Information Technology Institute

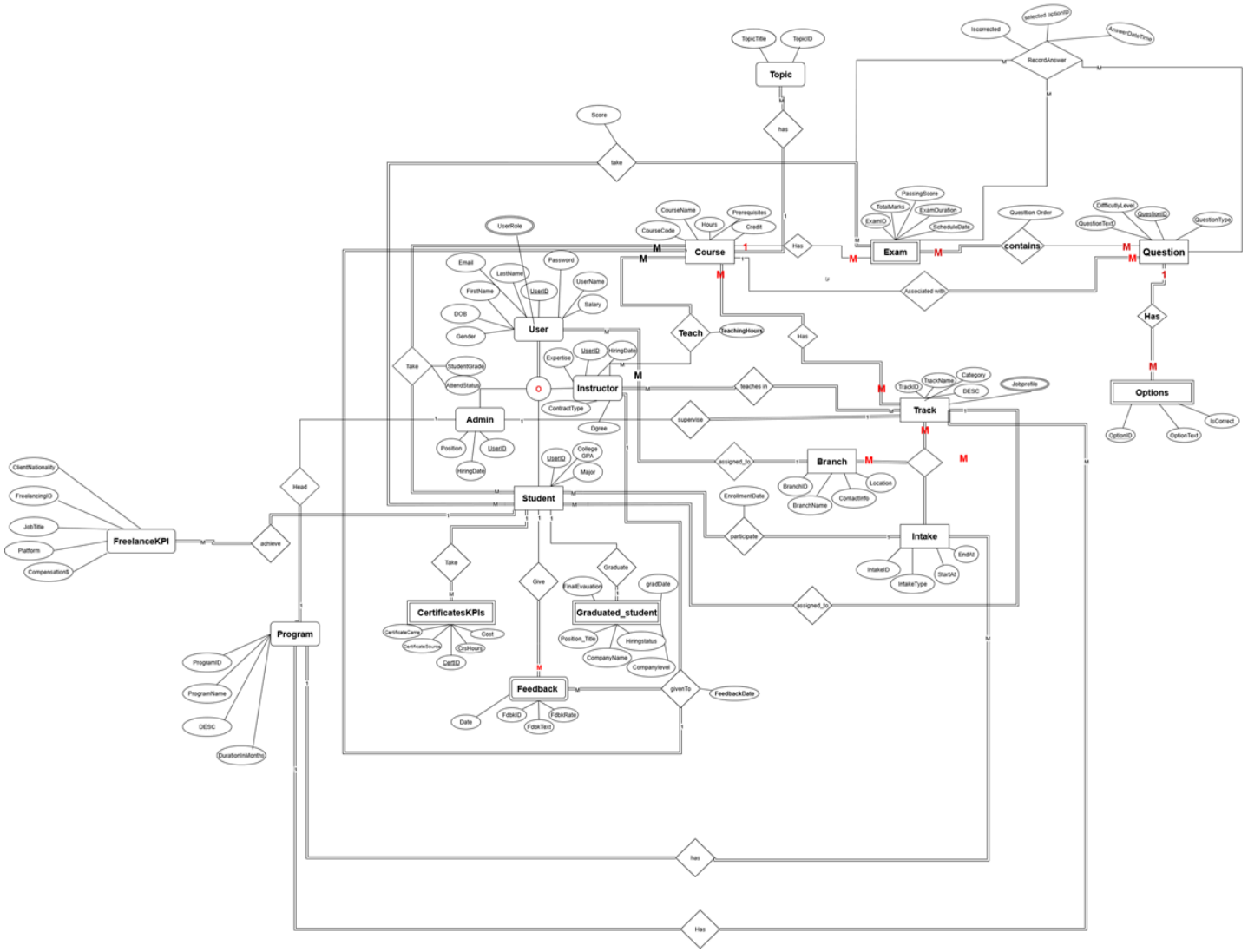
Intensive Code Camps - (ICC)
POWERBI DEVELOPER

Examination System

Introduction:

Entity-Relationship Diagram (ERD) Overview

The ERD visually represents the logical structure of the LMS database, illustrating the entities (tables), their attributes (columns), and the relationships between them. It serves as a blueprint for the database, ensuring data integrity and efficient data management.



Key Entities Include:

- **Core Administrative:** Branch, Users, Admin, Program, Intake, Track.
- **Educational Content:** Course, Topic, Question, Options, Exam.
- **User-Specific:** Student, Instructor, GraduatedStudent, CertificateKPIs, FreelanceJob, Feedback.
- **Job & Career:** JobProfile.

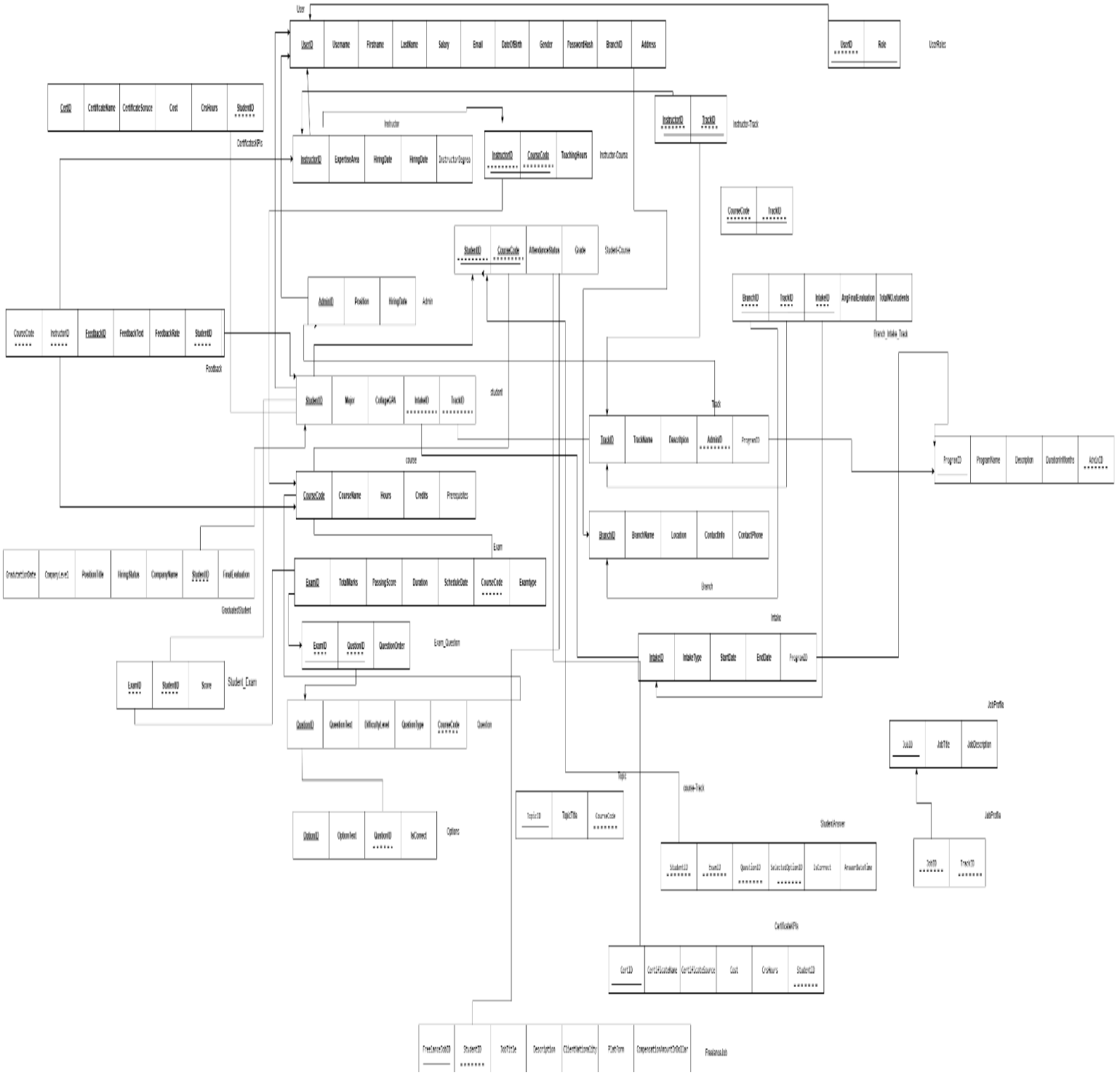
ERD to SQL Mapping

This section illustrates the direct translation of the conceptual ERD components into the relational database schema.

(A screenshot of the ERD to SQL mapping will be inserted here.)

Mapping Principles:

- **Entities to Tables:** Each entity in the ERD is mapped directly to a table in the SQL schema.
- **Attributes to Columns:** Attributes of entities become columns within their respective tables. Primary keys are explicitly defined.
- **Relationships to Foreign Keys / Junction Tables:**
  - **One-to-Many (1:M) relationships:** Typically implemented by adding a foreign key in the table on the "many" side, referencing the primary key of the table on the "one" side.
  - **Many-to-Many (M:M) relationships:** Resolved by creating a new junction (or associative) table. This junction table contains the primary keys of the participating entities as a composite primary key and foreign keys.
  - **Generalization/Specialization:** Handled by having the subtype's primary key also serve as a foreign key to the supertype's primary key.
- **Weak Entities:** Identified by a partial key and a foreign key to their identifying owner. In some cases, for practical implementation, they may receive an IDENTITY primary key while retaining the foreign key relationship.
- **Constraints:** ERD constraints (e.g., uniqueness, not null) are translated into SQL UNIQUE, NOT NULL, and CHECK constraints to enforce data integrity.


User
UserID
Username
Firstname
LastName
Salary
Email
DateOfBirth
Gender
PasswordHash
BranchID
Address
UserRoles
Role
Instructor
InstructorID
ExpertiseArea
HiringDate
InstructorTrack
TrackID
InstructorCourse
CourseCode
TeachingHours
StudentCourse
AttendanceStatus
Grade
Admin
AdminID
Position
Feedback
FeedbackID
FeedbackText
FeedbackRate
student
StudentID
Major
CollegeGPA
IntakeID
Track
TrackName
Description
ProgramID
course
CourseName
Hours
Credits
Prerequisites
Exam
ExamID
TotalMarks
PassingScore
Duration
ScheduleDate
ExamType
GraduatedStudent
Exam_Question
QuestionID
QuestionOrder
Student_Exam
Question
QuestionText
DifficultyLevel
QuestionType
Options
OptionText
IsCorrect
Topic
TopicID
TopicTitle
course-Track
StudentAnswer
SelectedOptionID
Answer
Intake
IntakeType
StartDate
EndDate
Branch
BranchName
Location
ContactInfo
ContactPhone
JobProfile
JobID
JobTitle
JobDescription
CertificatesPFs
CertificateName
CertificateSource
Cost
CreHours
FreelanceJob

Database Schema Implementation

The SQL database schema is designed to support the functionalities of the LMS, adhering closely to the ERD. It defines tables, primary keys, foreign keys, unique constraints, and check constraints to maintain data consistency and relationships.

Key Design Highlights:

- **Generalization/Specialization:** The Users table acts as a supertype, with Admin, Instructor, and Student as subtypes. This is effectively managed through foreign keys where the subtype's primary key is also a foreign key referencing Users.UserID.
- **Role-Based Access:** The UserRoles table allows for flexible assignment of multiple roles ('Admin', 'Instructor', 'Student') to a single user.
- **Associative Entities:** Many-to-many relationships are resolved using associative tables, which also capture attributes specific to the relationship:
    - Branch_Intake_Track
    - Course_Track
    - Instructor_Course (includes TeachingHours)
    - Instructor_Track
    - Student_Course (includes AttendanceStatus, Grade)
    - Exam_Question (includes QuestionOrder)
    - Student_Answer (links Student, Exam, Question, and Options, capturing IsCorrect and AnswerDateTime).
- **Weak Entities:** Entities like Exam, Question, Options, CertificateKPIs, Feedback, and FreelanceJob are dependent on other entities for their full identification. In the SQL, Feedback and FreelanceJob are implemented with their own IDENTITY primary keys for practical flexibility, while maintaining foreign key relationships to their parent entities.
- **Self-Referencing Relationship:** The Course table includes a Prerequisites foreign key, allowing courses to reference other courses as prerequisites.

- **Data Integrity:** Extensive use of NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK constraints ensures data validity and referential integrity across the database.

Data Warehouse Design:

A Data Warehouse is a centralized repository that stores large volumes of structured data from multiple sources, optimized for querying and analysis rather than transaction processing.

A Data Warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data that supports decision-making processes.

Data Warehouse Schema Types:

1. Star Schema
    - Structure:
        - Central Fact Table with keys to multiple Dimension Tables
        - Dimensions are denormalized (flattened structure)
    - Pros:
        - Simple and fast for querying
        - Good performance in OLAP systems
    - Cons:
        - Data redundancy in dimension tables
        - Not as flexible for complex relationships
2. Snowflake Schema
    - Structure:
        - Fact Table in the center
        - Normalized Dimension Tables (i.e., dimensions are broken into sub-dimensions)
    - Example:

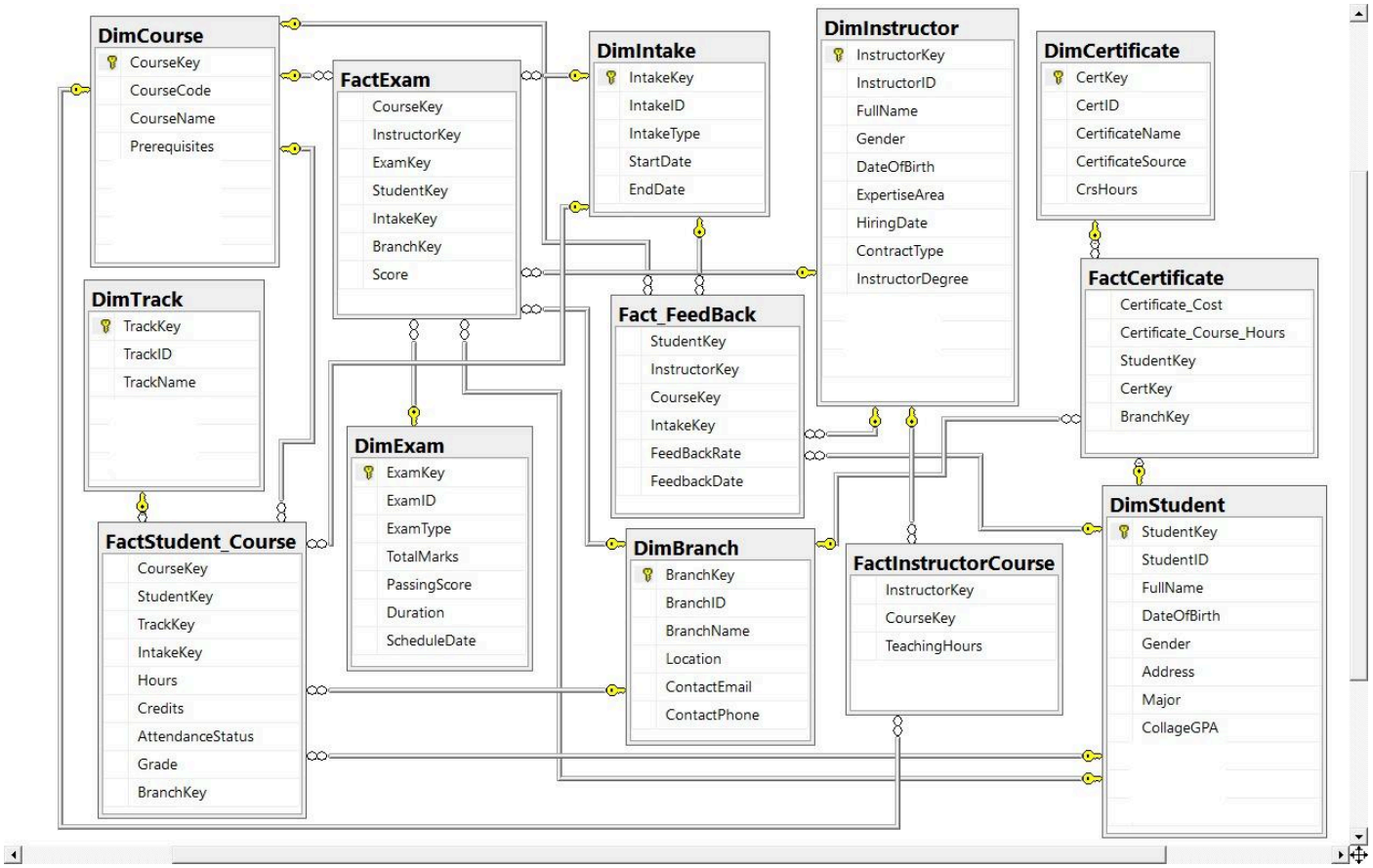
    - DimProduct → DimCategory → DimDepartment

- Pros:
    - Less redundancy (more space-efficient)
    - Easier to maintain and update dimensions
- Cons:
    - More complex joins
    - Slower query performance than star schema

3. Galaxy Schema

- Structure:
    - Multiple Fact Tables
    - Shared Dimension Tables
- Example:
    - FactSales and FactInventory both use DimProduct and DimStore
- Pros:
    - Can model complex business processes
    - Reusable dimensions across multiple facts
- Cons:
    - More complex design and maintenance
    - ETL is harder to manage

Use Case:



Data Warehouse Documentation

Title: Learning Management System (LMS) – Data Warehouse Schema

Schema Type: Galaxy Schema (Fact Constellation)

Objective

- To support decision-making and analytics across various academic activities such as student performance, instructor evaluations, course tracking, certification, and feedback, by integrating data from multiple subject areas into a central analytical repository.

Schema Overview

- This Data Warehouse schema consists of:
  - 5 Fact Tables: capturing key business processes
  - 8 Dimension Tables: providing context and descriptive information
  - Shared Dimensions across multiple facts to maintain consistency and analytical power

Dimension Tables

Dimension Table	Description	Key Attributes
DimStudent	Contains student profile information	StudentID, FullName, Gender, Major
DimInstructor	Contains instructor profile details	InstructorID, FullName, ExpertiseArea
DimCourse	Contains course catalog information	CourseCode, CourseName, Prerequisites
DimExam	Exam metadata	ExamType, TotalMarks, ScheduleDate
DimTrack	Educational track or program	TrackName
DimBranch	Physical or virtual campus locations	BranchName, Location, Contact Info
DimIntake	Academic intake periods	IntakeType, StartDate, EndDate
DimCertificate	Certification programs offered	CertificateName, CrsHours, CertificateSource

- Fact Tables

Fact Table	Description	Key Metrics
FactExam	Exam performance per student, per course, per intake	Score
FactStudent_Course	Student enrollment and academic performance per course	Grade, AttendanceStatus, Hours, Credits
Fact_FeedBack	Feedback ratings students give to instructors	FeedbackRate
FactInstructorCourse	Teaching activity per instructor per course	TeachingHours
FactCertificate	Certificate achievements by students	Certificate_Cost, Certificate_Course_Hours

- Relationships
  - FactExam connects with:
    - `DimStudent`, `DimInstructor`, `DimCourse`, `DimIntake`, `DimBranch`
  - FactStudent_Course connects with:
    - `DimStudent`, `DimCourse`, `DimTrack`, `DimIntake`, `DimBranch`
  - Fact_FeedBack connects with:
    - `DimStudent`, `DimInstructor`, `DimCourse`, `DimIntake`
  - FactInstructorCourse connects with:
    - `DimInstructor`, `DimCourse`, `DimBranch`
  - FactCertificate connects with:
    - `DimStudent`, `DimCertificate`, `DimBranch`
  - Shared Dimensions:
    - `DimStudent`, `DimInstructor`, `DimCourse`, `DimIntake`, and `DimBranch` are used by multiple fact tables.
- Analytical Capabilities Enabled
  - Student performance tracking across intakes and courses

- Course effectiveness and teaching hours per instructor
- Certification progress and cost analysis
- Feedback trends for instructors and course quality
- Enrollment behavior by branch, track, and intake

ETL Extract - Transform - Load Using SSIS

ETL stands for Extract, Transform, Load — it is a data integration process used to move data from multiple source systems into a Data Warehouse for analytics and reporting.

The Three Stages of ETL:

1. Extract
    - What it does: Retrieves raw data from various source systems
    - Sources can include:
        - Relational databases (e.g., SQL Server, Oracle)
        - Flat files (CSV, Excel)
        - APIs
        - ERP or LMS systems
    - Goal: Collect data as-is from all required sources
2. Transform
    - What it does: Cleans, restructures, and enriches the data to fit the data warehouse model

    - Typical Transformations:
        - Data cleansing (remove nulls, fix inconsistencies)
        - Data type conversions
        - Calculations or derivations
        - Applying business rules (e.g., converting letter grades to GPA)
        - Surrogate key mapping via dimension lookups
        - Goal: Convert data into a standardized, analysis-ready format
3. Load

- What it does: Inserts the transformed data into the Data Warehouse tables
- Types of load:
    - Full load: Load entire dataset from scratch
    - Incremental load: Load only new or changed records (more efficient)
- Goal: Store clean, structured data for querying and reporting

Full load VS Incremental load

- Full Load
  - Definition:
    - In a Full Load, all records from the source system are loaded into the destination every time, regardless of whether they've changed or not.
  - How it works:
    - The destination table is truncated (emptied)
    - Then, all data from the source is reloaded from scratch
  - Example:
    - You have a table with 10,000 student records. On each ETL run, you delete everything in the Data Warehouse table and reload all 10,000 records—even if only 5 changed.
  - Advantages:
    - Simple to implement
    - Guarantees data consistency (no partial updates)
  - Disadvantages:
    - Very slow and resource-heavy for large datasets
    - Can cause downtime or load performance issues
    - Not suitable for real-time or near-real-time needs

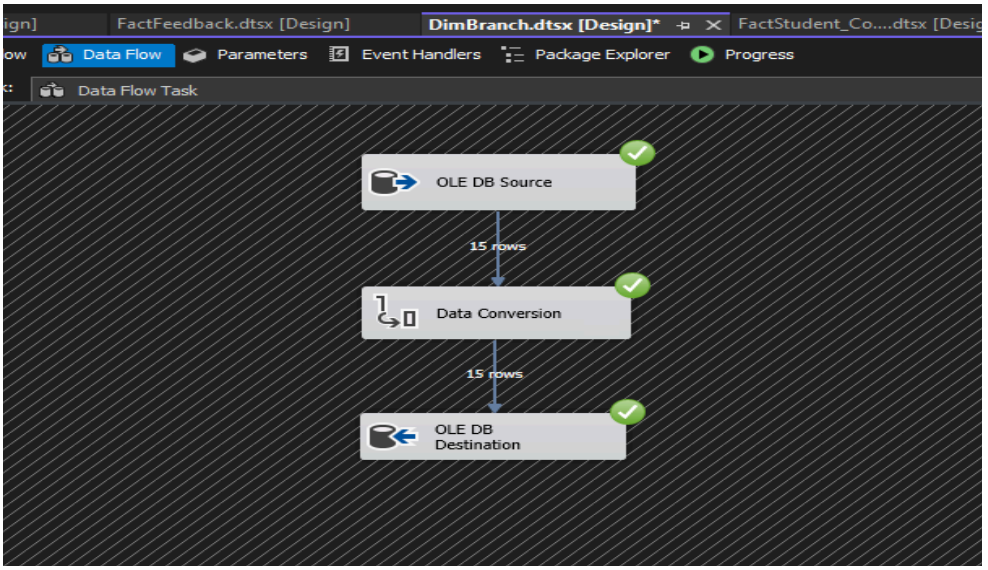
2. Incremental Load

- Definition:
  - In an Incremental Load, only new or changed data (since the last load) is extracted and loaded into the warehouse.
  - How it works:
    - Track data using a mechanism like:
    - `LastModifiedDate`
    - `CreatedDate`
    - Change Tracking/CDC (Change Data Capture
    - Compare source data with what's already in the warehouse
    - Insert new records or update changed ones
  - Example:
    - If 10 records were modified since yesterday, only those 10 get processed and added/updated in the warehouse.
  - Advantages:
    - Much faster and more efficient
    - Reduces load on network and system resources
    - Enables frequent or real-time updates

- Disadvantages:
  - More complex logic required
  - Requires reliable change tracking or audit columns

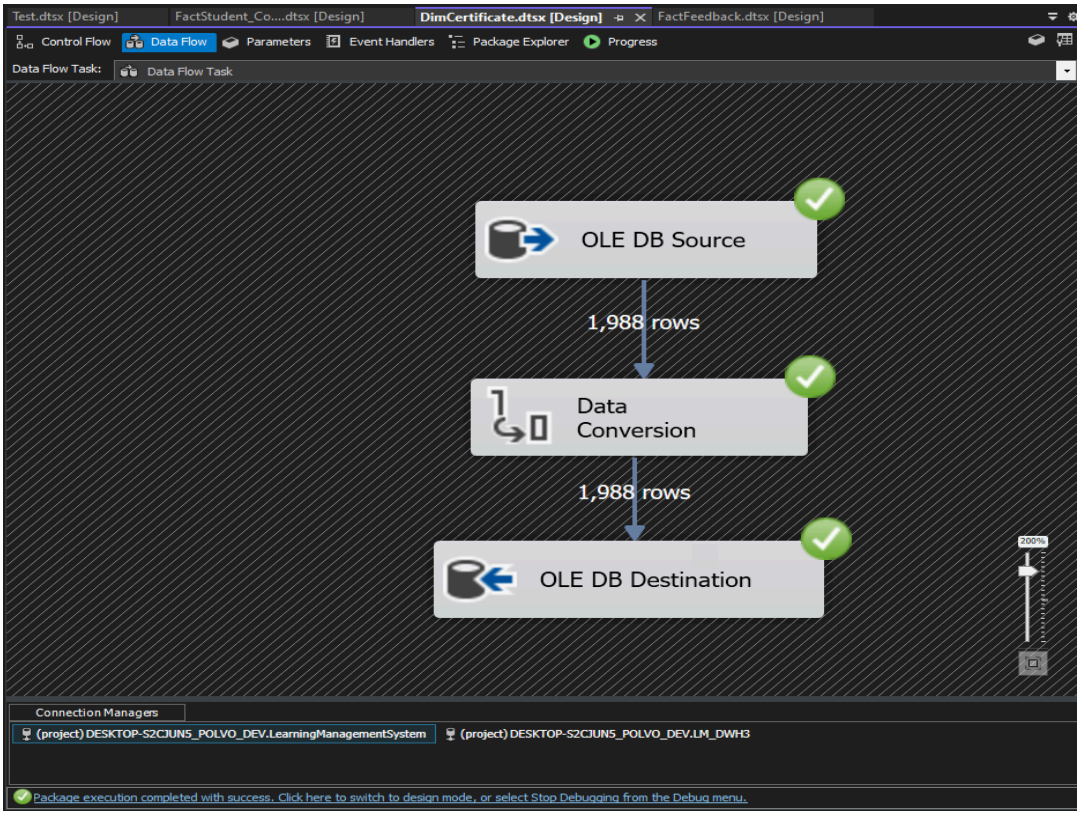
Use Case

- DimBranch:



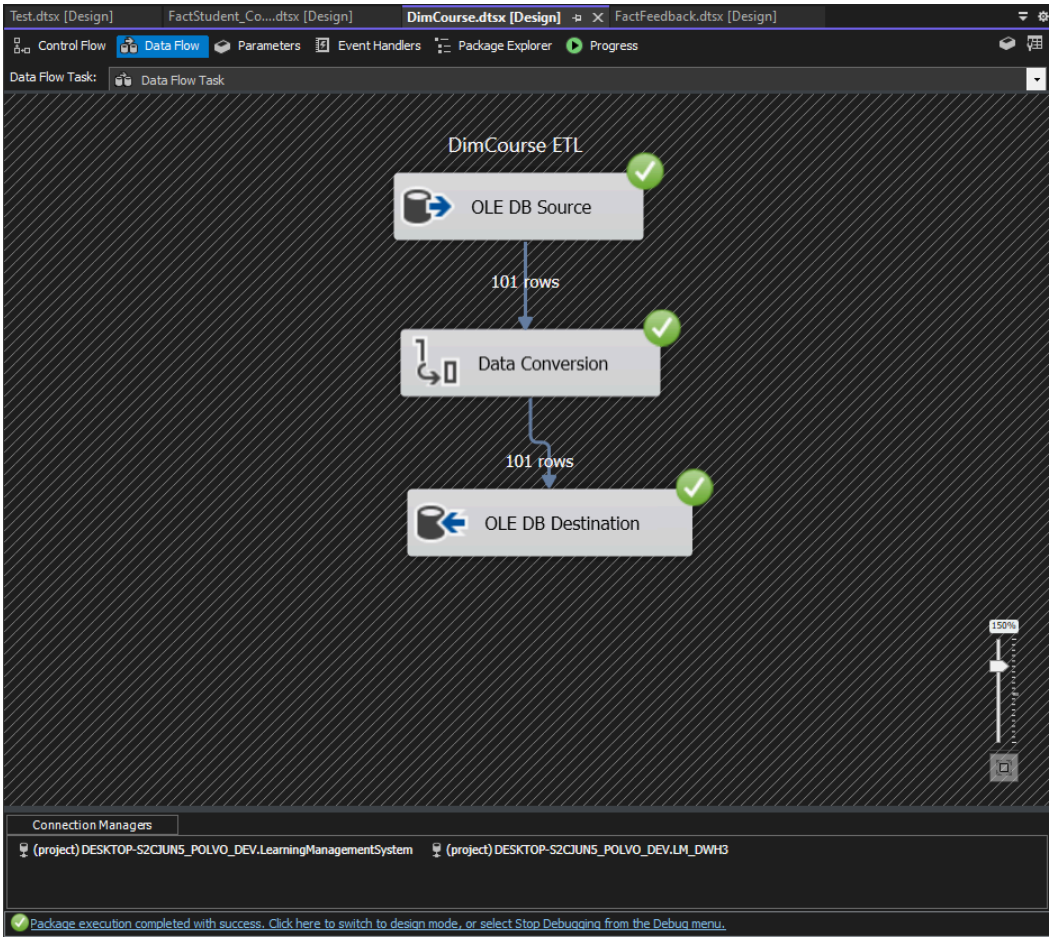
  - We used a full load in the dim branch cause its has no change in information across time, it's always be the same branch so we truncate the table and fill again for every batch.

- DimCertificate



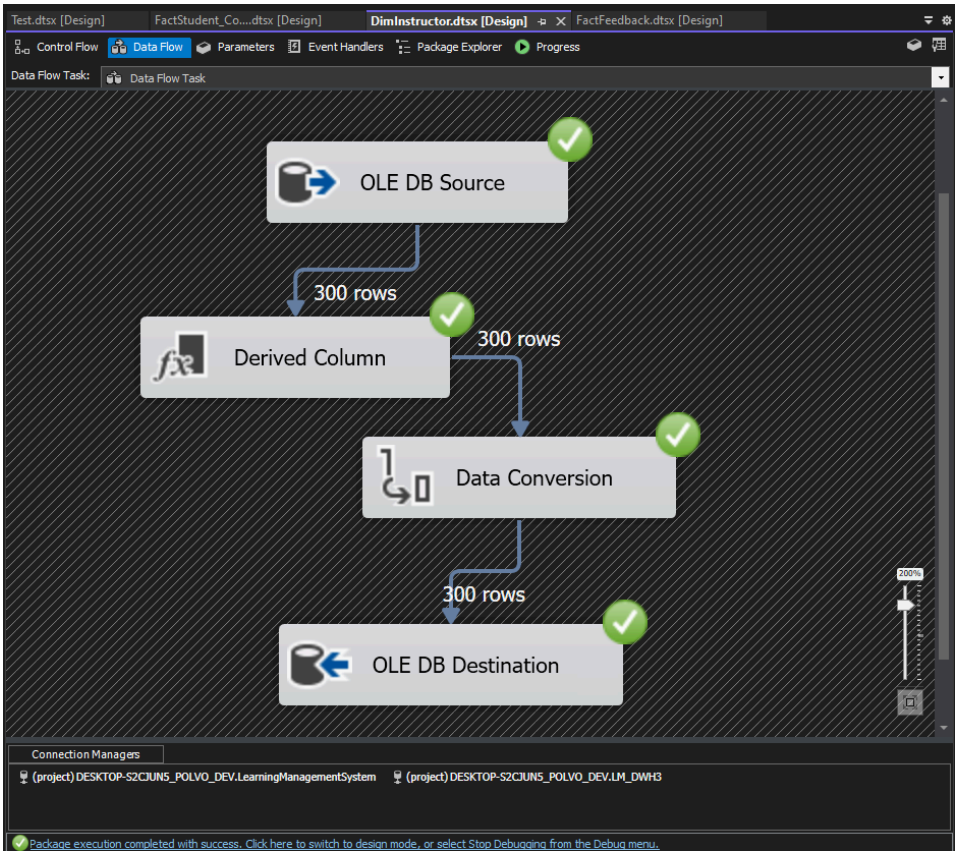
- We used the full load to transfer the certificate dimension because it’s never change over time.

- DimCourse
  - We used full load for transferring course dimension

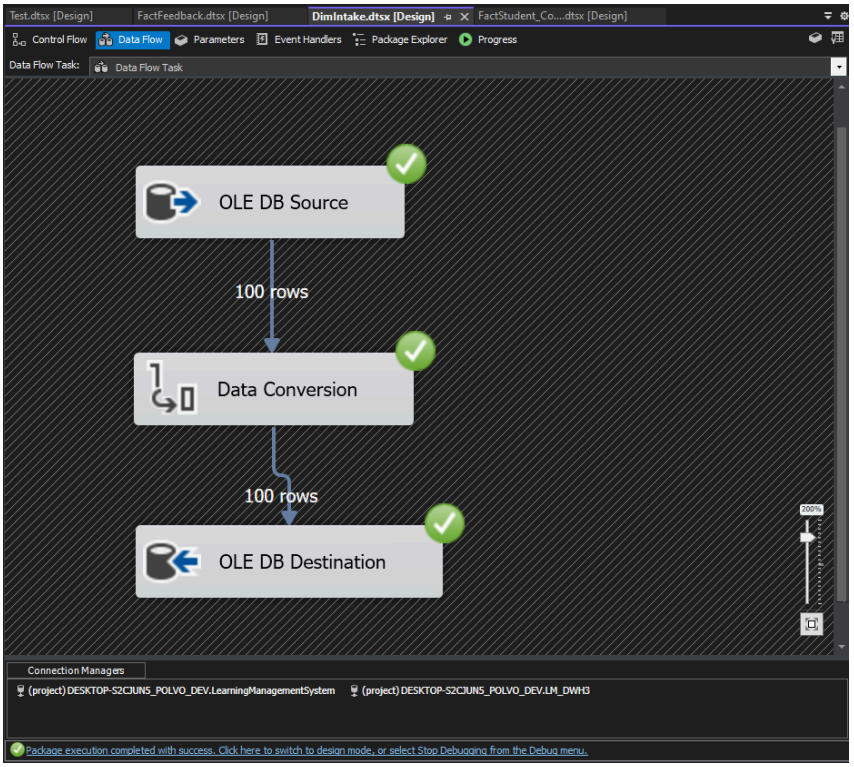


  - DimExam
    - We used full load for transferring Exam Dimension

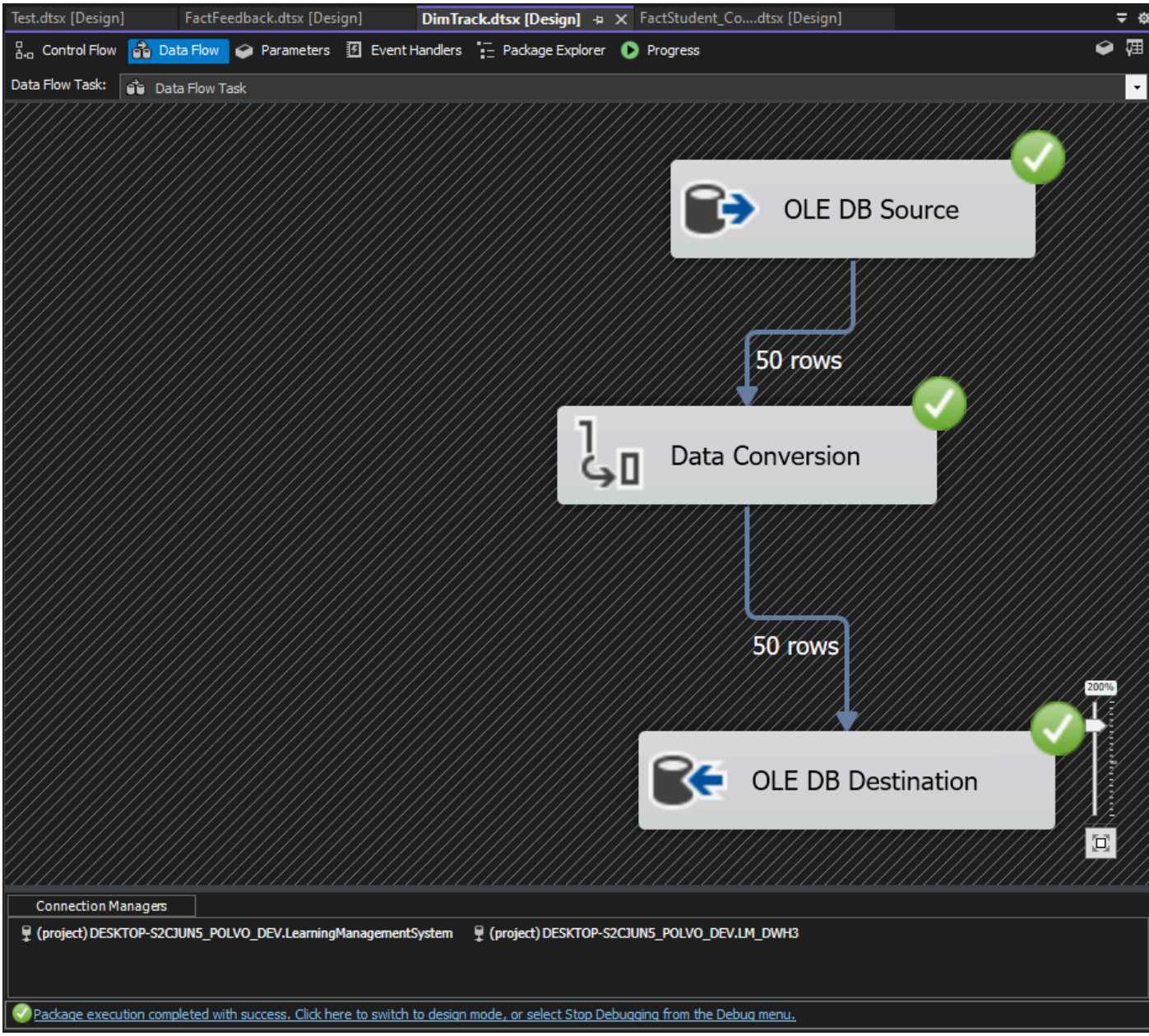
- DimInstructor :
  - We used full load to transferring Instructor Dimension



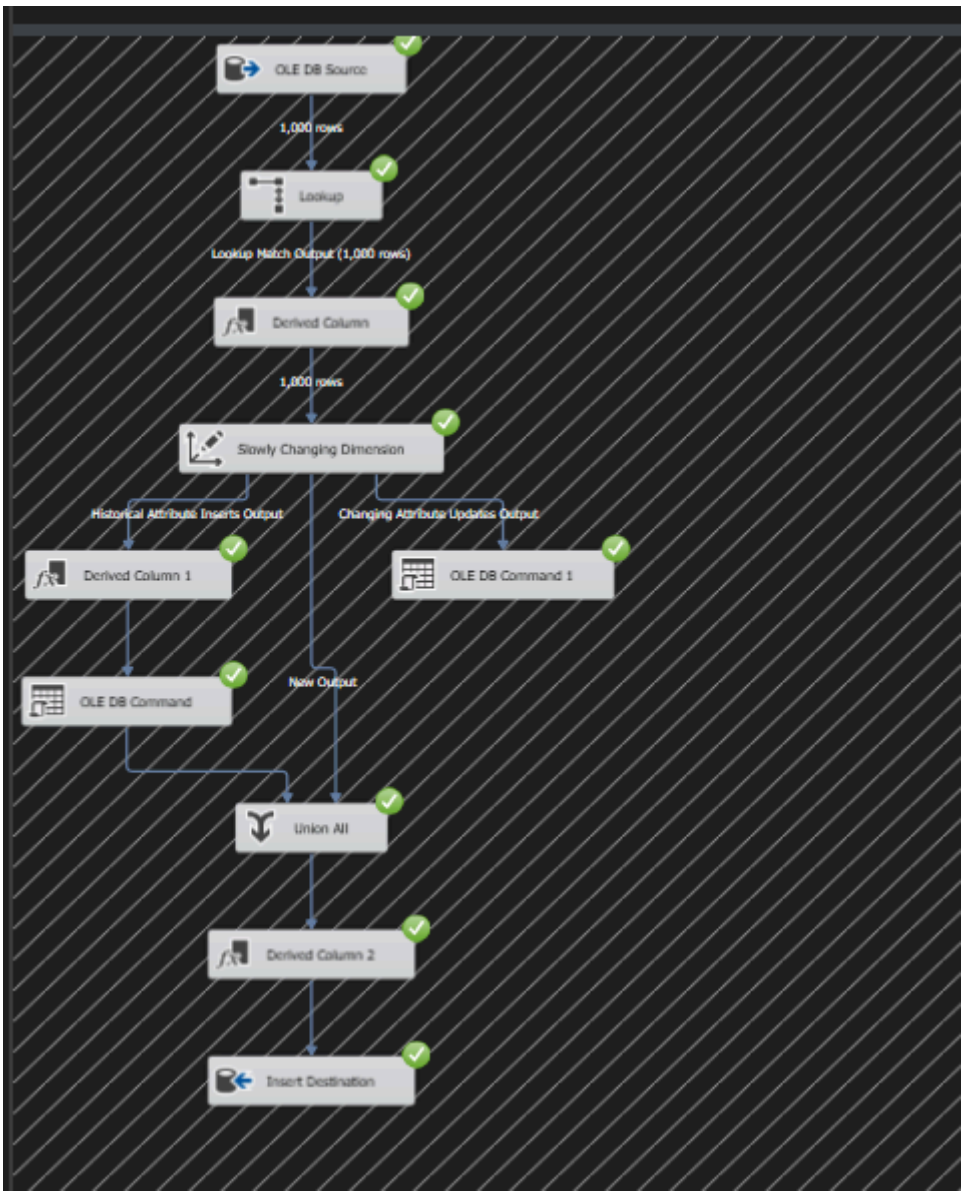
- DimIntake
  - We used full load to transfer intakes information



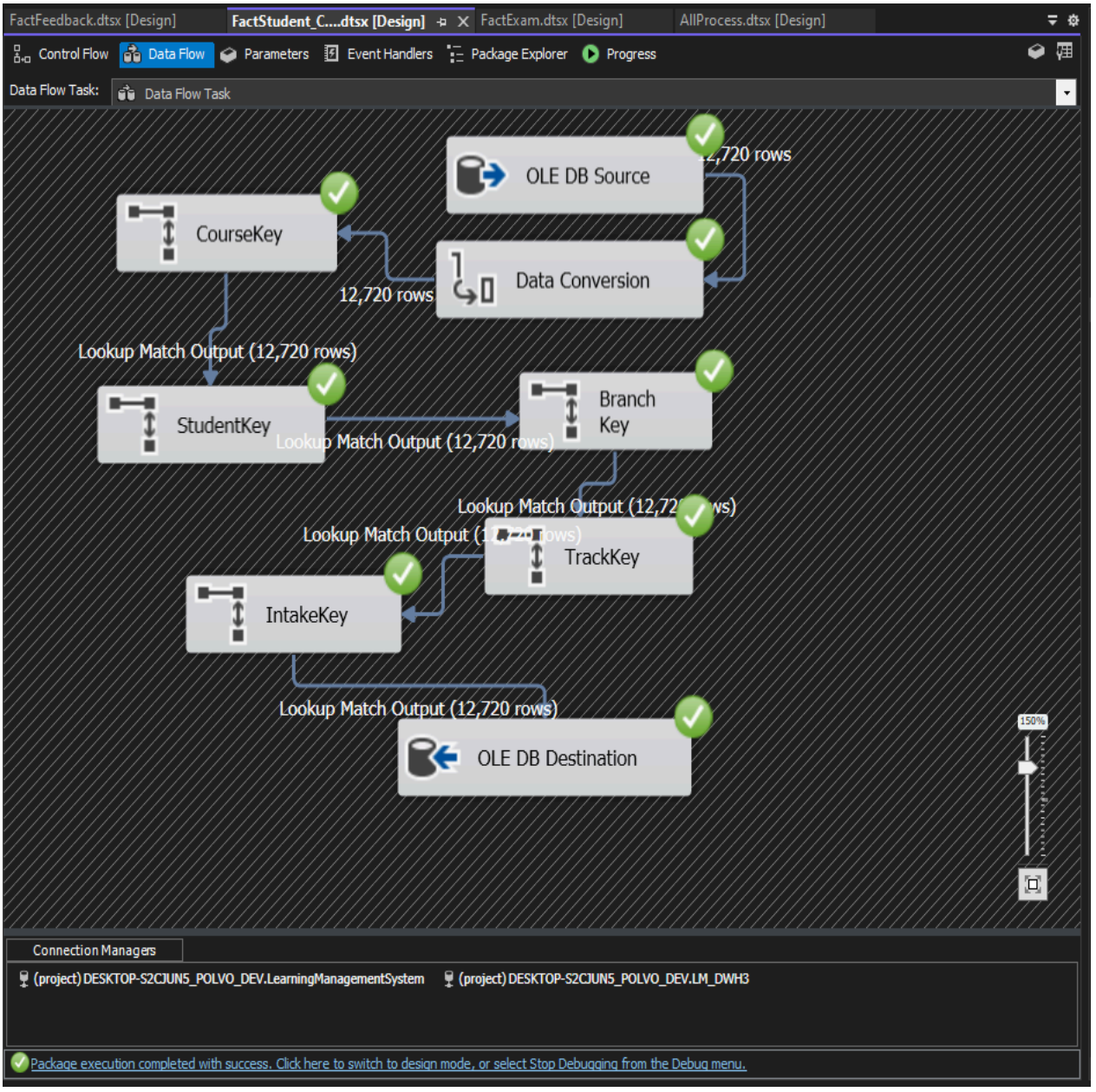
- Dim Track
    - We used full load to transfer Track Dimension


Test.dtsx [Design]
FactFeedback.dtsx [Design]
DimTrack.dtsx [Design]
FactStudent_Co....dtsx [Design]
Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Data Flow Task
OLE DB Source
50 rows
Data Conversion
50 rows
OLE DB Destination
200%
Connection Managers
(project) DESKTOP-S2CJUN5_POLVO_DEV.LearningManagementSystem
(project) DESKTOP-S2CJUN5_POLVO_DEV.LM_DWH3
Package execution completed with success. Click here to switch to design mode, or select Stop Debugging from the Debug menu.

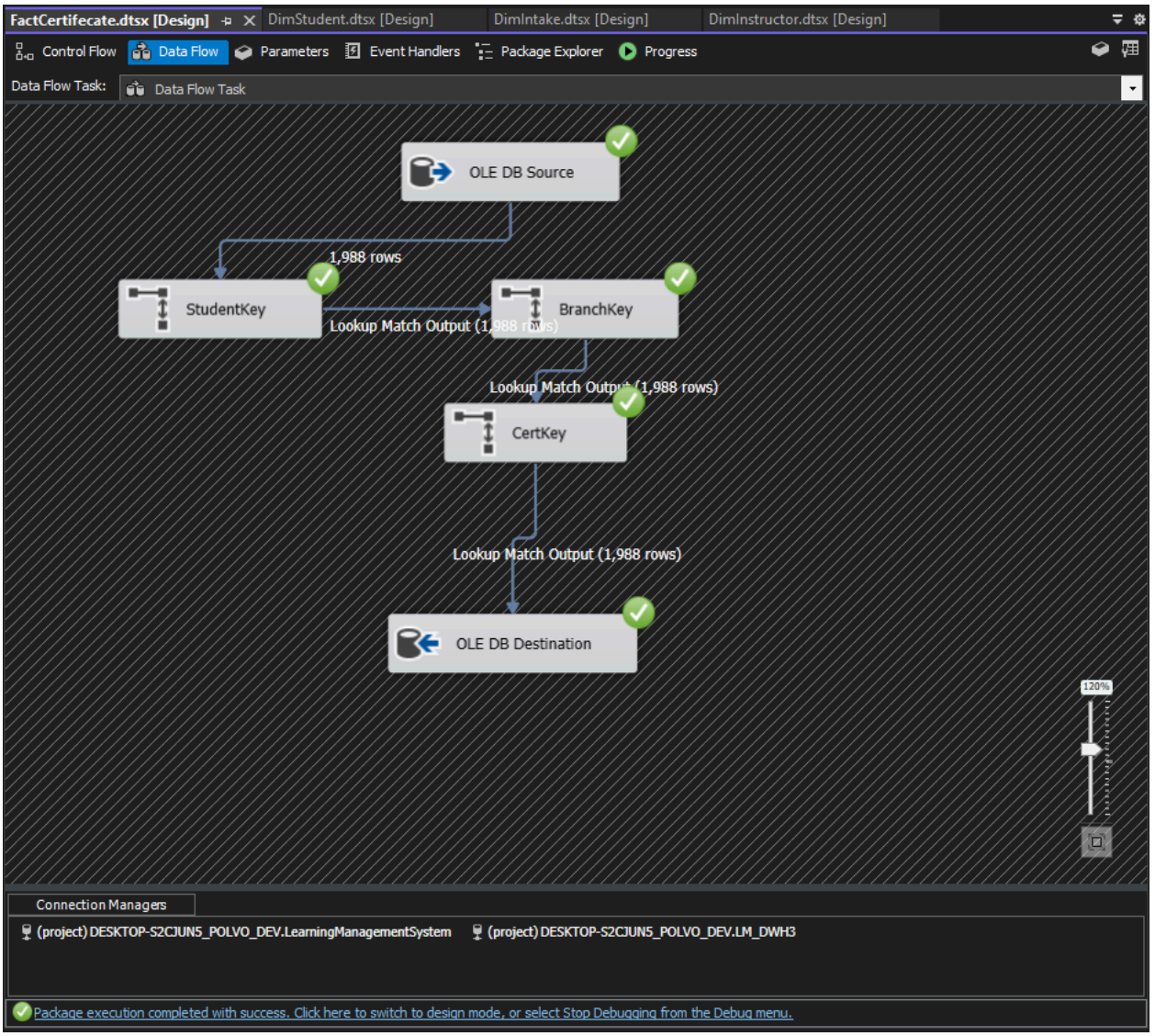
- DimStudent
  - We used incremental load for transferring the student information because it’s changing but in wide various time



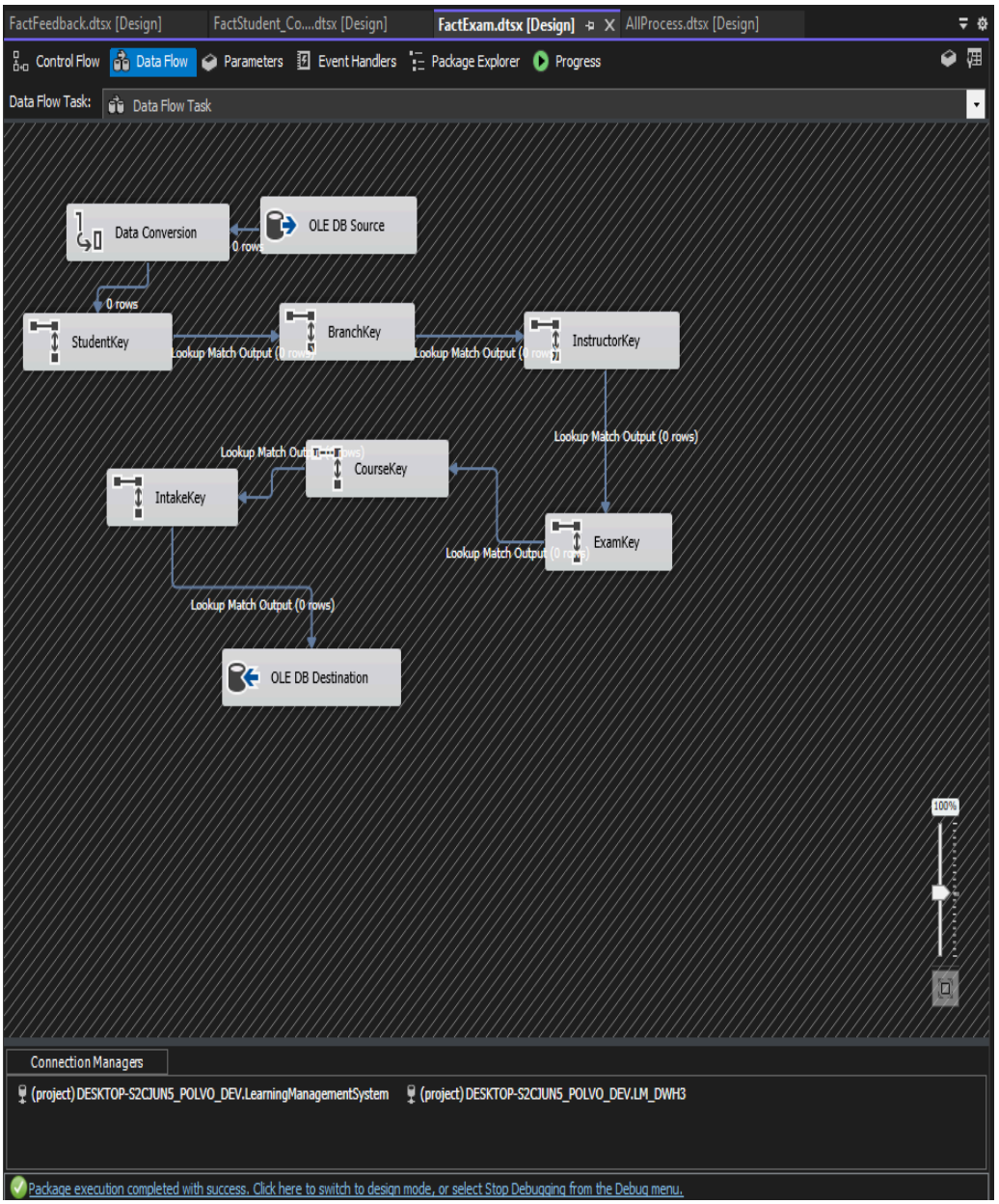
- Fact Student-Course
    - Measures the student performance per course



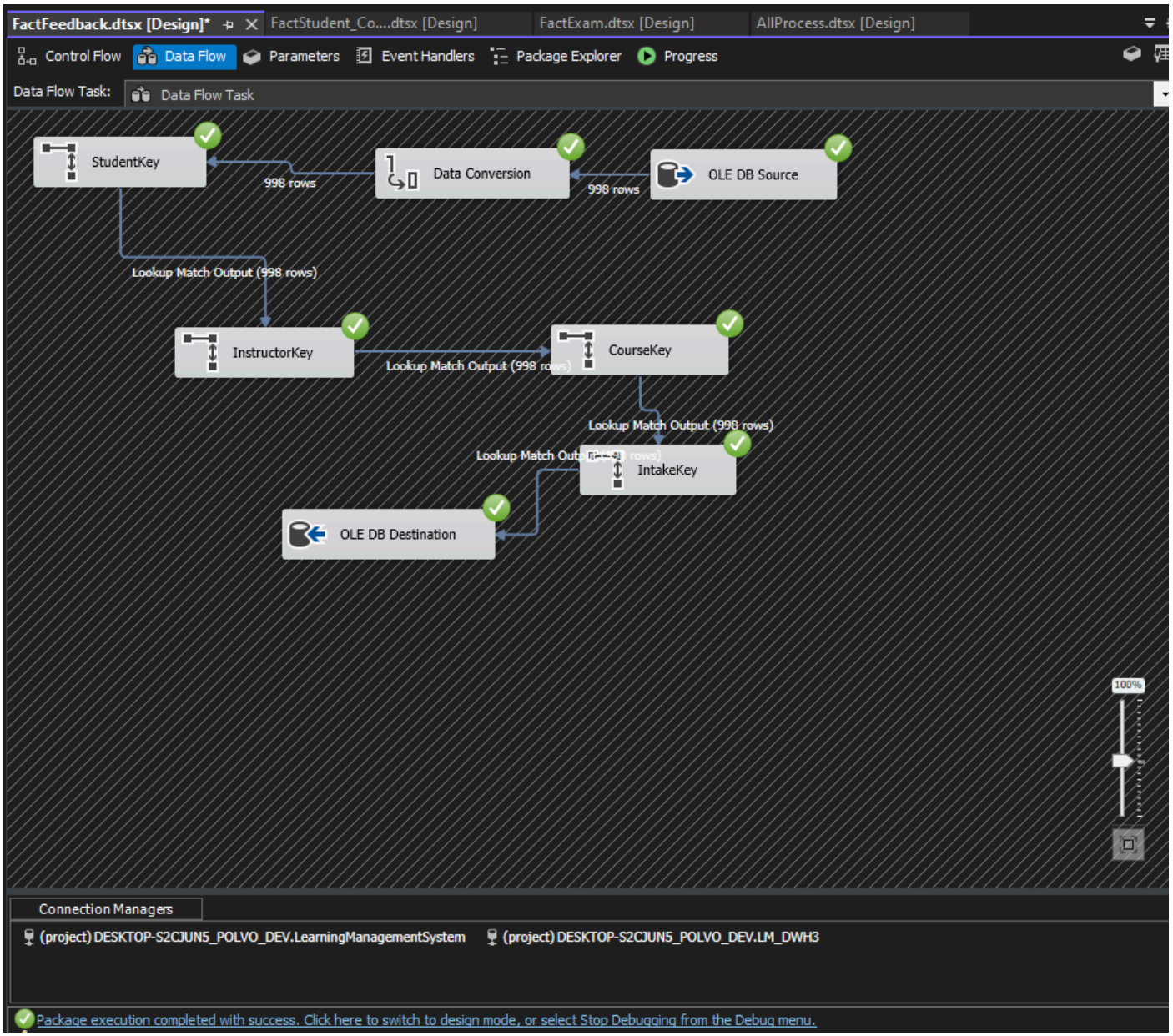
- Fact Certificate
    - Analyze certification per student



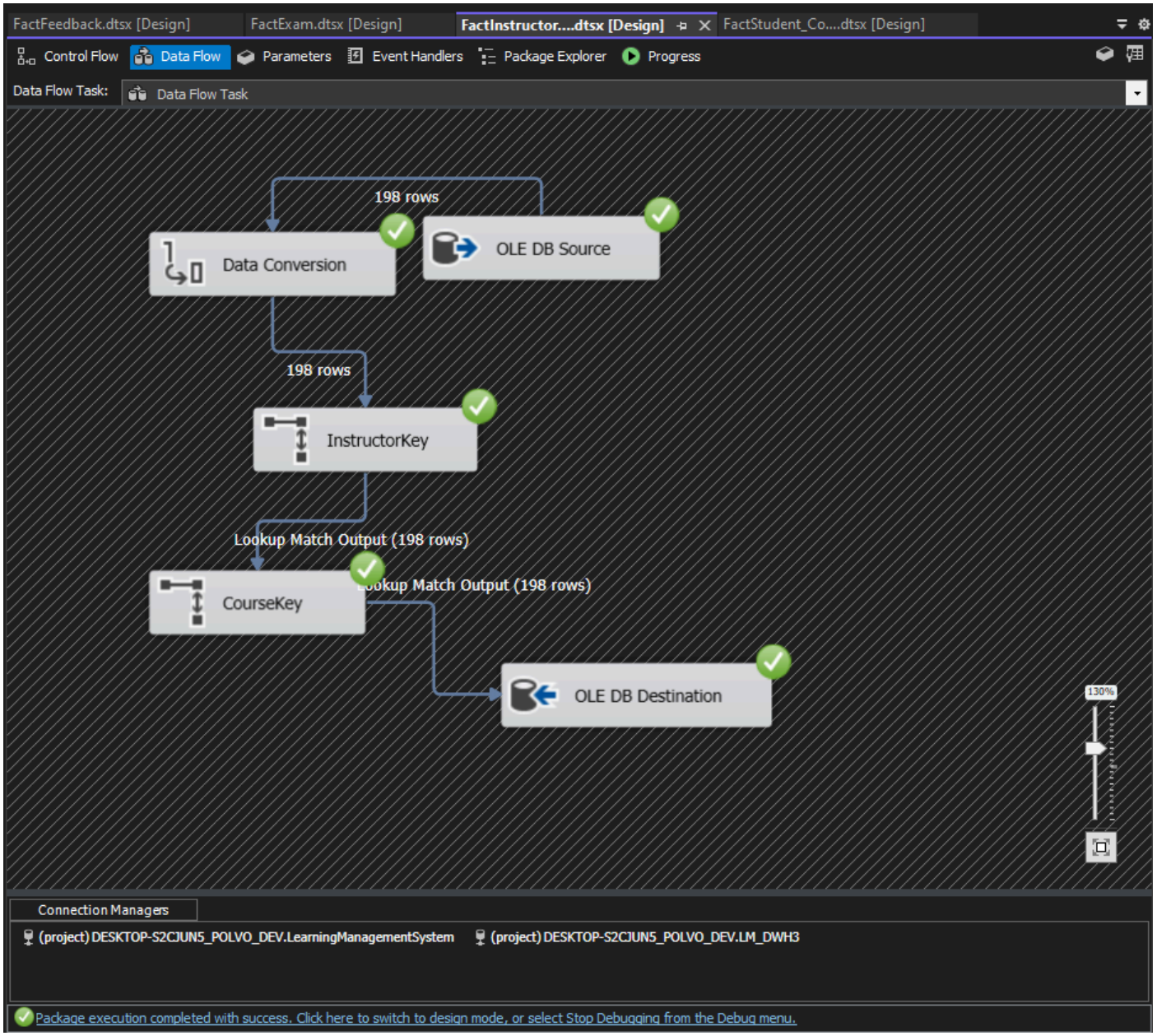
- Fact Exam
  - Measures Student performance per exam



- Fact Feedback
    - Analyze every feedback from each student to each course/ Instructor

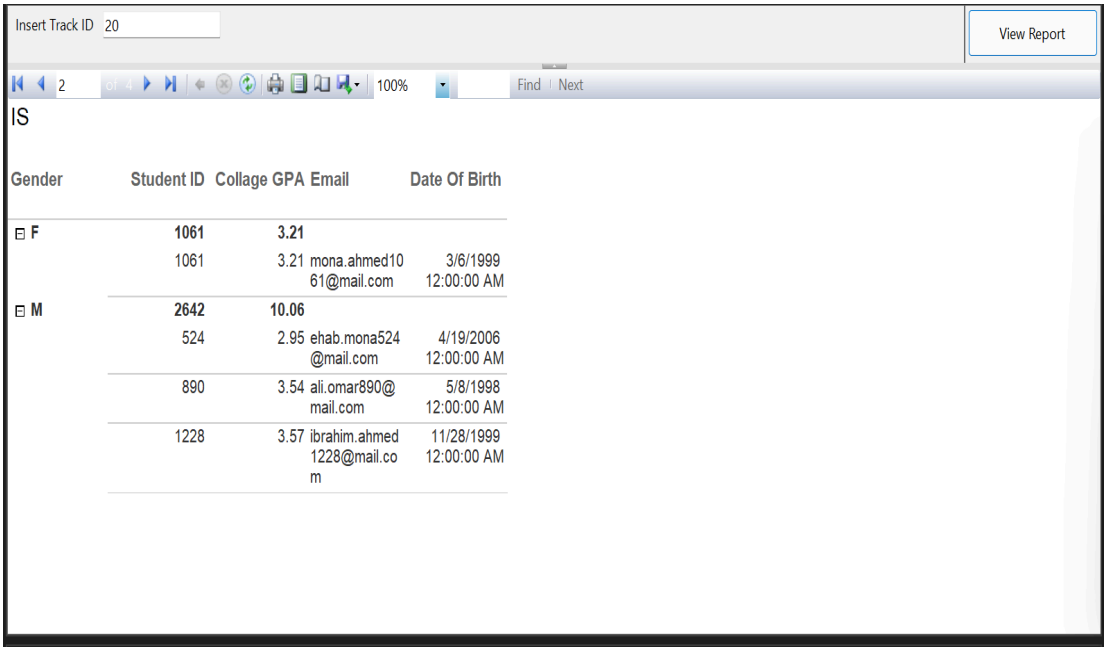


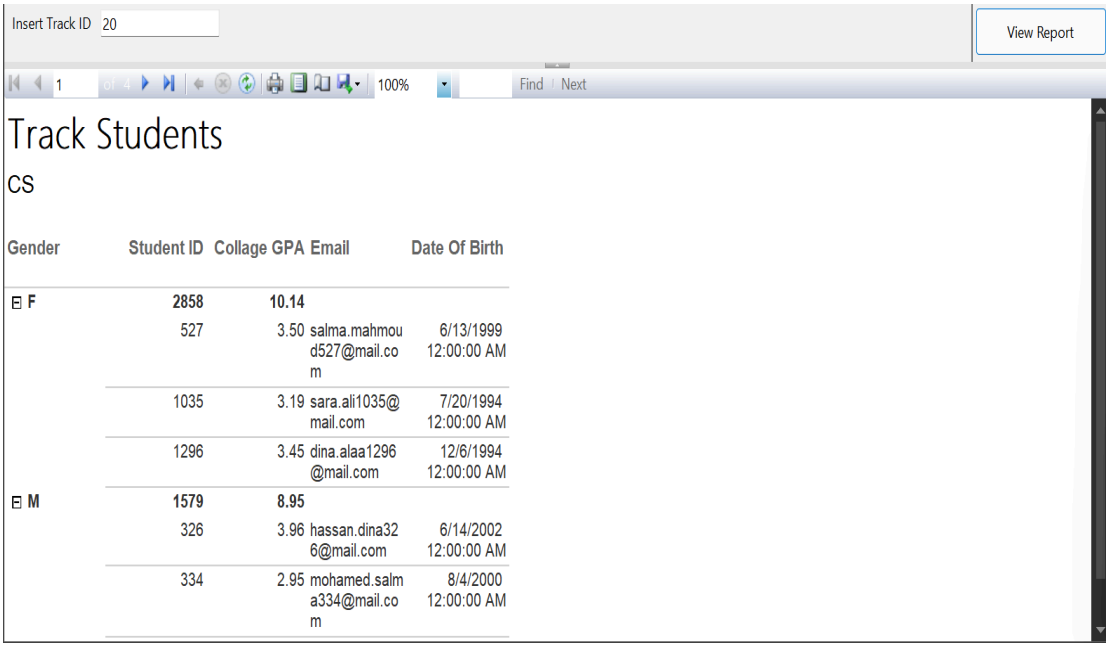
- Fact Instructor Course
    - Analyze the course details for each Instructor


FactFeedback.dtsx [Design]
FactExam.dtsx [Design]
FactInstructor....dtsx [Design]
FactStudent_Co....dtsx [Design]
Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Data Flow Task
198 rows
Data Conversion
OLE DB Source
198 rows
InstructorKey
Lookup Match Output (198 rows)
CourseKey
ookup Match Output (198 rows)
OLE DB Destination
130%
Connection Managers
(project) DESKTOP-S2CJUN5_POLVO_DEV.LearningManagementSystem
(project) DESKTOP-S2CJUN5_POLVO_DEV.LM_DWH3
Package execution completed with success. Click here to switch to design mode, or select Stop Debugging from the Debug menu.

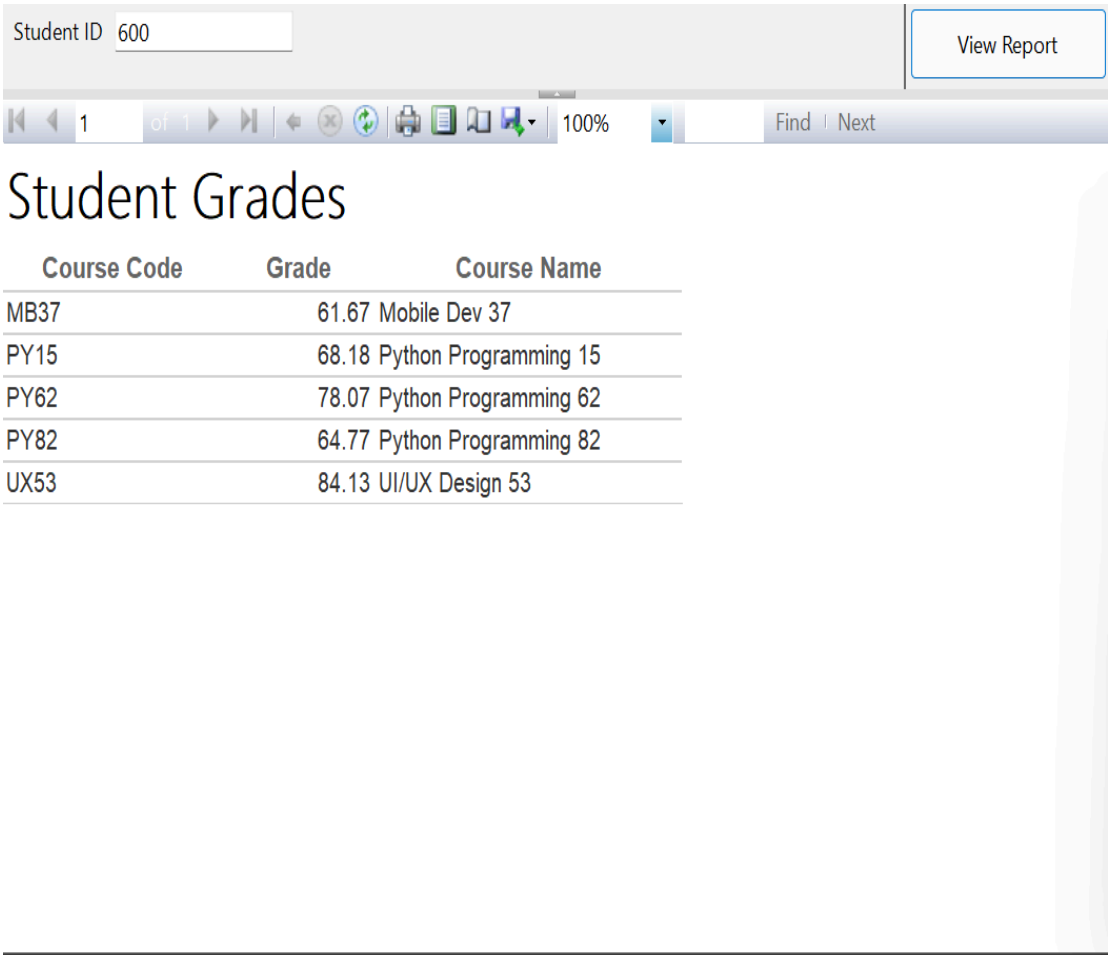
Reporting

- Using SSRS we created 6 Reports was required for business multiple search
    - Report that returns the students information according to Department No parameter.

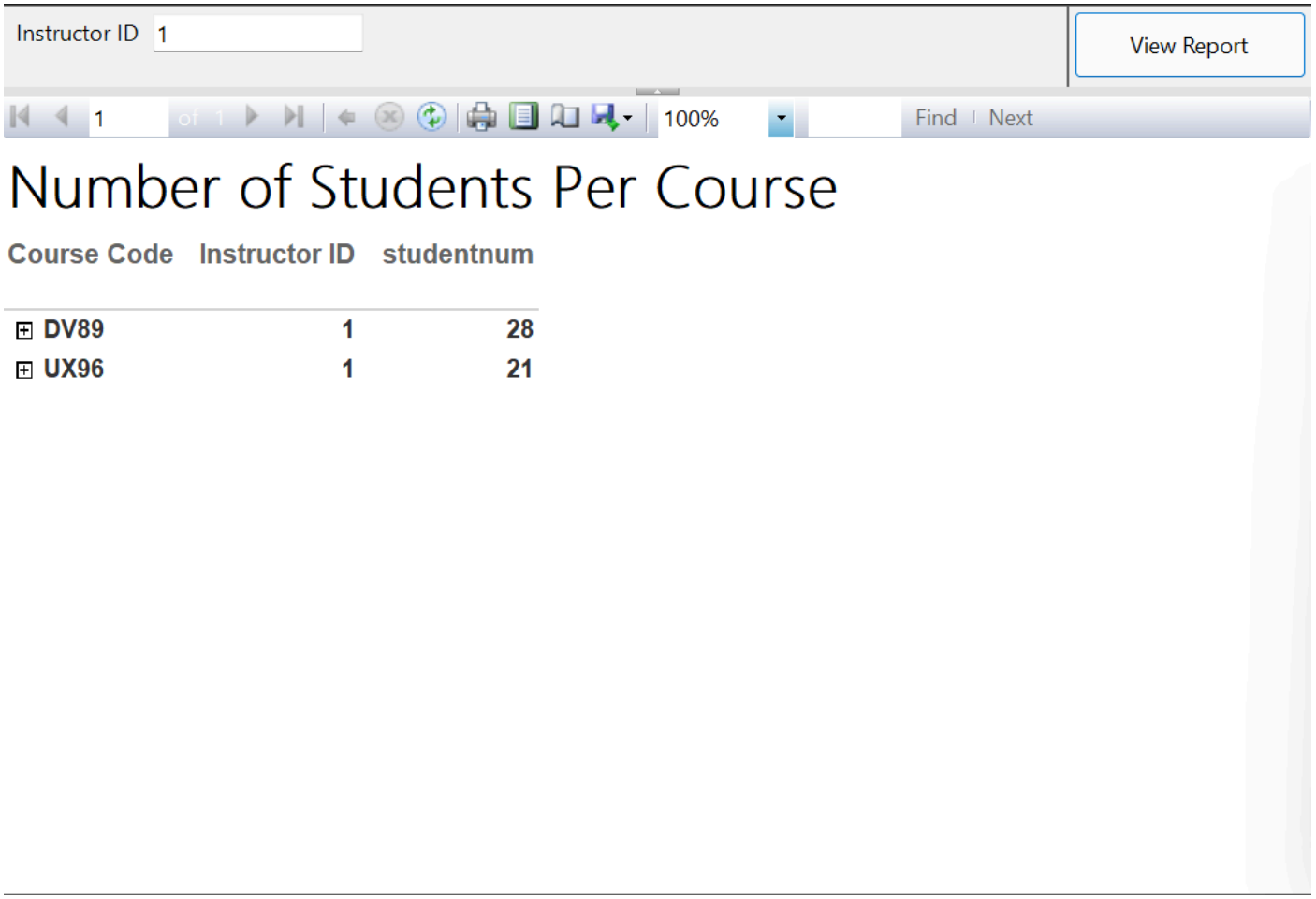




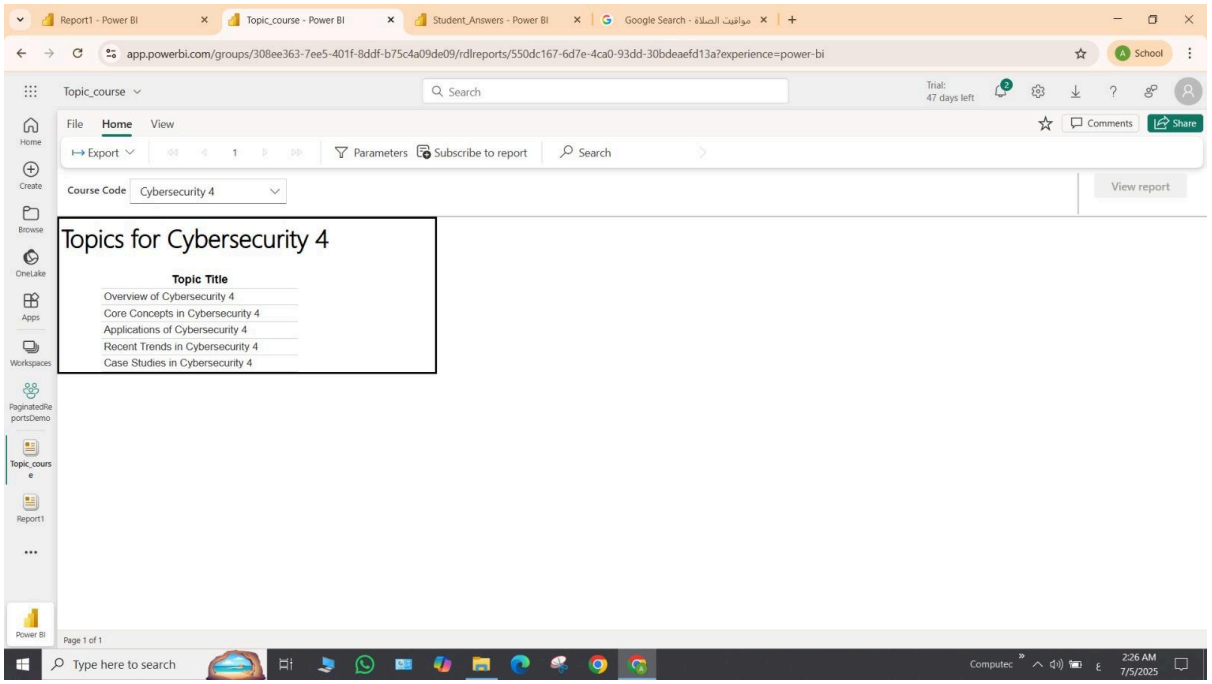
- Report that takes the student ID and returns the grades of the student in all courses. %



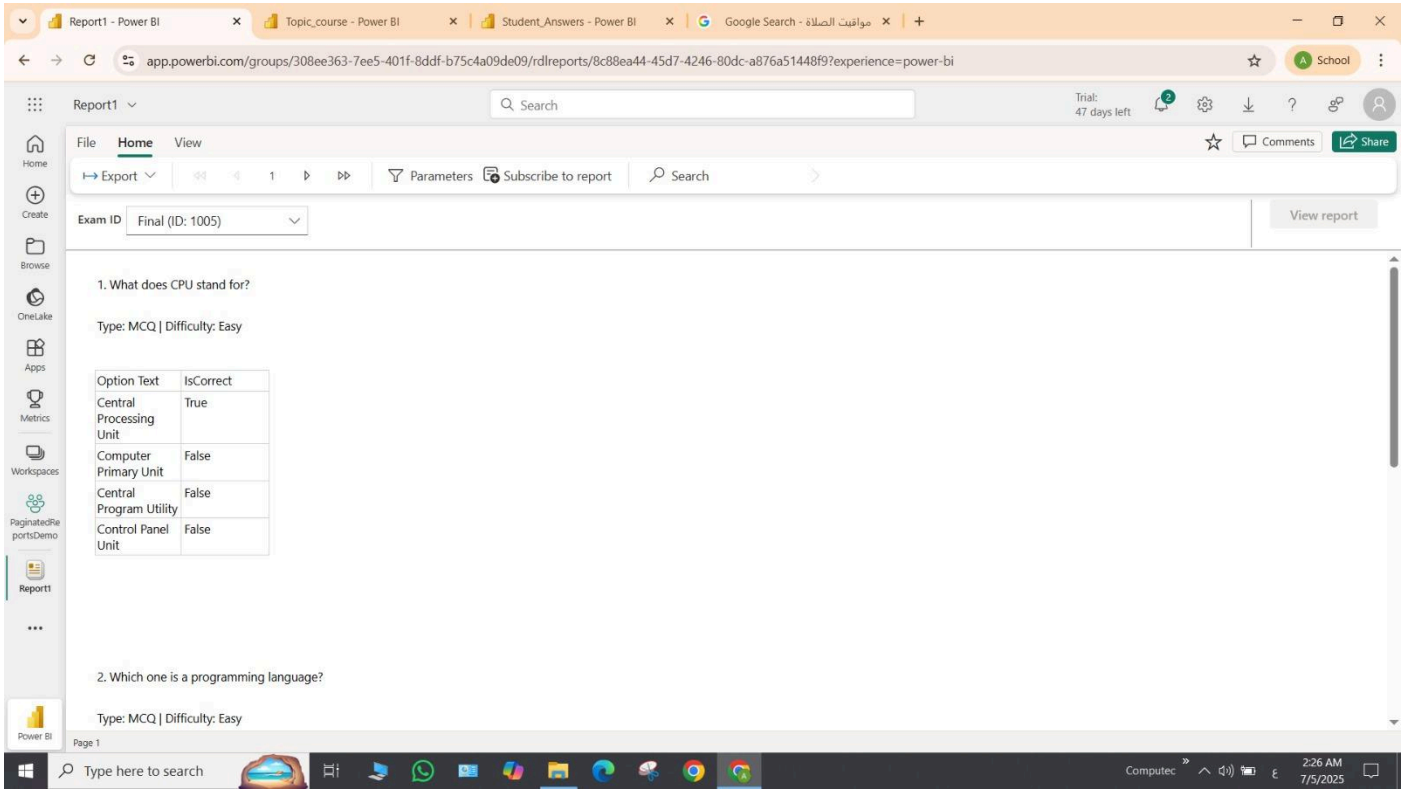
- Report that takes the instructor ID and returns the name of the courses that he teaches and the number of student per course.



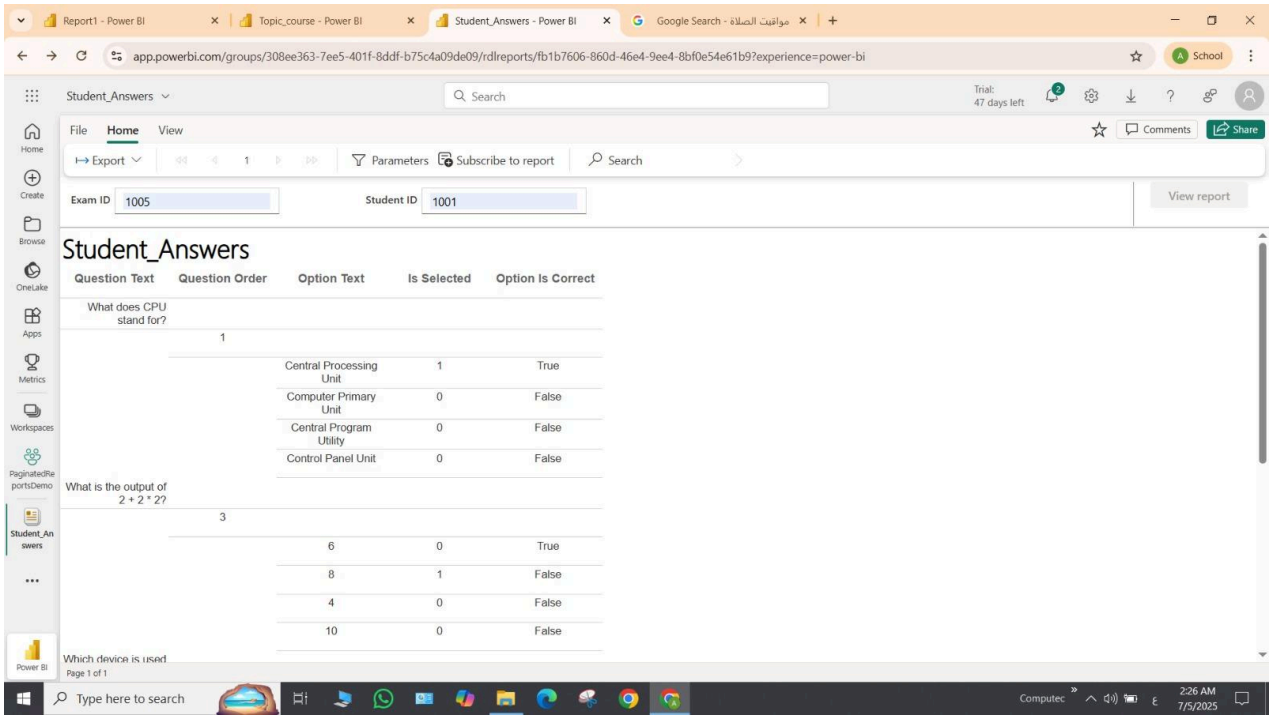
- Report that takes course ID and returns its topics



- Report that takes exam number and returns the Questions in it and chocies [freeform report]



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



1. BRANCH_SP.sql

Purpose:

Manages CRUD operations for branches (locations).

Procedures:

- `AddBranch`
- `UpdateBranch`
- `DeleteBranch`
- `GetAllBranches`

Parameters:

- `@BranchID INT`, `@BranchName VARCHAR(100)`, etc.

---

2. Certificate_SP.sql

Purpose:

Handles student certificate records.

Procedures:

- `InsertCertificate`
- `UpdateCertificate`
- `DeleteCertificate`
- `GetCertificatesByStudentID`

---

3. `Course_SP.sql`

Purpose:

Handles all course-related operations.

Procedures:

- `AddCourse`
- `UpdateCourse`
- `DeleteCourse`
- `GetCourseByID`
- `GetAllCourses`

---

4. `CreateUser_SP.sql`

Purpose:

Creates users and assigns them to roles (Admin/Instructor).

Procedures:

- `AddNewUserWithRole`

Parameters:

- `@Username`, `@Email`, `@PasswordHash`, `@RoleID`, e

---

5. `Enhanced_Exam_ST_Procedures.sql`

Purpose:

Enhanced procedures for exam management (standardized).

Procedures:

- Includes advanced logic: status tracking, validation, etc.

---

6. `Exam_SP.sql`

Purpose:

Core exam operations.

Procedures:

- `CreateExam`
- `UpdateExam`
- `DeleteExam`
- `GetExamByCourse`

---

7. `Freelancing_SP.sql`

Purpose:

Manages freelance job opportunities for students.

Procedures:

- `AddFreelanceJob`
- `UpdateFreelanceJob`
- `DeleteFreelanceJob`
- `GetJobsForStudent`

---

8. `GraduatedStudent_SP.sql`

Purpose:

Handles graduation records.

Procedures:

- `AddGraduatedStudent`
- `GetGraduatedStudents`
- `DeleteGraduatedStudent`

---

9. `Intake_SP.sql`

Purpose:

Manages academic intakes and cohorts.

Procedures:

- `AddIntake`
- `UpdateIntake`
- `DeleteIntake`
- `GetAllIntakes`

---

10. `Login_SP.sql`

Purpose:

Handles user authentication.

Procedures:

- `LoginUser`

Parameters:

- `@Username`, `@Password`

---

11. `Program_SP.sql`

Purpose:

Handles educational programs in the LMS.

Procedures:

- `AddProgram`
- `UpdateProgram`
- `DeleteProgram`
- `GetAllPrograms`

---

12. `QuestAndOptions_SP.sql`

Purpose:

Manages exam questions and their options.

Procedures:

- `AddQuestion`
- `AddOptions`
- `UpdateQuestion`
- `DeleteQuestion`
- `GetQuestionsByExamID`

---

13. `ST For Reporting.sql`

Purpose:

Reporting procedures for dashboards and analytics.

Content:

- Stored procedures that generate metrics (completion %, scores, etc.)

---

14. `Topic_SP.sql`

Purpose:

Handles course topics and submodules.

Procedures:

- `AddTopic`
- `UpdateTopic`
- `DeleteTopic`
- `GetTopicsByCourse`

---

15. `Track_SP.sql`

Purpose:

Handles learning tracks for grouping courses.

Procedures:

- `AddTrack`
- `UpdateTrack`
- `DeleteTrack`
- `GetAllTracks`

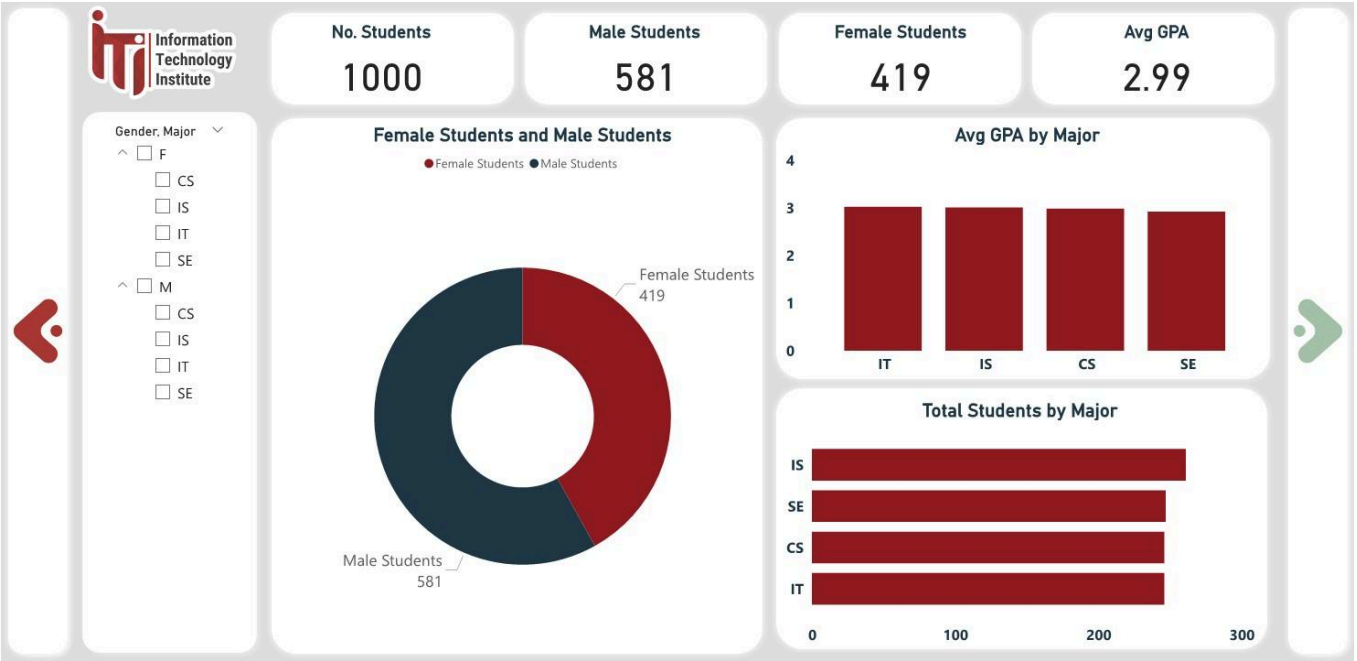
Project Overview

This comprehensive Power BI dashboard project provides a 360-degree view of educational institution performance through 20 specialized dashboards. The system enables data-driven decision making for administrators, instructors, and stakeholders by offering deep insights into student performance, academic operations, and institutional effectiveness.

The dashboard system transforms raw educational data into actionable insights, covering critical areas including student demographics, academic performance tracking, instructor effectiveness, certification management, and institutional operations. Each dashboard is designed to address specific analytical needs while maintaining integration with the overall institutional data ecosystem.

Dashboard Documentation

1. Student Overview Dashboard



Topic / Description:

Provides a high-level summary of student demographics and academic performance. Tracks total students, gender distribution, and GPA averages.

Benefits:

- Understand general student population characteristics.
- Support strategic academic and demographic monitoring.

Cards and Their Benefits:

- **Total Students:** Shows overall student count.
- **Female Students:** Tracks gender distribution.
- **Average GPA:** Indicates average academic performance.

Charts and Their Benefits:

- **Gender Distribution (Donut):** Visualizes gender balance.
- **Students by Major (Bar):** Shows student distribution by major.
- **GPA by Major (Column):** Compares academic performance across majors.

---

2. GPA by Track and Major

Topic / Description:

Analyzes GPA variations across tracks, majors, and branches to assess academic quality.

Benefits:

- Identify top-performing categories.
- Spot low-performing areas needing intervention.

Cards and Their Benefits:

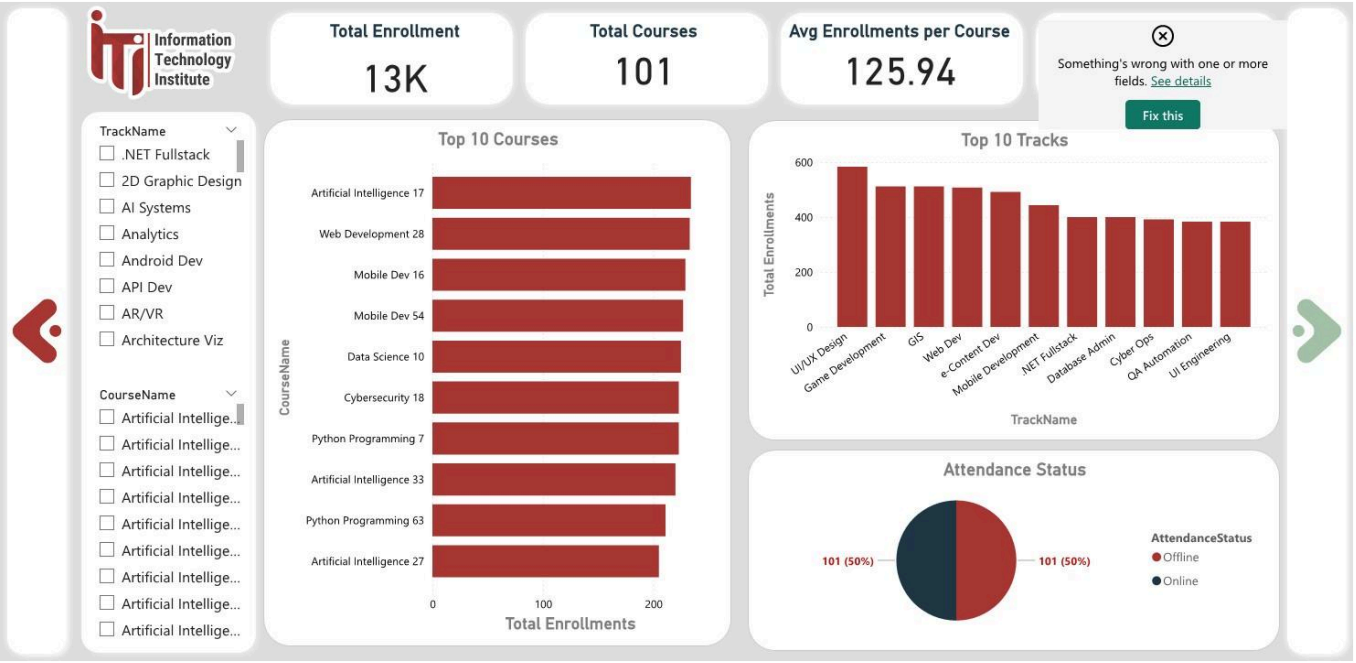
- **Avg GPA:** Measures general academic performance.
- **Highest GPA:** Highlights best achievers.
- **Lowest GPA:** Shows where students struggle the most.

Charts and Their Benefits:

- **Average Course Grade by Major (Column Chart):** Compares average academic results across different majors to assess subject-level performance.
- **Students Passed the Exam (Decomposition Tree):** Enables drill-down analysis of pass rates across dimensions like gender, track, and major to identify root factors influencing success.

- **Average Exam Score by Gender (Donut Chart):** Highlights gender differences in exam performance to support targeted academic support.

3. Course Enrollment Dashboard



Topic / Description:

Tracks course popularity and student engagement over time and across tracks.

Benefits:

- Understand student course preferences.
- Support curriculum planning.

Cards and Their Benefits:

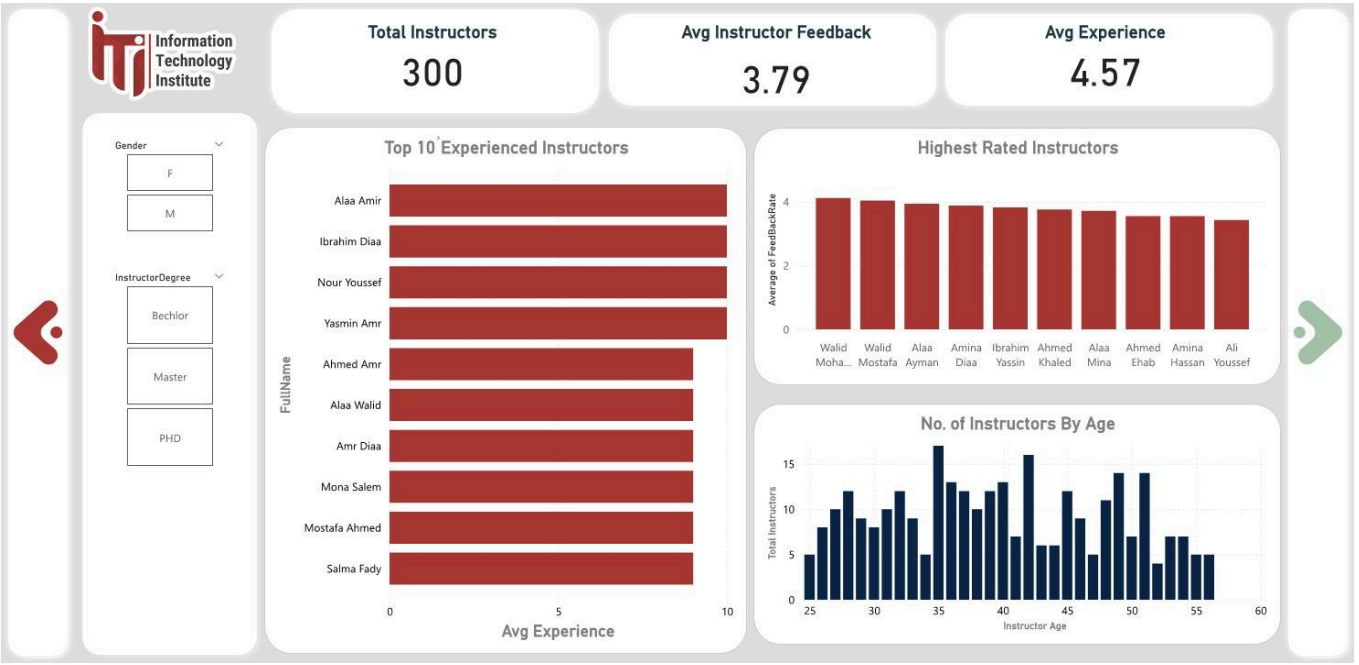
- **Total Courses:** Number of offered courses.
- **Total Enrollments:** Measures engagement across all courses.

Charts and Their Benefits:

- **Enrollments by Course (Bar):** Highlights most popular courses.
- **Attendance Status (Pie):** Identifies types of attendace.
- **Enrollments by Track (Column):** Links track design with course demand.

---

4. Instructor Insights



Topic / Description:

Analyzes instructor count, hiring trends, and feedback ratings.

Benefits:

- Helps HR monitor instructor performance and staffing.
- Supports quality assurance in teaching.

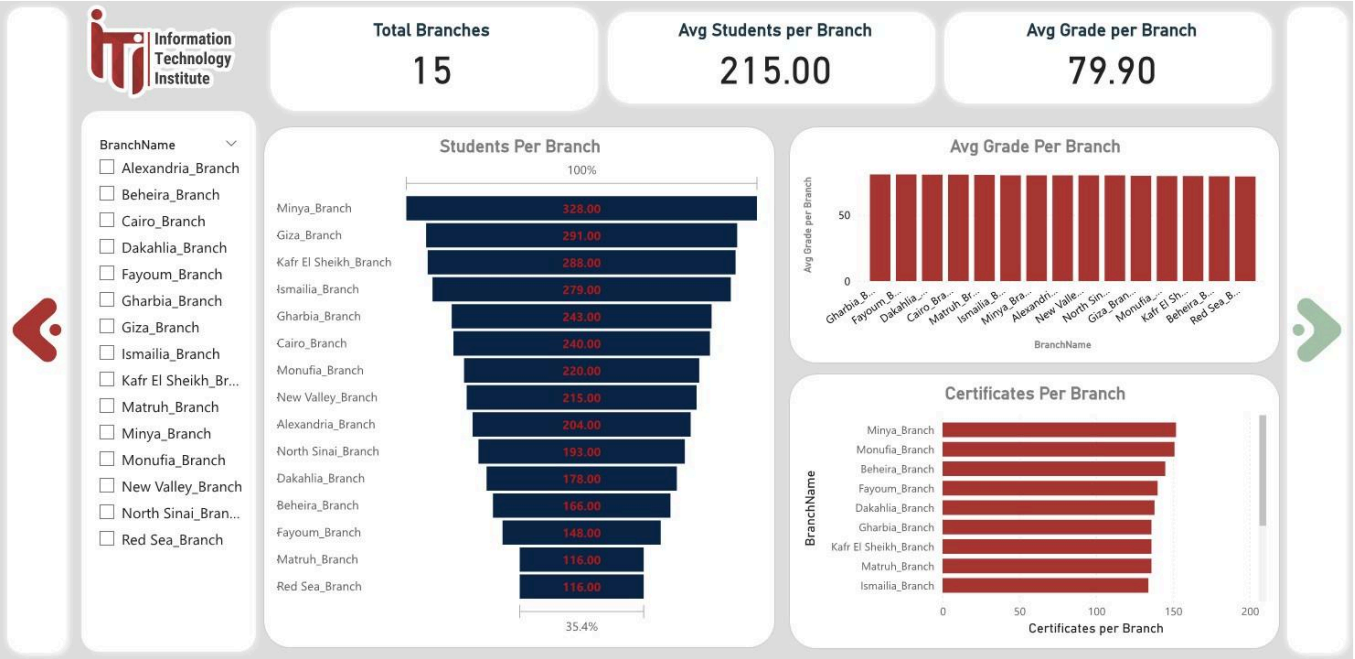
Cards and Their Benefits:

- Total Instructors: Shows workforce size.
- New Hires: Tracks instructor recruitment.
- Avg Instructor Feedback: Measures teaching quality.

Charts and Their Benefits:

- Instructors by Expertise (Bar): Shows distribution by subject.
- Instructors Age Groups (Column): Tracks age distribution of Instructors.
- Avg Feedback per Instructor (Column): Evaluates individual performance.

5. Branch Performance Dashboard



Topic / Description:

Analyzes student numbers and GPA performance across branches.

Benefits:

- Compare branch-level performance.
- Guide investment in branches.

Cards and Their Benefits:

- **Total Branches:** Shows branch spread.
- **Students per Branch:** Measures enrollment density.
- **Avg GPA per Branch:** Evaluates academic effectiveness.

Charts and Their Benefits:

- **Students by Branch (Funnel):** Identifies large or underused branches.
- **GPA by Branch (Column):** Tracks academic success by location.
- **Certificates by Branch (bar):** Visualizes achievement distribution.

6. Feedback Dashboard

Topic / Description:

Summarizes feedback from students on instructors and programs.

Benefits:

- Improves teaching based on feedback.
- Monitors satisfaction by track and intake.

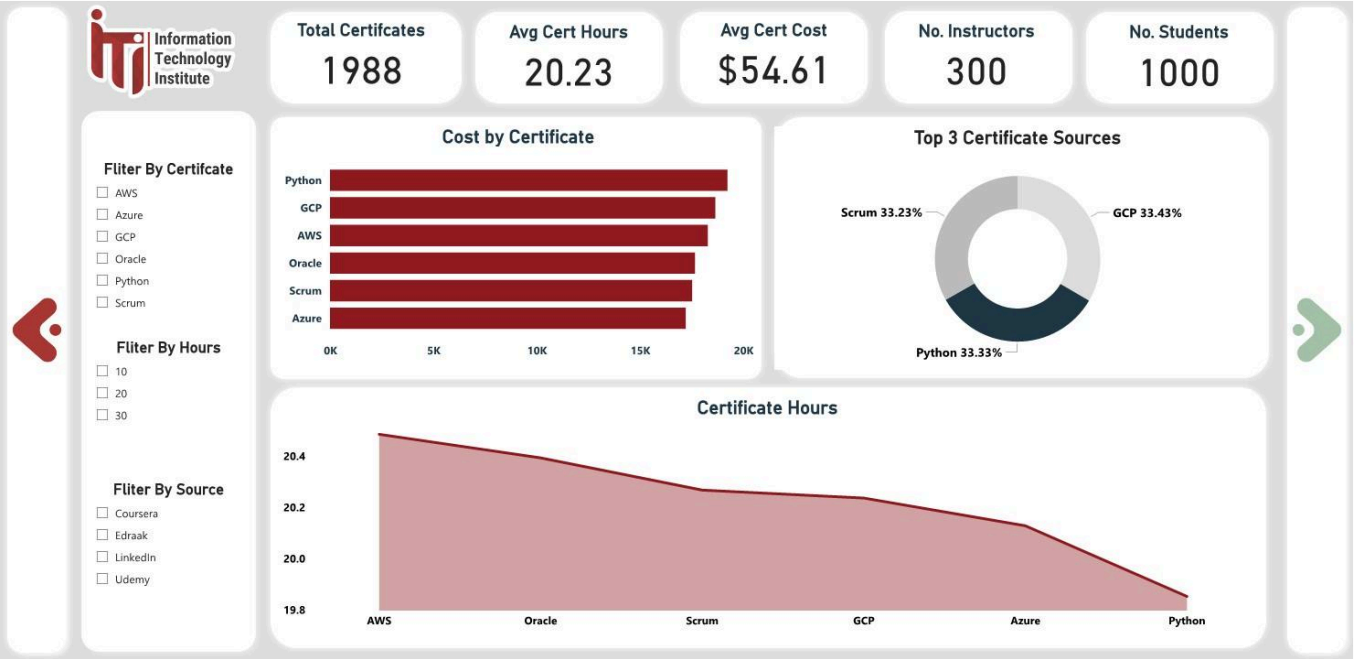
Cards and Their Benefits:

- **Avg Feedback:** Measures general satisfaction.
- **Total Feedbacks:** Tracks response volume.
- **Highest Rated Instructor:** Spotlights best-performing staff.

Charts and Their Benefits:

- **Feedback by Track (Bar):** Links satisfaction to curriculum.
- **Feedback by Instructor (Column):** Compares teaching effectiveness.
- **Feedback over Time (Line):** Monitors evolving trends.

7. Certification Insights



Topic / Description:

Monitors certification programs in terms of cost, hours, and participation.

Benefits:

- Budgeting and ROI analysis.
- Adjust certificate offerings.

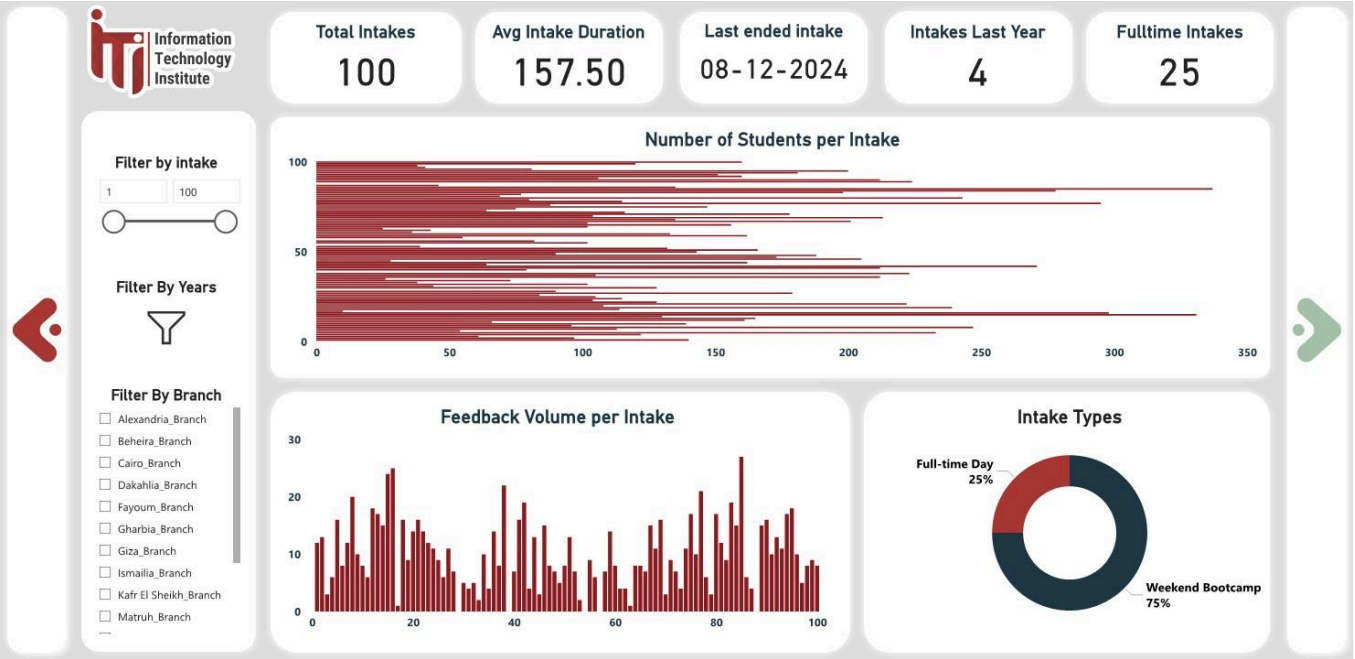
Cards and Their Benefits:

- **Total Certificates:** Measures certification reach.
- **Avg Certificate Hours:** Understand workload.
- **Avg Certificate Cost:** Evaluate financial burden.

Charts and Their Benefits:

- **Certificate Sources (Donut):** Categorize certifications.
- **Hours by Certificate (Line):** Evaluate intensity.
- **Cost by Certificate (Column):** Understand pricing.

8. Intake Analysis



Topic / Description:

Visualizes student intake trends, types, and durations.

Benefits:

- Plan recruitment and scheduling.
- Track historical growth.

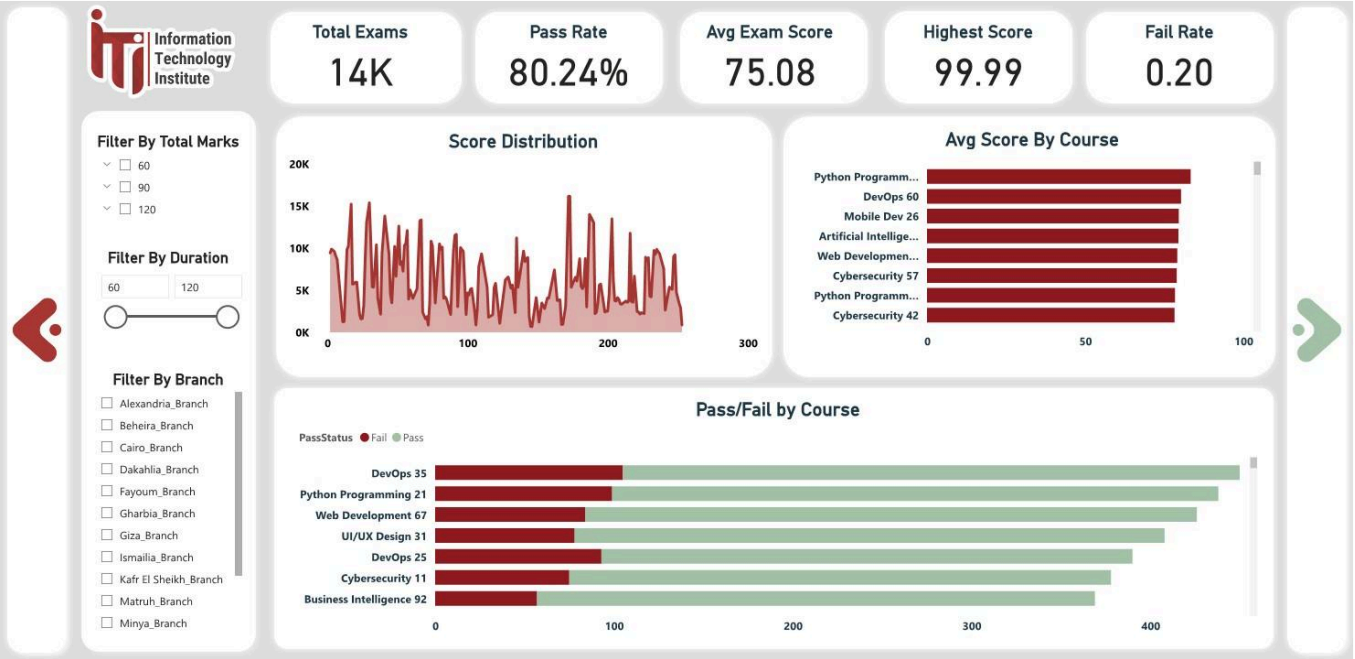
Cards and Their Benefits:

- **Total Intakes:** Measures frequency.
- **Avg Intake Duration:** Evaluate length and effort.
- **Intakes This Year:** Tracks current activity.

Charts and Their Benefits:

- **Students by Intake (Bar):** Compare cohort sizes.
- **Intake Timeline (Line):** View historical flow.
- **Intake Type (Donut):** Categorize intake structure.

9. Exam Performance Dashboard



Topic / Description:

Analyzes student exam scores and pass rates.

Benefits:

- Improve academic assessment.
- Detect weak areas in teaching.

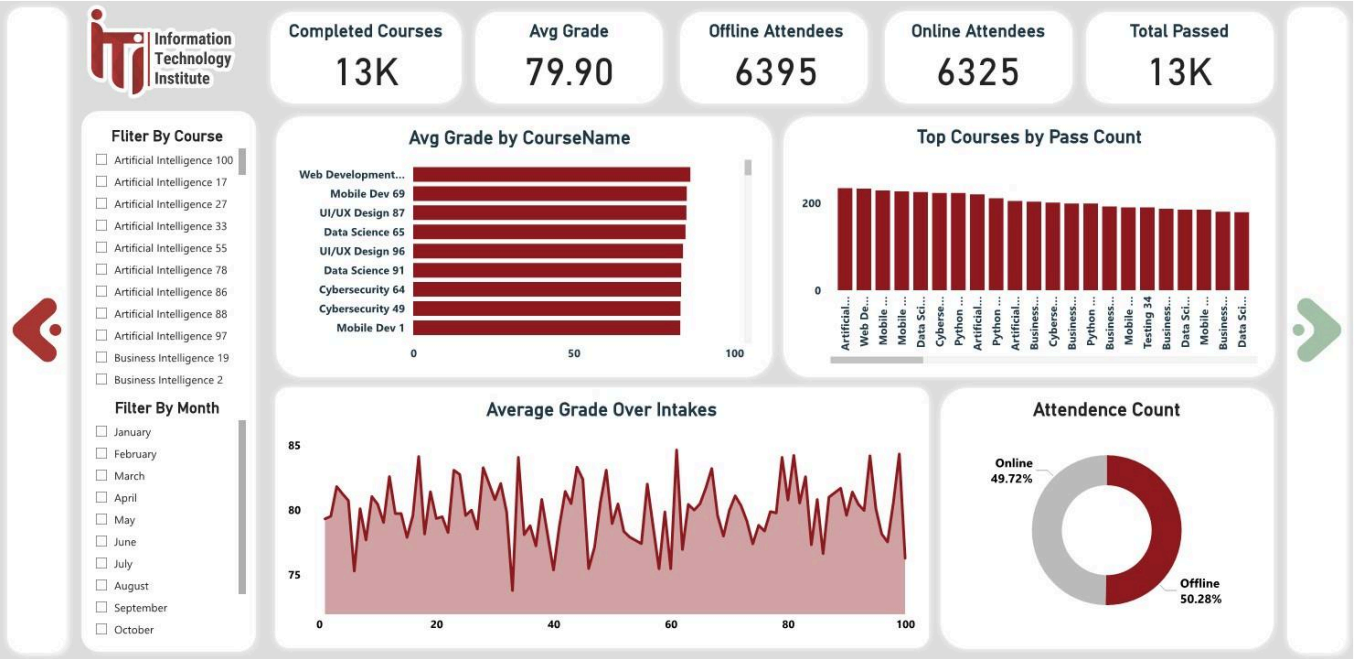
Cards and Their Benefits:

- **Avg Exam Score:** Overall academic level.
- **Total Exams:** Volume of testing.
- **Pass Rate:** Academic achievement ratio.

Charts and Their Benefits:

- **Score Distribution (Line):** Shows performance curve.
- **Avg Score by Course (Bar):** Evaluates subject difficulty.
- **Pass/Fail by Course (Stacked Column):** Pinpoints failing areas.

10. Course Completion Dashboard



Topic / Description:

Measures course outcomes and attendance.

Benefits:

- Academic success tracking.
- Identify course completion trends.

Cards and Their Benefits:

- **Avg Grade:** Overall academic result.
- **Total Completed:** Student throughput.
- **Attendance Present:** Participation indicator.

Charts and Their Benefits:

- **Grade by Course (Column):** Spot grade variations.
- **Attendance Distribution (Donut):** Monitor engagement.
- **Completion Rate by Track (Bar):** Track success by program.

11. Student Certificate Summary

Topic / Description:

Analyzes certificate ownership and student-related costs.

Benefits:

- Track certification impact.
- Estimate student investment.

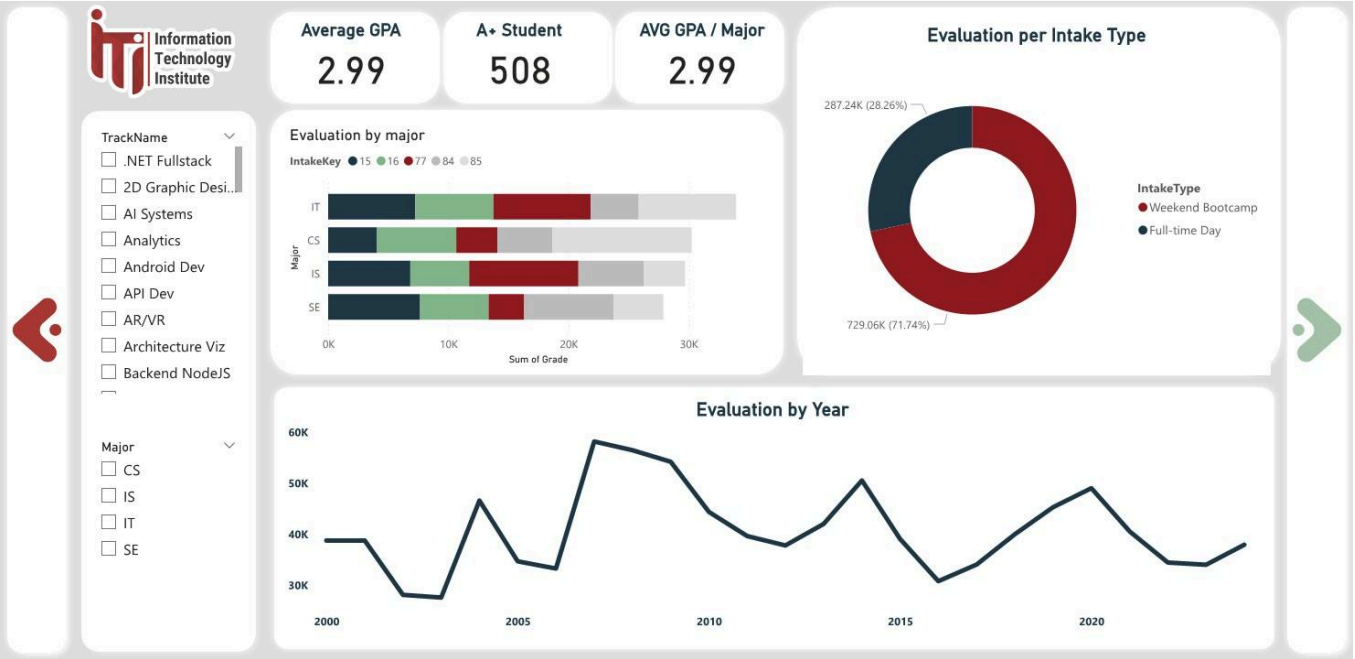
Cards and Their Benefits:

- **Students with Certificates:** Measures achievement rate.
- **Avg Certificates per Student:** Tracks student progress.
- **Avg Cost per Student:** Analyzes expense per learner.

Charts and Their Benefits:

- **Certificates by Student (Bar):** Visualize progress.
- **Certificate Cost Distribution (Histogram):** Understand cost spread.
- **Hours vs Cost (Scatter):** Balance effort and price.

12. GPA Progression Over Intakes



Topic / Description:

Tracks GPA changes over different intakes and types.

Benefits:

- Academic quality monitoring.
- Identify growth or decline trends.

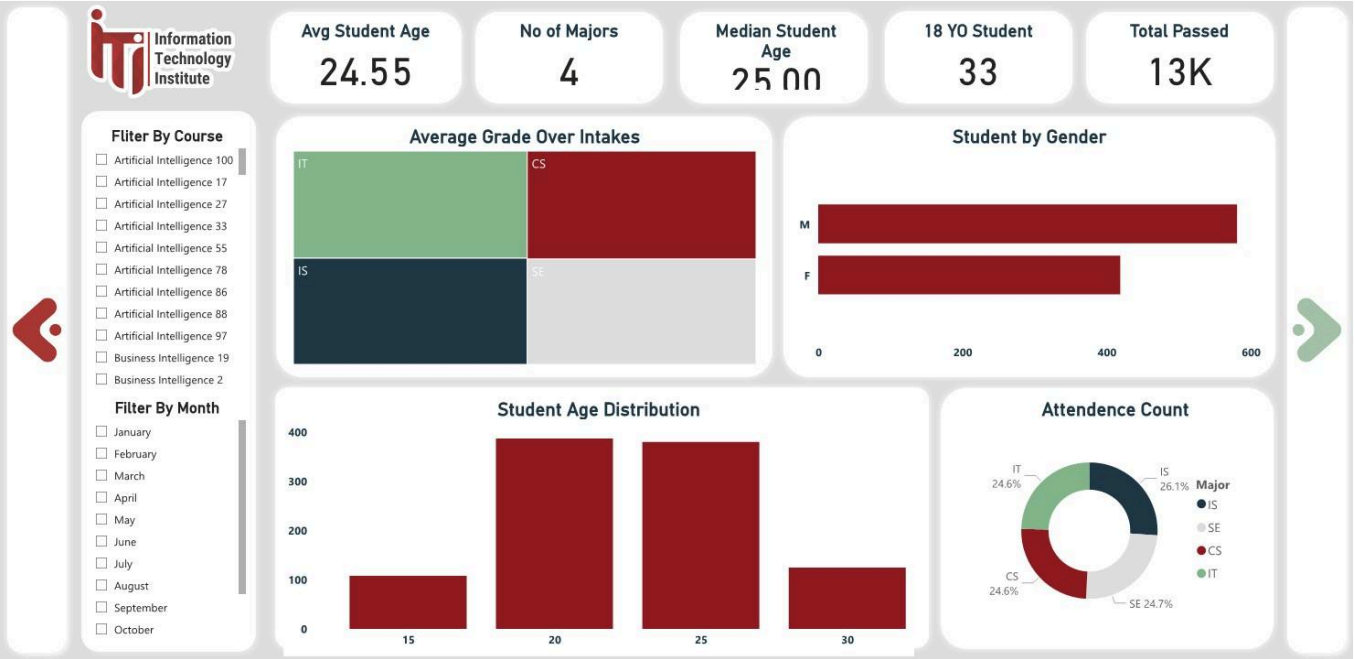
Cards and Their Benefits:

- **Avg GPA (Latest Intake):** Most recent performance.
- **GPA Improvement Rate:** Compare change over time.
- **Students GPA > 3.5:** Recognize high achievers.

Charts and Their Benefits:

- **GPA over Intakes (Line):** Shows performance trend.
- **GPA by Intake Type (Donut):** Compare formats.
- **GPA by Major over Intakes (Stacked Column):** Historical academic map.

13. Student Demographics Dashboard



Topic / Description:

Breaks down students by age, gender, and major.

Benefits:

- Support demographic analysis.
- Inform diversity and inclusion efforts.

Cards and Their Benefits:

- **Avg Age:** Shows age trends.
- **Students per Major:** Distribution across programs.
- **Gender Ratio:** Visualizes diversity.

Charts and Their Benefits:

- **Age Distribution (Histogram):** Track age bands.
- **Students by Major (Pie):** Program popularity.
- **Gender by Major (Stacked Bar):** Diversity by program.

14. Certificate Revenue Dashboard

Topic / Description:

Analyzes revenue generated from certificate programs.

Benefits:

- Financial planning.
- Measure certificate profitability.

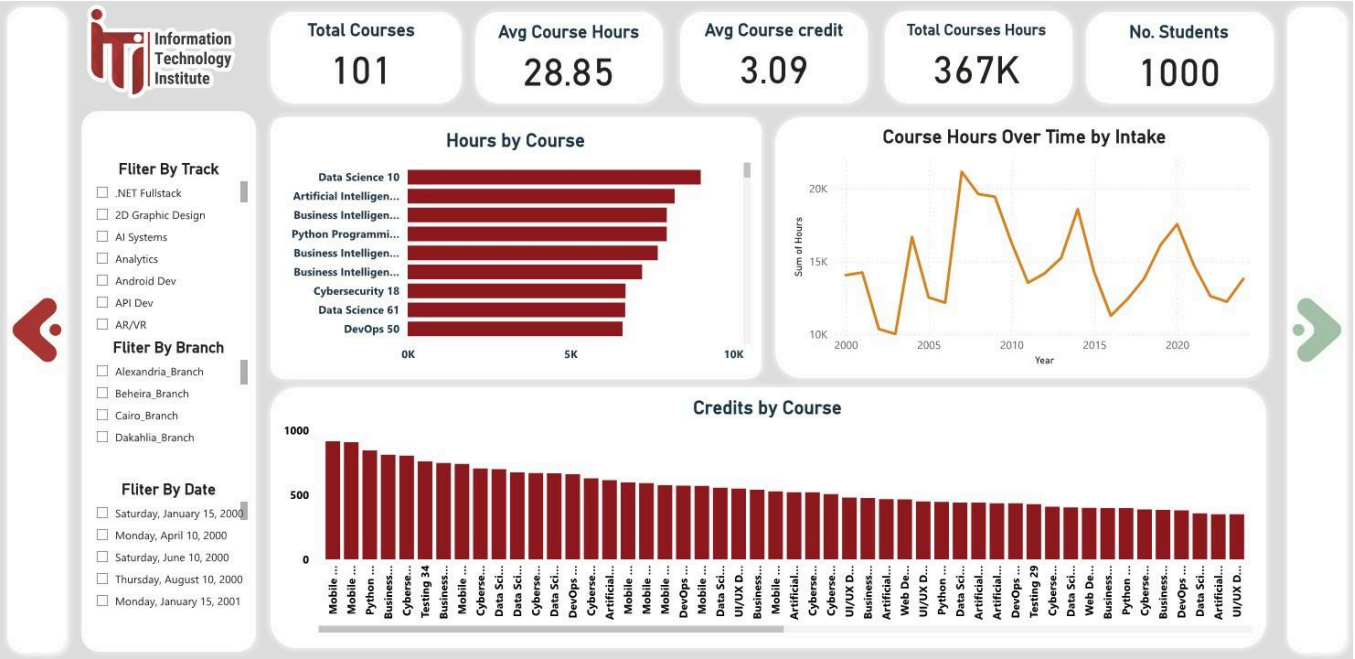
Cards and Their Benefits:

- **Total Certificate Revenue:** Total income.
- **Avg Revenue per Certificate:** Efficiency measure.
- **Revenue by Branch:** Contribution by location.

Charts and Their Benefits:

- **Revenue by Branch (Column):** Income per branch.
- **Certificate Source (Treemap):** source distribution .
- **Top 5 Certificates By Cost (Bar):** Financial efficiency.

15. Course Duration & Credits Dashboard



Topic / Description:

Monitors workload and credit distribution across courses.

Benefits:

- Assess course difficulty.
- Guide curriculum balance.

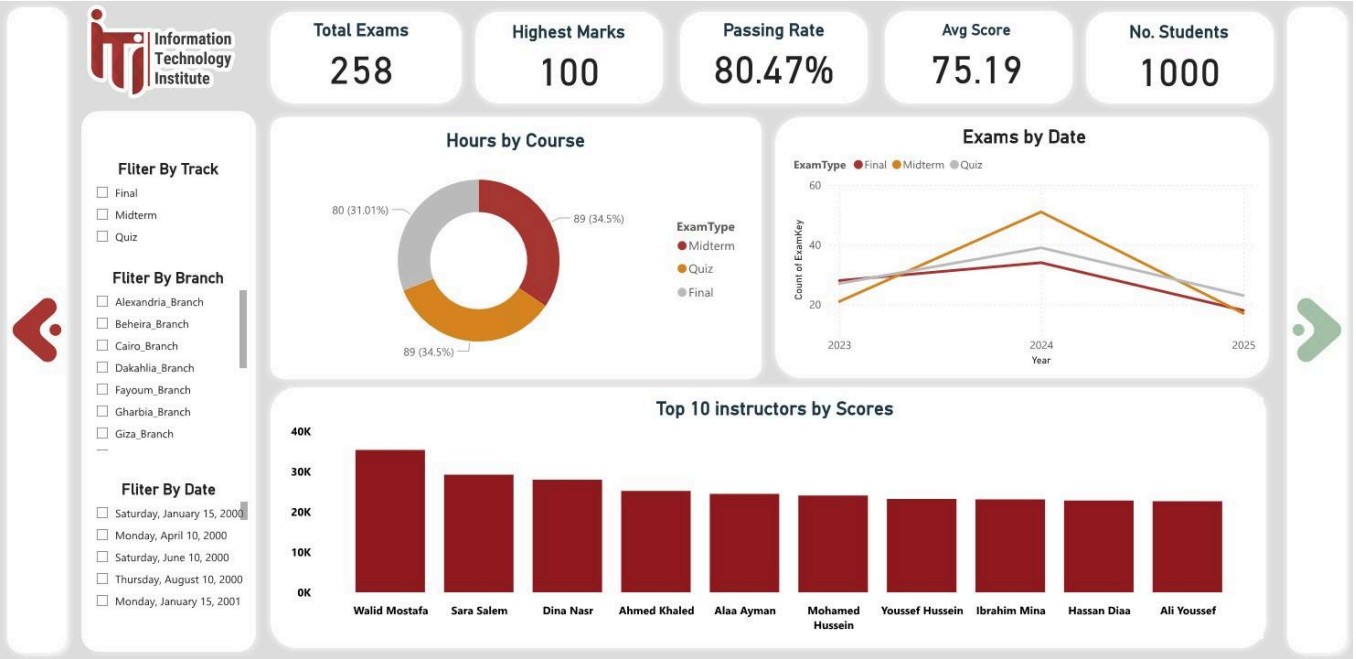
Cards and Their Benefits:

- **Avg Course Hours:** Average duration.
- **Avg Course Credits:** Academic value.
- **Total Course Hours:** Total student effort.

Charts and Their Benefits:

- **Hours by Course (Bar):** Duration by course.
- **Credits by Course (Column):** Compare credit weight.
- **Hours Trend By Intake (Line):** Analyzes hours trend in each intake .

16. Exam Scheduling Dashboard



Topic / Description:

Visualizes upcoming and past exam details.

Benefits:

- Assist scheduling and planning.
- Track exam intensity.

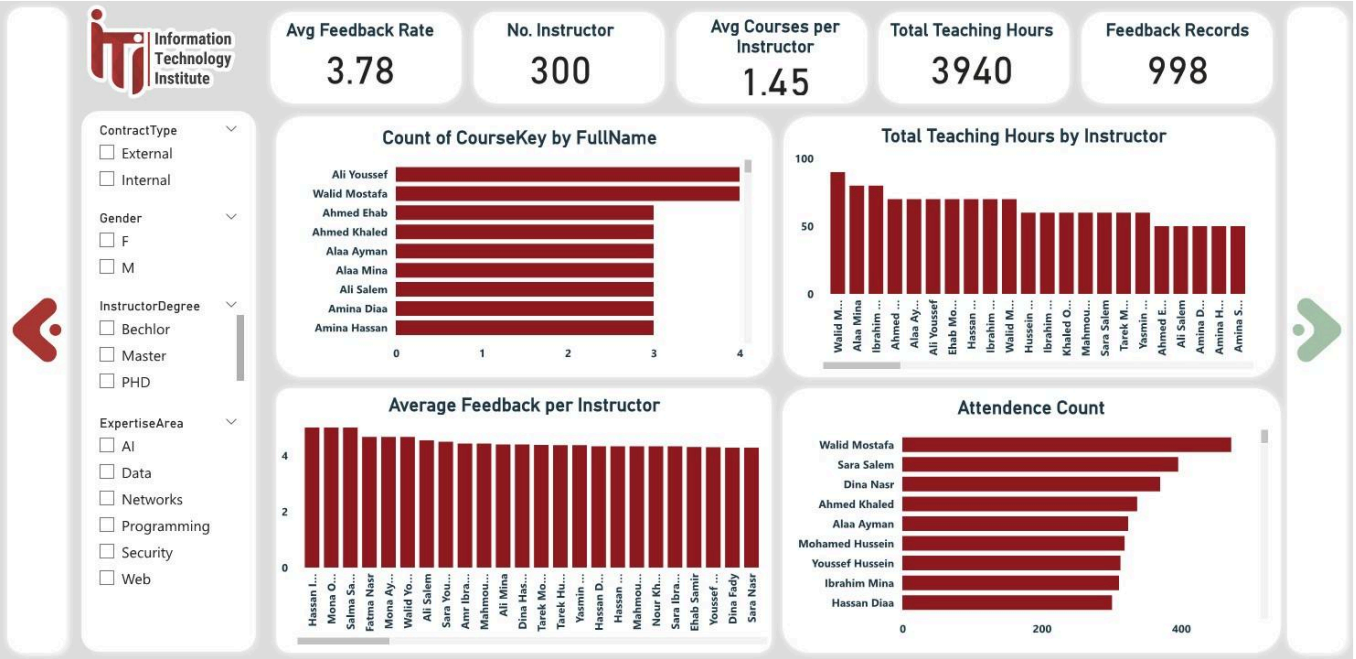
Cards and Their Benefits:

- **Upcoming Exams:** Future workload.
- **Avg Duration:** Time allocation analysis.
- **Total Marks:** Assessment weight.

Charts and Their Benefits:

- **Exams by Date (Timeline):** Scheduling map.
- **Exam Type (Donut):** Format comparison between exam types.
- **Top 10 instructors by exam score (Column):** Instructor efficiency indicator.

17. Instructor Load Dashboard



Topic / Description:

Tracks instructor workload and involvement in courses and exams.

Benefits:

- Ensure fair distribution of tasks.
- Evaluate staff effort.

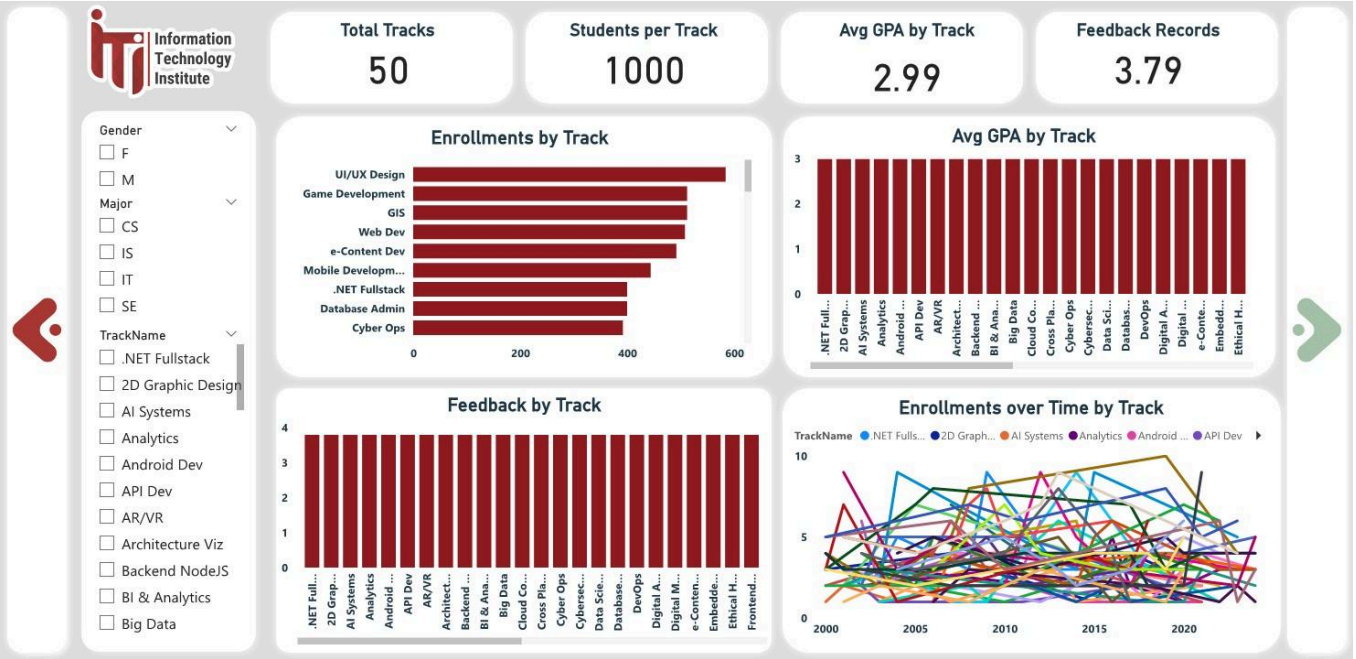
Cards and Their Benefits:

- **Courses per Instructor:** Teaching load.
- **Feedback per Instructor:** Teaching quality.
- **Exam Supervisions:** Monitoring participation.

Charts and Their Benefits:

- **Courses by Instructor (Bar):** Engagement levels.
- **Feedback by Instructor (Column):** Quality feedback.
- **Supervised Exams by Instructor (Bar):** Extra duties.

18. Track Popularity Dashboard



Topic / Description:

Evaluates track selection and student satisfaction.

Benefits:

- Support program offering decisions.
- Improve underperforming tracks.

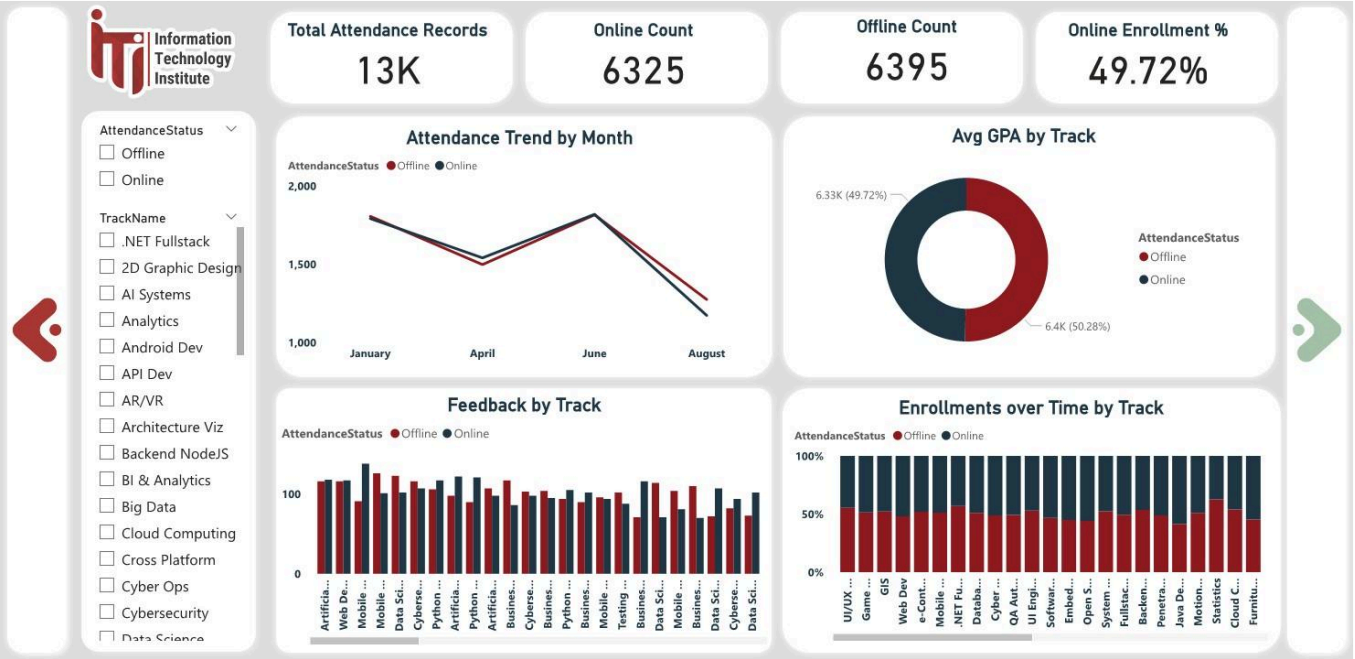
Cards and Their Benefits:

- **Students per Track:** Popularity indicator.
- **Avg Track Feedback:** Satisfaction measure.
- **Tracks Offered:** Curriculum variety.

Charts and Their Benefits:

- **Enrollments by Track (Bar):** Popularity map.
- **Feedback by Track (Column):** Quality analysis.
- **GPA by Track (column):** Academic outcome.
- **Enrollments Trend by Track (Line):** Popularity Trend map.

19. Attendance Trends Dashboard



Topic / Description:

Tracks attendance patterns and participation.

Benefits:

- Improve engagement.
- Spot absenteeism issues.

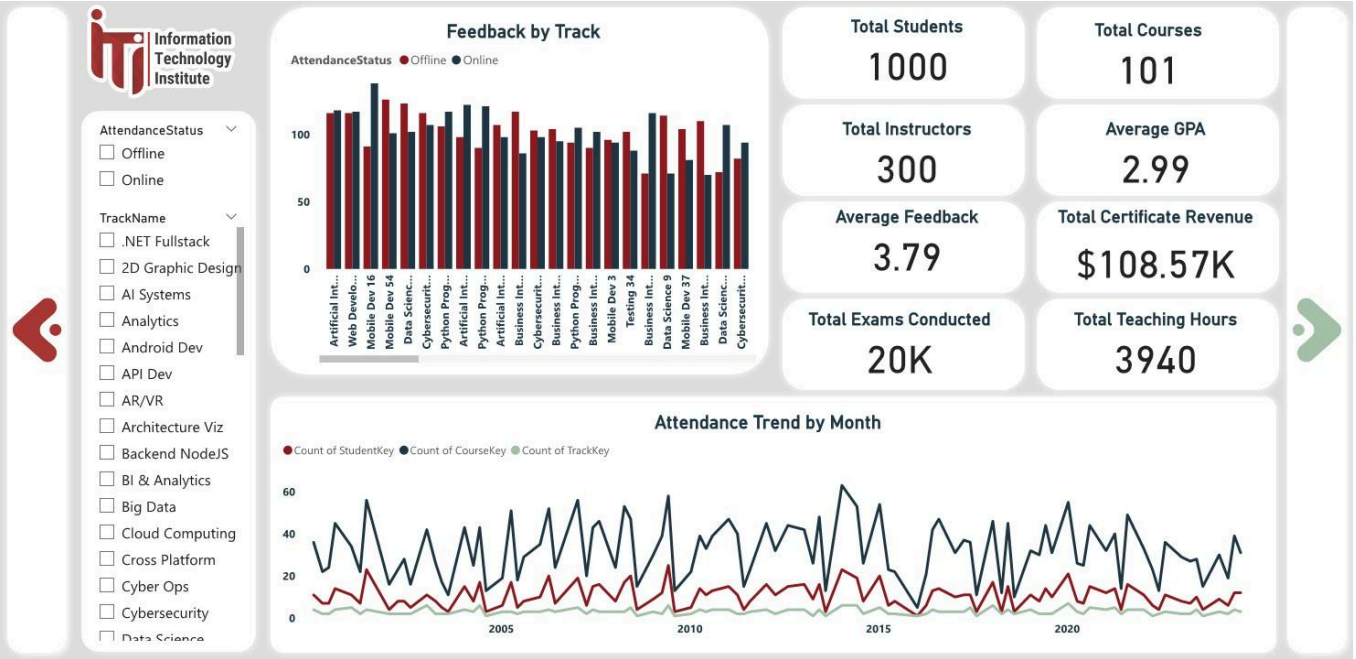
Cards and Their Benefits:

- **Present Count:** Measure of engagement.
- **Absent Count:** Risk indicator.
- **Total Attendance Records:** Dataset size.

Charts and Their Benefits:

- **Attendance Trend by Month (Line):** Monthly insights.
- **Attendance Status Distribution (Donut):** Summary view.
- **Attendance by Course (Column):** Course-level attendance.

20. Summary KPI Dashboard



Topic / Description:

Provides a quick snapshot of all major KPIs in the educational system.

Benefits:

- Supports strategic monitoring.
- One-stop summary for management.

Cards and Their Benefits:

- **Total Students, Courses, Instructors**: Key volume indicators.
- **Avg GPA, Avg Feedback, Total Revenue**: Core performance measures.

Charts and Their Benefits:

- **KPI Cards Grouped Visually:** Quick dashboard snapshot.
- **Overall Trends (Line):** Time-based changes.
- **Category-wise Contribution (Stacked Column):** Breakdown by type.

Implementation Strategy

The successful implementation of this comprehensive Power BI dashboard system requires a structured approach that considers data governance, user adoption, and ongoing maintenance. The implementation should follow a phased approach, beginning with foundational dashboards and progressively adding more specialized analytics.

Critical success factors include ensuring data quality and consistency across all sources, establishing clear data refresh schedules, and providing comprehensive training to end users. The system should be designed with scalability in mind, allowing for future expansion and integration with additional data sources.

Data Integration and Governance

The dashboard system relies on comprehensive data integration from multiple sources including student information systems, learning management systems, financial systems, and external certification platforms. A robust data governance framework ensures data accuracy, consistency, and security throughout the analytics pipeline.

Data quality monitoring and validation procedures are essential to maintain the integrity of insights generated by the dashboard system. Regular data audits and validation checks should be implemented to identify and resolve any data quality issues that may impact decision-making.

User Adoption and Training

Successful adoption of the dashboard system requires comprehensive training programs tailored to different user groups. Administrators, faculty, and support staff each have unique analytical needs and require specific training on relevant dashboards and features.

Change management strategies should be implemented to encourage adoption and ensure users understand the value proposition of data-driven decision making. Regular feedback sessions and user support mechanisms help maintain engagement and identify areas for improvement.

Conclusion

This comprehensive Power BI dashboard system transforms raw educational data into actionable insights, enabling data-driven decision making at all levels of the

institution. The 20 specialized dashboards provide stakeholders with the tools needed to monitor performance, identify opportunities for improvement, and make strategic decisions that enhance educational quality and student success.

The system's modular design allows for flexible implementation and scalability, while comprehensive analytics coverage ensures that all critical aspects of institutional performance are monitored and analyzed. Through effective implementation and ongoing maintenance, this dashboard system will serve as a cornerstone for institutional excellence and continuous improvement.

The investment in this analytics infrastructure will yield significant returns through improved operational efficiency, enhanced student outcomes, and more effective resource allocation. As the educational landscape continues to evolve, this robust analytical foundation will enable the institution to adapt and thrive in an increasingly data-driven environment.

Frontend Application Documentation (Flet.dev)

Project Overview

This frontend application is developed using **Flet.dev**, a Python framework that allows building real-time interactive web, desktop, and mobile apps using only Python. The frontend interacts with a backend API to perform operations such as user authentication, data submission, and UI updates.

Technologies Used

Tool	Purpose
Flet.dev	Python-based UI framework
Python	Main programming language
HTTP (via `requests`)	API communication
Node.js/FastAPI (optional)	Backend services

Application Structure

```
frontend/
│
├── main.py              # Entry point for Flet app
├── components/          # Custom UI components (optional)
│   ├── navbar.py
│   └── forms.py
├── services/
│   └── api_client.py    # Handles API communication
└── assets/              # Icons, images, etc.
```

UI Design Principles

- **Responsive Layout** using `Row`, `Column`, and `Container`.
- **Modular UI Elements**: Each screen or feature is split into reusable components.
- **User Feedback**: Real-time feedback through `SnackBar`, `Text`, or `Dialog`.
- **Routing**: Managed using `page.go()` for multi-page navigation (e.g., login, dashboard).

- **State Management**: Done using Flet's control references and update mechanism.

Key Functionalities

1. Login Screen

- Input fields for username and password using `TextField`.
- Form validation before submission.
- API request to backend for authentication using `requests.post`.
- Success response triggers a route change to the dashboard.

2. Admin Dashboard

- Role-based access: shows different options for admin vs instructor vs student.
- Dynamic form to create users (username, email, role, password).
- Form submission triggers POST request to `/users/create`.

3. Feedback and Navigation

- Real-time updates using `page.update()`.
- Navigation between views using `ft.View` and `page.views`.

API Integration

- All data operations are routed through a `services/api_client.py` module.
- Functions:
    - `login_user(username, password)`
    - `create_user(user_data)`

- Uses `requests` library for HTTP communication.

Example:

```
def login_user(username, password):
    response = requests.post(
        f"{BASE_URL}/login",
        json={"username": username, "password": password}
    )
    return response
```

Sample Code Snippet

```
def on_login_click(e):
    username = username_input.value
    password = password_input.value

    if not username or not password:
        error_text.value = "Fields cannot be empty"
        page.update()
        return

    response = login_user(username, password)
    if response.status_code == 200:
        page.go("/dashboard")
    else:
        error_text.value = "Login failed"
        page.update()
```

Deployment

- Run with: `flet run main.py`
- Can be packaged as a:
    - **Web App** (via Flet web server)
    - **Desktop App** (via Flet standalone app)
    - **Mobile App** (via Flutter bridge - future scope)

Testing and Validation

- Manual UI testing for button clicks, form validation, and layout consistency.
- API responses tested using mocked endpoints and actual backend.

Future Improvements

- Add client-side routing state (breadcrumbs, login tokens).
- Integrate charts/graphs using `plotly` with Flet canvas.
- Add themes and user settings.

References

- Flet.dev Documentation
- Flet GitHub Examples
- REST API documentation (OpenAPI/Swagger)

Backend Project Documentation

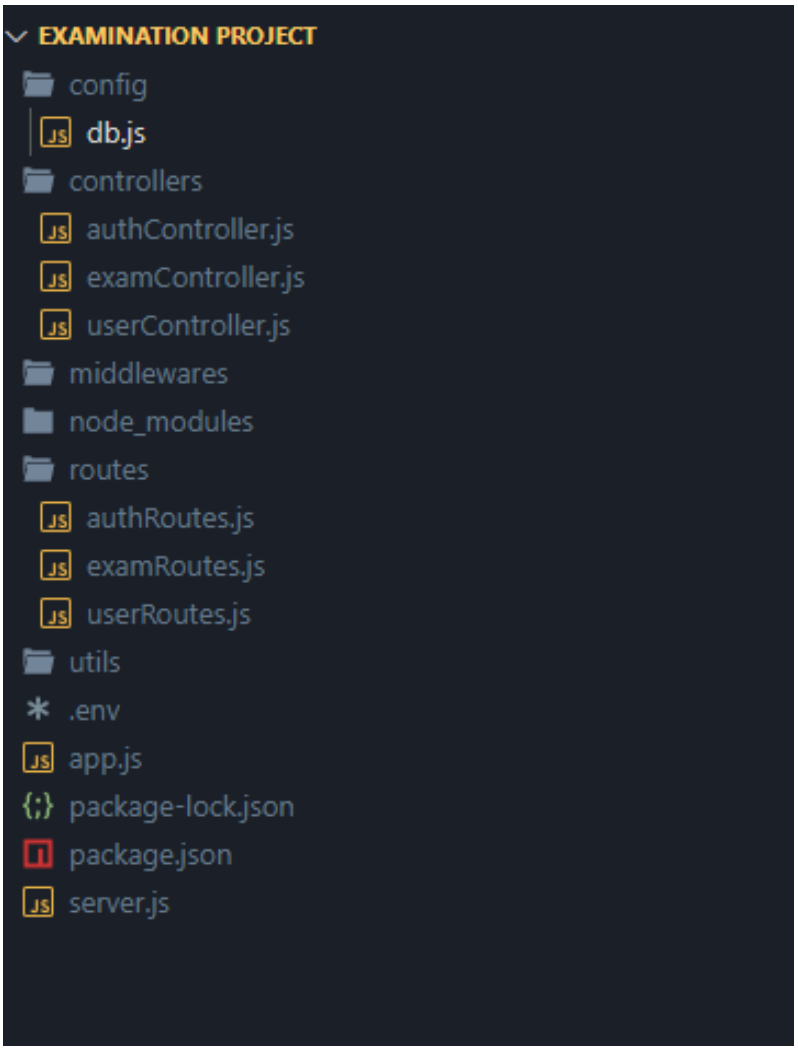
This document provides comprehensive documentation for the backend application, covering its structure, setup, API endpoints, and key functionalities.

1. Introduction

This project serves as the backend for an "Examination Project," likely handling user authentication, exam management, and general user data. It is built using Node.js and Express.js.

2. Project Structure

The project follows a modular and organized structure, separating concerns into distinct directories:



3. Setup & Installation

To set up and run the backend locally, follow these steps:

Prerequisites

- Node.js (LTS version recommended)
- npm (Node Package Manager)
- SQL Database

Installation Steps

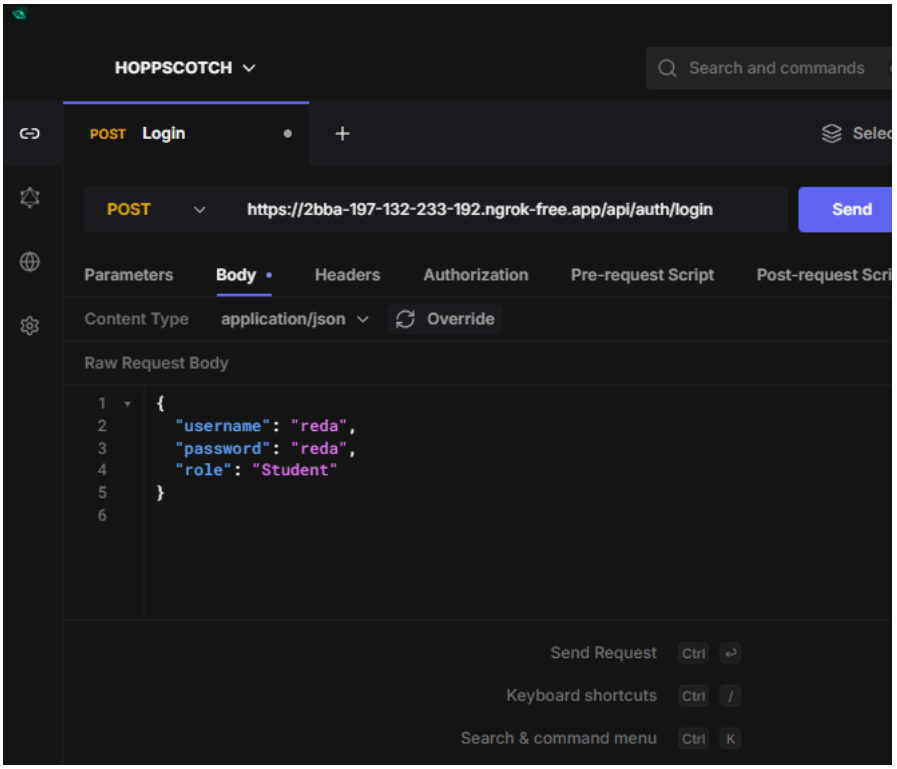
1. **Clone the repository**
2. **Install dependencies**
3. **Configure Environment Variables**
4. **Start the server**

4. API Endpoints

This section outlines the available API endpoints, their HTTP methods, and expected request/response formats.

4.1. Authentication Routes (`/api/auth`)

- **`POST /api/auth/register`**
    - **Description:** Registers a new user.
    - **Request Body:**



4.2. User Routes (`/api/users`)

- **`POST /api/users/add-user`**
  - **Description:** Adds a new user with a specified role (Student, Instructor, or Admin) to the system. This endpoint uses a stored procedure `AddNewUserWithRole` to handle the insertion and role-specific data.

Error Responses:

- `400 Bad Request`:
  - `{ "error": "❌ Username already exists." }`
  - `{ "error": "❌ Email already exists." }`
  - `{ "error": "❌ IntakeID is not valid." }`
  - `{ "error": "❌ TrackID is not valid." }`
  - `{ "error": "❌ Invalid role. Must be Student, Instructor, or Admin." }`
  - `{ "error": "❌ One or more required fields are missing." }` (for Cannot insert the value NULL errors)
- `500 Internal Server Error`:
  - `{ "error": "❌ Server error: [error message]" }`

4.3. Exam Routes (`/api/exams`)

- **`POST /api/exams`**
    - **Description:** Creates a new exam. (Likely for admin/teacher roles)
    - **Authentication:** Requires JWT token (and possibly role-based authorization).

Description: Retrieves a list of available exams.
Authentication: Requires JWT token.

5. Database Schema

SQL Server Configuration (`config/db.js`)

The `db.js` file configures the connection to a SQL Server database using the `mssql` package.

- **Database Type:** SQL Server
- **Connection Details:**
    - `user`: `'reda'`
    - `password`: `'reda13001322'`
    - `server`: `'localhost'`
    - `port`: `1433`
    - `database`: `'LearningManagementSystem'`
    - `options`:
        - `encrypt`: `false` (for local development, consider `true` for Azure SQL Database)
        - `trustServerCertificate`: `true` (required for self-signed certificates in development)

Collections/Tables

- **Users:**
    - This table stores information about all users (Students, Instructors, and Admins). It uses a common set of fields and then specific fields for each role, with `NULL` values for fields not applicable to a user's role.
    - `_id` (or `id`): Primary Key (likely auto-incrementing `INT` in SQL Server)
    - `Username`: `VARCHAR(30)`, Unique
    - `FirstName`: `VARCHAR(30)`
    - `LastName`: `VARCHAR(30)`
    - `Salary`: `DECIMAL(10, 2)`
    - `Email`: `VARCHAR(100)`, Unique
    - `DateOfBirth`: `DATE`
    - `Gender`: `CHAR(1)`
    - `PasswordHash`: `VARCHAR(255)`
    - `BranchID`: `INT`
    - `Address`: `VARCHAR(200)`
    - `Role`: `VARCHAR(30)` (e.g., 'Student', 'Instructor', 'Admin')
    - **Student Specific:**

        - `Major`: `VARCHAR(100)` (NULLable)
        - `GPA`: `DECIMAL(3, 2)` (NULLable)
        - `IntakeID`: `VARCHAR(20)` (NULLable)
        - `TrackID`: `INT` (NULLable)
    - **Instructor Specific:**
        - `ExpertiseArea`: `VARCHAR(100)` (NULLable)
        - `InstructorHiringDate`: `DATE` (NULLable)
        - `ContractType`: `VARCHAR(20)` (NULLable)
        - `InstructorDegree`: `VARCHAR(20)` (NULLable)
    - **Admin Specific:**
        - `AdminPosition`: `VARCHAR(50)` (NULLable)
        - `AdminHiringDate`: `DATE` (NULLable)
    - `createdAt`: Date (Assumed, for tracking creation time)
    - `updatedAt`: Date (Assumed, for tracking last update time)
- **Exams:**
    - `_id` (or `id`): Primary Key
    - `title`: String
    - `description`: String
    - `duration`: Number (minutes)
    - `questions`: Array of Objects (each object containing `questionText`, `options`, `correctAnswer`)
    - `createdAt`: Date
    - `updatedAt`: Date
- **Submissions (or Results):**
    - `_id` (or `id`): Primary Key
    - `userId`: Reference to User
    - `examId`: Reference to Exam
    - `answers`: Array of Objects (each object containing `questionId`, `selectedAnswer`)
    - `score`: Number
    - `submittedAt`: Date
    - `isCompleted`: Boolean
- *Add any other collections/tables and their fields.*

Stored Procedures

- **`GetExamResults`**
    - **Description:** Retrieves exam results for a given student.
    - **Parameters:**
        - `@StudentID`: `INT` - The ID of the student.
    - **Returns:** A result set containing exam details, score, submission date, and student information. (The exact columns depend on the stored procedure's definition in SQL Server).
- **`AddNewUserWithRole`**

- **Description:** A stored procedure used to insert a new user into the database, handling common user fields and role-specific fields (Student, Instructor, Admin). It likely contains logic to validate inputs and insert data into appropriate tables based on the `Role` provided.
- **Parameters:**
  - `@Username`: `VARCHAR(30)`
  - `@FirstName`: `VARCHAR(30)`
  - `@LastName`: `VARCHAR(30)`
  - `@Salary`: `DECIMAL(10, 2)`
  - `@Email`: `VARCHAR(100)`
  - `@DateOfBirth`: `DATE`
  - `@Gender`: `CHAR(1)`
  - `@PasswordHash`: `VARCHAR(255)`
  - `@BranchID`: `INT`
  - `@Address`: `VARCHAR(200)`
  - `@Role`: `VARCHAR(30)` (Expected values: 'Student', 'Instructor', 'Admin')
  - `@Major`: `VARCHAR(100)` (Can be `NULL`)
  - `@GPA`: `DECIMAL(3, 2)` (Can be `NULL`)
  - `@IntakeID`: `VARCHAR(20)` (Can be `NULL`)
  - `@TrackID`: `INT` (Can be `NULL`)
  - `@ExpertiseArea`: `VARCHAR(100)` (Can be `NULL`)
  - `@InstructorHiringDate`: `DATE` (Can be `NULL`)
  - `@ContractType`: `VARCHAR(20)` (Can be `NULL`)
  - `@InstructorDegree`: `VARCHAR(20)` (Can be `NULL`)
  - `@AdminPosition`: `VARCHAR(50)` (Can be `NULL`)
  - `@AdminHiringDate`: `DATE` (Can be `NULL`)
- **Behavior:** The stored procedure handles the insertion logic and returns success or specific error messages (e.g., "Username already exists", "Email already exists", "Invalid IntakeID", "Invalid TrackID", "Invalid role provided").

6. Authentication & Authorization

Please describe your authentication and authorization mechanisms here.

- **Authentication Method:** (e.g., JWT - JSON Web Tokens)
- **JWT Flow:**
  - User logs in with credentials.
  - Server verifies credentials and generates a JWT.
  - JWT is sent to the client.
  - Client stores the JWT (e.g., in `localStorage`).
  - For protected routes, the client sends the JWT in the `Authorization` header (`Bearer <token>`).
  - The `authMiddleware.js` (or similar) verifies the JWT.
- **Authorization (Roles):**

    - Are there different user roles (e.g., student, teacher, admin)?
    - How are roles enforced on specific routes (e.g., only teachers can create exams)?

7. Environment Variables

The application uses environment variables for sensitive information and configuration. These are loaded from the `.env` file.

- `PORT`: The port on which the server will run (e.g., `3000`).
- `DATABASE_URL`: The connection string for your database.
    - *Note: Currently, database credentials are hardcoded in `config/db.js`. It is highly recommended to move these sensitive details to environment variables for security and flexibility.*
- `JWT_SECRET`: A secret key used to sign and verify JWTs.
- *List all other environment variables used in your project with a brief description.*

8. Dependencies

The project's dependencies are listed in `package.json`. Key dependencies likely include:

- `express`: Web framework for Node.js.
- `mssql`: Microsoft SQL Server client for Node.js.
- `bcryptjs`: For hashing passwords.
- `jsonwebtoken`: For implementing JWTs.
- `dotenv`: For loading environment variables from a `.env` file.
- `cors`: For enabling Cross-Origin Resource Sharing.
- *List all other significant dependencies and their purpose.*

9. Application Setup (`app.js`)

The `app.js` file is the core Express application setup. It initializes the Express app, applies global middleware, and mounts the API routes.

- **Middleware:**
    - `cors()`: Enables Cross-Origin Resource Sharing for all routes, allowing requests from different origins.
    - `express.json()`: Parses incoming requests with JSON payloads. This middleware is essential for handling JSON data sent in POST and PUT requests.
- **Route Integration:**
    - `app.use('/api/auth', authRoutes);`: Mounts the authentication routes under the `/api/auth` base path.
    - `app.use('/api/users', userRoutes);`: Mounts the user management routes under the `/api/users` base path.

- **Root Endpoint:**
    - `app.get('/', (req, res) => { res.send('API is running...'); });`: A simple root endpoint to confirm that the API server is running.

10. Server Entry Point (`server.js`)

The `server.js` file is the entry point for starting the Node.js Express application.

- It imports the configured Express `app` from `app.js`.
- It defines the port on which the server will listen, prioritizing the `PORT` environment variable and defaulting to `5000` if not set.
- It starts the server, making it accessible at the specified port.