SQL-Driven Training System For Guardian Group Services

By Cesar Suriel-Luna

Introduction

Guardian Group Services is a security guard training school based in Brooklyn that offers a wide range of certification and renewal courses. The school provides both mandatory New York State security guard trainings: the 8 Hour Pre-Assignment, 16 Hour On-the-Job Training, and 8 Hour Annual In-Service, as well as additional courses including firearms certifications, CPR and first aid, FDNY certificate prep courses, and various OSHA safety trainings. Classes are held virtually over Zoom, in-person, or asynchronously online, depending on the course type.

With a packed weekly schedule, sometimes offering up to five classes per day, Guardian Group Services trains thousands of students each year. Students come either independently or through workforce development agencies and corporate sponsorships. Course offerings operate in cohorts with enrollment limits between 20–35 students per session.

The current system of record keeping is based on multiple interconnected Google Sheets. Each course has its own attendance sheet where staff manually input student information such as name, contact details, agency affiliation, package type, and payment amounts. Separate booking and intake forms populate into additional sheets, while security guard training classes require detailed rosters to be submitted to the New York State Division of Criminal Justice Services (DCJS). These rosters are manually created for each class and must include information like date of birth, last four digits of the student's SSN, and agreement to DCJS guidelines. While functional, this system is highly manual, error-prone, and difficult to scale. With over 56,000 rows and growing, the use of spreadsheets for core operations presents a significant opportunity for improvement through database design.

Objective

The primary goal of this database project is to replace the current spreadsheet-based workflow at Guardian Group Services with a centralized system that can more efficiently manage student records, training schedules, and administrative tasks. The new system aims to reduce the amount of manual data entry, improve roster generation for state reporting requirements, and create a clearer overview of student course history and package enrollment.

This system is designed for use by office administrators and management. It will serve as the central recordkeeping solution to track student progress through different courses, assign students to classes based on their bookings, and store student data used in creating and submitting state rosters. One major issue this solution will address is the lack of user accountability in the current workflow. All staff share access to the same spreadsheets which makes it difficult to identify who made changes or errors. This has led to recurring mistakes,

finger-pointing, and a decline in morale. By implementing a database-backed system with secure user roles and auditing abilities for management, the organization will be able to retrain staff when needed and promote better collaboration and trust. While no front-end interface will be developed as part of this project, the backend database will be capable of supporting a user-facing application in the future.

Target Customers / Users

The primary users of the system will be the office administrators and management staff at Guardian Group Services. Instructors do not directly interact with student data as their only involvement is teaching their respective courses and signing off on class rosters once a session has concluded. Office admins handle the bulk of operational needs, including bookings, roster creation, and student intake, while the Director of Operations assists with administrative responsibilities. Due to the small team structure, all users will have similar access levels and permissions within the system.

Value Proposition

By implementing this database solution, Guardian Group Services will be able to reduce manual work, streamline class and roster management, and allow staff to focus on higher-value business needs such as organizational growth and expansion. The system will help create a more reliable and searchable source of truth for student and class information. In turn, this will lead to fewer mistakes, improved staff morale, and the ability to make data-driven decisions. With a centralized, well-structured system in place, Guardian Group Services can finally build trust in the data they rely on, opening the door to smoother operations and future growth opportunities.

Application Features

The database will track key entities such as students, course offerings, individual class sessions, instructors, and booking records. It will also store course locations, instructor credentials, and which courses each instructor is authorized to teach. Students may enroll in individual classes or choose from three predefined security guard training packages, which include combinations of courses and administrative services like fingerprinting and application assistance. Office administrators will be able to manage bookings, assign students to class sessions, update student profiles, and search for students by name, course history, or contact information. The system will also support creating rosters for each class session and preparing them for upload to the New York State Division of Criminal Justice Services online service. In addition, the database will enable powerful reporting functionality. Staff can view upcoming classes for the week, generate lists of students who completed a specific course within a certain date range (useful for tracking certifications due for renewal), and analyze student registration

patterns, including how many came through workforce agencies versus individual sign-ups. These insights can support marketing and outreach strategies, helping the organization expand its services more strategically.

Tools and Resources

This project will be developed with SQL Server Management Studio (SSMS) as the primary development environment. Dbdiagram.io will be used to create the physical Entity Relationship Diagram (ERD). The system will be normalized to 3NF and populated with sample data to simulate real world use cases.

The solution will include views, stored procedures, triggers, JSON payloads, and user-defined functions to support reporting, automation, and integration with external systems in the future. Role based security will also be implemented to ensure appropriate access control. Read-only/reporting roles will be defined for admins-in-training or managerial users such as the Director of Operations or CEO, while office administrators will have read-write access for daily operations. All permissions will be scoped at the schema level for clarity and consistency.

Challenges

One of the most challenging aspects of this project will be designing stored procedures and backend logic that provide meaningful automation and value to the business. While technical implementation is straightforward, it will take detailed analysis to determine what kinds of reporting and processing workflows will most improve the school's day-to-day operations. Another difficulty lies in modeling a clean, normalized structure for student and course data that was previously stored redundantly across several different spreadsheets. For example, the same student or booking information may appear in both intake forms and class rosters, with slight inconsistencies between them. Developing a unified schema that prevents duplication while preserving necessary relationships will be essential.

It is important to note several business processes will intentionally be left out of scope. These include fingerprinting appointments, job placement services, and payment processing, all of which are handled externally or through separate systems. While the eventual goal is to integrate the website and intake forms directly with the database, those features are outside the scope of this current phase. For now, the system will be designed to accommodate those services in the future.

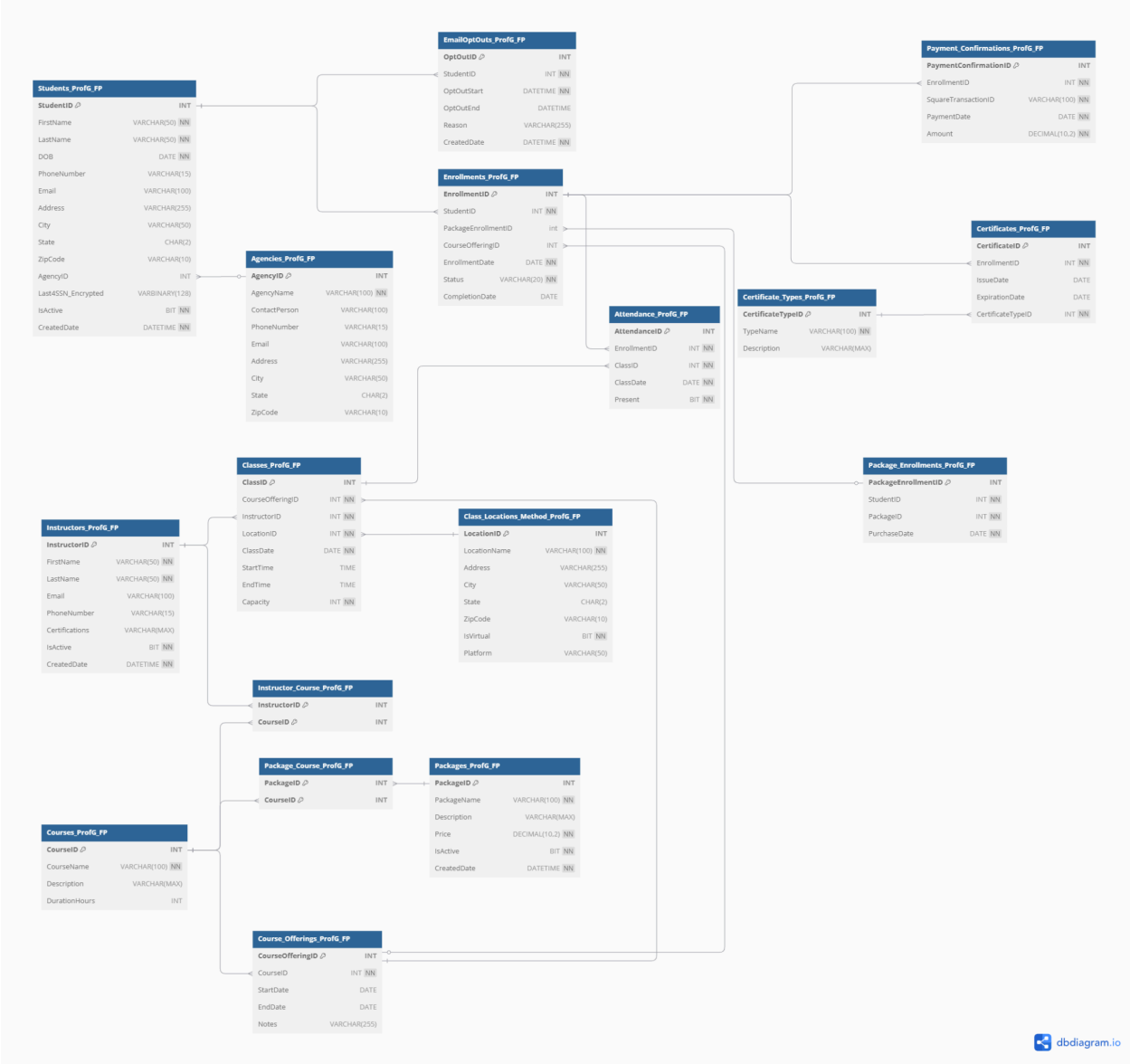
Expected ROI

The implementation of this database is expected to save significant time and effort for office staff at Guardian Group Services. By replacing manual data entry and spreadsheet

management with automated workflows, staff will be able to focus more on quality assurance, class monitoring, and outreach to potential students or agencies. Over time, this will lead to an increase in operational capacity, allowing the school to run more classes and better serve its growing client base.

Future enhancements such as automated certificate generation and email communication will further streamline the student experience while reinforcing compliance and follow-up processes. Additionally, the system's built-in auditing capabilities will promote greater accountability by enabling managers to identify training needs and reduce recurring administrative mistakes. The long-term benefits of this system include reduced dependence on error-prone tools like Google Sheets, greater staff morale, and a stronger foundation for business growth. As the organization continues to expand its services and explore new markets, a centralized, well-designed database will be essential.

ERD Diagram


Students_ProfG_FP
StudentID INT
FirstName VARCHAR(50) NN
LastName VARCHAR(50) NN
DOB DATE NN
PhoneNumber VARCHAR(15)
Email VARCHAR(100)
Address VARCHAR(255)
City VARCHAR(50)
State CHAR(2)
ZipCode VARCHAR(10)
AgencyID INT
Last4SSN_Encrypted VARBINARY(128)
IsActive BIT NN
CreatedDate DATETIME NN
EmailOptOuts_ProfG_FP
OptOutID INT
StudentID INT NN
OptOutStart DATETIME NN
OptOutEnd DATETIME
Reason VARCHAR(255)
CreatedDate DATETIME NN
Payment_Confirmations_ProfG_FP
PaymentConfirmationID INT
EnrollmentID INT NN
SquareTransactionID VARCHAR(100) NN
PaymentDate DATE NN
Amount DECIMAL(10,2) NN
Enrollments_ProfG_FP
EnrollmentID INT
StudentID INT NN
PackageEnrollmentID int
CourseOfferingID INT
EnrollmentDate DATE NN
Status VARCHAR(20) NN
CompletionDate DATE
Certificates_ProfG_FP
CertificateID INT
EnrollmentID INT NN
IssueDate DATE
ExpirationDate DATE
CertificateTypeID INT NN
Agencies_ProfG_FP
AgencyID INT
AgencyName VARCHAR(100) NN
ContactPerson VARCHAR(100)
PhoneNumber VARCHAR(15)
Email VARCHAR(100)
Address VARCHAR(255)
City VARCHAR(50)
State CHAR(2)
ZipCode VARCHAR(10)
Attendance_ProfG_FP
AttendanceID INT
EnrollmentID INT NN
ClassID INT NN
ClassDate DATE NN
Present BIT NN
Certificate_Types_ProfG_FP
CertificateTypeID INT
TypeName VARCHAR(100) NN
Description VARCHAR(MAX)
Classes_ProfG_FP
ClassID INT
CourseOfferingID INT NN
InstructorID INT NN
LocationID INT NN
ClassDate DATE NN
StartTime TIME
EndTime TIME
Capacity INT NN
Package_Enrollments_ProfG_FP
PackageEnrollmentID INT
StudentID INT NN
PackageID INT NN
PurchaseDate DATE NN
Instructors_ProfG_FP
InstructorID INT
FirstName VARCHAR(50) NN
LastName VARCHAR(50) NN
Email VARCHAR(100)
PhoneNumber VARCHAR(15)
Certifications VARCHAR(MAX)
IsActive BIT NN
CreatedDate DATETIME NN
Class_Locations_Method_ProfG_FP
LocationID INT
LocationName VARCHAR(100) NN
Address VARCHAR(255)
City VARCHAR(50)
State CHAR(2)
ZipCode VARCHAR(10)
IsVirtual BIT NN
Platform VARCHAR(50)
Instructor_Course_ProfG_FP
InstructorID INT
CourseID INT
Package_Course_ProfG_FP
PackageID INT
CourseID INT
Packages_ProfG_FP
PackageID INT
PackageName VARCHAR(100) NN
Description VARCHAR(MAX)
Price DECIMAL(10,2) NN
IsActive BIT NN
CreatedDate DATETIME NN
Courses_ProfG_FP
CourseID INT
CourseName VARCHAR(100) NN
Description VARCHAR(MAX)
DurationHours INT
Course_Offerings_ProfG_FP
CourseOfferingID INT
CourseID INT NN
StartDate DATE
EndDate DATE
Notes VARCHAR(255)
dbdiagram.io

GitHub

The link below is my GitHub repository for this project. You'll be able to inspect all of the code that was used to construct and test this database.

https://github.com/CesarASLuna/DatabaseProjectCST4714/tree/main

While GitHub contains everything, the section below has most of the code that was used to set up the stored procedures, triggers, auditing, and reporting that was required to fulfill the business requirements.

Solving Business Requirements

1. Add student securely with encrypted SSN if provided, then delete plaintext field.

   This

```
CREATE PROCEDURE sp_AddStudent_ProfG_FP
  @FirstName VARCHAR(50),
  @LastName VARCHAR(50),
  @DOB DATE,
  @PhoneNumber VARCHAR(15),
  @Email VARCHAR(100),
  @Address VARCHAR(255),
  @City VARCHAR(50),
  @State CHAR(2),
  @ZipCode VARCHAR(10),
  @AgencyID INT,
  @Last4SSN_Plaintext CHAR(4) = NULL
AS
BEGIN
  DECLARE @EncryptedSSN VARBINARY(128)

  IF @Last4SSN_Plaintext IS NOT NULL
  BEGIN
    SET @EncryptedSSN = EncryptByPassPhrase('YourStrongPassphrase!',
@Last4SSN_Plaintext)
  END

  INSERT INTO Students_ProfG_FP (
    FirstName, LastName, DOB, PhoneNumber, Email, Address,
    City, State, ZipCode, AgencyID, Last4SSN_Encrypted, IsActive, CreatedDate
  )
  VALUES (
```

```
    @FirstName, @LastName, @DOB, @PhoneNumber, @Email, @Address,
    @City, @State, @ZipCode, @AgencyID, @EncryptedSSN, 1, GETDATE()
  )
END
```

2. Validate certificate eligibility, to be used in certificate issuing procedure.

```
CREATE FUNCTION fn_ValidateCertificateEligibility_ProfG_FP (
  @EnrollmentID INT
)
RETURNS BIT
AS
BEGIN
  DECLARE @CourseOfferingID INT
  DECLARE @IsPaid BIT = 0
  DECLARE @TotalSessions INT
  DECLARE @PresentSessions INT
  DECLARE @Eligible BIT = 0

  -- Get course offering tied to this enrollment
  SELECT @CourseOfferingID = CourseOfferingID
  FROM Enrollments_ProfG_FP
  WHERE EnrollmentID = @EnrollmentID

  -- Confirm payment
  IF EXISTS (
    SELECT 1 FROM Payment_Confirmations_ProfG_FP
    WHERE EnrollmentID = @EnrollmentID
  )
    SET @IsPaid = 1

  -- Total expected sessions for that course offering
  SELECT @TotalSessions = COUNT(*)
  FROM Classes_ProfG_FP
  WHERE CourseOfferingID = @CourseOfferingID

  -- Total sessions marked Present
  SELECT @PresentSessions = COUNT(*)
  FROM Attendance_ProfG_FP
  WHERE EnrollmentID = @EnrollmentID AND Present = 1

  -- Final eligibility check
  IF @IsPaid = 1 AND @TotalSessions = @PresentSessions AND @TotalSessions > 0
    SET @Eligible = 1
```

```
  RETURN @Eligible
END
```

3. Issue certificate only if student has met attendance and payment criteria.

```
CREATE PROCEDURE sp_IssueCertificate_ProfG_FP
  @EnrollmentID INT,
  @CertificateTypeID INT
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @IsEligible BIT;

  -- Check eligibility
  SET @IsEligible = S23804121.fn_ValidateCertificateEligibility_ProfG_FP(@EnrollmentID);

  IF @IsEligible = 0
  BEGIN
    THROW 50001, 'Student is not eligible for certificate. Check attendance and payment.', 1;
    RETURN;
  END

  -- Issue certificate
  INSERT INTO Certificates_ProfG_FP (
    EnrollmentID,
    IssueDate,
    ExpirationDate,
    CertificateTypeID
  )
  VALUES (
    @EnrollmentID,
    GETDATE(),
    DATEADD(YEAR, 2, GETDATE()),  -- 2-year default expiration
    @CertificateTypeID
  );
END
```

4. Add a new course offering and auto-generate class sessions.

```
CREATE PROCEDURE sp_GenerateClassSessions_ProfG_FP
  @CourseOfferingID INT
AS
```

```
BEGIN
  DECLARE @CourseID INT, @DurationHours INT, @SessionCount INT, @i INT = 0

  -- Get course duration
  SELECT @CourseID = CourseID FROM Course_Offerings_ProfG_FP WHERE
CourseOfferingID = @CourseOfferingID
  SELECT @DurationHours = DurationHours FROM Courses_ProfG_FP WHERE CourseID =
@CourseID

  -- Assume 8 hours per session
  SET @SessionCount = CEILING(@DurationHours / 8.0)

  -- Generate sessions starting the day after StartDate
  WHILE @i < @SessionCount
  BEGIN
    INSERT INTO Classes_ProfG_FP (
      CourseOfferingID, InstructorID, LocationID,
      ClassDate, StartTime, EndTime, Capacity
    )
    SELECT
      co.CourseOfferingID,
      1, -- placeholder instructor
      1, -- placeholder location
      DATEADD(DAY, @i, co.StartDate),
      '09:00', '17:00',
      20
    FROM Course_Offerings_ProfG_FP co
    WHERE co.CourseOfferingID = @CourseOfferingID

    SET @i += 1
  END
END

CREATE PROCEDURE sp_AddCourseOffering_ProfG_FP
  @CourseID INT,
  @StartDate DATE,
  @EndDate DATE = NULL,
  @Notes VARCHAR(255) = NULL
AS
BEGIN
  DECLARE @NewOfferingID INT

  -- Insert the offering
  INSERT INTO Course_Offerings_ProfG_FP (CourseID, StartDate, EndDate, Notes)
```

```
  VALUES (@CourseID, @StartDate, @EndDate, @Notes)

  -- Get the new ID
  SET @NewOfferingID = SCOPE_IDENTITY()

  -- Call nested procedure to create class sessions
  EXEC sp_GenerateClassSessions_ProfG_FP @NewOfferingID
END
```

5. Drop a student from a course and log the removal.

```
CREATE TABLE Dropped_Enrollments_ProfG_FP (
  LogID INT IDENTITY(1,1) PRIMARY KEY,
  EnrollmentID INT,
  StudentID INT,
  CourseOfferingID INT,
  DropReason VARCHAR(255),
  DroppedBy VARCHAR(100),
  DropDate DATETIME DEFAULT GETDATE()
);


CREATE PROCEDURE sp_DropStudent_ProfG_FP
  @EnrollmentID INT,
  @DropReason VARCHAR(255) = NULL,
  @DroppedBy VARCHAR(100) = NULL
AS
BEGIN
  DECLARE @StudentID INT, @CourseOfferingID INT;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Get info before delete
    SELECT @StudentID = StudentID, @CourseOfferingID = CourseOfferingID
    FROM Enrollments_ProfG_FP
    WHERE EnrollmentID = @EnrollmentID;

    -- Validate: If nothing found, throw custom error
    IF @StudentID IS NULL
    BEGIN
      ROLLBACK;
      THROW 50002, 'Enrollment ID not found. Nothing to drop.', 1;
    END
```

```
    -- Clean related data
    DELETE FROM Attendance_ProfG_FP WHERE EnrollmentID = @EnrollmentID;
    DELETE FROM Payment_Confirmations_ProfG_FP WHERE EnrollmentID = @EnrollmentID;

    -- Delete from enrollments
    DELETE FROM Enrollments_ProfG_FP WHERE EnrollmentID = @EnrollmentID;

    -- Log to audit
    INSERT INTO Dropped_Enrollments_ProfG_FP (
      EnrollmentID, StudentID, CourseOfferingID, DropReason, DroppedBy
    )
    VALUES (
      @EnrollmentID, @StudentID, @CourseOfferingID, @DropReason, @DroppedBy
    );

    COMMIT;
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK;

    THROW 50003, 'Failed to drop enrollment. Check foreign keys or data integrity.', 1;
  END CATCH
END
```

6. Switch an instructor for all future sessions in a course offering.

```
CREATE PROCEDURE sp_SwitchInstructor_ProfG_FP
  @CourseOfferingID INT,
  @NewInstructorID INT
AS
BEGIN
  BEGIN TRY
    BEGIN TRANSACTION;

    UPDATE Classes_ProfG_FP
    SET InstructorID = @NewInstructorID
    WHERE CourseOfferingID = @CourseOfferingID
      AND ClassDate >= CAST(GETDATE() AS DATE);

    COMMIT;
  END TRY
  BEGIN CATCH
    ROLLBACK;
```

```
    THROW 50020, 'Error switching instructor. Check input IDs.', 1;
  END CATCH
END
```

7. Generate instructor workload report, showing number of students trained in the last 30 days

```
CREATE PROCEDURE sp_InstructorWorkloadReport_ProfG_FP
AS
BEGIN
  SET NOCOUNT ON;

  SELECT
    i.InstructorID,
    i.FirstName,
    i.LastName,
    COUNT(DISTINCT a.EnrollmentID) AS StudentsTrained
  FROM Instructors_ProfG_FP i
  JOIN Classes_ProfG_FP c ON i.InstructorID = c.InstructorID
  JOIN Attendance_ProfG_FP a ON c.ClassID = a.ClassID
  WHERE a.Present = 1
  GROUP BY i.InstructorID, i.FirstName, i.LastName
  FOR JSON AUTO
END
```

8. List all students enrolled in a specific class, including names, agency, and attendance status.

```
CREATE VIEW vw_ClassRoster_ProfG_FP AS
SELECT
  c.ClassID,
  s.StudentID,
  s.FirstName,
  s.LastName,
  a.AgencyName,
  att.ClassDate,
  att.Present
FROM Attendance_ProfG_FP att
JOIN Enrollments_ProfG_FP e ON att.EnrollmentID = e.EnrollmentID
JOIN Students_ProfG_FP s ON e.StudentID = s.StudentID
LEFT JOIN Agencies_ProfG_FP a ON s.AgencyID = a.AgencyID
JOIN Classes_ProfG_FP c ON att.ClassID = c.ClassID;
```

9. Show average class size attendance by course, across all offerings.

```
CREATE VIEW vw_AvgClassSizeByCourse_ProfG_FP AS
SELECT
  crs.CourseID,
  crs.CourseName,
  COUNT(DISTINCT cls.ClassID) AS TotalClasses,
  COUNT(DISTINCT att.EnrollmentID) AS TotalEnrollments,
  CAST(COUNT(DISTINCT att.EnrollmentID) * 1.0 / NULLIF(COUNT(DISTINCT cls.ClassID), 0)
AS DECIMAL(5,2)) AS AvgClassSize
FROM Courses_ProfG_FP crs
JOIN Course_Offerings_ProfG_FP cof ON crs.CourseID = cof.CourseID
JOIN Classes_ProfG_FP cls ON cof.CourseOfferingID = cls.CourseOfferingID
JOIN Attendance_ProfG_FP att ON cls.ClassID = att.ClassID
WHERE att.Present = 1
  AND cls.ClassDate < CAST(GETDATE() AS DATE)
GROUP BY crs.CourseID, crs.CourseName;
```

10. Courses held at a specific location with total number of class sessions per course

```
CREATE VIEW vw_CourseSessionsByLocation_ProfG_FP AS
SELECT
  l.LocationID,
  l.LocationName,
  c.CourseName,
  COUNT(cls.ClassID) AS TotalSessions
FROM Classes_ProfG_FP cls
JOIN Course_Offerings_ProfG_FP co ON cls.CourseOfferingID = co.CourseOfferingID
JOIN Courses_ProfG_FP c ON co.CourseID = c.CourseID
JOIN Class_Locations_Method_ProfG_FP l ON cls.LocationID = l.LocationID
GROUP BY l.LocationID, l.LocationName, c.CourseName;
```

11. On new enrollment, set timestamp and log enrollment automatically.

```
CREATE TRIGGER trg_AuditEnrollmentInsert_ProfG_FP
ON Enrollments_ProfG_FP
AFTER INSERT
AS
BEGIN
  INSERT INTO EnrollmentAudit_ProfG_FP (
    EnrollmentID, StudentID, CourseOfferingID, ActionTaken
  )
  SELECT
    i.EnrollmentID,
    i.StudentID,
    i.CourseOfferingID,
    'ENROLLMENT CREATED'
```

```
  FROM INSERTED i;
END
```

12. On new student insert, auto-create a blank email opt-out record.

```
CREATE TRIGGER trg_AutoInsertOptOut_ProfG_FP
ON Students_ProfG_FP
AFTER INSERT
AS
BEGIN
  INSERT INTO EmailOptOuts_ProfG_FP (StudentID, Reason)
  SELECT StudentID, NULL
  FROM INSERTED;
END
```

13. When a class is deleted, archive it to another table.

```
CREATE TRIGGER trg_ArchiveClassOnDelete_ProfG_FP
ON Classes_ProfG_FP
AFTER DELETE
AS
BEGIN
  SET NOCOUNT ON;

  INSERT INTO Archived_Classes_ProfG_FP (
    ClassID, CourseOfferingID, InstructorID, LocationID,
    ClassDate, StartTime, EndTime, Capacity
  )
  SELECT
    d.ClassID, d.CourseOfferingID, d.InstructorID, d.LocationID,
    d.ClassDate, d.StartTime, d.EndTime, d.Capacity
  FROM DELETED d;
END
```

14. When an instructor loses their certification for a course, archive it to another table.

```
CREATE TRIGGER trg_ArchiveInstructorCourseOnDelete_ProfG_FP
ON Instructor_Course_ProfG_FP
AFTER DELETE
AS
BEGIN
  SET NOCOUNT ON;
```

```
  INSERT INTO Archived_Instructor_Course_ProfG_FP (
    InstructorID, CourseID
  )
  SELECT
    InstructorID, CourseID
  FROM DELETED;
END
```

15. When an instructor updates their contact information, archive it to another table.

```
CREATE TRIGGER trg_AuditInstructorUpdate_ProfG_FP
ON Instructors_ProfG_FP
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  INSERT INTO InstructorAudit_ProfG_FP (
    InstructorID,
    OldEmail, NewEmail,
    OldPhone, NewPhone,
    OldCertifications, NewCertifications
  )
  SELECT
    i.InstructorID,
    d.Email, i.Email,
    d.PhoneNumber, i.PhoneNumber,
    d.Certifications, i.Certifications
  FROM INSERTED i
  JOIN DELETED d ON i.InstructorID = d.InstructorID
  WHERE
    ISNULL(d.Email, '') <> ISNULL(i.Email, '')
    OR ISNULL(d.PhoneNumber, '') <> ISNULL(i.PhoneNumber, '')
    OR ISNULL(d.Certifications, '') <> ISNULL(i.Certifications, '');
END
```

16. Update Class information and log it into a table.

```
CREATE TRIGGER trg_LogClassChange_ProfG_FP
ON Classes_ProfG_FP
AFTER UPDATE
AS
BEGIN
  SET NOCOUNT ON;
```

```
  INSERT INTO ClassChangeLog_ProfG_FP (
    ClassID,
    OldClassDate, NewClassDate,
    OldLocationID, NewLocationID
  )
  SELECT
    i.ClassID,
    d.ClassDate, i.ClassDate,
    d.LocationID, i.LocationID
  FROM INSERTED i
  JOIN DELETED d ON i.ClassID = d.ClassID
  WHERE
    ISNULL(d.ClassDate, '') <> ISNULL(i.ClassDate, '')
    OR ISNULL(d.LocationID, 0) <> ISNULL(i.LocationID, 0);
END
```

17. Anytime attendance is added, updated, or removed, log it.

```
CREATE TRIGGER trg_AuditAttendanceChanges_ProfG_FP
ON Attendance_ProfG_FP
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
  SET NOCOUNT ON;

  -- INSERT
  INSERT INTO AttendanceAudit_ProfG_FP (
    ActionType, AttendanceID, EnrollmentID, ClassID, ClassDate, Present
  )
  SELECT
    'INSERT', AttendanceID, EnrollmentID, ClassID, ClassDate, Present
  FROM INSERTED
  WHERE NOT EXISTS (SELECT 1 FROM DELETED WHERE DELETED.AttendanceID =
INSERTED.AttendanceID);

  -- DELETE
  INSERT INTO AttendanceAudit_ProfG_FP (
    ActionType, AttendanceID, EnrollmentID, ClassID, ClassDate, Present
  )
  SELECT
    'DELETE', AttendanceID, EnrollmentID, ClassID, ClassDate, Present
  FROM DELETED
  WHERE NOT EXISTS (SELECT 1 FROM INSERTED WHERE INSERTED.AttendanceID =
DELETED.AttendanceID);
```

```
  -- UPDATE
  INSERT INTO AttendanceAudit_ProfG_FP (
    ActionType, AttendanceID, EnrollmentID, ClassID, ClassDate, Present
  )
  SELECT
    'UPDATE', i.AttendanceID, i.EnrollmentID, i.ClassID, i.ClassDate, i.Present
  FROM INSERTED i
  JOIN DELETED d ON i.AttendanceID = d.AttendanceID
  WHERE
    ISNULL(i.Present, 0) <> ISNULL(d.Present, 0)
    OR ISNULL(i.ClassDate, '') <> ISNULL(d.ClassDate, '')
    OR ISNULL(i.ClassID, 0) <> ISNULL(d.ClassID, 0);
END
```

18. Export all certificates ever issued, including student names and certificate types, as a JSON payload

```
SELECT
  c.CertificateID,
  c.EnrollmentID,
  c.IssueDate,
  ct.TypeName,
  e.StudentID,
  s.FirstName,
  s.LastName
FROM Certificates_ProfG_FP c
JOIN Certificate_Types_ProfG_FP ct ON c.CertificateTypeID = ct.CertificateTypeID
JOIN Enrollments_ProfG_FP e ON c.EnrollmentID = e.EnrollmentID
JOIN Students_ProfG_FP s ON e.StudentID = s.StudentID
FOR JSON AUTO, INCLUDE_NULL_VALUES;
```

19. For each agency, return a JSON list of active students and the courses they are currently enrolled in

```
SELECT
  a.AgencyName,
  s.StudentID,
  s.FirstName,
  s.LastName,
  co.CourseOfferingID,
  c.CourseName
FROM Students_ProfG_FP s
JOIN Agencies_ProfG_FP a ON s.AgencyID = a.AgencyID
```

```
JOIN Enrollments_ProfG_FP e ON s.StudentID = e.StudentID
JOIN Course_Offerings_ProfG_FP co ON e.CourseOfferingID = co.CourseOfferingID
JOIN Courses_ProfG_FP c ON co.CourseID = c.CourseID
WHERE s.IsActive = 1
FOR JSON AUTO, INCLUDE_NULL_VALUES;
```

20. How full are classes relative to their max capacity?

```
CREATE FUNCTION S23804121.fn_ClassEnrollmentFillPercentage_ProfG_FP (
  @ClassID INT
)
RETURNS DECIMAL(5,2)
AS
BEGIN
  DECLARE @CourseOfferingID INT;
  DECLARE @Capacity INT = 0;
  DECLARE @EnrolledStudents INT = 0;
  DECLARE @Percent DECIMAL(5,2) = 0;

  -- Get the course offering and capacity for this class
  SELECT
    @CourseOfferingID = CourseOfferingID,
    @Capacity = Capacity
  FROM S23804121.Classes_ProfG_FP
  WHERE ClassID = @ClassID;

  -- Count how many students are enrolled in that offering
  SELECT @EnrolledStudents = COUNT(*)
  FROM S23804121.Enrollments_ProfG_FP
  WHERE CourseOfferingID = @CourseOfferingID;

  IF @Capacity > 0
    SET @Percent = (CAST(@EnrolledStudents AS DECIMAL(5,2)) / @Capacity) * 100;

  RETURN @Percent;
END;
```

21. Which classes are suffering from low enrollments?

```
CREATE FUNCTION S23804121.fn_GetEnrollmentCountForOffering_ProfG_FP (
  @CourseOfferingID INT
)
RETURNS INT
```

```
AS
BEGIN
  DECLARE @Count INT

  SELECT @Count = COUNT(*)
  FROM Enrollments_ProfG_FP
  WHERE CourseOfferingID = @CourseOfferingID

  RETURN @Count
END
GO

CREATE PROCEDURE sp_FindLowEnrollmentOfferings_ProfG_FP
  @MinStudents INT = 5
AS
BEGIN
  SET NOCOUNT ON;

  SELECT
    co.CourseOfferingID,
    c.CourseName,
    co.StartDate,
    co.EndDate,
    dbo.fn_GetEnrollmentCountForOffering_ProfG_FP(co.CourseOfferingID) AS EnrolledCount
  FROM Course_Offerings_ProfG_FP co
  JOIN Courses_ProfG_FP c ON co.CourseID = c.CourseID
  WHERE dbo.fn_GetEnrollmentCountForOffering_ProfG_FP(co.CourseOfferingID) <
@MinStudents
END
GO
```

Credits and Acknowledgements

Guardian Group Services Staff
I wanted to keep my project as realistic and aligned with a real-world scenario as possible. I thank them for answering any questions I had and letting me use their business as the basis for this project.

- Jurney Ramos - Office administrator
- Bruce Weiss - Chief Executive Officer, Owner
- Charles McNamara - Director of Operations

Academics

- Sona Brahmbhatt - Professor at Farmingdale University, fellow coworker. She helped me refine my database design.
- Kevin Gravesande - Professor at CUNY New York City College of Technology. Provided feedback and the environment I mostly used while testing.