

PK: Instructor_ID

FKs: Department_ID → **Department(Department_ID)**
ZipCode → **Address(ZipCode)**

[dbo].[Course]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Course_ID	int	☐
	Course_Name	varchar(100)	☑
	Duration_Hrs	int	☑
			☐

PK: Course_ID

[dbo].[Topic]

Columns

Column Name	Data Type	Allow Nulls
Topic_ID	int	☐
Topic_Name	varchar(100)	☑
Course_ID	int	☐
		☐

PK: Topic_ID

FKs: Course_ID → **Course(Course_ID)**

[dbo].[Track]

Columns

Column Name	Data Type	Allow Nulls
Track_ID	int	☐
Track_Name	varchar(100)	☑
Track_manager	int	☑
Department_id	int	☑
		☐

PK: Track_ID

FKs: Department_ID → **Department(Department_ID)**

[dbo].[Branch]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Branch_ID	int	☐
	Branch_Name	varchar(20)	☑
	Branch_Location	varchar(20)	☑
			☐

PK: Branch_ID

[dbo].[Company]

Columns

Column Name	Data Type	Allow Nulls
Company_ID	int	☐
Name	varchar(20)	☐
Location	varchar(50)	☑
Type	varchar(50)	☑
		☐

PK: Company_ID

[dbo].[Department]

Columns

Column Name	Data Type	Allow Nulls
Department_ID	int	☐
Department_Name	varchar(100)	☑
Department_Manager	int	☑
		☐

PK: Department_ID

[dbo].[Exam]

Columns

Column Name	Data Type	Allow Nulls
Exam_ID	int	☐
Exam_Date	date	☑
Exam_Duration	int	☑
Mark	int	☑
		☐

PK: Exam_ID

[dbo].[Question]

Columns

Column Name	Data Type	Allow Nulls
Question_ID	int	☐
Question	varchar(1000)	☑
Answer	varchar(200)	☑
Question_Type	varchar(4)	☑
Course_ID	int	☑
Instructor_ID	int	☑
		☐

PK: Question_ID

FKs: Course_ID → **Course(Course_ID)**

[dbo].[St_Takes_Exams]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Student_ID	int	☐
🔑	Exam_ID	int	☐
🔑	Question_ID	int	☐
	Student_Answer	varchar(255)	☑
	Grade	int	☑
			☐

PK: (Student_ID, Exam_ID, Question_ID)

FKs: Student_ID → **Student(Student_ID)**
Exam_ID → **Exam(Exam_ID)**
Question_ID → **Question(Question_ID)**

[dbo].[Exam_Questions]

Columns

Column Name	Data Type	Allow Nulls
Exam_ID	int	☐
Question_ID	int	☐
		☐

PK: (Exam_ID, Question_ID)

FKs: Exam_ID → **Exam(Exam_ID)**
Question_ID → **Question(Question_ID)**

[dbo].[St_Studies_Crs]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Student_ID	int	☐
🔑	Course_ID	int	☐
			☐

PK: (Student_ID, Course_ID)

FKs: Student_ID → **Student(Student_ID)**
Course_ID → **Course(Course_ID)**

[dbo].[Track Courses]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Track_ID	int	☐
🔑	Course_ID	int	☐
			☐

PK: (Track_ID, Course_ID)

FKs: Track_ID → **Track(Track_ID)**
Course_ID → **Course(Course_ID)**

[dbo].[Ins_Teaches_Crs]

Columns

Column Name	Data Type	Allow Nulls
Instructor_ID	int	☐
Course_ID	int	☐
		☐

- **PK:** (Instructor_ID, Course_ID)
  FKs: Instructor_ID → **Instructor(Instructor_ID)**
  Course_ID → **Course(Course_ID)**

[dbo].[Ins_Creates_Qs]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Instructor_ID	int	☐
🔑	Question_ID	int	☐
			☐

PK: (Instructor_ID, Question_ID)
FKs: Instructor_ID → **Instructor(Instructor_ID)**
Question_ID → **Question(Question_ID)**

[dbo].[Branch_Includes_Tracks]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Branch_ID	int	☐
🔑	Track_ID	int	☐
			☐

PK: (Branch_ID, Track_ID)

FKs: Branch_ID → **Branch(Branch_ID)**
Track_ID → **Track(Track_ID)**

[dbo].[Branch_Launches_Intakes]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Branch_ID	int	☐
🔑	Intake_ID	int	☐
			☐

PK: (Branch_ID, Intake_ID)

FKs: Branch_ID → **Branch(Branch_ID)**

Intake_ID → **Intake(Intake_ID)**

[dbo].[Question_Choices]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Question_ID	int	☐
🔑	Choice	varchar(200)	☐
			☐

PK: (Question_ID, Choice)

FKs: Question_ID → **Question(Question_ID**

[dbo].[Instructor_Phone]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Instructor_ID	int	☐
🔑	Phone	varchar(14)	☐
			☐

PK: (Instructor_ID, Phone)

FKs: Instructor_ID → **Instructor(Instructor_ID)**

[dbo].[Student_Phone]

Columns

	Column Name	Data Type	Allow Nulls
🔑	Student_ID	int	☐
🔑	Phone	varchar(14)	☐
			☐

PK: (Student_ID, Phone)

FKs: Student_ID → **Student(Student_ID)**

n

[dbo].[Address]

Columns

PC-1\SQLEXPRESS.E...tem - dbo.Address

Column Name	Data Type	Allow Nulls
ZipCode	varchar(5)	☐
City	varchar(20)	☑
		☐

PK: ZipCode

[dbo].[Intake]

Columns

Column Name	Data Type	Allow Nulls
Intake_ID	int	☐
Intake_Number	int	☑
Type	varchar(20)	☑
		☐

PK: Intake_ID

3.4 Data generation and table creation

- **Sample of table creation**

```sql
CREATE TABLE Student(
    Student_ID INT IDENTITY PRIMARY KEY,
    SSN VARCHAR(14) UNIQUE NOT NULL,
    FName VARCHAR(20) NOT NULL,
    LName VARCHAR(20) NOT NULL,
    Gender VARCHAR(1),
    Email VARCHAR(50) UNIQUE,
    DOB DATE,
    University VARCHAR(20),
    Faculty VARCHAR(50),
    Graduation_Year INT,
    Military_Status VARCHAR(15) DEFAULT 'NA',
    Freelance INT DEFAULT 0,
    NO_Certificates INT DEFAULT 0,
    Work_Status VARCHAR(15) DEFAULT 'Unemployed',
    Position VARCHAR(50),
    Hiring_Date DATE,
    Intake_Start_Date DATE,
    ZipCode VARCHAR(5), -- FK
    Track_ID INT,       -- FK
    Intake_ID INT,      -- FK
    Company_ID INT,     -- FK
    Branch_ID INT,      --FK
    CONSTRAINT checkFreelanced CHECK (Freelance IN (0,1)),
    CONSTRAINT checkNo_Certificates CHECK (No_Certificates IN (0, 1, 2, 3)),
    CONSTRAINT checkGraduation_Year CHECK ((YEAR(GETDATE()) - Graduation_Year) <= 10),
    CONSTRAINT FK_SZ FOREIGN KEY (ZipCode) REFERENCES Address(ZipCode),
    CONSTRAINT FK_ST FOREIGN KEY (Track_ID) REFERENCES Track(Track_ID),
    CONSTRAINT FK_SI FOREIGN KEY (Intake_ID) REFERENCES Intake(Intake_ID),
    CONSTRAINT FK_SC FOREIGN KEY (Company_ID) REFERENCES Company(Company_ID)
);
```

- **Samples of data generation**

```sql
--- Inserting Topics for Courses with Duration 6 Hrs
INSERT INTO Topic (Topic_ID, Topic_Name, Course_ID)
VALUES
-- Git & GitHub Version Control
(1, 'Git Basics', 9),
(2, 'Branching & Merging', 9),
(3, 'GitHub Actions', 9),

-- Secure Coding in .NET
(4, 'Input Validation', 18),
(5, 'Authentication & Authorization', 18),
(6, 'Secure Coding Standards', 18),

-- Data Viz with Power BI & Tableau
(7, 'Power BI Dashboards', 27),
(8, 'Tableau Calculated Fields', 27),
(9, 'Data Storytelling', 27),

-- Business Analytics with Excel
(10, 'Pivot Tables', 36),
(11, 'What-If Analysis', 36),
(12, 'Power Query', 36),

-- Kubernetes & Containers
(13, 'Kubernetes Architecture', 45),
(14, 'Docker Fundamentals', 45),
(15, 'Kubernetes Networking', 45),

-- Web App Security
(16, 'Cross-Site Scripting (XSS)', 54),
(17, 'SQL Injection Prevention', 54),
(18, 'OAuth & JWT Security', 54),
```

```
WITH Numbers AS (
    SELECT 1 AS Intake_Number
    UNION ALL
    SELECT Intake_Number + 1 FROM Numbers WHERE Intake_Number < 45
)
INSERT INTO Intake (Intake_Number, Type)
SELECT Intake_Number, 'PTP' FROM Numbers WHERE Intake_Number <= 15
UNION ALL
SELECT Intake_Number, 'ICC' FROM Numbers;
```

```
INSERT INTO Exam (Exam_Date, Exam_Duration, Mark) VALUES
('2019-08-26', 80, 55),
('2025-01-28', 60, 55),
('2024-07-18', 60, 60),
('2024-10-22', 50, 70),
('2019-07-22', 45, 65),
('2024-10-28', 30, 20),
('2015-07-12', 20, 35),
('2023-02-17', 85, 90),
('2020-03-08', 35, 70),
('2016-05-25', 25, 75),
('2021-11-29', 45, 20),
('2025-01-05', 55, 90),
('2017-01-03', 90, 95),
('2020-10-13', 40, 10),
('2017-12-30', 75, 90),
('2016-02-14', 40, 55),
('2023-06-26', 40, 75),
('2017-01-10', 20, 40),
('2018-08-17', 30, 25),
('2018-09-19', 45, 5),
('2020-09-10', 45, 20),
('2016-10-13', 90, 30),
('2024-01-02', 20, 65),
('2025-01-31', 55, 45),
('2021-11-14', 80, 10),
```

```
INSERT INTO Course (Course_Name, Duration_Hrs) VALUES
('Intro to Linux', 9),
('Web Dev with PHP & MySQL', 12),
('JS & Frontend Frameworks', 15),
('Python for Web Dev', 18),
('Backend Dev with Node.js', 21),
('API Dev & Integration', 24),
('Software Eng Principles', 27),
('DB Design & Optimization', 30),
('Git & GitHub Version Ctrl', 6),
('CI/CD Pipelines', 9),
('C# Programming Basics', 12),
('ASP.NET Core Dev', 15),
('Entity Framework & DB Mgmt', 18),
('Frontend with Angular & Vue', 21),
('Web API Dev with .NET', 24),
('Microservices in .NET', 27),
('Cloud Deploy with Azure', 30),
('Secure Coding in .NET', 6),
('Software Testing & Debugging', 9),
('Agile & Scrum for Devs', 12),
('Python for Data Science', 15),
```

```
INSERT INTO Company (Name, Location, Type) VALUES
('Maridive & Oil Servi', 'Egypt', 'National'),
('El Araby Group', 'Egypt', 'National'),
('Etisalat Misr', 'Egypt', 'National'),
('Emaar Misr', 'Egypt', 'National'),
('Juhayna', 'Egypt', 'National'),
('Ezz Steel', 'Egypt', 'National'),
('Edita Food Industrie', 'Egypt', 'National'),
('B-Tech', 'Egypt', 'National'),
('Orascom Construction', 'Egypt', 'National'),
('Ezz Steel', 'Egypt', 'National'),
('Palm Hills Developme', 'Egypt', 'National'),
('El Araby Group', 'Egypt', 'National'),
('B-Tech', 'Egypt', 'National'),
('Arab Contractors', 'Egypt', 'National'),
('Palm Hills Developme', 'Egypt', 'National'),
('Juhayna', 'Egypt', 'National'),
('Banque Misr', 'Egypt', 'National'),
('Banque Misr', 'Egypt', 'National'),
('Telecom Egypt', 'Egypt', 'National'),
('SODIC', 'Egypt', 'National'),
('El Sewedy Electric', 'Egypt', 'National'),
```

```sql
INSERT INTO Branch (Branch_Name, Branch_Location)
VALUES
    ('New Capital', 'Cairo'),
    ('Cairo University', 'Cairo'),
    ('Alexandria', 'Alexandria'),
    ('Assiut', 'Assiut'),
    ('Aswan', 'Aswan'),
    ('Beni Suef', 'Beni Suef'),
    ('Fayoum', 'Fayoum'),
    ('Ismailia', 'Ismailia'),
    ('Mansoura', 'Mansoura'),
    ('Menofia', 'Menofia'),
    ('Minya', 'Minya'),
    ('Qena', 'Qena'),
    ('Sohag', 'Sohag'),
    ('Smart Village', 'Cairo');
```

```sql
INSERT INTO Branch_Launches_Intakes (Branch_ID, Intake_ID)
-- Intake IDs 1 to 15 with specific branches
SELECT b.Branch_ID, i.Intake_ID
FROM (VALUES (14)) AS b(Branch_ID)
CROSS JOIN (SELECT Intake_ID FROM Intake WHERE Intake_ID BETWEEN 1 AND 15) i

UNION ALL

-- Intake IDs 1 to 15 with specific branches
SELECT b.Branch_ID, i.Intake_ID
FROM (VALUES (3), (9), (10)) AS b(Branch_ID)
CROSS JOIN (SELECT Intake_ID FROM Intake WHERE Intake_ID BETWEEN 10 AND 15) i

UNION ALL

-- Intake IDs 16 to 60 with branches
SELECT b.Branch_ID, i.Intake_ID
FROM (SELECT Branch_ID FROM Branch where Branch_ID IN (14, 10, 9)) b
CROSS JOIN (SELECT Intake_ID FROM Intake WHERE Intake_ID BETWEEN 16 AND 60) i
```

```
INSERT INTO Address (ZipCode, City) VALUES
('44933', 'Fayoum'),
('93320', 'Suez'),
('10038', 'Fayoum'),
('75130', 'Sharm El Sheikh'),
('28662', 'Ismailia'),
('16173', 'Aswan'),
('88353', 'Sohag'),
('64633', 'Alexandria'),
('68656', 'Hurghada'),
('30659', 'Sohag'),
('81367', 'Sohag'),
('47348', 'Tanta'),
('85926', 'Mansoura'),
('88681', 'Hurghada'),
('93639', 'Ismailia'),
('18379', 'Sohag'),
('51347', 'Mansoura'),
('50329', 'Alexandria'),
('32743', 'Sohag'),
('28324', 'Giza'),
('76218', 'Minya'),
```

3.5 Stored Procedures

Stored Procedures

dbo.AddTrack

Adds a new track to the system after validating the existence of the specified department and track manager.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[AddTrack]    Script Date: 24/03/2025 6:17:38 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Create
ALTER PROCEDURE [dbo].[AddTrack]
    @Track_ID INT,
    @Track_Name VARCHAR(100),
    @Track_Manager INT,
    @Department_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Department WHERE Department_ID = @Department_ID)
    BEGIN
        PRINT 'Error: Department_ID does NOT exist';
        RETURN;
    END
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Track_Manager)
    BEGIN
        PRINT 'Error: The Instructor to be the Track Manager does NOT exist';
        RETURN;
    END


    INSERT INTO Track (Track_ID, Track_Name, Track_Manager, Department_ID)
    VALUES (@Track_ID, @Track_Name, @Track_Manager, @Department_ID);
END;
```

dbo.AnswerExam

Processes a student's submitted answers for an exam and records their responses linked to the corresponding questions.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[AnswerExam]    Script Date: 24/03/2025 6:19:59 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[AnswerExam]
@exam_id INT,
@student_id INT,
@St_answers VARCHAR(MAX)
AS
BEGIN
    -- tempView to hold the Student Answers
    DELETE FROM EneteredStudentAnswers;    -- Empty the Student Answers Table
    INSERT INTO EneteredStudentAnswers (row_num, st_ans)
        SELECT  ROW_NUMBER() OVER (ORDER BY (select null)) AS ans_rn, value AS St_Answer
        FROM STRING_SPLIT(@St_answers, ',');


    -- tempView for the exam questions that the student took
    DELETE FROM StudentExamQuestions;   -- Empty the Exam Table
    INSERT INTO StudentExamQuestions (row_num, exam_id, student_id, q_id)
    SELECT ROW_NUMBER() OVER (ORDER BY (select null)) AS eq_rn,
    @exam_id as exam_id,
    @student_id as student_id,
    Question_ID
    FROM Exam_Questions
    WHERE Exam_ID = @exam_id

    INSERT INTO St_Takes_Exams (Exam_ID, Student_ID, Question_ID, Student_Answer)
    SELECT SEQ.exam_id, SEQ.student_id, SEQ.q_id, SA.st_ans
    FROM StudentExamQuestions SEQ
    JOIN EneteredStudentAnswers SA
    ON SEQ.row_num = SA.row_num
END
```

dbo.CorrectExam

Corrects a student's exam by comparing submitted answers to the correct ones and assigns a grade.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[CorrectExam]    Script Date: 24/03/2025 6:22:30 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[CorrectExam]
@exam_id INT,
@student_id INT
AS
BEGIN
    UPDATE ST
    SET ST.Grade = CASE
                       WHEN Q.Answer = ST.Student_Answer THEN 1
                       ELSE 0
                   END
    FROM St_Takes_Exams ST
    JOIN Question Q ON ST.Question_ID = Q.Question_ID
    JOIN Exam_Questions QE ON Q.Question_ID = QE.Question_ID
    WHERE ST.Student_ID = @student_id
    AND QE.Exam_ID = @exam_id;
END
```

dbo.DeleteBranchByID

Deletes a branch by ID after ensuring it has no associated students, intakes, or tracks.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteBranchByID]    Script Date: 24/03/2025 6:28:26 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteBranchByID]
    @BranchID INT
AS
BEGIN
    SET NOCOUNT ON;
     IF NOT EXISTS (SELECT 1 FROM Branch WHERE Branch_ID = @BranchID)
        BEGIN
            PRINT 'This Branch ID does not exist';
            RETURN;
        END
         IF EXISTS (SELECT 1 FROM Student WHERE Branch_ID = @BranchID)
        BEGIN
            PRINT 'Cannot delete Branch. It has associated students.';
            RETURN;
        END
         IF EXISTS (SELECT 1 FROM Branch_Launches_Intakes WHERE Branch_ID = @BranchID)
        BEGIN
            PRINT 'Cannot delete Branch. It has associated intakes.';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Branch_Includes_Tracks WHERE Branch_ID = @BranchID)
        BEGIN
            PRINT 'Cannot delete Branch. It has associated tracks.';
            RETURN;
        END
        DELETE FROM Branch WHERE Branch_ID = @BranchID;
        PRINT 'Branch deleted successfully.';
END;
```

dbo.DeleteCompanyByID

Deletes a company by ID if it exists, otherwise prints an error message.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteCompanyByID]    Script Date: 24/03/2025 6:32:00 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Delete Company
ALTER PROC [dbo].[DeleteCompanyByID]
@company_id INT
AS
BEGIN
    IF EXISTS  (SELECT 1 FROM Company WHERE Company_ID = @company_id)
    BEGIN
        DELETE FROM Company
        WHERE Company_ID = @company_id
    END
    ELSE
    BEGIN
        PRINT 'There is NO Company with this ID, Please Enter Another Company ID!';
    END
END
```

dbo.DeleteCompanyByName

Deletes a company by name if it exists, otherwise prints an error message.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteCompanyByName]    Script Date: 24/03/2025 6:33:10 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Delete Company by its name
ALTER PROC [dbo].[DeleteCompanyByName]
@company_name INT
AS
BEGIN
    IF EXISTS  (SELECT 1 FROM Company WHERE Name = @company_name)
    BEGIN
        DELETE FROM Company
        WHERE Name = @company_name
    END
    ELSE
    BEGIN
        PRINT 'There is NO Company with this ID, Please Enter Another Company ID!';
    END
END
```

dbo.DeleteCourse

Deletes a course using either its ID or name, and displays the deleted record or an error if not found.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteCourse]    Script Date: 24/03/2025 6:36:07 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteCourse]
    @CrsID INT = NULL,
    @CrsName NVARCHAR(100) = NULL
AS
BEGIN
--    SET NOCOUNT ON;

    DECLARE @DeletedRecord NVARCHAR(500);

    -- Check if want to delete with  ID or Name is
    IF @CrsID IS NULL AND @CrsName IS NULL
    BEGIN
        PRINT 'Error: Please enter either an ID or a Name to delete the Course.'
        RETURN;
    END

    --catch the deleted record
    SELECT @DeletedRecord = CONCAT('Deleted Course - ID: ', Course_ID, ', Name: ', Course_Name)
    FROM Course
    WHERE (Course_ID = @CrsID OR @CrsName IS NOT NULL AND Course_Name = @CrsName);

    -- Delete the record
    DELETE FROM Course
    WHERE (Course_ID = @CrsID OR @CrsName IS NOT NULL AND Course_Name = @CrsName);

    -- Check if a record was deleted
    IF @@ROWCOUNT > 0
        PRINT @DeletedRecord;
    ELSE
        PRINT 'No matching record found to delete.';
END;
```

dbo.DeleteDepartment

Deletes a department by ID only if it has no associated tracks or instructors.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteDepartment]    Script Date: 24/03/2025 6:37:58 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteDepartment]
    @DepartmentID INT
AS
BEGIN
    SET NOCOUNT ON;
        IF NOT EXISTS (SELECT 1 FROM Department WHERE Department_ID = @DepartmentID)
        BEGIN
            PRINT 'Cannot delete Department. does not exist.';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Track WHERE Department_ID = @DepartmentID)
        BEGIN
            PRINT 'Cannot delete Department. It has associated tracks.';
            RETURN;
        END
         IF EXISTS (SELECT 1 FROM Instructor WHERE Department_ID = @DepartmentID)
        BEGIN
            PRINT 'Cannot delete Department. It has associated instructor.';
            RETURN;
        END
        DELETE FROM Department WHERE Department_ID = @DepartmentID;
        PRINT 'Department deleted successfully.';
END;
```

dbo.DeleteExam

Deletes an exam by ID after verifying its existence.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteExam]    Script Date: 24/03/2025 6:38:02 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteExam]
    @Exam_ID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if Exam_ID exists
    IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam_ID = @Exam_ID)
    BEGIN
        PRINT 'Error: Exam ID does not exist.';
        RETURN;
    END

    -- Delete the exam record
    DELETE FROM Exam WHERE Exam_ID = @Exam_ID;

    PRINT 'Exam deleted successfully.';
END;
```

dbo.DeleteInstructor

Deletes an instructor if no related records exist in dependent tables; otherwise returns an error.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteInstructor]    Script Date: 24/03/2025 6:41:09 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
---Delete
ALTER PROCEDURE [dbo].[DeleteInstructor]
    @Instructor_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Instructor_ID)
    BEGIN
        PRINT 'Instructor not found!';
        RETURN;
    END;
    IF EXISTS (SELECT 1 FROM Ins_Teaches_Crs WHERE Instructor_ID = @Instructor_ID) OR
       EXISTS (SELECT 1 FROM Ins_Creates_Qs WHERE Instructor_ID = @Instructor_ID) OR
       EXISTS (SELECT 1 FROM Instructor_Phone WHERE Instructor_ID = @Instructor_ID) OR
       EXISTS (SELECT 1 FROM Track WHERE Track_Manager = @Instructor_ID) OR
       EXISTS (SELECT 1 FROM Department WHERE Department_Manager = @Instructor_ID)
    BEGIN
        PRINT 'Cannot delete instructor. It has related records in other tables!';
        RETURN;
    END;
    DELETE FROM Instructor WHERE Instructor_ID = @Instructor_ID;

    PRINT 'Instructor deleted successfully!';
END;
```

dbo.DeleteIntakeByID

Deletes an intake record by ID if it has no related students or branches.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteIntakeByID]    Script Date: 24/03/2025 6:41:12 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteIntakeByID]
    @IntakeID INT
AS
BEGIN
    SET NOCOUNT ON;
        IF NOT EXISTS (SELECT 1 FROM Intake WHERE Intake_ID = @IntakeID)
        BEGIN
            PRINT 'This ID does not exist.';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Student WHERE Intake_ID = @IntakeID)
        BEGIN
            PRINT 'Cannot delete Department. It has associated students.';
            RETURN;
        END
         IF EXISTS (SELECT 1 FROM Branch_Launches_Intakes WHERE Intake_ID = @IntakeID)
        BEGIN
            PRINT 'Cannot delete Department. It has associated branches.';
            RETURN;
        END
        DELETE FROM Intake WHERE Intake_ID = @IntakeID;
        PRINT 'Intake deleted successfully';
END;
```

dbo.DeleteQuestion

Deletes a question and its choices if the question ID exists.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteQuestion]    Script Date: 24/03/2025 6:44:09 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[DeleteQuestion]
    @q_id INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE Question_ID = @q_id)
    BEGIN
        DELETE FROM Question_Choices WHERE Question_ID = @q_id;
        DELETE FROM Question WHERE Question_ID = @q_id;
        PRINT 'Question and its Choices are Deleted Successfully!';
    END
    ELSE
    BEGIN
        PRINT 'Question ID Does NOT Exist!';
    END
END;
```

dbo.DeleteStudent

Deletes a student by either Student ID or SSN, after validating the student has no dependent records.

```sql
ALTER PROC [dbo].[DeleteStudent]
(   @StudentID INT = NULL,
    @SSN VARCHAR(14) = NULL)

AS
BEGIN
    BEGIN TRY
        -- Ensure at least one parameter is provided
        IF @StudentID IS NULL AND @SSN IS NULL
        BEGIN
            SELECT CAST('Error: You must provide either StudentID or SSN to delete a record.' AS VARCHAR(100));
            RETURN;
        END

        -- Check if the student exists
        IF NOT EXISTS (SELECT 1 FROM student WHERE Student_ID = @StudentID OR SSN = @SSN)
        BEGIN
            SELECT CAST('Error: No student found with the provided StudentID or SSN.' AS VARCHAR(100));
            RETURN;
        END
        -- Prevent deletion if student has related records in other tables
            IF @SSN IS NOT NULL AND @StudentID IS NULL
            BEGIN
                SET @StudentID = (SELECT Student_ID from Student WHERE SSN = @SSN)
            END
            IF EXISTS (SELECT 1 FROM Student_Phone WHERE Student_ID = @StudentID)
            OR EXISTS (SELECT 1 FROM ST_Studies_Crs WHERE Student_ID = @StudentID)
            OR EXISTS (SELECT 1 FROM St_Takes_Exams WHERE Student_ID = @StudentID)

        BEGIN
            SELECT CAST('Error: Cannot delete student because related records exist in other tables.' AS VARCHAR(150));
            RETURN;
        END

        -- Delete the student record
        DELETE FROM student WHERE Student_ID = @StudentID OR SSN = @SSN;

        -- Confirm deletion
        SELECT CAST('Success: Student record deleted successfully.' AS VARCHAR(100));
    END TRY

    BEGIN CATCH
        -- Handle any unexpected errors
        SELECT CAST('Error: An unexpected error occurred during deletion.' AS VARCHAR(100));
    END CATCH
END;
```

dbo.DeleteTopic

Deletes a topic by topic ID, course ID, or name, and shows details of the deleted topic or an error.

```
ALTER PROCEDURE [dbo].[DeleteTopic]
    @TID INT = null,
    @TName NVARCHAR(100) =null ,
    @CrsID INT =null
AS
BEGIN
DECLARE @DeletedRecord NVARCHAR(500); ;

--    SET NOCOUNT ON;
if @TID is null and @CrsID is null and @TName is not null
begin
SELECT @DeletedRecord = CONCAT('Deleted Course - ID: ', Topic_ID, ', Name: ', Topic_Name ,'course id :' ,Course_ID   )
    FROM Topic
    WHERE (Topic_Name = @TName);
end

if @TID is not null and @CrsID is not null and @TName is null
begin
SELECT @DeletedRecord = CONCAT('Deleted Topic - ID: ', Topic_ID, ', Name: ', Topic_Name ,' course id :' ,Course_ID   )
    FROM Topic
    WHERE (Course_ID = @CrsID and Topic_ID = @TID );
end


    IF @TID IS NULL and @CrsID IS NULL and @TName is null
    BEGIN
        PRINT 'Error:Please enter both the topic ID and the course id  or enter the Topic name.'
        RETURN;
    END
    else if @TID IS not NULL and @CrsID IS not NULL and @TName is  null
    begin
    delete from Topic
        WHERE Topic_ID = @TID
        and Course_ID = @CrsID  ;
    end
    else if @TID IS NULL and @CrsID IS  NULL and @TName is not null
    begin
    delete from Topic
        WHERE Topic_Name = @TName   ;
    end

    IF @@ROWCOUNT > 0
        PRINT @DeletedRecord;
    ELSE
        PRINT 'No matching Topic or Course found to delete.';


END;
```

dbo.DeleteTrack

Deletes a track only if it is not linked to any students or branches.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[DeleteTrack]    Script Date: 24/03/2025 6:50:37 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
---delete
ALTER PROCEDURE [dbo].[DeleteTrack]
    @Track_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Track WHERE Track_ID = @Track_ID)
    BEGIN
        PRINT 'Track not found!';
        RETURN;
    END;
    IF EXISTS (SELECT 1 FROM Student WHERE Track_ID = @Track_ID)
    BEGIN
        PRINT 'Cannot delete Track: Students are assigned to this track!';
        RETURN;
    END;
    IF EXISTS (SELECT 1 FROM Branch_Includes_Tracks WHERE Track_ID = @Track_ID)
    BEGIN
        PRINT 'Cannot delete Track: It is linked to a Branch!';
        RETURN;
    END;
    DELETE FROM Track WHERE Track_ID = @Track_ID;

    PRINT 'Track deleted successfully!';
END;
```

dbo.GenerateExam

Generates a new exam by randomly selecting MCQ and TF questions for a specified course.

```
ALTER PROC [dbo].[GenerateExam]
@course_name VARCHAR(100),
@no_of_MCQs INT,
@no_of_TFs INT,
@exam_duration INT = 30
AS
BEGIN
IF NOT EXISTS (SELECT 1 FROM Course WHERE Course_Name = @course_name)
BEGIN
    PRINT 'This Course Does NOT Exist! Please Enter an Existing Course';
    RETURN;
END
ELSE
BEGIN
-- delete all previous questions from the temporary ExamQuestions Table
DELETE FROM ExamQuestionsTemp;

WITH RandomQuestions AS (
    SELECT
        Q.Question_ID,
        Q.Question,
        ROW_NUMBER() OVER (ORDER BY NEWID()) AS rn
    FROM Question Q
    INNER JOIN Course C ON Q.Course_ID = C.Course_ID
    WHERE Q.Course_ID = (SELECT Course_ID FROM Course WHERE Course_Name = @course_name)
    AND Q.Question_Type = 'MCQ'
)
INSERT INTO ExamQuestionsTemp (Question_ID, Question, Choice)
SELECT Q_MCQ.Question_ID, Q_MCQ.Question, QC.Choice
FROM (
    SELECT Question_ID, Question
    FROM RandomQuestions
    WHERE rn <= @no_of_MCQs
) AS Q_MCQ
JOIN Question_Choices QC
ON Q_MCQ.Question_ID = QC.Question_ID;

WITH RandomTFQuestions AS (
    SELECT
        Q.Question_ID,
        Q.Question,
        ROW_NUMBER() OVER (ORDER BY NEWID()) AS rn
    FROM Question Q
    INNER JOIN Course C ON Q.Course_ID = C.Course_ID
    WHERE Q.Course_ID = (SELECT Course_ID FROM Course WHERE Course_Name = @course_name)
    AND Q.Question_Type = 'T/F'
)
INSERT INTO ExamQuestionsTemp (Question_ID, Question, Choice)
SELECT Q_TF.Question_ID, Q_TF.Question, QC.Choice
FROM (
    SELECT Question_ID, Question
    FROM RandomTFQuestions
    WHERE rn <= @no_of_TFs
) AS Q_TF
JOIN Question_Choices QC
ON Q_TF.Question_ID = QC.Question_ID;


declare @e_date date = CONVERT(DATE, GETDATE());
-- Generate Exam data in the Exam Table
EXEC InsertExam @Exam_Date = @e_date, @Exam_Duration =  @exam_duration

END
END
```

dbo.GetAllCompanies

Retrieves all companies from the Company table.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetAllCompanies]    Script Date: 24/03/2025 6:56:54 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[GetAllCompanies]
AS
BEGIN
        SELECT *
        FROM Company
END
```

dbo.GetAllInstructors

Retrieves all instructors from the Instructor table.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetAllInstructors]    Script Date: 24/03/2025 6:57:59 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
---READ
ALTER PROCEDURE [dbo].[GetAllInstructors]
AS
BEGIN
    SELECT * FROM Instructor;
END;
```

dbo.GetCompany

Retrieves a specific company by ID if it exists, otherwise returns an error message.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetCompany]    Script Date: 25/03/2025 1:37:00 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[GetCompany]
@company_id INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Company WHERE Company_ID = @company_id)
    BEGIN
        SELECT *
        FROM Company
        WHERE Company_ID = @company_id
    END
    ELSE
    BEGIN
        PRINT 'This Company ID Does NOT Exist, Please Enter Another Company ID!';
    END
END
```

dbo.GetCompanyByName

Retrieves a specific company by name if it exists, otherwise returns an error message.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetCompanyByName]    Script Date: 25/03/2025 1:37:07 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- READ A COMPANY INFO BY NAME
ALTER PROC [dbo].[GetCompanyByName]
@company_name INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Company WHERE Name = @company_name)
    BEGIN
        SELECT *
        FROM Company
        WHERE Name = @company_name
    END
    ELSE
    BEGIN
        PRINT 'This Company Does NOT Exist, Please Enter Another Company!';
    END
END
```

dbo.GetCourseTopics

Retrieves all topics related to a specific course by Course ID.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetCourseTopics]    Script Date: 25/03/2025 1:37:11 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetCourseTopics]   -- Report 4
    @Course_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Course WHERE Course_ID = @Course_ID)
    BEGIN
        PRINT 'Course not found!';
        RETURN;
    END;
    SELECT
        Topic_ID,
        Topic_Name,
        Course_ID
    FROM Topic
    WHERE Course_ID = @Course_ID;
END;
```

dbo.GetExamQuestionAndChoices

Retrieves all questions and choices for a specific exam by Exam ID.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetExamQuestionsAndChoices]    Script Date: 25/03/2025 1:39:09 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetExamQuestionsAndChoices]   -- Report 5
    @Exam_Number INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam_ID = @Exam_Number)
    BEGIN
        PRINT 'Exam not found!';
        RETURN;
    END;
    SELECT
        Q.Question_ID,
        Q.Question,
        Qc.Choice
    FROM Question_Choices Qc
    INNER JOIN Question Q ON Qc.Question_ID = Q.Question_ID
    INNER JOIN St_Takes_Exams SE ON Q.Question_ID = SE.Question_ID
    inner join Exam E on E.Exam_ID = SE.Exam_ID
    WHERE E.Exam_ID = @Exam_Number;
END;
```

dbo.GetInstructorByDepartmentID

Retrieves all instructors who belong to a specific department by Department ID.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetInstructorsByDepartmentID]    Script Date: 25/03/2025 1:39:31 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetInstructorsByDepartmentID]
    @Department_ID INT
AS
BEGIN
    SELECT
        *
    FROM Instructor
    WHERE Department_ID = @Department_ID;
END;
```

dbo.GetInstructorByID

Retrieves a specific instructor by Instructor ID.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetInstructorByID]    Script Date: 25/03/2025 1:39:13 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetInstructorByID]
    @Instructor_ID INT
AS
BEGIN
    SELECT  *
    FROM Instructor
    WHERE Instructor_ID = @Instructor_ID;
END;
```

dbo.GetQuestionWithChoices

Retrieves a question and its choices using the provided Question ID.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetQuestionWithChoices]    Script Date: 25/03/2025 1:41:42 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Get Question and Its Choices by Question_ID
ALTER PROCEDURE [dbo].[GetQuestionWithChoices]
    @q_id INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE Question_ID = @q_id)
    BEGIN
        SELECT q.Question_ID, q.Question, qc.Choice
        FROM Question q
        LEFT JOIN Question_Choices qc ON q.Question_ID = qc.Question_ID
        WHERE q.Question_ID = @q_id;
    END
    ELSE
    BEGIN
        PRINT 'No Question Found with this ID!';
    END
END;
```

dbo.GetStudentByDepartment

Retrieves students who are enrolled in tracks within a specific department.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetStudentsByDepartment]    Script Date: 25/03/2025 1:46:48 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[GetStudentsByDepartment]
@DepartmentNo INT   ---GetStudentsByDepartment
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Department WHERE Department_ID = @DepartmentNo)
    BEGIN
        SELECT s.*
        FROM student s
        INNER JOIN Track t
        ON s.track_id = t.track_id
        INNER JOIN department d
        ON d.department_id = t.department_id
        WHERE d.department_id = @DepartmentNo
    END
    ELSE
    BEGIN
        PRINT 'This Department Does NOT Exist!';
    END
END;
```

dbo.GetStudentCountByInstructor

Returns the number of students taught by a specific instructor for each course.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetStudentCountByInstructor]    Script Date: 25/03/2025 1:42:01 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetStudentCountByInstructor]
    @InstructorID INT
AS
BEGIN
--    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @InstructorID)
    BEGIN
        PRINT 'Instructor ID not found.';
        RETURN;
    END

    SELECT i.Instructor_Name,
           c.Course_Name,
           COUNT(s.Student_ID) AS 'Students Count'
    FROM Instructor i
    JOIN Ins_Teaches_Crs it ON i.Instructor_ID = it.Instructor_ID
    JOIN Course c ON c.Course_ID = it.Course_ID
    JOIN St_Studies_Crs sc ON c.Course_ID = sc.Course_ID
    JOIN Student s ON s.Student_ID = sc.Student_ID
    WHERE i.Instructor_ID = @InstructorID
    GROUP BY c.Course_Name, i.Instructor_Name;
END
```

dbo.GetStudentExamAnswers

Displays student answers and whether each answer is correct or incorrect for a specific exam.

```sql
ALTER PROCEDURE [dbo].[GetStudentExamAnswers]
    @ExamID INT,
    @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

        -- Check if the Exam exists
        IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam_ID = @ExamID)
        BEGIN
            PRINT 'Error: Exam does not exist.';
            RETURN;
        END

        -- Check if the Student exists
        IF NOT EXISTS (SELECT 1 FROM Student WHERE Student_ID = @StudentID)
        BEGIN
            PRINT 'Error: Student does not exist.';
            RETURN;
        END

        -- Check if the Student has taken the Exam
        IF NOT EXISTS (SELECT 1 FROM St_Takes_Exams WHERE Exam_ID = @ExamID AND Student_ID = @StudentID)
        BEGIN
            PRINT 'This Student did not take exam: No answers found for this student in the given exam.';
            RETURN;
        END

        -- Retrieve Exam Questions and Student Answers
        SELECT
            Q.Question_ID,
            Q.Question,
            Q.Answer AS Correct_Choice,
            SE.Student_Answer AS Student_Choice,
            CASE
                WHEN SE.Student_Answer = Q.Answer THEN 'Correct'
                ELSE 'Incorrect'
            END AS Answer_Status
        FROM Question Q
        LEFT JOIN St_Takes_Exams SE
            ON Q.Question_ID = SE.Question_ID AND SE.Exam_ID = @ExamID AND SE.Student_ID = @StudentID;

        PRINT 'Success: Exam answers retrieved successfully.';
END;
```

dbo.GetStudentGrades

Retrieves a student's grades across exams and courses with calculated percentages.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetStudentGrades]    Script Date: 25/03/2025 1:45:57 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[GetStudentGrades]
@StudentID INT     -- Report 2
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Student WHERE Student_ID = @StudentID)
    BEGIN
            SELECT
    s.Student_Id, ste.Exam_Id,
    CONCAT(s.fname, ' ', s.lname) AS[ Full Name],
    c.Course_Name,
    CONCAT(CAST(SUM(ste.Grade) * 100.0 / E.MARK AS DECIMAL(5,0)), '%') AS [Grade %]
        FROM Student s
        JOIN St_Studies_Crs stc ON s.Student_ID = stc.Student_ID
        JOIN Course c ON stc.Course_ID = c.Course_ID
        JOIN Question q ON q.Course_ID = c.Course_ID
        JOIN St_Takes_Exams ste ON q.Question_ID = ste.Question_ID AND s.Student_ID = ste.Student_ID
        JOIN Exam e ON e.Exam_ID = ste.Exam_ID
        WHERE s.Student_ID = @StudentID
        GROUP BY c.Course_Name, s.student_id,ste.Exam_Id, s.fname, s.lname, E.MARK;
    END
    ELSE
        PRINT 'This Student ID Does NOT Exist!'
END
```

dbo.GetTrackByID

Retrieves track details by the provided Track ID.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[GetTrackByID]    Script Date: 25/03/2025 1:49:15 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetTrackByID]
    @Track_ID INT
AS
BEGIN
    SELECT  *
    FROM Track
    WHERE Track_ID = @Track_ID;
END;
```

dbo.InsertBranch

Inserts a new branch into the system after ensuring it doesn't already exist.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertBranch]    Script Date: 25/03/2025 1:50:01 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertBranch]
    @BranchName VARCHAR(20),
    @Branchloc  VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS (SELECT 1 FROM Branch WHERE Branch_Name = @BranchName)
        BEGIN
            PRINT 'This Branch Already Exists!';
            RETURN;
        END
        INSERT INTO Branch (Branch_Name, Branch_Location) VALUES (@Branchname, @Branchloc);
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while inserting into Branch';
    END CATCH
END;
```

dbo.InsertCompany

Inserts a new company into the system after validating the type value.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertCompany]    Script Date: 25/03/2025 1:50:38 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[InsertCompany]
@company_name VARCHAR(20),
@location VARCHAR(50),
@type VARCHAR(50)
AS
BEGIN
    IF lower(@type) NOT IN ('national', 'multi-national', 'multi_national', 'multinational')
    BEGIN
        PRINT 'The Company type should be either [National] or [Multi-National]';
        PRINT 'Please, Enter a valid type';
        RETURN;
    END
    ELSE
    BEGIN
        INSERT INTO Company (Name, Location, Type)
        VALUES (@company_name, @location, @type)
    END
END
```

dbo.InsertCourse

Adds a new course with its name and duration.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertCourse]    Script Date: 25/03/2025 1:51:09 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertCourse]
    @CrsName NVARCHAR(100) ,
    @CrsDuration INT
AS
BEGIN
--    SET NOCOUNT ON;
    insert into Course(Course_Name , Duration_Hrs)
    values(@CrsName , @CrsDuration)


    PRINT 'Course inserted successfully!';
END;
```

dbo.InsertDepartment

Inserts a new department after checking for existing ID, name, or manager conflicts.

```sql
ALTER PROCEDURE [dbo].[InsertDepartment]
    @DepartmentID INT,
    @DepartmentName VARCHAR(100),
    @Departmentmanager INT
AS
BEGIN
    SET NOCOUNT ON;
   -- BEGIN TRY
        IF EXISTS (SELECT 1 FROM Department WHERE Department_ID = @DepartmentID)
        BEGIN
            PRINT 'This Department Already Exists!';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Department WHERE Department_Name = @DepartmentName)
        BEGIN
            PRINT 'This Department Already Exists!';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Department WHERE Department_Manager = @Departmentmanager)
        BEGIN
            PRINT 'This Instructor is Already A Manager for another Department!';
            RETURN;
        END
        IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Departmentmanager)
        BEGIN
            PRINT 'This Department Manager ID Does NOT Exist in the Instructor Table!';
            RETURN;
        END
        INSERT INTO Department (Department_ID, Department_Name, Department_Manager) VALUES (@DepartmentID, @DepartmentName, @Departmentmanager);
   /* END TRY
    BEGIN CATCH
        PRINT 'Error occurred while inserting into Department';
    END CATCH*/
END;
```

dbo.InsertExam

Inserts a new exam into the system after validating input parameters.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertExam]    Script Date: 25/03/2025 1:51:47 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertExam]
    @Exam_Date DATE = NULL,
    @Exam_Duration INT = NULL,
    @Mark INT = 10
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate input parameters
    IF @Exam_Date IS NULL OR @Exam_Duration IS NULL OR @Mark IS NULL
    BEGIN
        PRINT 'Error: All parameters (Exam_Date, Exam_Duration, and Mark) must be provided.';
        RETURN;
    END

    INSERT INTO Exam (Exam_Date, Exam_Duration, Mark)
    VALUES (@Exam_Date, @Exam_Duration, @Mark);

    PRINT 'Exam  inserted successfully.';
END;
```

dbo.InsertInstructor

Inserts a new instructor after validating department and address references.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertInstructor]    Script Date: 25/03/2025 1:51:50 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertInstructor]
    @Instructor_ID INT,
    @Instructor_Name VARCHAR(20),
    @SSN VARCHAR(14),
    @Salary INT,
    @DOB DATE,
    @Email VARCHAR(100),
    @ZipCode VARCHAR(5),
    @University VARCHAR(100),
    @Hiring_Date DATE,
    @Department_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Department WHERE Department_ID = @Department_ID)
    BEGIN
        PRINT 'Error: Department_ID does not exist';
        RETURN;
    END
    IF NOT EXISTS (SELECT 1 FROM Address WHERE ZipCode = @ZipCode)
    BEGIN
        PRINT 'Error: ZipCode does not exist';
        RETURN;
    END

    INSERT INTO Instructor (Instructor_ID, Instructor_Name, SSN, Salary, DOB, Email, ZipCode, University, Hiring_Date, Department_ID)
    VALUES (@Instructor_ID, @Instructor_Name, @SSN, @Salary, @DOB, @Email, @ZipCode, @University, @Hiring_Date, @Department_ID);
END;
```

dbo.InsertIntake

Inserts a new intake if it doesn't already exist and has a valid intake type (PTP or ICC).

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[InsertIntake]    Script Date: 25/03/2025 1:54:42 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[InsertIntake]
    @IntakeNum INT,
    @Intaketype VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS (SELECT 1 FROM Intake WHERE Intake_Number = @IntakeNum)
        BEGIN
            PRINT 'This Intake Already Exist'
        END
        ELSE IF UPPER(@Intaketype) NOT IN ('PTP', 'ICC')
        BEGIN
            PRINT 'The Intake Type should be either PTP or ICC'
        END
        ELSE
        BEGIN
            INSERT INTO Intake (Intake_Number, Type) VALUES (@IntakeNum, Upper(@Intaketype));
        END
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while inserting into Intake';
    END CATCH
END;
```

dbo.InsertMCQ_Question

Inserts a new multiple-choice question and its four choices for a given course and instructor.

```
ALTER PROC [dbo].[InsertMCQ_Question]
    @ins_id INT,
    @crs_id INT,
    @q VARCHAR(1000),
    @choice1 VARCHAR(200),
    @choice2 VARCHAR(200),
    @choice3 VARCHAR(200),
    @choice4 VARCHAR(200),
    @ans VARCHAR(200)
AS
BEGIN
    DECLARE @new_q_id INT

    IF EXISTS (SELECT 1 FROM Course WHERE Course_ID = @crs_id) AND EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @ins_id)
    BEGIN
        IF EXISTS (SELECT 1 FROM Question WHERE Question = @q AND Course_ID = @crs_id AND Instructor_ID = @ins_id)
        BEGIN
            PRINT 'This Question Already Exists!';
            RETURN;
        END
        ELSE
        BEGIN
        -- Insert the question
        INSERT INTO Question (Question, Answer, Question_Type, Course_ID, Instructor_ID)
        VALUES (@q, @ans, 'MCQ', @crs_id, @ins_id)
        -- Get the current question_id
        SET @new_q_id = SCOPE_IDENTITY();
        -- INSSERT the choices for this question
        INSERT INTO Question_Choices (Question_ID, Choice)
        VALUES (@new_q_id, @choice1),
               (@new_q_id, @choice2),
               (@new_q_id, @choice3),
               (@new_q_id, @choice4)
        END
    END
    ELSE
    BEGIN
        PRINT 'This Course Or Instructor Does NOT Exist, Please Choose Another Course ID or Enter another Instructor ID!';
    END

END;
```

dbo.InsertStudent

Validates and inserts a new student after checking all input constraints and foreign key references.

```
ALTER PROC [dbo].[InsertStudent]
(@SSN VARCHAR(14), @FName VARCHAR(20), @LName VARCHAR(20), @Gender VARCHAR(1), @Email VARCHAR(50), @DOB DATE,
 @University VARCHAR(100), @Faculty VARCHAR(50), @GraduationYear INT, @MilitaryStatus VARCHAR(15), @Freelance INT = 0,
 @No_Certificates INT = 0, @WorkStatus VARCHAR(15) = 'Unemployed', @Position VARCHAR(50) = 'NA', @HiringDate DATE = NULL,
 @IntakeStartDate DATE, @ZipCode VARCHAR(5), @Track_ID INT, @Intake_ID INT, @Company_ID INT = NULL, @Branch_ID INT)
AS
BEGIN
    BEGIN TRY
        -- Validate SSN length
        IF LEN(CAST(@SSN AS VARCHAR(14))) <> 14
        BEGIN
            SELECT CAST('Error: National ID must be 14 digits.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Gender input
        IF Upper(@Gender) NOT IN ('M', 'F')
        BEGIN
            SELECT CAST('Error: Gender must be M or F.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Graduation Year (must be within the last 10 years and not in the future)
        IF @GraduationYear < (YEAR(GETDATE()) - 10) OR @GraduationYear > YEAR(GETDATE())
        BEGIN
            SELECT CAST('Error: Graduation Year must be within the last 10 years and not in the future.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Work_Status input (must be 'Employed' or 'Unemployed')
        IF lower(@WorkStatus) NOT IN ('employed', 'unemployed')
        BEGIN
            SELECT CAST('Error: Work Status must be either Employed or Unemployed.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Hiring Date (cannot be in the future)
        IF @HiringDate > GETDATE()
        BEGIN
            SELECT CAST('Error: Hiring Date cannot be in the future.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Military Status based on Gender
        IF Upper(@Gender) = 'M' AND @MilitaryStatus NOT IN ('Completed', 'Exempted', 'Postponed')
        BEGIN
            SELECT CAST('Error: Military Status for males must be Completed, Exempted, or Postponed.' AS VARCHAR(100));
            RETURN;
        END
        ELSE IF Upper(@Gender) = 'F' AND @MilitaryStatus <> 'NA'
        BEGIN
            SELECT CAST('Error: Military Status for females must be NA.' AS VARCHAR(100));
            RETURN;
```

Continued logic for inserting a student, including further validations on IDs and relationships.

```
        BEGIN
            SELECT CAST('Error: Military Status for females must be NA.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate that Intake Start Date is before Hiring Date
        IF @IntakeStartDate > @HiringDate
        BEGIN
            SELECT CAST('Error: Intake Start Date must be before Hiring Date.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Freelance (must be 0 or 1)
        IF @Freelance NOT IN (0, 1)
        BEGIN
            SELECT CAST('Error: Freelance must be either 0 or 1 as 1: Yes or 0: No.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate No_Certificates (must be max 3)
        IF @No_Certificates > 3 OR @No_Certificates < 0
        BEGIN
            SELECT CAST('Error: No_Certificates must be in (0,1,2,3).' AS VARCHAR(100));
            RETURN;
        END
        -- Check if the student already exists (by StudentID or SSN)
        IF EXISTS (SELECT 1 FROM student WHERE SSN = @SSN)
        BEGIN
            SELECT CAST('Error: Student with this SSN already exists.' AS VARCHAR(100));
            RETURN;
        END
          -- Check if foreign key values exist in parent tables
        IF NOT EXISTS (SELECT 1 FROM address WHERE ZipCode = @ZipCode)
        BEGIN
            SELECT CAST('Error: Invalid ZipCode. It is not a valid address.' AS VARCHAR(100));
            RETURN;
        END
        IF NOT EXISTS (SELECT 1 FROM Track WHERE Track_ID = @Track_ID)
        BEGIN
            SELECT CAST('Error: Invalid Track_ID. Please choose an existing Track_ID' AS VARCHAR(100));
            RETURN;
        END
        IF NOT EXISTS (SELECT 1 FROM Intake WHERE Intake_ID = @Intake_ID)
        BEGIN
            SELECT CAST('Error: Invalid Intake_ID. Please choose an existing Intake_ID.' AS VARCHAR(100));
            RETURN;
        END
         IF @Company_ID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM company WHERE Company_ID = @Company_ID)
        BEGIN
            SELECT CAST('This Company ID does NOT exist, Please First Insert The Company Details in the Company Table.' AS VARCHAR(100));
            RETURN;
```

Final segment for inserting a student record, including data insertion and error handling.

```
         IF @Company_ID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM company WHERE Company_ID = @Company_ID)
        BEGIN
            SELECT CAST('This Company ID does NOT exist, Please First Insert The Company Details in the Company Table.' AS VARCHAR(100));
            RETURN;
        END
        IF NOT EXISTS (SELECT 1 FROM Branch WHERE Branch_ID = @Branch_ID)
        BEGIN
            SELECT CAST('Error: Invalid Branch_ID. Please choose an existing Branch_ID.' AS VARCHAR(100));
            RETURN;
        END
        -- Insert the new student record
        INSERT INTO student
        (SSN, FName, LName, Gender, Email, DOB, University, Faculty, Graduation_Year, Military_Status, Freelance, No_Certificates,
         Work_Status, Position, Hiring_Date, Intake_Start_Date, ZipCode, Track_ID, Intake_ID, Company_ID,Branch_ID)
        VALUES
        (@SSN, @FName, @LName, @Gender, @Email, @DOB, @University, @Faculty, @GraduationYear, @MilitaryStatus, @Freelance,
         @No_Certificates, @WorkStatus, @Position, @HiringDate, @IntakeStartDate, @ZipCode, @Track_ID, @Intake_ID, @Company_ID,
         @Branch_ID);
        -- Confirm successful insertion
        SELECT CAST('Success: Student record inserted successfully.' AS VARCHAR(100));
    END TRY
BEGIN CATCH
        -- Handle unexpected SQL errors
        SELECT CAST('Error: An unexpected error occurred.' AS VARCHAR(100));
    END CATCH
END;
```

dbo.InsertTF_Question

Inserts a true/false type question with associated choices for a given instructor and course.

```
ALTER PROC [dbo].[InsertTF_Quesstion]
@ins_id INT,
@crs_id INT,
@q VARCHAR(1000),
@ans VARCHAR(200)
AS
BEGIN
    DECLARE @new_q_id INT
    IF EXISTS (SELECT 1 FROM Course WHERE Course_ID = @crs_id) AND EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @ins_id)
    BEGIN
        IF EXISTS (SELECT 1 FROM Question WHERE Question = @q AND Course_ID = @crs_id AND Instructor_ID = @ins_id)
        BEGIN
            PRINT 'This Question Already Exists!';
            RETURN;
        END
        ELSE
        BEGIN
        -- Insert the question
        INSERT INTO Question (Question, Answer, Question_Type, Course_ID, Instructor_ID)
        VALUES (@q, @ans, 'T/F', @crs_id, @ins_id)
        -- get the current question_id
        SET @new_q_id = SCOPE_IDENTITY();
        -- insert the true and false choices along with the question_id
        INSERT INTO Question_Choices (Question_ID, Choice)
        VALUES (@new_q_id, 'True'),
               (@new_q_id, 'False')
        END
    END
    ELSE
    BEGIN
        PRINT 'This Course Or Instructor Does NOT Exist, Please Choose Another Course ID or Enter another Instructor ID!';
    END
END;
```

dbo.InsertTopic

Inserts a new topic under a specific course if it doesn't already exist.

```
ALTER PROCEDURE [dbo].[InsertTopic]
    @Topic_Id int ,
    @Topic_name varchar(500) ,
    @crs_id int
as
begin



declare @Crsavailability int

        IF EXISTS (SELECT 1 FROM Topic WHERE Topic_ID = @Topic_Id AND Course_ID = @crs_id)
        BEGIN
            PRINT 'This Topic ID Already Exists With This Course, Please Enter New Data'
            RETURN;
        END
select @Crsavailability = count(*)
from Course
where Course_ID = @crs_id

    if @Crsavailability = 0
    begin
        print 'course is not found'
    end
    else if  @Crsavailability > 0 and  @Topic_Id is not null and @Topic_name is not null and @crs_id is not null
    begin
    insert into Topic values(@Topic_Id,@Topic_name,@crs_id)
    end
    else
    begin
       print 'Enter The Topic ID The Topic Name and The Course ID'
    end

end;
```

dbo.ReadAllExams

Retrieves all exams from the Exam table.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadAllExams]    Script Date: 25/03/2025 2:11:23 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[ReadAllExams]
AS
BEGIN
Select *
FROM Exam
END
```

dbo.ReadAllStudent

Retrieves all student records from the Student table.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadAllStudent]    Script Date: 25/03/2025 2:11:56 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[ReadAllStudent]
AS
BEGIN
SELECT * FROM dbo.Student
END;
```

dbo.ReadBranch

Retrieves all branch records with error handling for data retrieval.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadBranch]    Script Date: 25/03/2025 2:12:25 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[ReadBranch]
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        SELECT * FROM Branch;
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while fetching Branch records';
    END CATCH
END;
```

dbo.ReadDepartment

Retrieves all department records with error handling.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadDepartment]    Script Date: 25/03/2025 2:12:53 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[ReadDepartment]
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        SELECT * FROM Department;
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while fetching Department records';
    END CATCH
END;
```

dbo.ReadExam

Retrieves an exam by ID after checking its existence.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadExam]    Script Date: 25/03/2025 2:13:21 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-----------------------------------------------------------------------------------------------------
ALTER PROCEDURE [dbo].[ReadExam]
    @Exam_ID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if Exam_ID exists
    IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam_ID = @Exam_ID)
    BEGIN
        PRINT 'Error: Exam ID does not exist.';
        RETURN;
    END

    -- Delete the exam record
    select * FROM Exam WHERE Exam_ID = @Exam_ID;

END;
```

dbo.ReadIntake

Retrieves all intake records with error handling.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadIntake]    Script Date: 25/03/2025 2:13:49 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[ReadIntake]
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        SELECT * FROM Intake;
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while fetching Intake records';
    END CATCH
END;
```

dbo.ReadStudent

Searches for student records using various criteria such as ID, SSN, or name and email combinations.

```sql
ALTER PROC [dbo].[ReadStudent]
    ( @StudentID INT= NULL,
    @SSN VARCHAR(14) = NULL,
    @FName VARCHAR(20) = NULL,
    @LName VARCHAR(20) = NULL,
    @Email VARCHAR(50) = NULL)

AS
BEGIN
    -- Search by StudentID
    IF @StudentID IS NOT NULL
    BEGIN
        IF EXISTS (SELECT 1 FROM student WHERE Student_ID = @StudentID)
            SELECT * FROM student WHERE Student_ID = @StudentID;
        ELSE
            SELECT CAST('Student ID is not found' AS VARCHAR(100));
        RETURN;
     END

    -- Search by SSN
        IF @SSN IS NOT NULL
        BEGIN
            IF LEN(@SSN) = 14
            BEGIN
                IF EXISTS (SELECT 1 FROM student WHERE SSN = @SSN)
                    SELECT * FROM student WHERE SSN = @SSN;
                ELSE
                    SELECT CAST('National ID is not found' AS VARCHAR(100));
            END
            ELSE
                SELECT CAST('National ID must be 14 digits.' AS VARCHAR(100));

            RETURN;
        END
    -- Search by First Name, Last Name, AND Email together
        IF @FName IS NOT NULL AND @LName IS NOT NULL AND @Email IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM student WHERE FName = @FName AND LName = @LName AND Email = @Email)
            BEGIN
                SELECT * FROM Student WHERE FName = @FName AND LName = @LName AND Email = @Email;
            END
            ELSE
            BEGIN
                SELECT CAST('Error: There is erorr with First Name or Last Name or and Email.' AS VARCHAR(100)) AS Message;
            END
            RETURN;
        END
```

Continued logic of the ReadStudent procedure, handling search by name combinations and email.

```
    -- Search by First Name AND Last Name together
        IF @FName IS NOT NULL AND @LName IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM student WHERE FName = @FName AND LName = @LName)
            BEGIN
                SELECT * FROM Student WHERE FName = @FName AND LName = @LName;
            END
            ELSE
            BEGIN
                SELECT CAST('Error: First and Last Name do not exist.' AS VARCHAR(100)) AS Message;
            END
            RETURN;
        END

    -- If the user provides only one of First Name or Last Name
        IF (@FName IS NOT NULL AND @LName IS NULL) OR (@FName IS NULL AND @LName IS NOT NULL)
        BEGIN
            SELECT CAST('Error: You must provide both First Name and Last Name together.' AS VARCHAR(100)) AS Message;
            RETURN;
        END
    -- Search by Email separately
        IF @Email IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM student WHERE Email = @Email)
            BEGIN
                SELECT * FROM Student WHERE Email = @Email;
            END
            ELSE
                SELECT CAST('Error: Email is not exists.' AS VARCHAR(100)) AS Message;
            RETURN;
        END

    -- If no parameters are provided
    SELECT CAST('No search criteria provided.' AS VARCHAR(100));
END;
```

dbo.ReadTopic

Searches for topics based on topic ID, name, or course ID, with flexible query support.

```
ALTER PROCEDURE [dbo].[ReadTopic]
    @TID INT = NULL,
    @TName NVARCHAR(100) = NULL,
    @CrsID INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    IF @TID IS NOT NULL OR @TName IS NOT NULL OR @CrsID IS NOT NULL
    BEGIN
        IF EXISTS (
            SELECT 1 FROM Topic
            WHERE (@TID IS NULL OR Topic_ID = @TID)
              AND (@TName IS NULL OR Topic_Name = @TName)
              AND (@CrsID IS NULL OR Course_ID = @CrsID)
        )
        BEGIN
            SELECT * FROM Topic
            WHERE (@TID IS NULL OR Topic_ID = @TID)
              AND (@TName IS NULL OR Topic_Name = @TName)
              AND (@CrsID IS NULL OR Course_ID = @CrsID);
        END
        ELSE
        BEGIN
            PRINT 'NOT EXISTS';
        END
    END
    ELSE
    BEGIN
        PRINT 'Please provide at least one parameter to search.';
    END
END;
```

dbo.ReadTrack

Retrieves all tracks from the Track table.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[ReadTrack]    Script Date: 25/03/2025 2:16:48 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[ReadTrack]
AS
BEGIN
    SELECT * FROM Track;
END;
```

dbo.SelectCourse

Selects a course by ID or name, with validation to ensure at least one input is provided.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[SelectCourse]    Script Date: 25/03/2025 2:17:18 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[SelectCourse]
    @CrsID INT = NULL,
    @CrsName NVARCHAR(100) = NULL
AS
BEGIN
--    SET NOCOUNT ON;


    -- Check if want to select with  ID or Name is
    IF @CrsID IS NULL AND @CrsName IS NULL
    BEGIN
        PRINT 'Error: Please enter either an ID or a Name to delete the Course.'
        RETURN;
    END

    Select * FROM Course
    WHERE (Course_ID = @CrsID OR @CrsName IS NOT NULL AND Course_Name = @CrsName);

END;
```

dbo.UpdateAnswer

Updates the answer text for a specific question by its ID.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateAnswer]    Script Date: 25/03/2025 2:17:55 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateAnswer]
    @q_id INT,
    @new_answer VARCHAR(200)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE Question_ID = @q_id)
    BEGIN
        UPDATE Question
        SET Answer = @new_answer
        WHERE Question_ID = @q_id;
    END
    ELSE
    BEGIN
        PRINT 'Question ID Does NOT Exist!';
    END
END;
```

dbo.UpdateBranchLocation

Updates the location of a specific branch by its ID, handling errors with TRY...CATCH.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateBranchLocation]    Script Date: 25/03/2025 2:18:23 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateBranchLocation]
    @BranchID INT,
    @NewBranchLoc VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS (SELECT 1 FROM Branch WHERE Branch_ID = @BranchID)
        BEGIN
            UPDATE Branch SET Branch_Location = @NewBranchLoc WHERE Branch_ID = @BranchID;
        END
        ELSE
        BEGIN
            PRINT 'This Branch Does NOT Exist!'
        END
    END TRY
    BEGIN CATCH
        PRINT 'Error Occurred while updating Branch';
    END CATCH
END;
```

dbo.UpdateBranchName

Updates the name of a specific branch by its ID with exception handling.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateBranchName]    Script Date: 25/03/2025 2:18:55 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateBranchName]
    @BranchID INT,
    @NewBranchName VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS (SELECT 1 FROM Branch WHERE Branch_ID = @BranchID)
        BEGIN
            UPDATE Branch SET Branch_Name = @NewBranchName WHERE Branch_ID = @BranchID;
        END
        ELSE
        BEGIN
            PRINT 'This Branch Does NOT Exist!'
        END
    END TRY
    BEGIN CATCH
        PRINT 'Error Occurred while updating Branch';
    END CATCH
END;
```

dbo.UpdateChoices

Replaces all choices for a specific question by deleting old ones and inserting new ones.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateChoices]    Script Date: 25/03/2025 2:19:27 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateChoices]
    @q_id INT,
    @choice1 VARCHAR(200),
    @choice2 VARCHAR(200),
    @choice3 VARCHAR(200),
    @choice4 VARCHAR(200)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE Question_ID = @q_id)
    BEGIN
        DELETE FROM Question_Choices WHERE Question_ID = @q_id;

        INSERT INTO Question_Choices(Question_ID, Choice) VALUES
            (@q_id, @choice1),
            (@q_id, @choice2),
            (@q_id, @choice3),
            (@q_id, @choice4);
    END
    ELSE
    BEGIN
        PRINT 'Question ID Does NOT Exist!';
    END
END;
```

dbo.UpdateCompanyLocation

Updates the location of a specific company based on its ID.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateCompanyLocation]    Script Date: 25/03/2025 2:19:58 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- Update
ALTER PROC [dbo].[UpdateCompanyLocation]
@company_id INT,
@new_location VARCHAR(50)
AS
BEGIN
    IF EXISTS  (SELECT 1 FROM Company WHERE Company_ID = @company_id)
    BEGIN
        UPDATE Company
        SET Location = @new_location
        WHERE Company_ID = @company_id
    END
    ELSE
    BEGIN
        PRINT 'There is NO Company with this ID, Please Enter Another Company ID!';
    END
END
```

dbo.UpdateCompanyName

Updates the name of a specific company based on its ID.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateCompanyName]    Script Date: 25/03/2025 2:20:38 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
 -- Update Company's Name
ALTER PROC [dbo].[UpdateCompanyName]
@company_id INT,
@new_name VARCHAR(50)
AS
BEGIN
    IF EXISTS  (SELECT 1 FROM Company WHERE Company_ID = @company_id)
    BEGIN
        UPDATE Company
        SET Name = @new_name
        WHERE Company_ID = @company_id
    END
    ELSE
    BEGIN
        PRINT 'There is NO Company with this ID, Please Enter Another Company ID!';
    END
END
```

dbo.UpdateCompanyType

Updates the type of a company (e.g., national or multinational) by company ID.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateCompanyType]    Script Date: 25/03/2025 2:21:13 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[UpdateCompanyType]
@company_id INT,
@new_type VARCHAR(50)
AS
BEGIN
    IF EXISTS  (SELECT 1 FROM Company WHERE Company_ID = @company_id)
    BEGIN
        UPDATE Company
        SET Type = @new_type
        WHERE Company_ID = @company_id
    END
    ELSE
    BEGIN
        PRINT 'There is NO Company with this ID, Please Enter Another Company ID!';
    END
END
```

dbo.UpdateCourse

Updates the name and duration of a course by its ID, using ISNULL for optional updates.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateCourse]    Script Date: 25/03/2025 2:21:47 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateCourse]
    @CrsID INT,
    @CrsName NVARCHAR(100) = NULL,
    @CrsDuration INT = NULL
AS
BEGIN
--    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM Course WHERE Course_ID = @CrsID)
    BEGIN
        UPDATE Course
        SET
            Course_Name = ISNULL(@CrsName, Course_Name),
            Duration_Hrs = ISNULL(@CrsDuration, Duration_Hrs)
        WHERE Course_ID = @CrsID;

        PRINT 'Course updated successfully!';
    END
    ELSE
    BEGIN
        PRINT 'This Course ID Does NOT Exist, Please Enter an Existing Course!'
    END
END;
```

dbo.UpdateDepartmentManager

Updates the manager of a department if the department exists and the new manager is valid.

```sql
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateDepartmentManager]    Script Date: 25/03/2025 2:22:17 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateDepartmentManager]
    @DepartmentID INT,
    @NewManagerID INT
AS
BEGIN
    SET NOCOUNT ON;
        IF NOT EXISTS (SELECT 1 FROM Department WHERE Department_ID = @DepartmentID)
        BEGIN
            PRINT 'Error: Department does not exist.';
            RETURN;
        END
        IF EXISTS (SELECT 1 FROM Department WHERE Department_Manager = @NewManagerID)
        BEGIN
            PRINT 'This Instructor ID is Already A Manager for another Department.';
            RETURN;
        END
        UPDATE Department
        SET Department_Manager = @NewManagerID
        WHERE Department_ID = @DepartmentID;
        PRINT 'Success: Department Manager updated successfully.';
END;
```

dbo.UpdateDepartmentName

Updates the name of a department by its ID using a TRY...CATCH block for error handling.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateDepartmentName]    Script Date: 25/03/2025 2:22:56 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateDepartmentName]
    @DepartmentID INT,
    @NewDepartmentName VARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        IF EXISTS (SELECT 1 FROM Department WHERE Department_ID = @DepartmentID)
            BEGIN
                UPDATE Department SET Department_Name = @NewDepartmentName WHERE Department_ID = @DepartmentID;
            END
        ELSE
            BEGIN
                PRINT 'This Department ID Does NOT Exist'
            END
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while updating Department';
    END CATCH
END;
```

dbo.UpdateExam

Updates an exam record if it exists, using COALESCE to keep existing values.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateExam]    Script Date: 25/03/2025 2:23:54 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER procedure [dbo].[UpdateExam]
    @E_id int ,
    @E_date date =null,
    @E_duration int =null ,
    @E_mark int =null
as
BEGIN
    SET NOCOUNT ON;

    -- Check if Exam_ID exists
    IF NOT EXISTS (SELECT 1 FROM Exam WHERE Exam_ID = @E_id)
    BEGIN
        PRINT 'Exam ID does not exist.';
        RETURN;
    END

    -- Update the record with only the provided parameters
    UPDATE Exam
    SET Exam_Date = COALESCE(@E_date, Exam_Date),
        Exam_Duration = COALESCE(@E_duration, Exam_Duration),
        Mark = COALESCE(@E_mark, Mark)
    WHERE Exam_ID = @E_id;

    PRINT 'Exam updated successfully.';
END;
```

dbo.UpdateFullQuestion

Updates a question's text and answer, deletes old choices, and reinserts new ones.

```
ALTER PROCEDURE [dbo].[UpdateFullQuestion]
    @q_id INT,
    @new_question_text VARCHAR(1000),
    @new_answer VARCHAR(200),
    @choice1 VARCHAR(200),
    @choice2 VARCHAR(200),
    @choice3 VARCHAR(200),
    @choice4 VARCHAR(200)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE question_id = @q_id)
    BEGIN
        UPDATE Question
        SET Question = @new_question_text, Answer = @new_answer
        WHERE Question_ID = @q_id;

        DELETE FROM Question_Choices WHERE Question_ID = @q_id;

        INSERT INTO Question_Choices (Question_ID, Choice) VALUES
            (@q_id, @choice1),
            (@q_id, @choice2),
            (@q_id, @choice3),
            (@q_id, @choice4);
    END
    ELSE
    BEGIN
        PRINT 'Question ID Does NOT Exist!';
    END
END;
```

dbo.UpdateInstructor

Updates instructor details; stops execution if the instructor doesn't exist.

```
ALTER PROCEDURE [dbo].[UpdateInstructor]
    @Instructor_ID INT,
    @Instructor_Name VARCHAR(20),
    @SSN VARCHAR(14),
    @Salary INT,
    @DOB DATE,
    @Email VARCHAR(100),
    @ZipCode VARCHAR(5),
    @University VARCHAR(100),
    @Hiring_Date DATE,
    @Department_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Instructor_ID)
    BEGIN
        PRINT 'Instructor not found!';
        RETURN;
    END;
    UPDATE Instructor
    SET
        Instructor_Name = @Instructor_Name,
        SSN = @SSN,
        Salary = @Salary,
        DOB = @DOB,
        Email = @Email,
        ZipCode = @ZipCode,
        University = @University,
        Hiring_Date = @Hiring_Date,
        Department_ID = @Department_ID
    WHERE Instructor_ID = @Instructor_ID;
END;
```

dbo.UpdateIntake

Updates an intake, ensuring uniqueness and valid intake types (PTP or ICC).

```
CREATE PROCEDURE UpdateIntake
    @IntakeID INT,
    @NewIntakeNum INT,
    @Intaketype VARCHAR(20)
WITH ENCRYPTION
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM Intake WHERE Intake_ID = @IntakeID)
        BEGIN
            PRINT 'This Intake Does NOT Exist'
        END
        IF EXISTS (SELECT 1 FROM Intake WHERE Intake_Number = @NewIntakeNum)
        BEGIN
            PRINT 'This Intake Number Already Exist'
        END
        ELSE IF UPPER(@Intaketype) NOT IN ('PTP', 'ICC')
        BEGIN
            PRINT 'The Intake Type should be either PTP or ICC'
        END
        ELSE
        BEGIN
            UPDATE Intake SET Intake_Number = @NewIntakeNum, Type = @Intaketype WHERE Intake_ID = @IntakeID;
        END
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred while updating Intake';
    END CATCH
END;
```

dbo.UpdateQuestionText

Updates only the text of a question if the question exists.

```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateQuestionText]    Script Date: 25/03/2025 2:26:21 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateQuestionText]
    @q_id INT,
    @new_question_text VARCHAR(1000)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Question WHERE question_id = @q_id)
    BEGIN
        UPDATE Question
        SET Question = @new_question_text
        WHERE Question_ID = @q_id;
    END
    ELSE
    BEGIN
        PRINT 'Question ID Does NOT Exist!';
    END
END;
```

dbo.UpdateStudent

Updates student info with validations for existence, gender input, and graduation year.

```
-- Stored Procedure to Update data on student table
ALTER PROC [dbo].[UpdateStudent]
    (@StudentID INT, @SSN VARCHAR = NULL, @FName VARCHAR(20) = NULL, @LName VARCHAR(20) = NULL, @Gender VARCHAR(1) = NULL,
     @Email VARCHAR(50) = NULL, @DOB DATE = NULL, @University VARCHAR(100) = NULL, @Faculty VARCHAR(50) = NULL,
     @GraduationYear INT = NULL, @MilitaryStatus VARCHAR(15) = NULL, @Freelance INT = NULL, @No_Certificates INT = NULL,
     @WorkStatus VARCHAR(15) = NULL, @Position VARCHAR(50) = NULL, @HiringDate DATE = NULL, @IntakeStartDate DATE = NULL,
     @ZipCode VARCHAR(5) = NULL, @Track_ID INT = NULL, @Intake_ID INT = NULL, @Company_ID INT = NULL, @Branch_ID INT = NULL)
AS
BEGIN
    BEGIN TRY
        -- Check if student exists
        IF NOT EXISTS (SELECT 1 FROM student WHERE Student_ID = @StudentID)
        BEGIN
            SELECT CAST('Error: No student found with this SSN with this ID.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate SSN exists
        IF NOT EXISTS (SELECT 1 FROM student WHERE SSN = COALESCE(@SSN, SSN))
        BEGIN
            SELECT CAST('Error: No student found with this SSN.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Gender input (if provided)
        IF @Gender NOT IN ('M', 'F')
        BEGIN
            SELECT CAST('Error: Gender must be M or F.' AS VARCHAR(100));
            RETURN;
        END
        -- Validate Graduation Year (must be within the last 10 years)
        IF (@GraduationYear < YEAR(GETDATE()) - 10 OR @GraduationYear > YEAR(GETDATE()))
        BEGIN
            SELECT CAST('Error: Graduation Year must be within the last 10 years and not in the future.' AS VARCHAR(100));
            RETURN;
        END
```

Validates and updates student records, ensuring correct references.

```
-- Validate Hiring Date (cannot be in the future)
IF @HiringDate > GETDATE()
BEGIN
    SELECT CAST('Error: Hiring Date cannot be in the future.' AS VARCHAR(100));
    RETURN;
END
-- Validate Military Status based on Gender (if provided)
IF UPPER(@Gender) = 'M' AND @MilitaryStatus NOT IN ('Completed', 'Exempted', 'Postponed')
BEGIN
    SELECT CAST('Error: Military Status for males must be Completed, Exempted, or Postponed.' AS VARCHAR(100));
    RETURN;
END
ELSE IF UPPER(@Gender) = 'F' AND @MilitaryStatus <> 'NA'
BEGIN
    SELECT CAST('Error: Military Status for females must be NA.' AS VARCHAR(100));
    RETURN;
END
-- Validate that Intake Start Date is before Hiring Date
IF @IntakeStartDate > @HiringDate
BEGIN
    SELECT CAST('Error: Intake Start Date must be before Hiring Date.' AS VARCHAR(100));
    RETURN;
END
-- Validate Freelance (must be 0 or 1)
IF @Freelance NOT IN (0, 1)
BEGIN
    SELECT CAST('Error: Freelance must be either 0 or 1.' AS VARCHAR(100));
    RETURN;
END
-- Validate No_Certificates (must be max 3)
IF @No_Certificates > 3 OR @No_Certificates < 0
BEGIN
    SELECT CAST('Error: No_Certificates must be in (0,1,2,3).' AS VARCHAR(100));
    RETURN;
END
```

```sql
-- Validate Work_Status input (must be 'Employed' or 'Unemployed')
IF lower(@WorkStatus) NOT IN ('employed', 'unemployed')
BEGIN
    SELECT CAST('Error: Work Status must be either Employed or Unemployed.' AS VARCHAR(100));
    RETURN;
END
 -- Check if foreign key values exist in parent tables
IF NOT EXISTS (SELECT 1 FROM address WHERE ZipCode = @ZipCode)
BEGIN
    SELECT CAST('Error: Invalid ZipCode. It is not a valid address.' AS VARCHAR(100));
    RETURN;
END
IF NOT EXISTS (SELECT 1 FROM Track WHERE Track_ID = @Track_ID)
BEGIN
    SELECT CAST('Error: Invalid Track_ID. Please choose an existing Track_ID.' AS VARCHAR(100));
    RETURN;
END
IF NOT EXISTS (SELECT 1 FROM Intake WHERE Intake_ID = @Intake_ID)
BEGIN
    SELECT CAST('Error: Invalid Intake_ID.  Please choose an existing Intake_ID.' AS VARCHAR(100));
    RETURN;
END
IF @company_ID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Company WHERE Company_ID = @Company_ID)
BEGIN
    SELECT CAST('Error: This Company ID does NOT exist, Please First Insert The Company Details in the Company Table.' AS VARCHAR(100));
    RETURN;
END
IF NOT EXISTS (SELECT 1 FROM Branch WHERE Branch_ID = @Branch_ID)
BEGIN
    SELECT CAST('Error: Invalid Branch_ID. Please choose an existing Branch_ID.' AS VARCHAR(100));
    RETURN;
END
        -- Perform the update
        UPDATE student
        SET
            SSN = COALESCE(@SSN, SSN), FName = COALESCE(@FName, FName), LName = COALESCE(@LName, LName),
            Gender = COALESCE(@Gender, Gender), Email = COALESCE(@Email, Email), DOB = COALESCE(@DOB, DOB),
            University = COALESCE(@University, University), Faculty = COALESCE(@Faculty, Faculty),
            Graduation_Year = COALESCE(@GraduationYear, Graduation_Year),
            Military_Status = COALESCE(@MilitaryStatus, Military_Status), Freelance = COALESCE(@Freelance, Freelance),
            No_Certificates = COALESCE(@No_Certificates, No_Certificates), Work_Status = COALESCE(@WorkStatus, Work_Status),
            Position = COALESCE(@Position, Position), Hiring_Date = COALESCE(@HiringDate, Hiring_Date),
            Intake_Start_Date = COALESCE(@IntakeStartDate, Intake_Start_Date), ZipCode = COALESCE(@ZipCode, ZipCode),
            Track_ID = COALESCE(@Track_ID, Track_ID), Intake_ID = COALESCE(@Intake_ID, Intake_ID),
            Company_ID = COALESCE(@Company_ID, Company_ID), Branch_ID = COALESCE(@Branch_ID, Branch_ID)
        WHERE Student_ID = @StudentID OR SSN = @SSN
        -- Confirm successful update
        SELECT CAST('Success: Student record updated successfully.' AS VARCHAR(100));
    END TRY
    BEGIN CATCH
        -- Handle unexpected SQL errors
        SELECT CAST('Error: An unexpected error occurred.' AS VARCHAR(100));
    END CATCH
END;
```

dbo.UpdateTopic

Updates a topic’s name within a course.

```
ALTER PROCEDURE [dbo].[UpdateTopic]
    @TID INT,
    @TName NVARCHAR(100) ,
    @CrsID INT
AS
BEGIN
DECLARE @Crsavailability int ;

--    SET NOCOUNT ON;
      IF NOT EXISTS (SELECT 1 FROM Topic WHERE Topic_ID = @TId AND Course_ID = @CrsID)
      BEGIN
          PRINT 'This Topic ID Does NOT Exist With This Course, Please Enter Existing Data'
          RETURN;
      END

select @Crsavailability = count(*)
from Course
where Course_ID = @CrsID

    IF @TID IS NULL Or @CrsID IS NULL
    BEGIN
        PRINT 'Error:Please enter both the topic ID and the course id .'
        RETURN;
    END
74
    if @Crsavailability = 0
    begin
    print 'Course Not Found : please insert the course first'
    end

    else
    begin
    UPDATE Topic
    SET
        Topic_Name = ISNULL(@TName, Topic_Name)
        WHERE Topic_ID = @TID
        and Course_ID = @CrsID  ;

    PRINT 'Topic updated successfully!';
    end
END;
```

dbo.UpdateTrack

Modifies track details, ensuring valid dependencies.

```
ALTER PROCEDURE [dbo].[UpdateTrack]
    @Track_ID INT,
    @Track_Name VARCHAR(100),
    @Track_Manager INT,
    @Department_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Track WHERE Track_ID = @Track_ID)
    BEGIN
        PRINT 'Track not found!';
        RETURN;
    END;
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Track_Manager)
    BEGIN
        PRINT 'Track Manager (Instructor) not found!';
        RETURN;
    END;
    IF NOT EXISTS (SELECT 1 FROM Department WHERE Department_ID = @Department_ID)
    BEGIN
        PRINT 'Department not found!';
        RETURN;
    END;
    UPDATE Track
    SET
        Track_Name = @Track_Name,
        Track_Manager = @Track_Manager,
        Department_ID = @Department_ID
    WHERE Track_ID = @Track_ID;;
END;
```

dbo.UpdateTrackManager

Updates the manager (instructor) of a specific track.

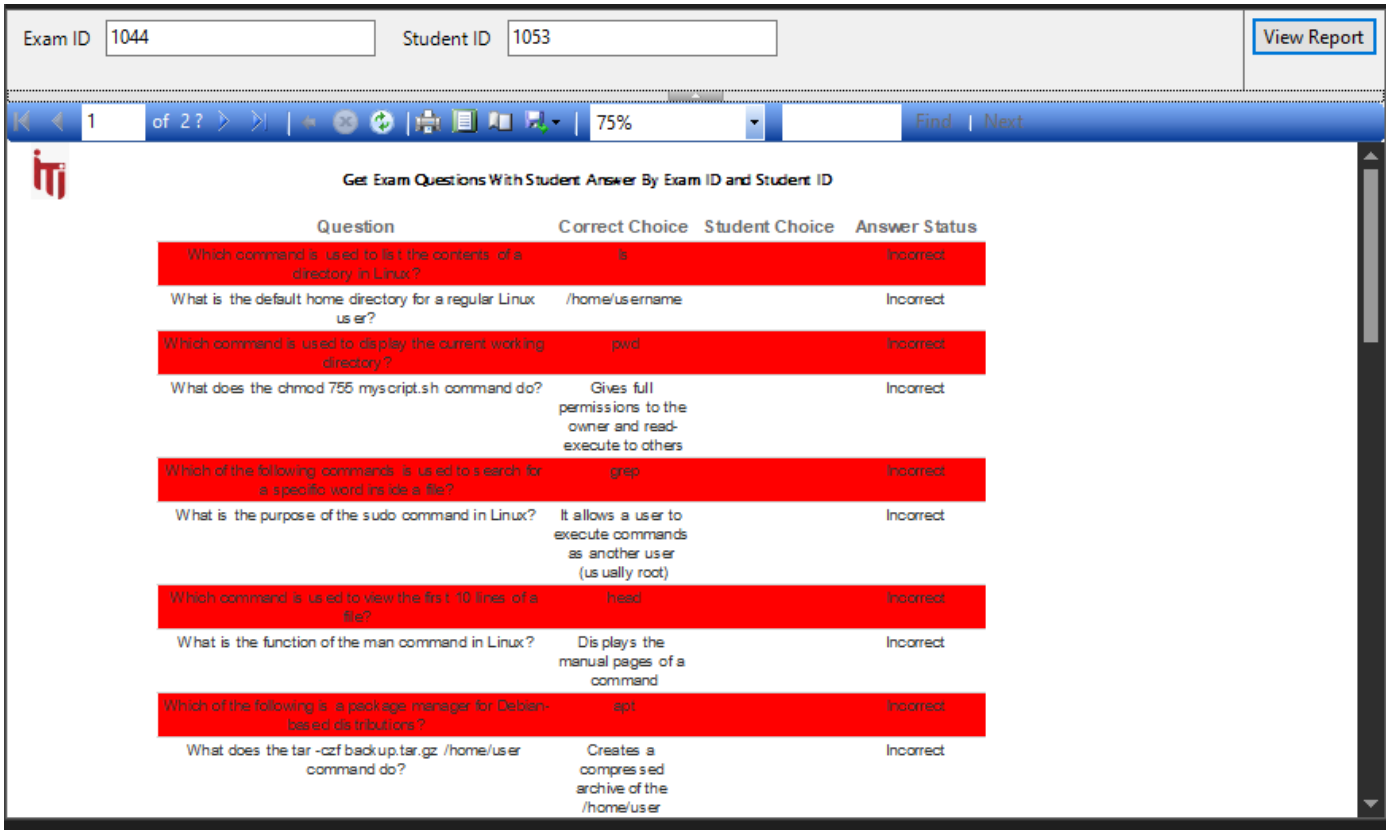
```
USE [Examination System]
GO
/****** Object:  StoredProcedure [dbo].[UpdateTrackManager]    Script Date: 25/03/2025 2:37:46 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateTrackManager]
    @Track_ID INT,
    @Instructor_ID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Track WHERE Track_ID = @Track_ID)
    BEGIN
        PRINT 'Track not found!';
        RETURN;
    END;
    IF NOT EXISTS (SELECT 1 FROM Instructor WHERE Instructor_ID = @Instructor_ID)
    BEGIN
        PRINT 'Instructor not found!';
        RETURN;
    END;

    UPDATE Track
    SET Track_Manager = @Instructor_ID
    WHERE Track_ID = @Track_ID;
END;
```

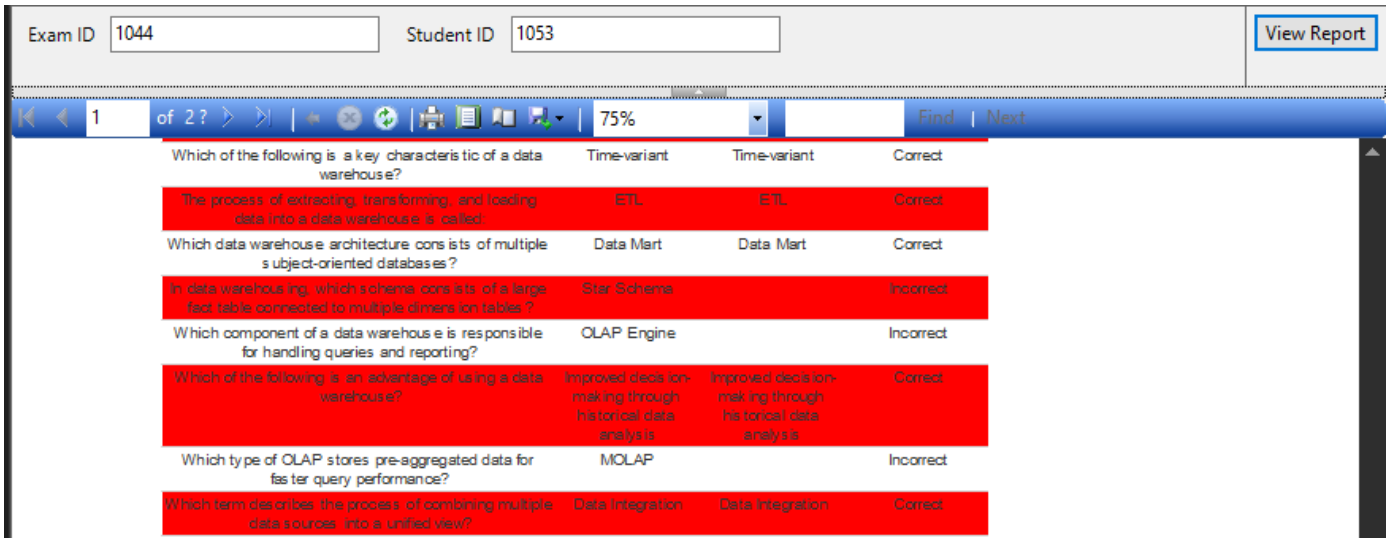
Chapter 4: Dashboards and reports

4.1 SSRS Reports

- **Get exam Question and answer by Exam ID and Student Id**

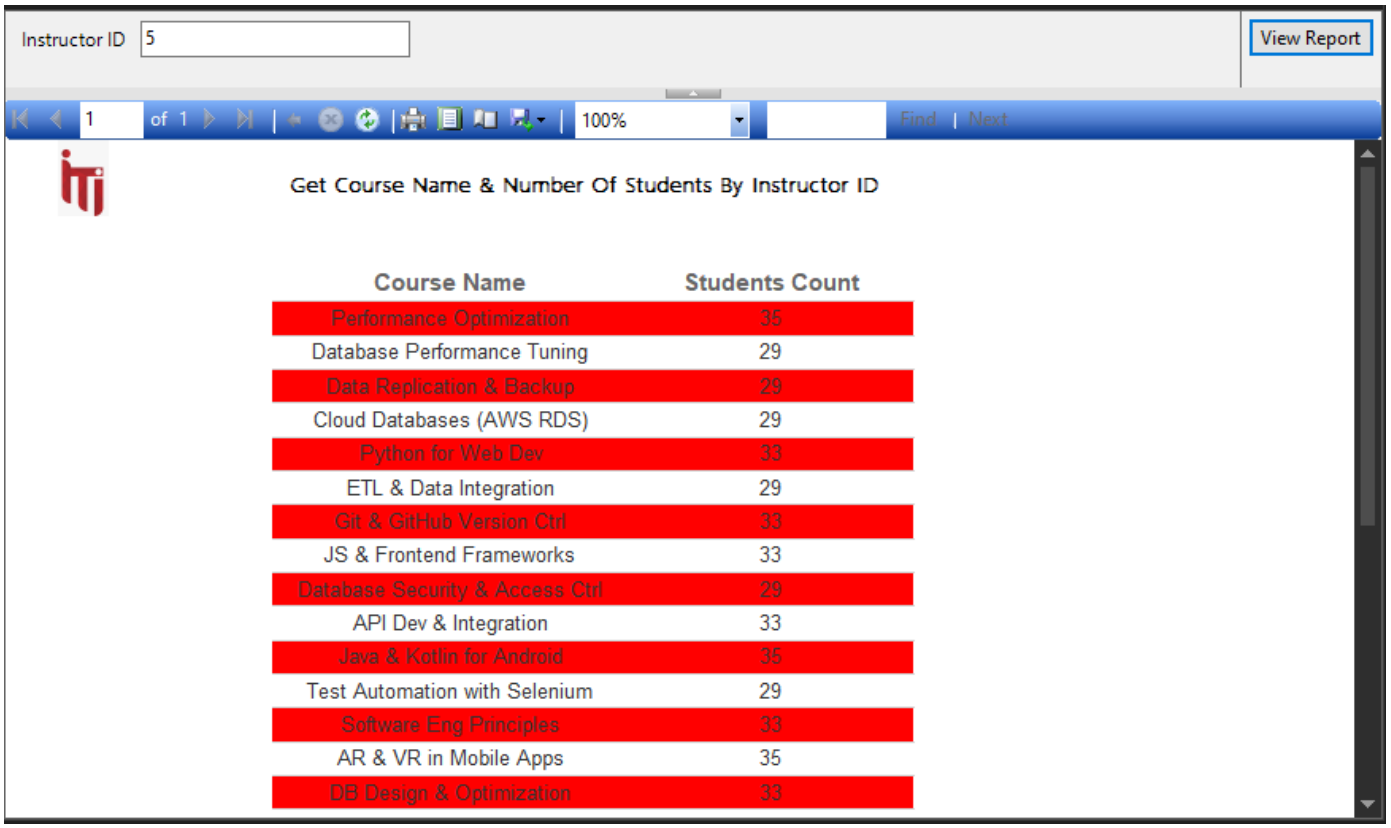


Question	Correct Choice	Student Choice	Answer Status
Which command is used to list the contents of a directory in Linux?	ls		Incorrect
What is the default home directory for a regular Linux user?	/home/username		Incorrect
Which command is used to display the current working directory?	pwd		Incorrect
What does the chmod 755 myscript.sh command do?	Gives full permissions to the owner and read-execute to others		Incorrect
Which of the following commands is used to search for a specific word inside a file?	grep		Incorrect
What is the purpose of the sudo command in Linux?	It allows a user to execute commands as another user (usually root)		Incorrect
Which command is used to view the first 10 lines of a file?	head		Incorrect
What is the function of the man command in Linux?	Displays the manual pages of a command		Incorrect
Which of the following is a package manager for Debian-based distributions?	apt		Incorrect
What does the tar -czf backup.tar.gz /home/user command do?	Creates a compressed archive of the /home/user		Incorrect



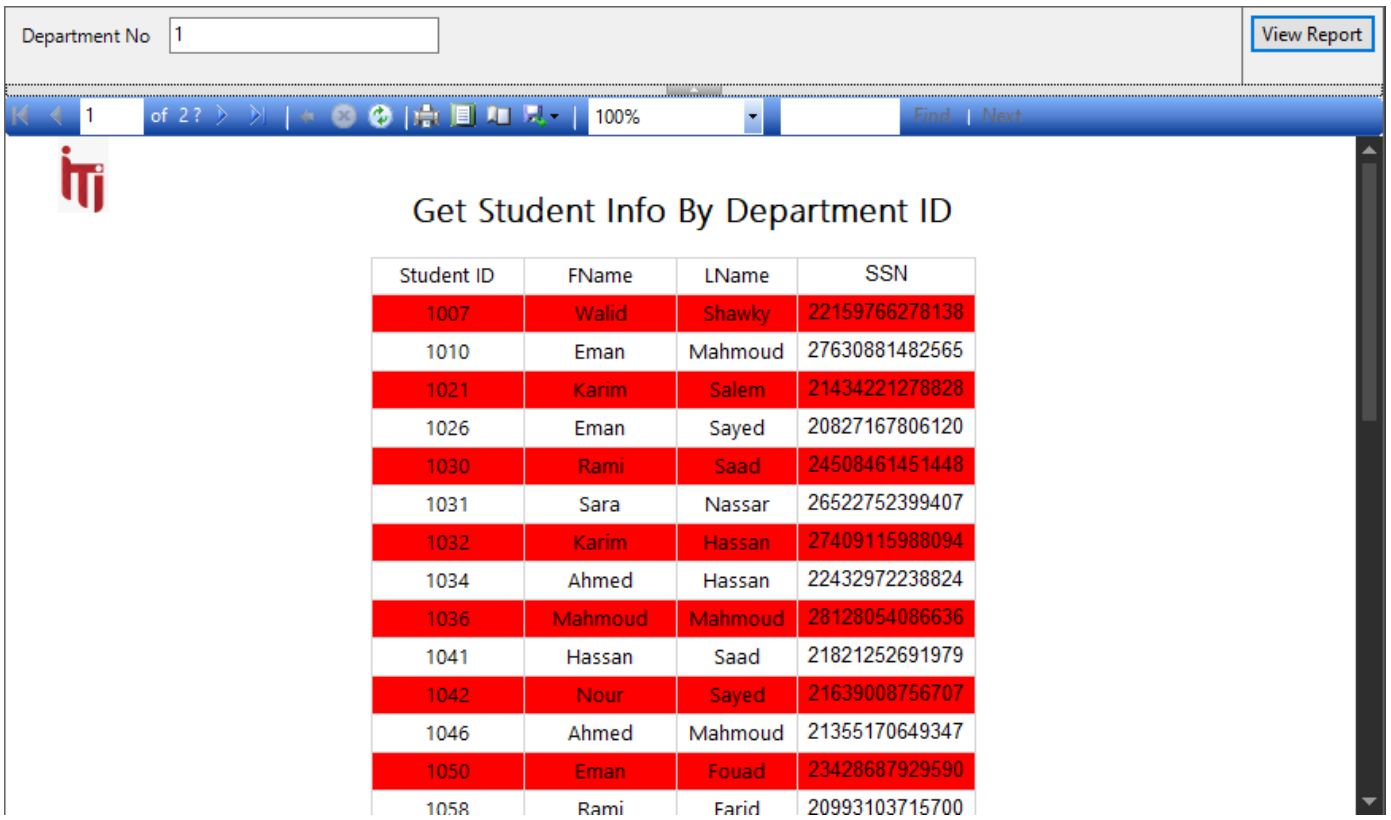
Which of the following is a key characteristic of a data warehouse?	Time-variant	Time-variant	Correct
The process of extracting, transforming, and loading data into a data warehouse is called:	ETL	ETL	Correct
Which data warehouse architecture consists of multiple subject-oriented databases?	Data Mart	Data Mart	Correct
In data warehousing, which schema consists of a large fact table connected to multiple dimension tables?	Star Schema		Incorrect
Which component of a data warehouse is responsible for handling queries and reporting?	OLAP Engine		Incorrect
Which of the following is an advantage of using a data warehouse?	Improved decision-making through historical data analysis	Improved decision-making through historical data analysis	Correct
Which type of OLAP stores pre-aggregated data for faster query performance?	MOLAP		Incorrect
Which term describes the process of combining multiple data sources into a unified view?	Data Integration	Data Integration	Correct

- **Get Course Name and Students Count by Instructor ID**



Course Name	Students Count
Performance Optimization	35
Database Performance Tuning	29
Data Replication & Backup	29
Cloud Databases (AWS RDS)	29
Python for Web Dev	33
ETL & Data Integration	29
Git & GitHub Version Ctrl	33
JS & Frontend Frameworks	33
Database Security & Access Ctrl	29
API Dev & Integration	33
Java & Kotlin for Android	35
Test Automation with Selenium	29
Software Eng Principles	33
AR & VR in Mobile Apps	35
DB Design & Optimization	33

- **Get Student Info By Department ID**



Student ID	FName	LName	SSN
1007	Walid	Shawky	22159766278138
1010	Eman	Mahmoud	27630881482565
1021	Karim	Salem	21434221278828
1026	Eman	Sayed	20827167806120
1030	Rami	Saad	24508461451448
1031	Sara	Nassar	26522752399407
1032	Karim	Hassan	27409115988094
1034	Ahmed	Hassan	22432972238824
1036	Mahmoud	Mahmoud	28128054086636
1041	Hassan	Saad	21821252691979
1042	Nour	Sayed	21639008756707
1046	Ahmed	Mahmoud	21355170649347
1050	Eman	Fouad	23428687929590
1058	Rami	Farid	20993103715700

- **Get Exam Questions and Choices by Exam Number**

Exam Number 1012 View Report

1 of 2 100% Find | Next

What does SQL stand for?

Sequential Query Logic
Simple Query Language
Standard Question Language
Structured Query Language

Which SQL statement is used to retrieve data from a database?

FETCH
GET
RETRIEVE
SELECT

- **Get Student Grade in Courses by Student ID**

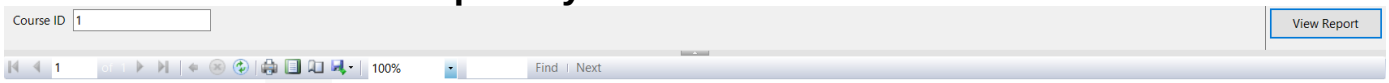
Student ID 1125 View Report

1 of 1 100% Find | Next

Get student grades

Student id	Full Name	Course Name	Grade
1125	Mona Rizq	Data Cleaning & Preprocessing	100%
1125	Mona Rizq	Data Warehousing Basics	70%
1125	Mona Rizq	Data Cleaning & Preprocessing	100%
1125	Mona Rizq	SQL for Data Analysis	100%
1125	Mona Rizq	Data Warehousing Basics	100%
1125	Mona Rizq	Power BI for Business Intel	100%
1125	Mona Rizq	Tableau for Data Visualization	30%
1125	Mona Rizq	Business Analytics with Excel	90%

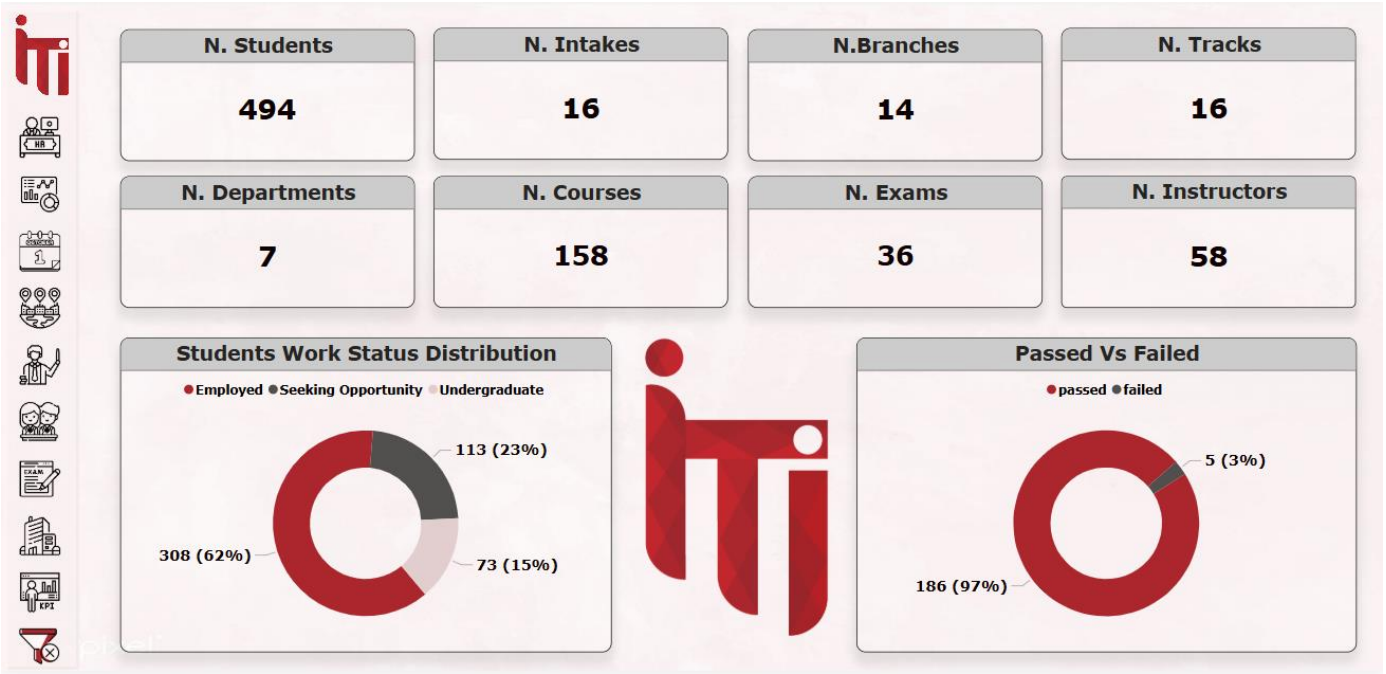
- **Get Course Topics by Course ID**

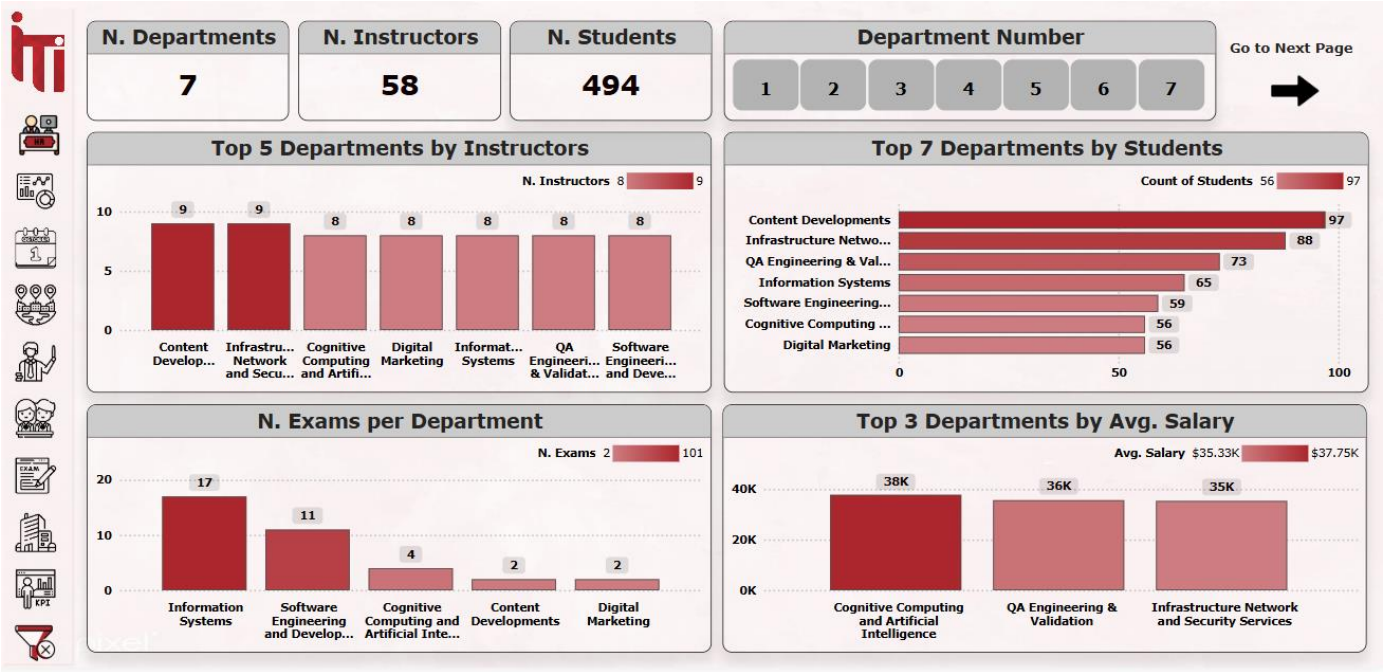


Get Course topics

Topic Name	Topic ID
Linux File System Basics	69
Shell Scripting	70
Process & Memory Management	71
User & Permission Management	72

4.2 Dashboards

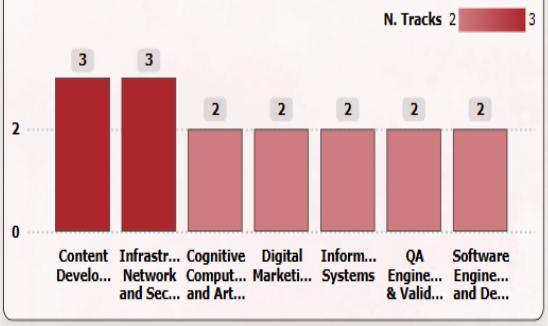




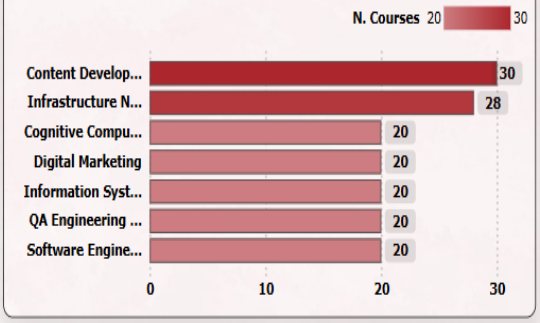
N. Departments	N. Instructors	N. Students
7	58	494

Department Number: 1 2 3 4 5 6 7

N. Tracks per Department



N. Courses per Department



Intake: All

Branch: All

Track: All

Department ID	Department Name	Average of Salary
5	Cognitive Computing and Artificial Intelligence	37,750.00
6	QA Engineering & Validation	35,625.00
2	Infrastructure Network and Security Services	35,333.33
1	Content Developments	33,444.44
4	Software Engineering and Development	32,625.00
3	Information Systems	32,500.00
7	Digital Marketing	30,000.00

Location: All

Course: All

Year: All

N. Students	N. Departments	N. Tracks	Avg. Students Grade
494	7	16	83.7%

Go to Next Page

Department: All

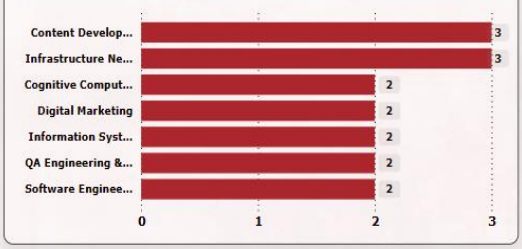
Track: All

Branch: All

Intake type: All

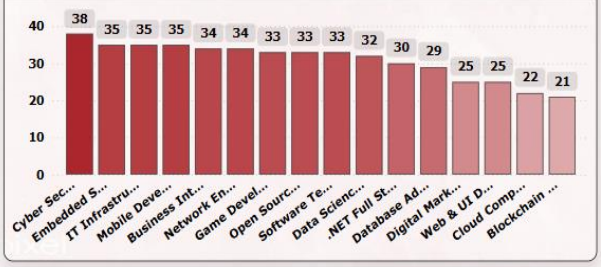
Intake number: All

N.Tracks Per Department

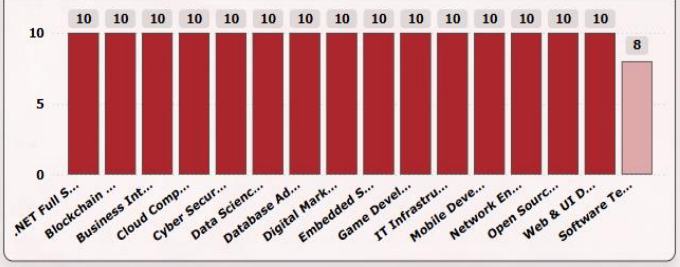


Track Name	Avg.Grade
Open Source Application Development	100.00...
Business Intelligence & Analytics	88.21%
Cloud Computing & DevOps	83.94%
Data Science & AI	79.78%
Blockchain Development	75.00%

N. Students per Track



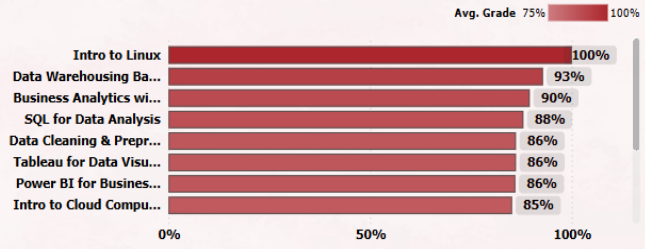
N. Courses per Track



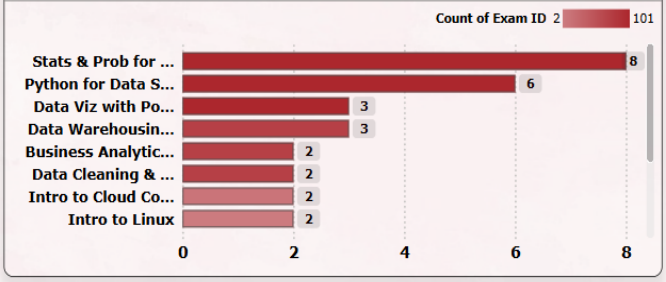
N. Students	N. Courses	Avg. Course Hours	N. Topics
494	158	18	646

Course Name	N. Tpoics	N. Students Take Course
AI in Games	20	33
App Deployment & Monetization	20	35
Branding & Online Reputation	20	25
Cloud Deploy with Azure	20	30
Cloud Infra Services	20	35
Computer Vision & Imaging	20	32
DApps & Blockchain Security	20	21
Data Cleaning & Preprocessing	20	33
Database Security & Access Ctrl	20	29
DB Design & Optimization	20	33
Google Cloud Platform (GCP)	20	22
Java & Kotlin for Android	20	35
Microcontroller Programming	20	35
Penetration Testing	20	38
Performance Testing with JMeter	20	33
Progressive Web Apps (PWA)	20	25
VoIP & Telecommunication	20	34
API Testing with Postman	18	33
AR & VR in Mobile Apps	18	35
Cryptocurrency & Tokenization	18	21

Avg. Grade Per Course



N. Exam per Course



N. Students	N. Intake	Avg. students	Avg. Students Grade
494	18	27	83.7%

Go to Next Page

Department: All

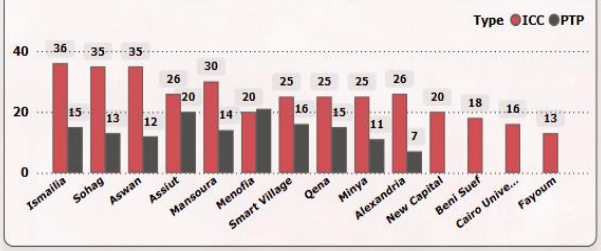
Track: All

Branch: All

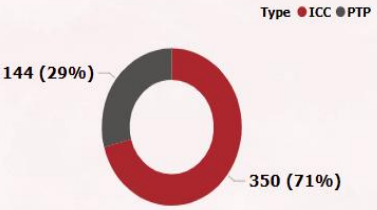
Intake type: All

Intake number: All

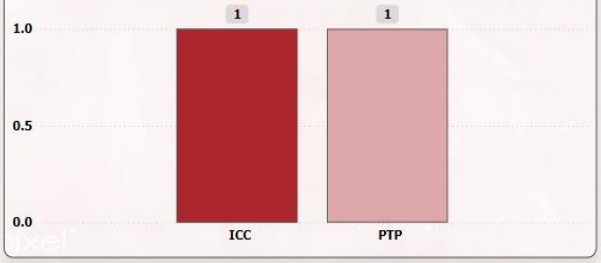
N. Students per Branch According to Intake Type



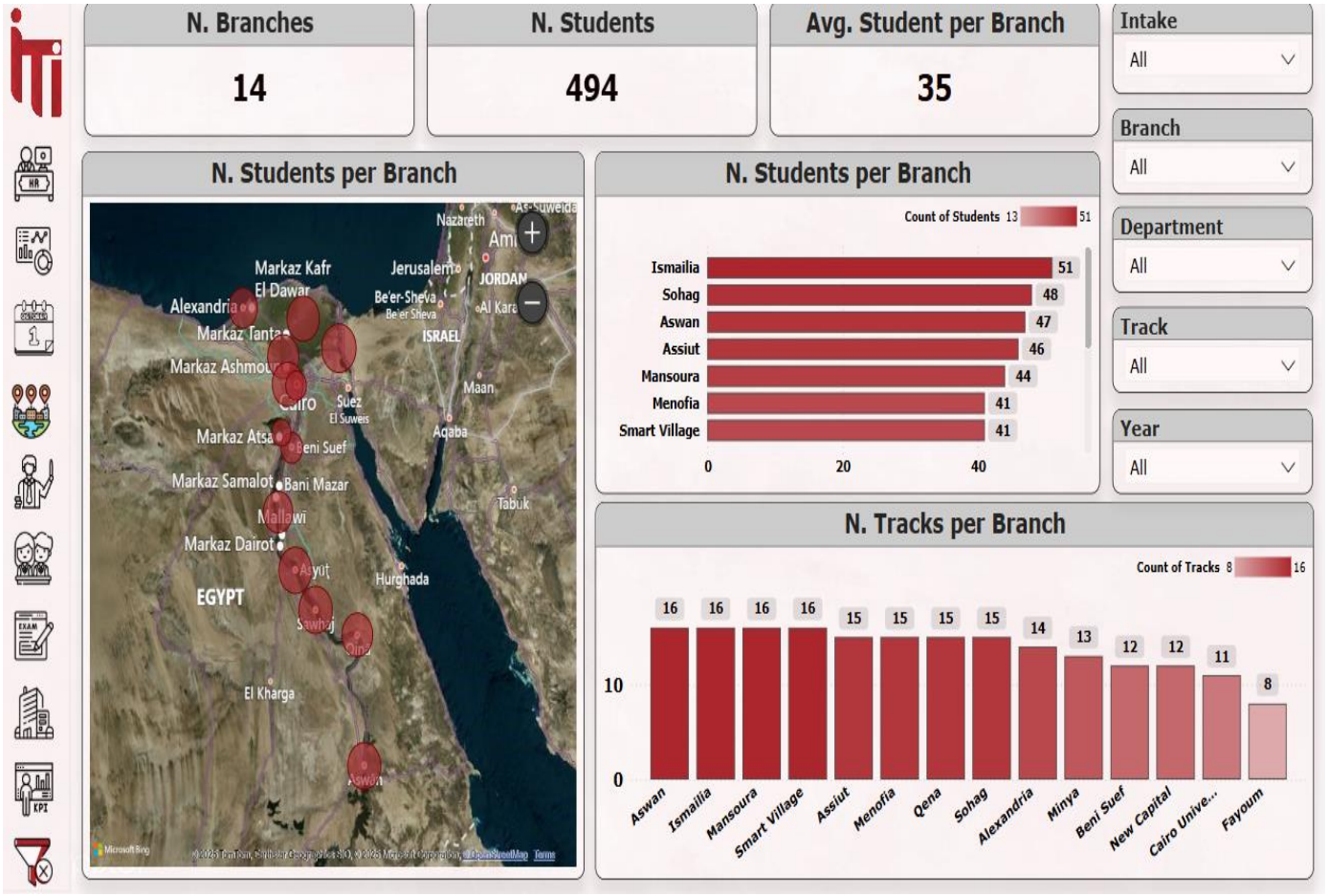
N. Students per Intake Type

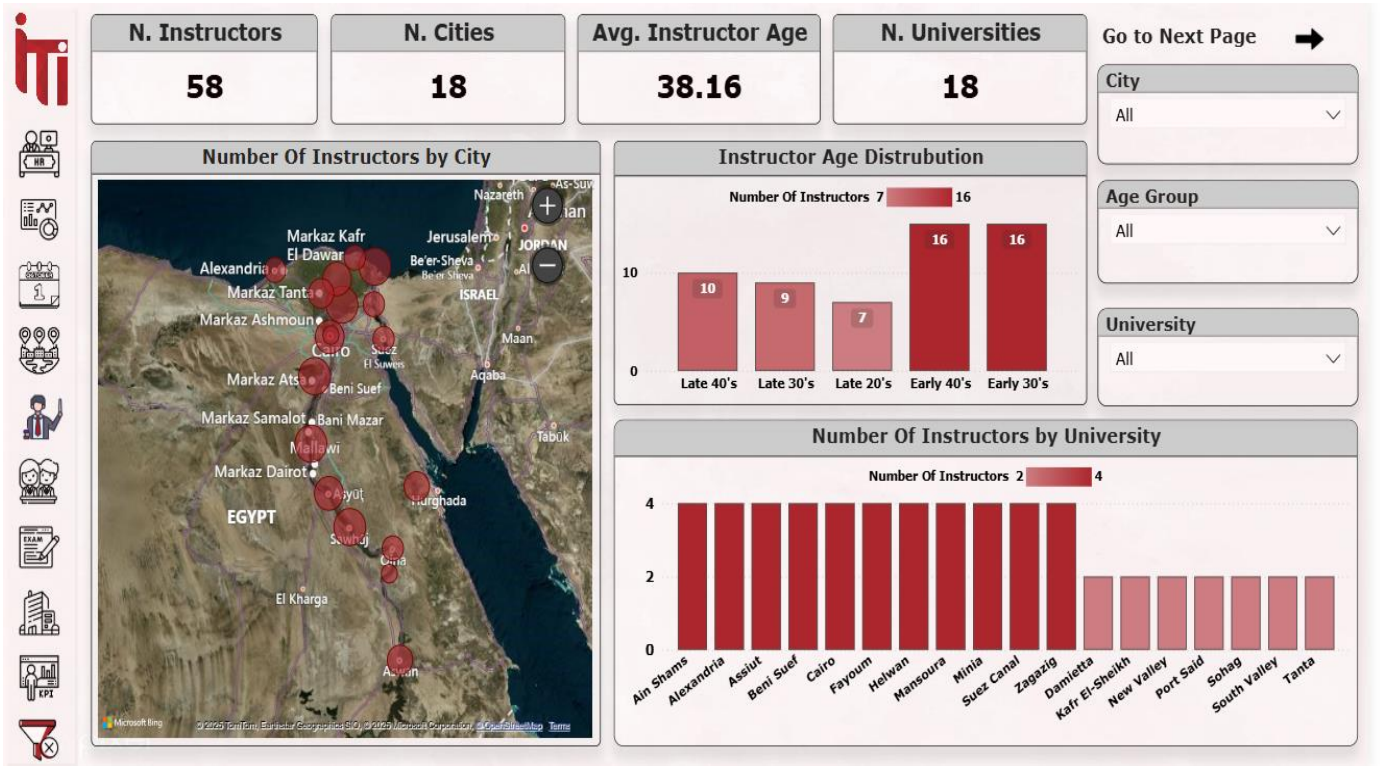


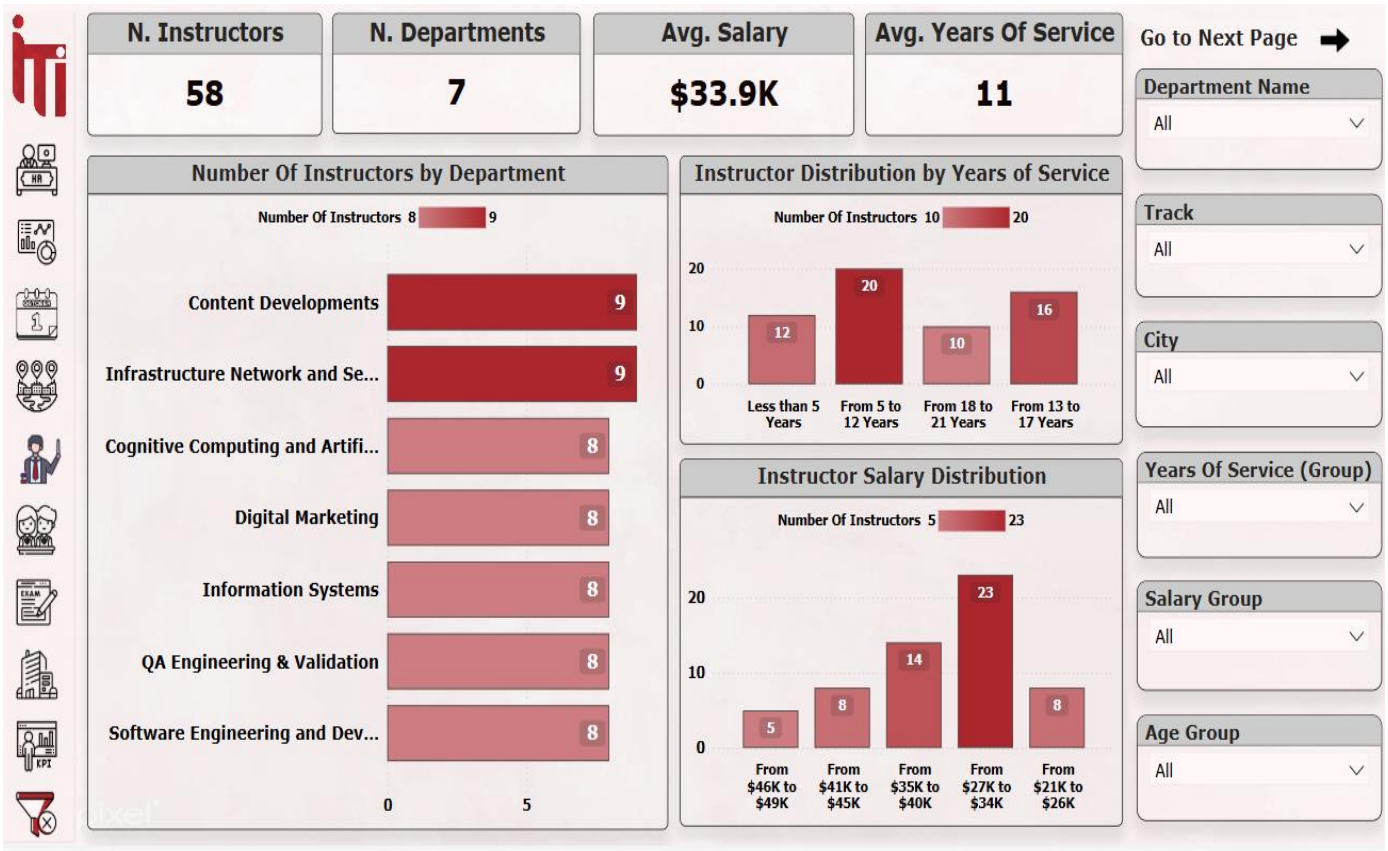
Avg. Success Rate per Intake Type



N. Students per Intake

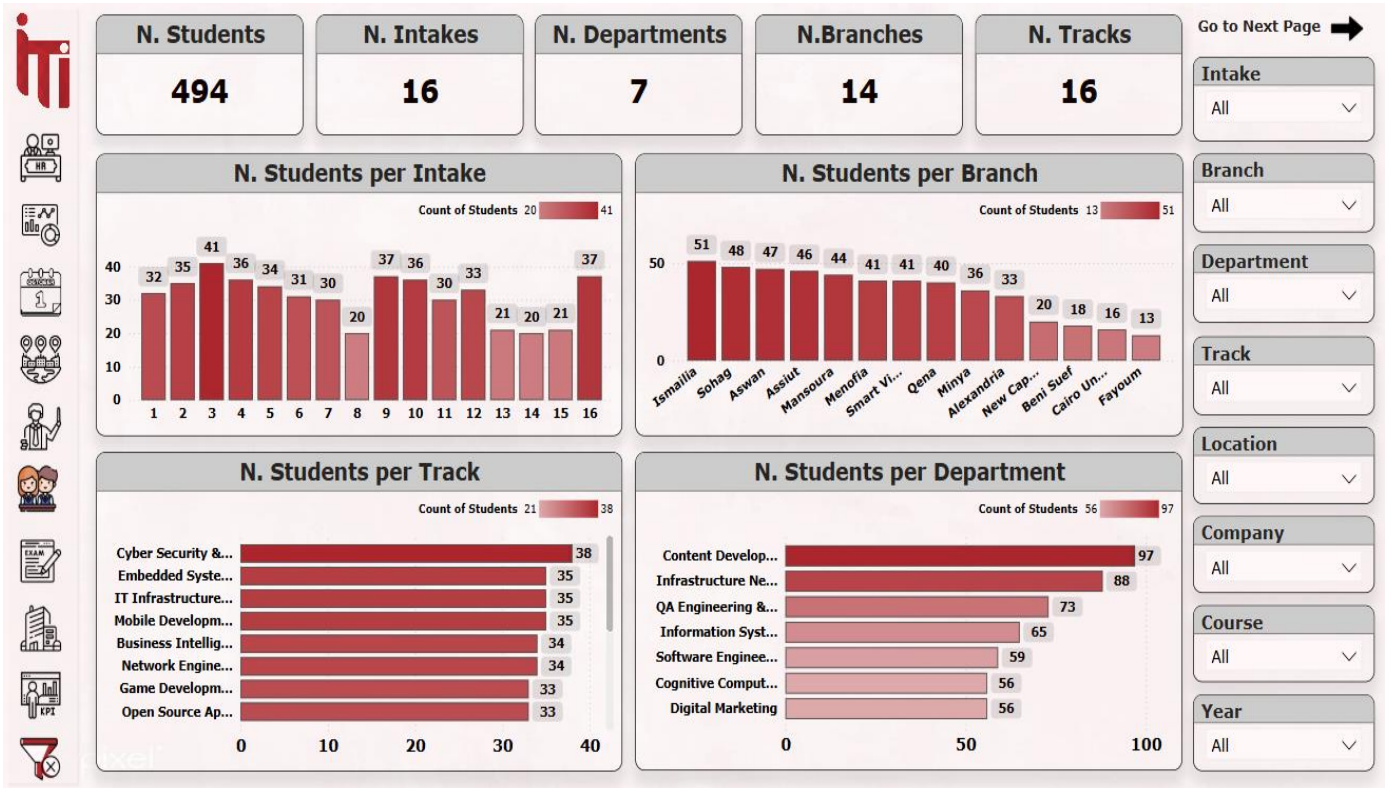

N. Branches
14
N. Students
494
Avg. Student per Branch
35
Intake
All
Branch
All
Department
All
Track
All
Year
All
N. Students per Branch
Markaz Kafr El Dawar
Jerusalem
JORDAN
Alexandria
Be'er-Sheva
ISRAEL
Markaz Tanta
Markaz Ashmoun
Cairo
Suez
Maan
Aqaba
Markaz Atsa
Beni Suef
Markaz Samalot
Bani Mazar
Tabuk
Mallawi
Markaz Dairot
Hurghada
EGYPT
El Kharga
Aswan
N. Students per Branch
Count of Students 13 51
Ismailia 51
Sohag 48
Aswan 47
Assiut 46
Mansoura 44
Menofia 41
Smart Village 41
0 20 40
N. Tracks per Branch
Count of Tracks 8 16
16 16 16 16 15 15 15 15 14 13 12 12 11 8
10 0
Aswan Ismailia Mansoura Smart Village Assiut Menofia Qena Sohag Alexandria Minya Beni Suef New Capital Cairo Unive... Fayoum

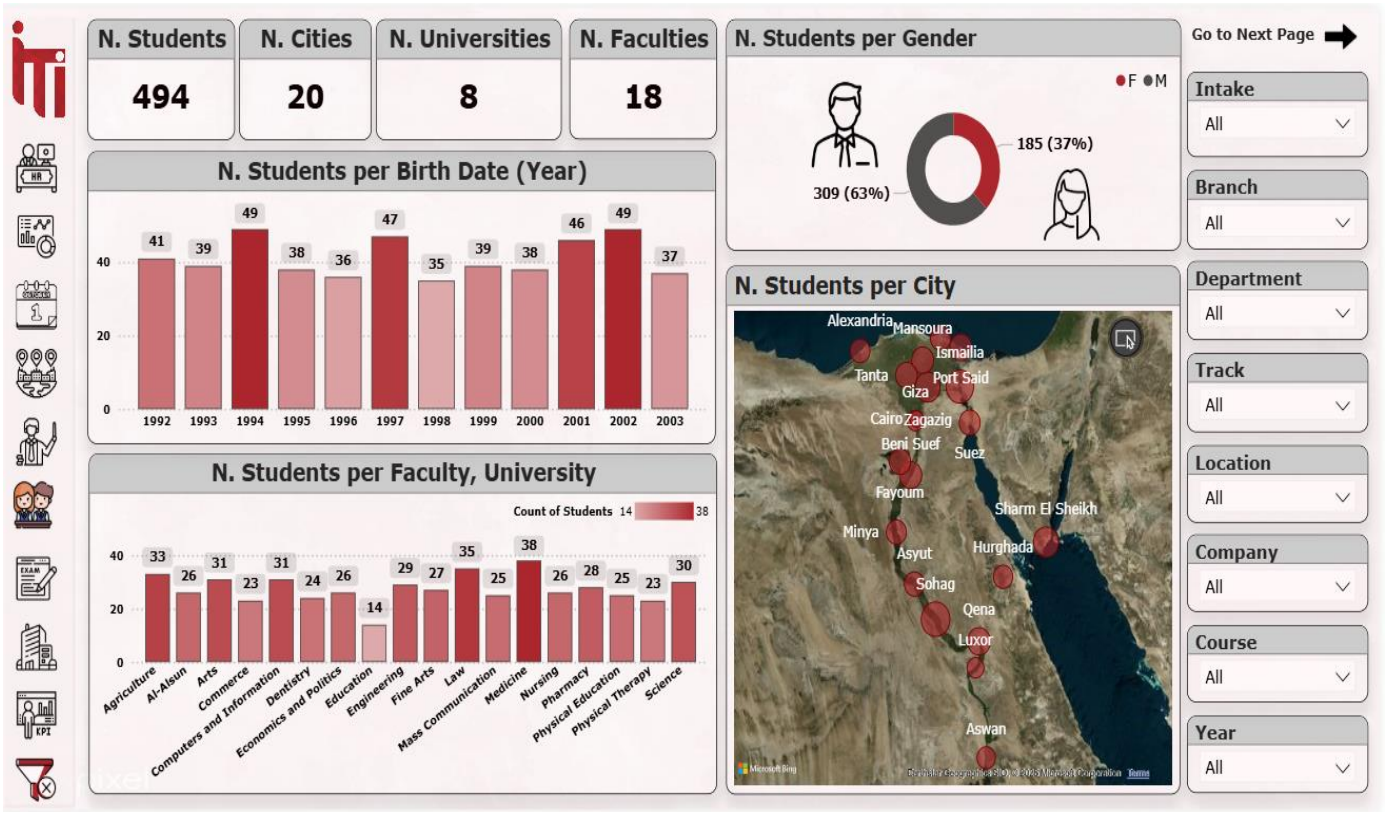

N. Instructors
58
N. Cities
18
Avg. Instructor Age
38.16
N. Universities
18
Go to Next Page
City
All
Age Group
All
University
All
Number Of Instructors by City
Nazareth
Markaz Kafr El Dawar
Jerusalem
JORDAN
Alexandria
Be'er-Sheva
ISRAEL
Markaz Tanta
Markaz Ashmoun
Cairo
Suez
Maan
Aqaba
Markaz Atsa
Beni Suef
Markaz Samalot
Bani Mazar
Tabuk
Mallawi
Markaz Dairot
Hurghada
EGYPT
El Kharga
Aswan
Instructor Age Distrubution
Number Of Instructors 7 16
10 0
10 9 7 16 16
Late 40's Late 30's Late 20's Early 40's Early 30's
Number Of Instructors by University
Number Of Instructors 2 4
4 2 0
Ain Shams Alexandria Assiut Beni Suef Cairo Fayoum Helwan Mansoura Minia Suez Canal Zagazig Damietta Kafr El-Sheikh New Valley Port Said Sohag South Valley Tanta

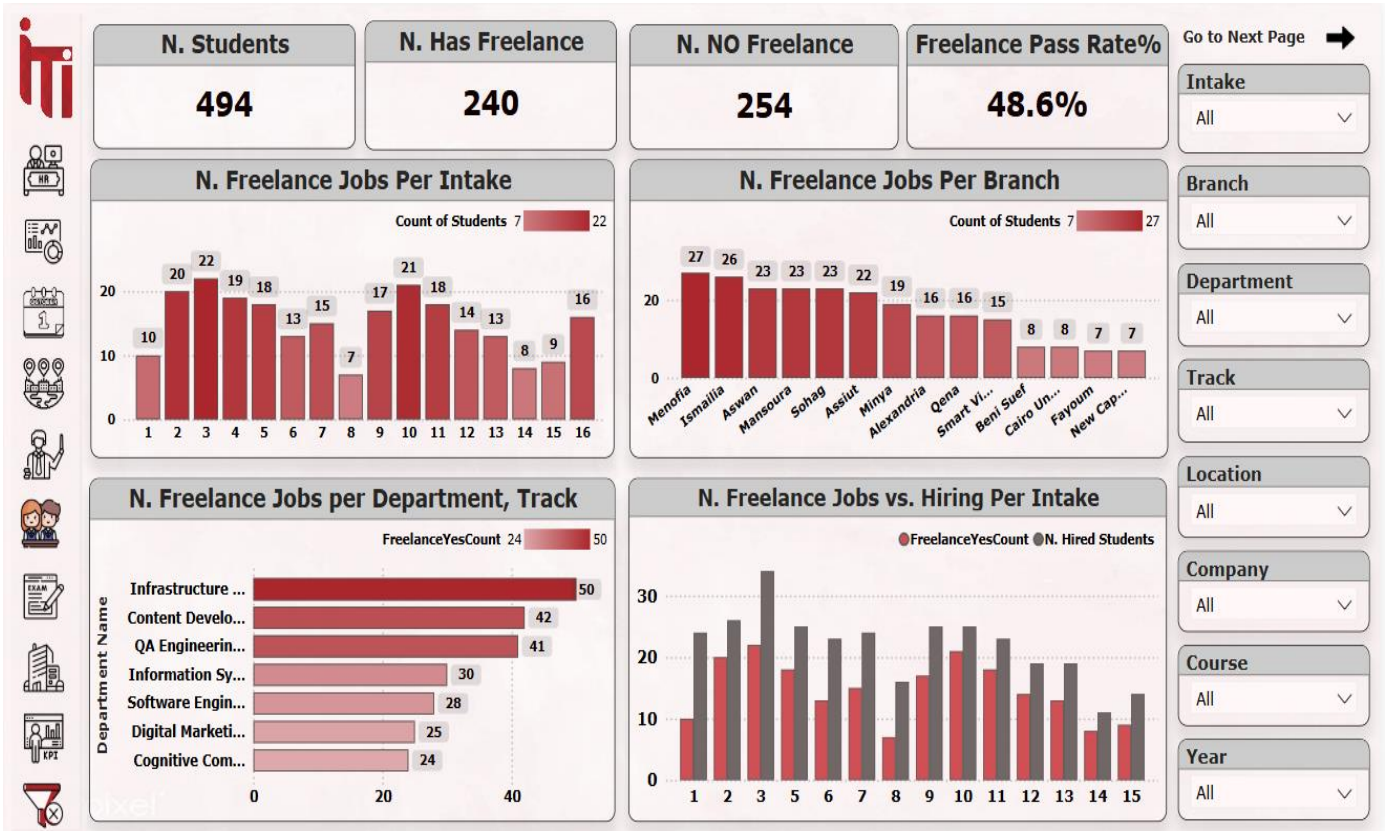


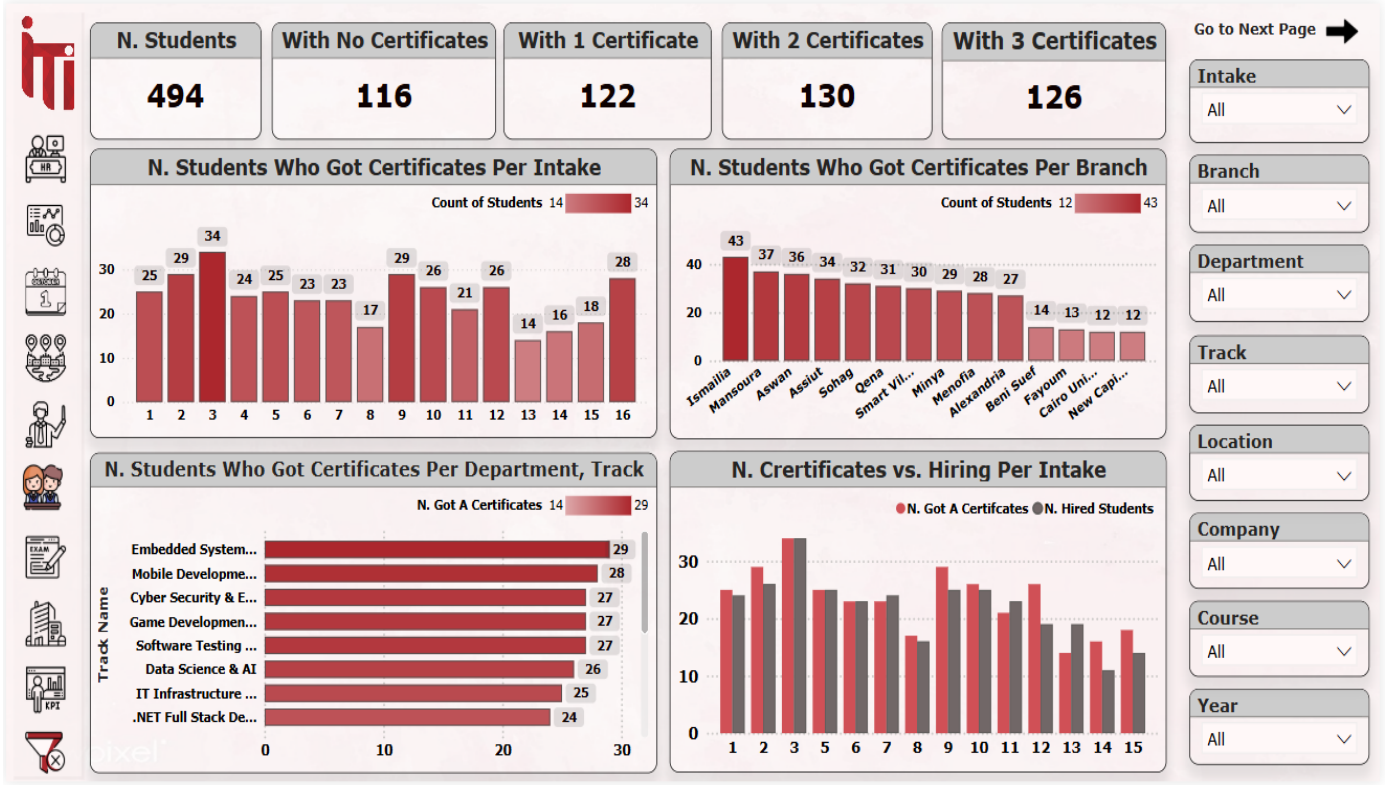
Department Name	Instructor	Years Of Service	Salary
Software Engineering and Development	Yasmine Tamer_57	9	27,000
Software Engineering and Development	Dina Ehab_22	8	29,000
Software Engineering and Development	Khaled Nasser_36	8	29,000
Software Engineering and Development	Farida Khaled_43	13	30,000
Software Engineering and Development	Amr Magdy_50	13	34,000
Software Engineering and Development	Mohamed Adel_29	4	35,000
Software Engineering and Development	Hany Saad_15	6	37,000
Software Engineering and Development	Fatma Ali_8	4	40,000
QA Engineering & Validation	Tamer Abdelrahman_17	19	26,000
QA Engineering & Validation	Mostafa Tamer_31	16	29,000
QA Engineering & Validation	Salma Mohamed_10	17	32,000
QA Engineering & Validation	Mahmoud Hassan_52	2	33,000
QA Engineering & Validation	Reem Khaled_59	10	39,000
QA Engineering & Validation	Sara Khalil_24	14	41,000
QA Engineering & Validation	Nourhan Mohamed_45	2	42,000
QA Engineering & Validation	Omar Elsaid_38	21	43,000
Infrastructure Network and Security Services	Sherif Ali_55	1	23,000
Infrastructure Network and Security Services	Ramy Hassan_34	6	27,000
Infrastructure Network and Security Services	Lina Hisham_41	8	28,000
Infrastructure Network and Security Services	Mona Ibrahim_6	13	34,000
Infrastructure Network and Security Services	Nada Ahmed_20	5	35,000
Infrastructure Network and Security Services	Omar Gamal 27	17	37,000

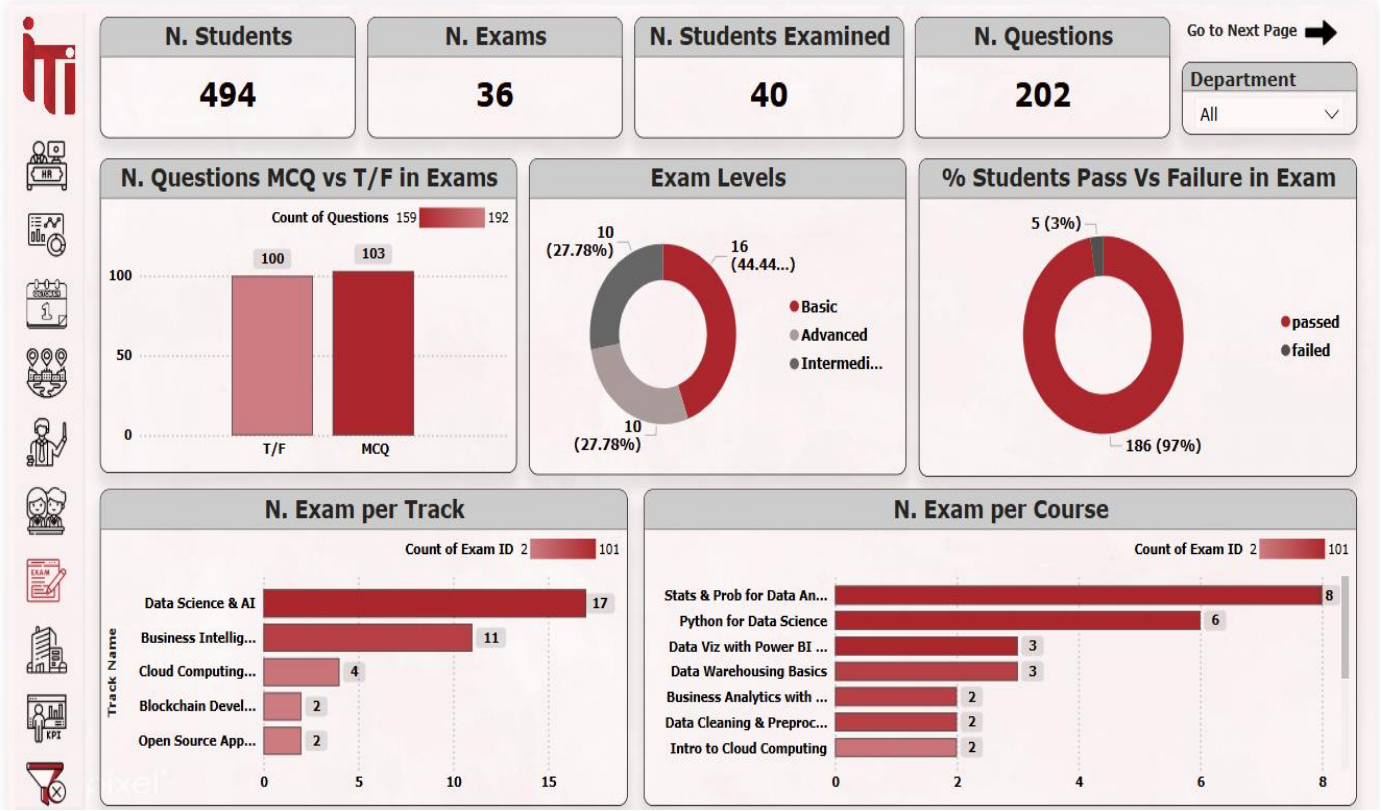
Department Name
All
Track
All
City
All
Years Of Service (Group)
All
Salary Group
All
Age Group
All

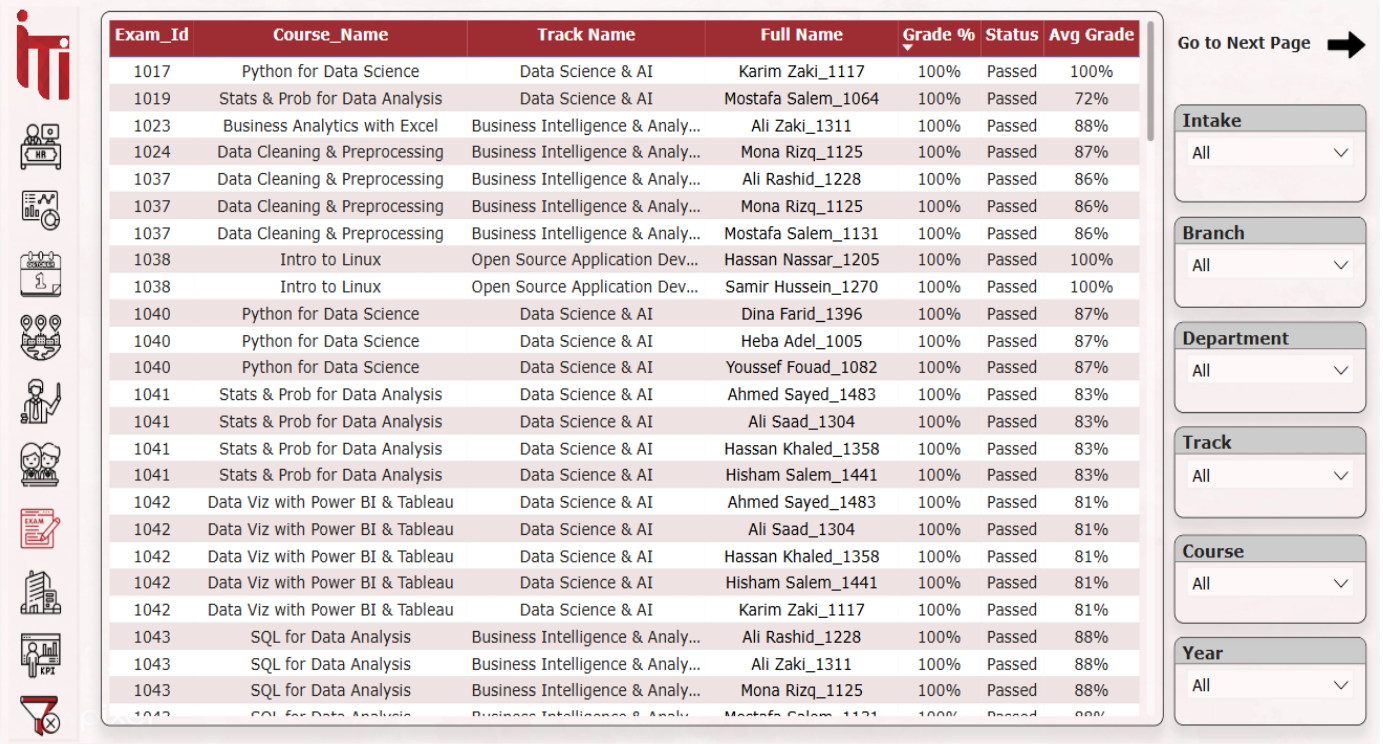

N. Students
494
N. Intakes
16
N. Departments
7
N.Branches
14
N. Tracks
16
Go to Next Page
N. Students per Intake
Count of Students 20 41
32 35 41 36 34 31 30 20 37 36 30 33 21 20 21 37
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
N. Students per Branch
Count of Students 13 51
51 48 47 46 44 41 41 40 36 33 20 18 16 13
Ismailia Sohag Aswan Assiut Mansoura Menofia Smart Vi... Qena Minya Alexandria New Cap... Beni Suef Cairo Un... Fayoum
N. Students per Track
Count of Students 21 38
Cyber Security &... 38
Embedded Syste... 35
IT Infrastructure... 35
Mobile Developm... 35
Business Intellig... 34
Network Engine... 34
Game Developm... 33
Open Source Ap... 33
N. Students per Department
Count of Students 56 97
Content Develop... 97
Infrastructure Ne... 88
QA Engineering &... 73
Information Syst... 65
Software Enginee... 59
Cognitive Comput... 56
Digital Marketing 56
Intake
All
Branch
All
Department
All
Track
All
Location
All
Company
All
Course
All
Year
All


N. Students
494
N. Cities
20
N. Universities
8
N. Faculties
18
N. Students per Gender
F M
309 (63%)
185 (37%)
Go to Next Page
N. Students per Birth Date (Year)
41 39 49 38 36 47 35 39 38 46 49 37
1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003
N. Students per City
Alexandria Mansoura Ismailia Tanta Port Said Giza Cairo Zagazig Beni Suef Suez Fayoum Sharm El Sheikh Minya Asyut Hurghada Sohag Qena Luxor Aswan
N. Students per Faculty, University
Count of Students 14 38
33 26 31 23 31 24 26 14 29 27 35 25 38 26 28 25 23 30
Agriculture Al-Alsun Arts Commerce Computers and Information Dentistry Economics and Politics Education Engineering Fine Arts Law Mass Communication Medicine Nursing Pharmacy Physical Education Physical Therapy Science
Intake
All
Branch
All
Department
All
Track
All
Location
All
Company
All
Course
All
Year
All

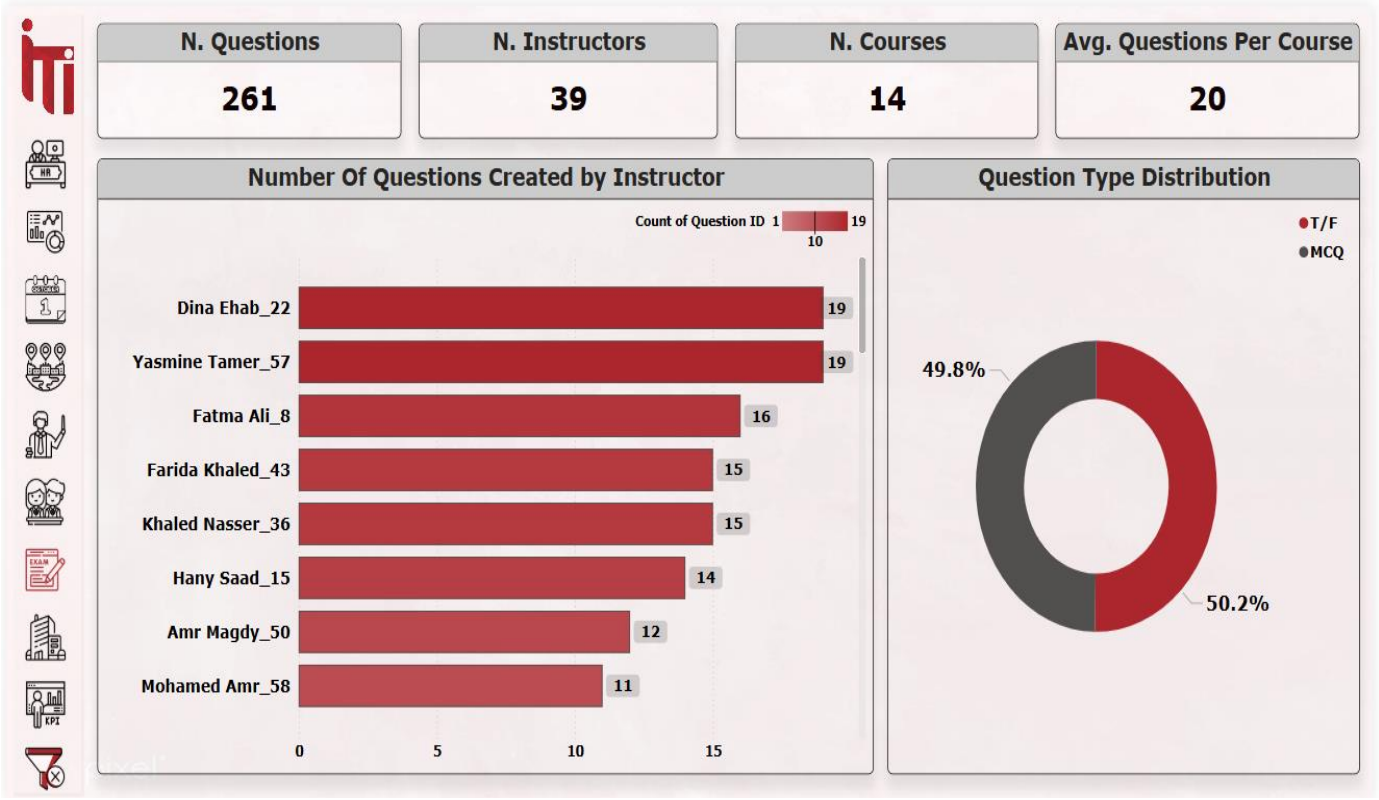

N. Students
494
N. Has Freelance
240
N. NO Freelance
254
Freelance Pass Rate%
48.6%
Go to Next Page
N. Freelance Jobs Per Intake
Count of Students 7 22
N. Freelance Jobs Per Branch
Count of Students 7 27
Menofia
Ismailia
Aswan
Mansoura
Sohag
Assiut
Minya
Alexandria
Qena
Smart Vi...
Beni Suef
Cairo Un...
Fayoum
New Cap...
N. Freelance Jobs per Department, Track
FreelanceYesCount 24 50
Department Name
Infrastructure ...
Content Develo...
QA Engineerin...
Information Sy...
Software Engin...
Digital Marketi...
Cognitive Com...
N. Freelance Jobs vs. Hiring Per Intake
FreelanceYesCount
N. Hired Students
Intake
All
Branch
All
Department
All
Track
All
Location
All
Company
All
Course
All
Year
All

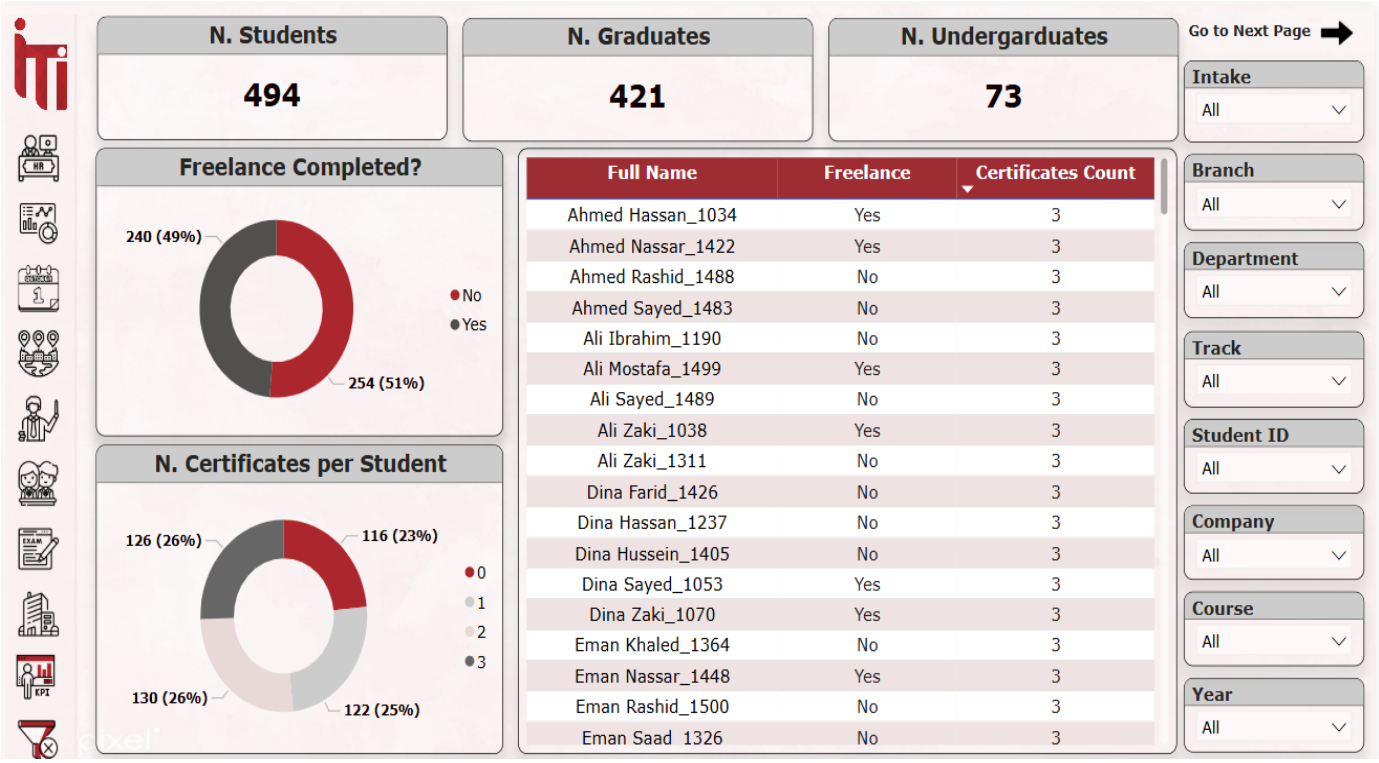

N. Students
494
With No Certificates
116
With 1 Certificate
122
With 2 Certificates
130
With 3 Certificates
126
Go to Next Page
N. Students Who Got Certificates Per Intake
Count of Students 14 34
N. Students Who Got Certificates Per Branch
Count of Students 12 43
Ismailia
Mansoura
Aswan
Assiut
Sohag
Qena
Smart Vil...
Minya
Menofia
Alexandria
Beni Suef
Fayoum
Cairo Uni...
New Capi...
N. Students Who Got Certificates Per Department, Track
N. Got A Certificates 14 29
Track Name
Embedded System...
Mobile Developme...
Cyber Security & E...
Game Developmen...
Software Testing ...
Data Science & AI
IT Infrastructure ...
.NET Full Stack De...
N. Crertificates vs. Hiring Per Intake
N. Got A Certifcates
N. Hired Students
Intake
All
Branch
All
Department
All
Track
All
Location
All
Company
All
Course
All
Year
All





Exam_Id	Course_Name	Track Name	Full Name	Grade %	Status	Avg Grade
1017	Python for Data Science	Data Science & AI	Karim Zaki_1117	100%	Passed	100%
1019	Stats & Prob for Data Analysis	Data Science & AI	Mostafa Salem_1064	100%	Passed	72%
1023	Business Analytics with Excel	Business Intelligence & Analy...	Ali Zaki_1311	100%	Passed	88%
1024	Data Cleaning & Preprocessing	Business Intelligence & Analy...	Mona Rizq_1125	100%	Passed	87%
1037	Data Cleaning & Preprocessing	Business Intelligence & Analy...	Ali Rashid_1228	100%	Passed	86%
1037	Data Cleaning & Preprocessing	Business Intelligence & Analy...	Mona Rizq_1125	100%	Passed	86%
1037	Data Cleaning & Preprocessing	Business Intelligence & Analy...	Mostafa Salem_1131	100%	Passed	86%
1038	Intro to Linux	Open Source Application Dev...	Hassan Nassar_1205	100%	Passed	100%
1038	Intro to Linux	Open Source Application Dev...	Samir Hussein_1270	100%	Passed	100%
1040	Python for Data Science	Data Science & AI	Dina Farid_1396	100%	Passed	87%
1040	Python for Data Science	Data Science & AI	Heba Adel_1005	100%	Passed	87%
1040	Python for Data Science	Data Science & AI	Youssef Fouad_1082	100%	Passed	87%
1041	Stats & Prob for Data Analysis	Data Science & AI	Ahmed Sayed_1483	100%	Passed	83%
1041	Stats & Prob for Data Analysis	Data Science & AI	Ali Saad_1304	100%	Passed	83%
1041	Stats & Prob for Data Analysis	Data Science & AI	Hassan Khaled_1358	100%	Passed	83%
1041	Stats & Prob for Data Analysis	Data Science & AI	Hisham Salem_1441	100%	Passed	83%
1042	Data Viz with Power BI & Tableau	Data Science & AI	Ahmed Sayed_1483	100%	Passed	81%
1042	Data Viz with Power BI & Tableau	Data Science & AI	Ali Saad_1304	100%	Passed	81%
1042	Data Viz with Power BI & Tableau	Data Science & AI	Hassan Khaled_1358	100%	Passed	81%
1042	Data Viz with Power BI & Tableau	Data Science & AI	Hisham Salem_1441	100%	Passed	81%
1042	Data Viz with Power BI & Tableau	Data Science & AI	Karim Zaki_1117	100%	Passed	81%
1043	SQL for Data Analysis	Business Intelligence & Analy...	Ali Rashid_1228	100%	Passed	88%
1043	SQL for Data Analysis	Business Intelligence & Analy...	Ali Zaki_1311	100%	Passed	88%
1043	SQL for Data Analysis	Business Intelligence & Analy...	Mona Rizq_1125	100%	Passed	88%


N. Questions
261
N. Instructors
39
N. Courses
14
Avg. Questions Per Course
20
Number Of Questions Created by Instructor
Count of Question ID 1
19
10
Dina Ehab_22
19
Yasmine Tamer_57
19
Fatma Ali_8
16
Farida Khaled_43
15
Khaled Nasser_36
15
Hany Saad_15
14
Amr Magdy_50
12
Mohamed Amr_58
11
0
5
10
15
Question Type Distribution
T/F
MCQ
49.8%
50.2%

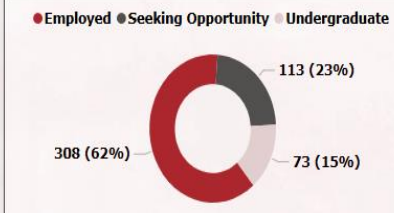

N. Students
494
N. Graduates
421
N. Undergarduates
73
Go to Next Page
Freelance Completed?
240 (49%)
254 (51%)
No
Yes
N. Certificates per Student
126 (26%)
116 (23%)
130 (26%)
122 (25%)
0
1
2
3
Full Name
Freelance
Certificates Count
Ahmed Hassan_1034 Yes 3
Ahmed Nassar_1422 Yes 3
Ahmed Rashid_1488 No 3
Ahmed Sayed_1483 No 3
Ali Ibrahim_1190 No 3
Ali Mostafa_1499 Yes 3
Ali Sayed_1489 No 3
Ali Zaki_1038 Yes 3
Ali Zaki_1311 No 3
Dina Farid_1426 No 3
Dina Hassan_1237 No 3
Dina Hussein_1405 No 3
Dina Sayed_1053 Yes 3
Dina Zaki_1070 Yes 3
Eman Khaled_1364 No 3
Eman Nassar_1448 Yes 3
Eman Rashid_1500 No 3
Eman Saad 1326 No 3
Intake
All
Branch
All
Department
All
Track
All
Student ID
All
Company
All
Course
All
Year
All

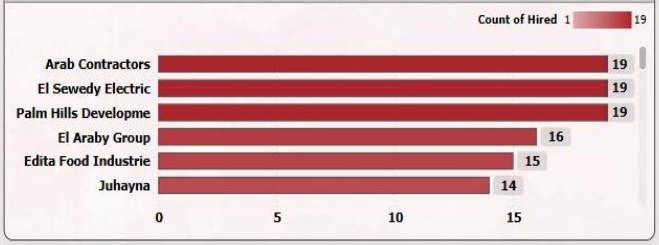
Intake: All | Department: All | Track: All | City: All | Student Name: All | Graduation Year: All

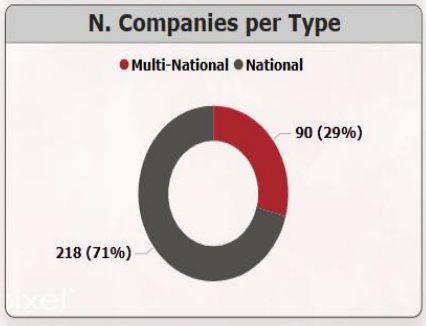
Full Name	Graduation Year	Location	Intake	Intake Type	Branch Name	Track Name	N. Exam	Avg.Grade	Freelance	Certificate Count	Facebook URL
Mona Rizq_1125	2017	Mansoura	16	3 Months	Mansoura	Business Intelligence & Analytics	8	86.25%	Yes	3	
Mostafa Salem_...	2018	Cairo	12	3 Months	New Capital	Business Intelligence & Analytics	8	95.00%	No	2	
Ali Rashid_1228	2020	Fayoum	12	3 Months	Fayoum	Business Intelligence & Analytics	7	95.71%	Yes	2	
Ali Zaki_1311	2018	Cairo	12	3 Months	Cairo Univ...	Business Intelligence & Analytics	7	92.65%	No	3	
Dina Sayed_1053	2021	Cairo	10	3 Months	New Capital	Business Intelligence & Analytics	7	85.71%	Yes	3	
Heba Gamal_11...	2018	Sohag	5	3 Months	Sohag	Business Intelligence & Analytics	7	85.71%	No	0	
Karim Gamal_1...	2016	Assiut	2	9 Months	Assiut	Business Intelligence & Analytics	7	85.71%	Yes	3	
Mai Salem_1163	2021	Qena	3	9 Months	Qena	Business Intelligence & Analytics	7	86.76%	Yes	1	
Mostafa Khaled...	2017	Beni Suef	8	3 Months	Beni Suef	Business Intelligence & Analytics	7	81.43%	No	1	
Walid Shawky_...	2016	Aswan	14	3 Months	Aswan	Business Intelligence & Analytics	7	87.14%	Yes	0	
Ali Saad_1304	2019	Sohag	2	9 Months	Sohag	Data Science & AI	6	84.75%	No	2	
Ahmed Sayed_1...	2021	Qena	3	9 Months	Qena	Data Science & AI	5	81.63%	No	3	
Hassan Khaled_...	2015	Sohag	2	9 Months	Sohag	Data Science & AI	5	80.00%	Yes	2	
Hisham Salem_...	2018	Minya	7	3 Months	Minya	Data Science & AI	5	81.63%	No	2	
Karim Zaki_1117	2018	Sohag	16	3 Months	Sohag	Data Science & AI	5	87.76%	Yes	3	
Nour Mostafa_1...	2016	Aswan	6	3 Months	Aswan	Data Science & AI	5	81.63%	No	0	
Nour Saad_1057	2015	Menofia	4	9 Months	Menofia	Data Science & AI	5	68.75%	Yes	1	
Youssef Fouad_...	2016	Cairo	1	9 Months	Smart Vill...	Data Science & AI	5	81.63%	No	3	
Dina Farid_1396	2016	Fayoum	10	3 Months	Fayoum	Data Science & AI	4	87.50%	Yes	2	
Heba Adel_1005	2018	Mansoura	5	3 Months	Mansoura	Data Science & AI	4	80.00%	Yes	3	
Mahmoud Adel_...	2016	Qena	2	9 Months	Qena	Data Science & AI	4	80.00%	No	1	
Mahmoud Ibrahi...	2016	Cairo	11	3 Months	New Capital	Data Science & AI	4	77.50%	No	3	

N. Students	N. Companies	N. Employed	Employement %
494	39	308	73.2%

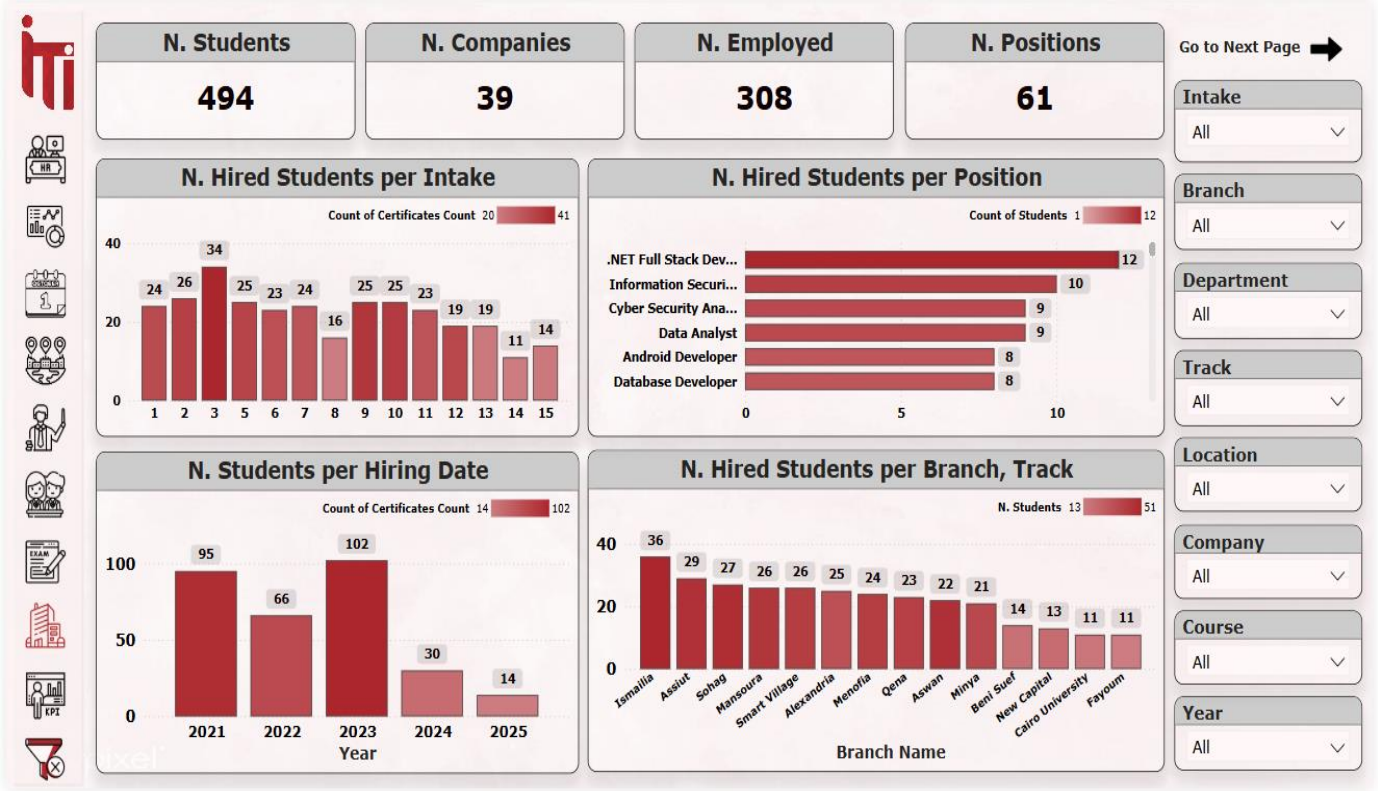
Go to Next Page

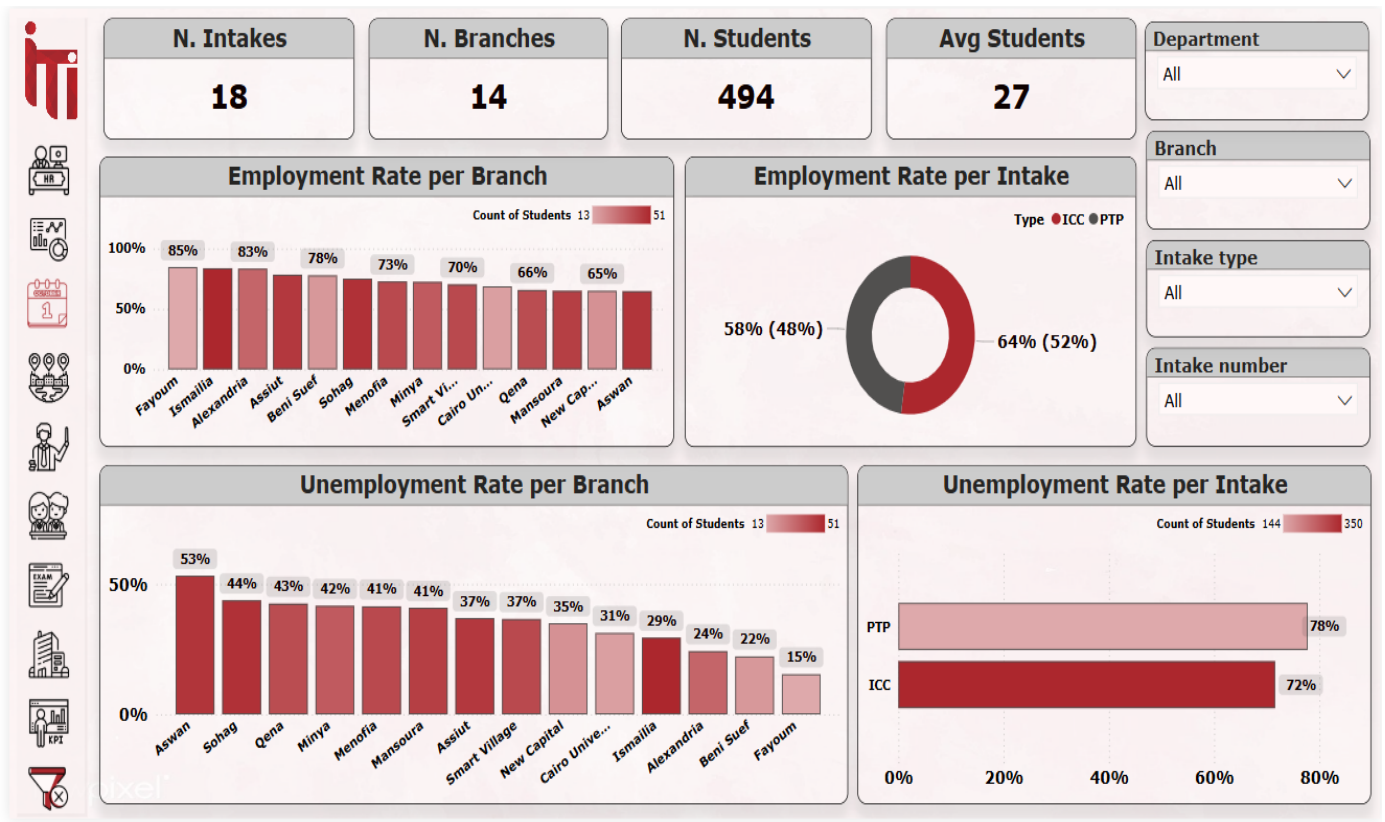






Intake: All | Branch: All | Department: All | Track: All | Location: All | Company: All | Course: All | Year: All


N. Students
494
N. Companies
39
N. Employed
308
N. Positions
61
Go to Next Page
N. Hired Students per Intake
Count of Certificates Count 20 41
N. Hired Students per Position
Count of Students 1 12
.NET Full Stack Dev... 12
Information Securi... 10
Cyber Security Ana... 9
Data Analyst 9
Android Developer 8
Database Developer 8
N. Students per Hiring Date
Count of Certificates Count 14 102
2021 2022 2023 2024 2025
Year
N. Hired Students per Branch, Track
N. Students 13 51
Ismailia Assiut Sohag Mansoura Smart Village Alexandria Menofia Qena Aswan Minya Beni Suef New Capital Cairo University Fayoum
Branch Name
Intake
All
Branch
All
Department
All
Track
All
Location
All
Company
All
Course
All
Year
All


N. Intakes
18
N. Branches
14
N. Students
494
Avg Students
27
Employment Rate per Branch
Count of Students 13 51
Fayoum Ismailia Alexandria Assiut Beni Suef Sohag Menofia Minya Smart Vi... Cairo Un... Qena Mansoura New Cap... Aswan
Employment Rate per Intake
Type ICC PTP
58% (48%)
64% (52%)
Unemployment Rate per Branch
Count of Students 13 51
Aswan Sohag Qena Minya Menofia Mansoura Assiut Smart Village New Capital Cairo Unive... Ismailia Alexandria Beni Suef Fayoum
Unemployment Rate per Intake
Count of Students 144 350
PTP 78%
ICC 72%
Department
All
Branch
All
Intake type
All
Intake number
All

5 Conclusion

The ITI Examination System Database was designed to efficiently manage and analyze student records, courses, exams, instructors, and employment tracking. By implementing a structured relational model, the system ensures data consistency, accuracy, and integrity, making it suitable for educational institutions to track student performance and institutional efficiency.

To simulate real-world scenarios, the data used in this project was generated, allowing for comprehensive testing of database functionalities, relationships, and analytical capabilities. The system successfully supports:

Student Enrollment & Tracking – Managing student progress from enrollment to employment.

Course & Exam Management – Organizing academic data efficiently.

Performance Analysis – Enabling insights through structured data storage.

This project demonstrates how a well-designed relational database can improve data management and decision-making in an educational institution. Future enhancements may include real-time data updates, advanced reporting features, and integration with external academic platforms for deeper analysis.