Strategic Data and Dimensional Warehouse Modeling for University Analytics

Satyanarayana Vinay Achanta (A02395874)

Hari Chandana Kotnani (A02396013)

Course: Advanced Database Systems

Table of Contents

1. Introduction

The goal of this project is to design and implement a comprehensive database and data warehouse for Utah State University to facilitate the analysis of admissions, enrollment, and educational data. This project focuses on several key areas:

- **Data Modeling**: Developing a robust data model to capture detailed information about students, admissions, programs, courses, enrollments, grades, departments, faculty, campuses, scholarships, and their relationships.
- **Dimensional Modeling**: Designing a data warehouse schema to support analytical queries and reporting, enabling efficient data retrieval for trend analysis and decision-making.

This database and data warehouse will support various departments, including Domestic Admissions, International Admissions, and Study Abroad programs. It will cover multiple education levels such as Undergraduate, Graduate, and PhD, providing a comprehensive platform for analyzing educational data, making informed decisions, and identifying trends.

The project includes the following critical components:

- **Conceptual Data Model**: Defines high-level entities and relationships to capture essential data elements and their interactions.
- **Logical Data Model**: Organizes data into normalized tables to ensure data integrity and reduce redundancy.
- **Physical Data Model**: Specifies the physical schema, indexing strategy, and partitioning requirements to optimize storage and query performance.
- **Dimensional Model Design**: Constructs fact and dimension tables, such as EnrollmentFact, GradeFact, and StudentDim, to support efficient data analysis and reporting.

Analytical Queries: To facilitate in-depth analysis, several key analytical queries are included, focusing on different aspects of educational data:

- **Student Performance Analysis**: Evaluates average grades per course.
- **Scholarship Distribution**: Analyzes total scholarship amounts awarded by department.
- **Enrollment Trends**: Tracks the number of enrollments in different programs over time.
- **Faculty Workload**: Measures the number of courses taught by faculty members.
- **Fee Collection Analysis**: Reviews the total fees collected for each course and semester.
- **Admission Status**: Reports on the status of student admissions by program.
- **Student Graduation Trends**: Examines graduation rates over the years.
- **Course Enrollment**: Counts the number of students enrolled in each course.

This system aims to enhance the university's ability to manage and analyze its data, providing valuable insights for strategic planning and operational improvements.

2. Project Objectives

- **Design a Normalized Database Schema:**
    - Develop a detailed, normalized schema to capture and manage educational data efficiently, ensuring data integrity and eliminating redundancy.
- **Develop a Denormalized Data Warehouse Schema:**
    - Create a denormalized data warehouse schema to facilitate efficient querying and reporting, optimized for analytical and business intelligence purposes.
- **Create Analytical Queries:**
    - Develop analytical queries and reports to support various stakeholders, including admissions, enrollment management, and program evaluation.

3. Scope of the Project

The project will cover the following areas:

- **Admissions Data:** Manage and analyze admissions data for domestic, international, and study abroad students.
- **Educational Levels:** Capture and analyze data across different educational levels, including undergraduate, graduate, and PhD.
- **Enrollment Details:** Track and manage enrollment details, including courses, grades, and academic records.
- **Database and Data Warehouse Development:** Develop a normalized operational database for detailed record-keeping and a denormalized data warehouse for efficient querying and reporting.

4. Requirements Analysis

- **Stakeholders:**
    - **Admissions Department:** Requires data to manage and analyze applications, admissions processes, and trends for domestic, international, and study abroad students.
    - **Academic Departments:** Needs data on student enrollments, course performance, and academic records to support curriculum planning and academic advising.
    - **University Administration:** Requires comprehensive data to make informed decisions about university operations, resource allocation, and strategic planning.
- **Key Metrics:**
    - **Student Enrollment Trends:** Analyzing patterns and changes in student enrollments over time to support planning and resource management.
    - **Admission Rates and Acceptance Statistics:** Tracking admission rates, acceptance ratios, and application trends to assess the effectiveness of recruitment strategies.
    - **Academic Performance and Grade Distributions:** Evaluating academic performance across different programs and courses to identify trends, strengths, and areas for improvement.
    - **Course Enrollment Statistics:** Monitoring course enrollments to manage class sizes, staffing needs, and curriculum adjustments.

5. Conceptual Data Model

Entities:

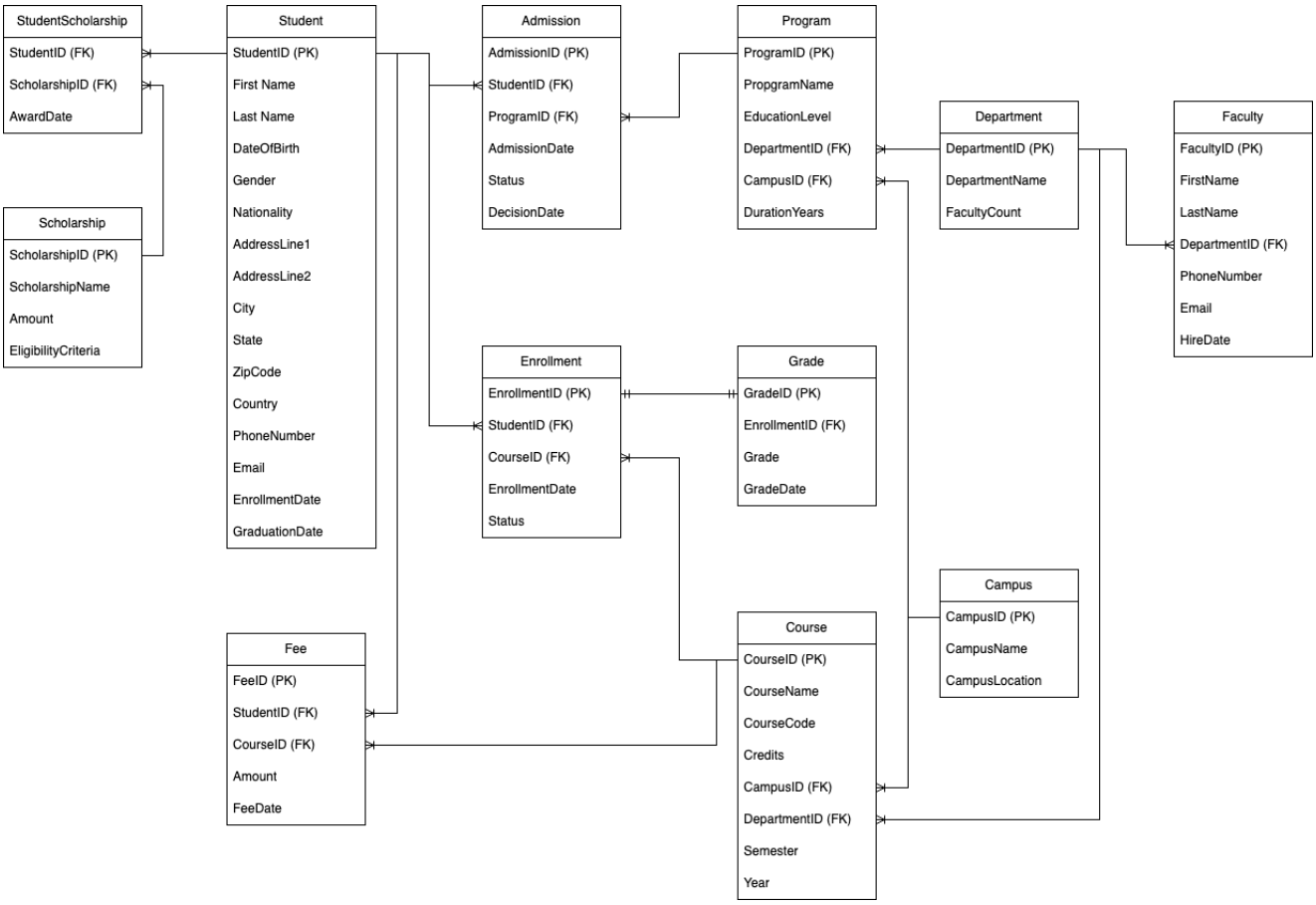
1. Student
2. Admission
3. Program
4. Course
5. Enrollment
6. Grade
7. Department
8. Faculty
9. Campus
10. Scholarship
11. StudentScholarship
12. Fee

Relationships:

- Student to Admission: One-to-Many
- Admission to Program: Many-to-One
- Student to Enrollment: One-to-Many
- Enrollment to Course: Many-to-One
- Enrollment to Grade: One-to-One
- Program to Department: Many-to-One
- Course to Department: Many-to-One
- Course to Campus: Many-to-One
- Student to StudentScholarship: One-to-Many
- Scholarship to StudentScholarship: One-to-Many
- Faculty to Department: Many-to-One
- Program to Campus: Many-to-One
- Fee to Student (Many-to-One)
- Fee to Course (Many-to-One)

6. Logical Data Model

Entity-Relationship Model



7. Physical Data Model

Normalized Schema (3NF):

- **Student:**

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Gender CHAR(1) CHECK (Gender IN ('M', 'F')),
    Nationality NVARCHAR(50),
    AddressLine1 NVARCHAR(100) NOT NULL,
    AddressLine2 NVARCHAR(100),
    City NVARCHAR(50) NOT NULL,
    State NVARCHAR(50) NOT NULL,
    ZipCode NVARCHAR(10) NOT NULL,
    Country NVARCHAR(50) NOT NULL,
    PhoneNumber NVARCHAR(15),
    Email NVARCHAR(100) UNIQUE NOT NULL,
    EnrollmentDate DATE NOT NULL,
    GraduationDate DATE
);

CREATE INDEX IX_Student_Email ON Student (Email);
CREATE INDEX IX_Student_Name ON Student (FirstName, LastName);
```

- **Admission:**

```sql
-- Admission Table
CREATE TABLE Admission (
    AdmissionID INT PRIMARY KEY IDENTITY(1,1),
    StudentID INT NOT NULL,
    ProgramID INT NOT NULL,
    AdmissionDate DATE NOT NULL,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Pending', 'Accepted', 'Rejected')),
    DecisionDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (ProgramID) REFERENCES Program(ProgramID)
);
-- Indexes
CREATE INDEX IX_Admission_StudentID ON Admission (StudentID);
CREATE INDEX IX_Admission_ProgramID ON Admission (ProgramID);
```

- **Program:**

```sql
-- Program Table
CREATE TABLE Program (
    ProgramID INT PRIMARY KEY IDENTITY(1,1),
    ProgramName NVARCHAR(100) NOT NULL,
    EducationLevel NVARCHAR(50) NOT NULL,
    DepartmentID INT NOT NULL,
    CampusID INT NOT NULL,
    DurationYears INT NOT NULL CHECK (DurationYears > 0),
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    FOREIGN KEY (CampusID) REFERENCES Campus(CampusID)
);
-- Indexes
CREATE INDEX IX_Program_Name ON Program (ProgramName);
```

- **Course:**

```sql
-- Course Table
CREATE TABLE Course (
    CourseID INT PRIMARY KEY IDENTITY(1,1),
    CourseName NVARCHAR(100) NOT NULL,
    CourseCode NVARCHAR(10) UNIQUE NOT NULL,
    Credits INT NOT NULL CHECK (Credits > 0),
    DepartmentID INT NOT NULL,
    CampusID INT NOT NULL,
    Semester NVARCHAR(10) NOT NULL,
    Year INT NOT NULL CHECK (Year > 2000),
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID),
    FOREIGN KEY (CampusID) REFERENCES Campus(CampusID)
);
-- Indexes
CREATE INDEX IX_Course_Code ON Course (CourseCode);
CREATE INDEX IX_Course_Name ON Course (CourseName);
CREATE INDEX IX_Course_DepartmentID ON Course (DepartmentID);
```

- **Enrollment:**

```sql
-- Enrollment Table
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY IDENTITY(1,1),
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    EnrollmentDate DATE NOT NULL,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Enrolled', 'Completed', 'Withdrawn')),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
-- Indexes
CREATE INDEX IX_Enrollment_StudentID ON Enrollment (StudentID);
CREATE INDEX IX_Enrollment_CourseID ON Enrollment (CourseID);
```

- **Grade:**

```sql
-- Grade Table
CREATE TABLE Grade (
    GradeID INT PRIMARY KEY IDENTITY(1,1),
    EnrollmentID INT NOT NULL,
    Grade CHAR(2) NOT NULL CHECK (Grade IN ('A', 'B', 'C', 'D', 'F')),
    GradeDate DATE NOT NULL,
    FOREIGN KEY (EnrollmentID) REFERENCES Enrollment(EnrollmentID)
);
-- Indexes
CREATE INDEX IX_Grade_EnrollmentID ON Grade (EnrollmentID);
CREATE INDEX IX_Grade_Grade ON Grade (Grade);
```

- **Department:**

```sql
-- Department Table
CREATE TABLE Department (
    DepartmentID INT PRIMARY KEY IDENTITY(1,1),
    DepartmentName NVARCHAR(100) NOT NULL,
    FacultyCount INT CHECK (FacultyCount >= 0)
);
-- Indexes
CREATE INDEX IX_Department_Name ON Department (DepartmentName);
```

- **Faculty:**

```sql
-- Faculty Table
CREATE TABLE Faculty (
    FacultyID INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DepartmentID INT NOT NULL,
    PhoneNumber NVARCHAR(15),
    Email NVARCHAR(100) UNIQUE NOT NULL,
    HireDate DATE NOT NULL,
    FOREIGN KEY (DepartmentID) REFERENCES Department(DepartmentID)
);
-- Indexes
CREATE INDEX IX_Faculty_Name ON Faculty (FirstName, LastName);
CREATE INDEX IX_Faculty_DepartmentID ON Faculty (DepartmentID);
```

- **Campus:**

```sql
-- Campus Table
CREATE TABLE Campus (
    CampusID INT PRIMARY KEY IDENTITY(1,1),
    CampusName NVARCHAR(100) NOT NULL,
    CampusLocation NVARCHAR(100) NOT NULL
);
-- Indexes
CREATE INDEX IX_Campus_Name ON Campus (CampusName);
```

- **Scholarship:**

```sql
-- Scholarship Table
CREATE TABLE Scholarship (
    ScholarshipID INT PRIMARY KEY IDENTITY(1,1),
    ScholarshipName NVARCHAR(100) NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL CHECK (Amount > 0),
    EligibilityCriteria TEXT
);
-- Indexes
CREATE INDEX IX_Scholarship_Name ON Scholarship (ScholarshipName);
```

- **StudentScholarship:**

```sql
-- StudentScholarship Table
CREATE TABLE StudentScholarship (
    StudentID INT NOT NULL,
    ScholarshipID INT NOT NULL,
    AwardDate DATE NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (ScholarshipID) REFERENCES Scholarship(ScholarshipID),
    PRIMARY KEY (StudentID, ScholarshipID)
);
-- Indexes
CREATE INDEX IX_StudentScholarship_StudentID ON StudentScholarship (StudentID);
CREATE INDEX IX_StudentScholarship_ScholarshipID ON StudentScholarship (ScholarshipID);
```

- **Fee**
  - ```
    CREATE TABLE Fee (
        FeeID INT PRIMARY KEY IDENTITY(1,1),
        StudentID INT NOT NULL,
        CourseID INT NOT NULL,
        Amount DECIMAL(10, 2) NOT NULL CHECK (Amount >= 0),
        FeeDate DATE NOT NULL,
        FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
        FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
    );
    -- Indexes
    CREATE INDEX IX_Fee_StudentID ON Fee (StudentID);
    CREATE INDEX IX_Fee_CourseID ON Fee (CourseID);
    CREATE INDEX IX_Fee_FeeDate ON Fee (FeeDate);
    ```

8.Dimensional Model Design

Dimension Tables:

- Student: Describes students.
- Faculty: Describes faculty members.
- Department: Describes departments.
- Campus: Describes campuses.
- Course: Describes courses.
- Scholarship: Describes scholarships.
- Program: Describes programs.
- Date: Will be used to manage date-related attributes for various events (e.g., enrollment, fee payment).

Fact Tables:

- Enrollment: Stores information about student enrollments.
- Grade: Stores grades received by students.
- Fee: Stores fee payments made by students.
- Admission: Stores admission details.
    ```

Code:

- **DateDim**

```
-- Create Date Dimension Table
CREATE TABLE DateDim (
    DateKey INT,
    Date DATE,
    Day INT,
    Month INT,
    Year INT,
    Quarter INT,
    MonthName NVARCHAR(20),
    QuarterName NVARCHAR(20),
    PRIMARY KEY (DateKey, Date)
)

-- Populate Date Dimension Table
DECLARE @StartDate DATE = '2015-01-01';
DECLARE @EndDate DATE = '2025-12-31';

WITH DateRange AS (
    SELECT @StartDate AS DateValue
    UNION ALL
    SELECT DATEADD(DAY, 1, DateValue)
    FROM DateRange
    WHERE DateValue < @EndDate
)
INSERT INTO DateDim (DateKey, Date, Day, Month, Year, Quarter, MonthName, QuarterName)
SELECT
    CONVERT(INT, FORMAT(DateValue, 'yyyyMMdd')) AS DateKey,
    DateValue AS Date,
    DAY(DateValue) AS Day,
    MONTH(DateValue) AS Month,
    YEAR(DateValue) AS Year,
    DATEPART(QUARTER, DateValue) AS Quarter,
    DATENAME(MONTH, DateValue) AS MonthName,
    'Q' + CAST(DATEPART(QUARTER, DateValue) AS NVARCHAR) AS QuarterName
FROM DateRange
OPTION (MAXRECURSION 0);
GO
```

- **StudentDim**

```
-- Create Student Dimension Table
CREATE TABLE StudentDim (
    StudentID INT,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DateOfBirth DATE NOT NULL,
    Gender CHAR(1) CHECK (Gender IN ('M', 'F')),
    Nationality NVARCHAR(50),
    AddressLine1 NVARCHAR(100) NOT NULL,
    AddressLine2 NVARCHAR(100),
    City NVARCHAR(50) NOT NULL,
    State NVARCHAR(50) NOT NULL,
    ZipCode NVARCHAR(10) NOT NULL,
    Country NVARCHAR(50) NOT NULL,
    PhoneNumber NVARCHAR(15),
    Email NVARCHAR(100) UNIQUE NOT NULL,
    EnrollmentDate DATE NOT NULL,
    GraduationDate DATE,
    PRIMARY KEY (StudentID)
)
-- Indexes for Student Dimension
CREATE NONCLUSTERED INDEX IX_Email ON StudentDim (Email);
CREATE NONCLUSTERED INDEX IX_EnrollmentDate ON StudentDim (EnrollmentDate);
CREATE NONCLUSTERED INDEX IX_Country ON StudentDim (Country);
CREATE NONCLUSTERED INDEX IX_State ON StudentDim (State);
```

- **FacultyDim**

```
-- Create Faculty Dimension Table
CREATE TABLE FacultyDim (
    FacultyID INT PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    DepartmentID INT NOT NULL,
    PhoneNumber NVARCHAR(15),
    Email NVARCHAR(100) UNIQUE NOT NULL,
    HireDate DATE NOT NULL
)
-- Indexes for Faculty Dimension
CREATE NONCLUSTERED INDEX IX_Email ON FacultyDim (Email);
CREATE NONCLUSTERED INDEX IX_DepartmentID ON FacultyDim (DepartmentID);
CREATE NONCLUSTERED INDEX IX_HireDate ON FacultyDim (HireDate);
```

- **DepartmentDim**

```sql
-- Create Department Dimension Table
CREATE TABLE DepartmentDim (
    DepartmentID INT PRIMARY KEY,
    DepartmentName NVARCHAR(100) NOT NULL,
    FacultyCount INT CHECK (FacultyCount >= 0)
);
-- Indexes for Department Dimension
CREATE NONCLUSTERED INDEX IX_DepartmentName ON DepartmentDim (DepartmentName);
```

- **CampusDim**

```sql
-- Create Campus Dimension Table
CREATE TABLE CampusDim (
    CampusID INT PRIMARY KEY,
    CampusName NVARCHAR(100) NOT NULL,
    CampusLocation NVARCHAR(100) NOT NULL
);
-- Indexes for Campus Dimension
CREATE NONCLUSTERED INDEX IX_CampusName ON CampusDim (CampusName);
CREATE NONCLUSTERED INDEX IX_CampusLocation ON CampusDim (CampusLocation);
```

- **CourseDim**

```sql
-- Create Course Dimension Table
CREATE TABLE CourseDim (
    CourseID INT PRIMARY KEY,
    CourseName NVARCHAR(100) NOT NULL,
    CourseCode NVARCHAR(10) UNIQUE NOT NULL,
    Credits INT NOT NULL CHECK (Credits > 0),
    DepartmentID INT NOT NULL,
    CampusID INT NOT NULL,
    Semester NVARCHAR(10) NOT NULL,
    Year INT NOT NULL CHECK (Year > 2000)
);
-- Indexes for Course Dimension
CREATE NONCLUSTERED INDEX IX_CourseCode ON CourseDim (CourseCode);
CREATE NONCLUSTERED INDEX IX_DepartmentID ON CourseDim (DepartmentID);
CREATE NONCLUSTERED INDEX IX_CampusID ON CourseDim (CampusID);
CREATE NONCLUSTERED INDEX IX_Semester_Year ON CourseDim (Semester, Year);
```

- **ScholarshipDim**

```sql
-- Create Scholarship Dimension Table
CREATE TABLE ScholarshipDim (
    ScholarshipID INT PRIMARY KEY,
    ScholarshipName NVARCHAR(100) NOT NULL,
    Amount DECIMAL(10, 2) NOT NULL CHECK (Amount > 0),
    EligibilityCriteria TEXT
);
-- Indexes for Scholarship Dimension
CREATE NONCLUSTERED INDEX IX_ScholarshipName ON ScholarshipDim (ScholarshipName);
```

- **ProgramDim**

```sql
-- Create Program Dimension Table
CREATE TABLE ProgramDim (
    ProgramID INT PRIMARY KEY,
    ProgramName NVARCHAR(100) NOT NULL,
    EducationLevel NVARCHAR(50) NOT NULL,
    DepartmentID INT NOT NULL,
    CampusID INT NOT NULL,
    DurationYears INT NOT NULL CHECK (DurationYears > 0)
);
-- Indexes for Program Dimension
CREATE NONCLUSTERED INDEX IX_ProgramName ON ProgramDim (ProgramName);
CREATE NONCLUSTERED INDEX IX_DepartmentID ON ProgramDim (DepartmentID);
CREATE NONCLUSTERED INDEX IX_CampusID ON ProgramDim (CampusID);
CREATE NONCLUSTERED INDEX IX_EducationLevel ON ProgramDim (EducationLevel);
```

- **AdmissionDim**

```sql
-- Create Admission Dimension Table
CREATE TABLE AdmissionDim (
    AdmissionID INT PRIMARY KEY,
    StudentID INT NOT NULL,
    ProgramID INT NOT NULL,
    AdmissionDate DATE NOT NULL,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Pending', 'Accepted',
      'Rejected')),
    DecisionDate DATE
)
-- Indexes for Admission Dimension
CREATE NONCLUSTERED INDEX IX_StudentID ON AdmissionDim (StudentID);
CREATE NONCLUSTERED INDEX IX_ProgramID ON AdmissionDim (ProgramID);
CREATE NONCLUSTERED INDEX IX_AdmissionDate ON AdmissionDim (AdmissionDate);
CREATE NONCLUSTERED INDEX IX_Status ON AdmissionDim (Status);
CREATE NONCLUSTERED INDEX IX_DecisionDate ON AdmissionDim (DecisionDate);
```

- **Partition Function and Scheme**

```sql
-- Partition Function
CREATE PARTITION FUNCTION PF_Date (INT)
AS RANGE LEFT FOR VALUES (
    20150101, 20160101, 20170101, 20180101, 20190101,
    20200101, 20210101, 20220101, 20230101, 20240101
);

-- Partition Scheme
CREATE PARTITION SCHEME PS_Date
AS PARTITION PF_Date
ALL TO ([PRIMARY]);
```

- **EnrollmentFact**

```sql
-- Create Enrollment Fact Table
CREATE TABLE EnrollmentFact (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDateKey INT,
    Status NVARCHAR(20),
    FOREIGN KEY (StudentID) REFERENCES StudentDim(StudentID),
    FOREIGN KEY (CourseID) REFERENCES CourseDim(CourseID),
    FOREIGN KEY (EnrollmentDateKey) REFERENCES DateDim(DateKey),
)

-- Create Non-Clustered Index on StudentID
CREATE NONCLUSTERED INDEX idx_EnrollmentFact_StudentID
ON EnrollmentFact (StudentID);

-- Create Non-Clustered Index on CourseID
CREATE NONCLUSTERED INDEX idx_EnrollmentFact_CourseID
ON EnrollmentFact (CourseID);

-- Create Non-Clustered Index on EnrollmentDateKey
CREATE NONCLUSTERED INDEX idx_EnrollmentFact_EnrollmentDateKey
ON EnrollmentFact (EnrollmentDateKey);

-- Create Composite Non-Clustered Index on StudentID, CourseID, and
  EnrollmentDateKey
CREATE NONCLUSTERED INDEX idx_EnrollmentFact_StudentID_CourseID_EnrollmentDateKey
ON EnrollmentFact (StudentID, CourseID, EnrollmentDateKey);
```

- **GradeFact**

```
-- Create Grade Fact Table
CREATE TABLE GradeFact (
    GradeID INT,
    EnrollmentID INT,
    Grade CHAR(2),
    GradeDateKey INT,
    FOREIGN KEY (EnrollmentID) REFERENCES EnrollmentFact(EnrollmentID),
    FOREIGN KEY (GradeDateKey) REFERENCES DateDim(DateKey),
    PRIMARY KEY (GradeID, GradeDateKey)
)
-- Create Non-Clustered Index on EnrollmentID
CREATE NONCLUSTERED INDEX idx_GradeFact_EnrollmentID
ON GradeFact (EnrollmentID);

-- Create Non-Clustered Index on GradeDateKey
CREATE NONCLUSTERED INDEX idx_GradeFact_GradeDateKey
ON GradeFact (GradeDateKey);

-- Create Composite Non-Clustered Index on EnrollmentID and GradeDateKey
CREATE NONCLUSTERED INDEX idx_GradeFact_EnrollmentID_GradeDateKey
ON GradeFact (EnrollmentID, GradeDateKey);
```

- **FeeFact**

```
-- Create Fee Fact Table
CREATE TABLE FeeFact (
    FeeID INT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    FeeAmount DECIMAL(10, 2),
    FeeDateKey INT,
    FOREIGN KEY (StudentID) REFERENCES StudentDim(StudentID),
    FOREIGN KEY (CourseID) REFERENCES CourseDim(CourseID),
    FOREIGN KEY (FeeDateKey) REFERENCES DateDim(DateKey)
)
-- Create Non-Clustered Index on StudentID
CREATE NONCLUSTERED INDEX idx_FeeFact_StudentID
ON FeeFact (StudentID);

-- Create Non-Clustered Index on CourseID
CREATE NONCLUSTERED INDEX idx_FeeFact_CourseID
ON FeeFact (CourseID);

-- Create Non-Clustered Index on FeeDateKey
CREATE NONCLUSTERED INDEX idx_FeeFact_FeeDateKey
ON FeeFact (FeeDateKey);

-- Create Composite Non-Clustered Index on StudentID, CourseID, and FeeDateKey
CREATE NONCLUSTERED INDEX idx_FeeFact_StudentID_CourseID_FeeDateKey
ON FeeFact (StudentID, CourseID, FeeDateKey);
```

- **AdmissionFact**

```
-- Create Admission Fact Table
CREATE TABLE AdmissionFact (
    AdmissionID INT PRIMARY KEY,
    StudentID INT,
    ProgramID INT,
    AdmissionDateKey INT,
    Status NVARCHAR(20),
    DecisionDateKey INT,
    FOREIGN KEY (StudentID) REFERENCES StudentDim(StudentID),
    FOREIGN KEY (ProgramID) REFERENCES ProgramDim(ProgramID),
    FOREIGN KEY (AdmissionDateKey) REFERENCES DateDim(DateKey),
    FOREIGN KEY (DecisionDateKey) REFERENCES DateDim(DateKey)
)
-- Create Non-Clustered Index on StudentID
CREATE NONCLUSTERED INDEX idx_AdmissionFact_StudentID
ON AdmissionFact (StudentID);

-- Create Non-Clustered Index on ProgramID
CREATE NONCLUSTERED INDEX idx_AdmissionFact_ProgramID
ON AdmissionFact (ProgramID);

-- Create Non-Clustered Index on AdmissionDateKey
CREATE NONCLUSTERED INDEX idx_AdmissionFact_AdmissionDateKey
ON AdmissionFact (AdmissionDateKey);

-- Create Non-Clustered Index on DecisionDateKey
CREATE NONCLUSTERED INDEX idx_AdmissionFact_DecisionDateKey
ON AdmissionFact (DecisionDateKey);

-- Create Composite Non-Clustered Index on StudentID, ProgramID, and
  AdmissionDateKey
CREATE NONCLUSTERED INDEX idx_AdmissionFact_StudentID_ProgramID_AdmissionDateKey
ON AdmissionFact (StudentID, ProgramID, AdmissionDateKey);
```

- **ScholarshipAwardFact**

```sql
-- Create Scholarship Award Fact Table
CREATE TABLE ScholarshipAwardFact (
    ScholarshipAwardID INT PRIMARY KEY,
    StudentID INT,
    ScholarshipID INT,
    AwardDateKey INT,
    Amount DECIMAL(10, 2),
    FOREIGN KEY (StudentID) REFERENCES StudentDim(StudentID),
    FOREIGN KEY (ScholarshipID) REFERENCES ScholarshipDim(ScholarshipID),
    FOREIGN KEY (AwardDateKey) REFERENCES DateDim(DateKey)
)
-- Create Non-Clustered Index on StudentID
CREATE NONCLUSTERED INDEX idx_ScholarshipAwardFact_StudentID
ON ScholarshipAwardFact (StudentID);

-- Create Non-Clustered Index on ScholarshipID
CREATE NONCLUSTERED INDEX idx_ScholarshipAwardFact_ScholarshipID
ON ScholarshipAwardFact (ScholarshipID);

-- Create Non-Clustered Index on AwardDateKey
CREATE NONCLUSTERED INDEX idx_ScholarshipAwardFact_AwardDateKey
ON ScholarshipAwardFact (AwardDateKey);

-- Create Composite Non-Clustered Index on StudentID, ScholarshipID, and
  AwardDateKey
CREATE NONCLUSTERED INDEX
  idx_ScholarshipAwardFact_StudentID_ScholarshipID_AwardDateKey
ON ScholarshipAwardFact (StudentID, ScholarshipID, AwardDateKey);
```

9.Reporting and Analysis

Types of Reports

- **Performance Reports**: Analyzing student performance and faculty workload.
- **Financial Reports**: Reviewing fee collections and scholarship distributions.
- **Enrollment Reports**: Tracking enrollment trends and admission statuses.

Key Performance Indicators (KPIs)

- **Average Grade**: Measures overall student performance.
- **Total Scholarship Amount**: Evaluates scholarship distribution by department.
- **Enrollment Count**: Assesses program popularity and trends.

Reporting Tools and Technologies

- **Reporting Tools**: Tools like Power BI or Tableau for visualizing data.
- **Technologies**: SQL for querying, ETL tools for data integration.

10. Analytical Queries

- **Student Performance Analysis**

```sql
-- Student Performance Analysis
SELECT
    c.CourseName,
    c.CourseCode,
    AVG(CASE
        WHEN g.Grade = 'A' THEN 4.0
        WHEN g.Grade = 'B' THEN 3.0
        WHEN g.Grade = 'C' THEN 2.0
        WHEN g.Grade = 'D' THEN 1.0
        ELSE 0.0
    END) AS AvgGrade
FROM
    GradeFact gf
JOIN
    CourseDim c ON gf.CourseID = c.CourseID
JOIN
    DateDim d ON gf.GradeDateKey = d.DateKey
WHERE
    d.Semester = 'Fall' AND d.Year = 2023
GROUP BY
    c.CourseName, c.CourseCode
ORDER BY
    AvgGrade DESC;
```

- **Scholarship Distribution**

```sql
-- Scholarship Distribution
SELECT
    d.DepartmentName,
    SUM(saf.Amount) AS TotalScholarshipAmount
FROM
    ScholarshipAwardFact saf
JOIN
    StudentDim sd ON saf.StudentID = sd.StudentID
JOIN
    DepartmentDim d ON sd.DepartmentID = d.DepartmentID
GROUP BY
    d.DepartmentName
ORDER BY
    TotalScholarshipAmount DESC;
```

- **Enrollment Trends**

```sql
-- Enrollment Trends
SELECT
    p.ProgramName,
    COUNT(e.StudentID) AS NumberOfEnrollments
FROM
    EnrollmentFact e
JOIN
    ProgramDim p ON e.ProgramID = p.ProgramID
JOIN
    DateDim d ON e.EnrollmentDateKey = d.DateKey
WHERE
    d.Year >= YEAR(GETDATE()) - 3
GROUP BY
    p.ProgramName
ORDER BY
    NumberOfEnrollments DESC;
```

- **Faculty Workload**

```sql
-- Faculty Workload
SELECT
    f.FirstName + ' ' + f.LastName AS FacultyName,
    d.DepartmentName,
    COUNT(c.CourseID) AS NumberOfCourses
FROM
    FacultyDim f
JOIN
    CourseDim c ON f.FacultyID = c.FacultyID
JOIN
    DepartmentDim d ON f.DepartmentID = d.DepartmentID
GROUP BY
    f.FirstName, f.LastName, d.DepartmentName
ORDER BY
    NumberOfCourses DESC;
```

- **Fee Collection Analysis**

```sql
-- Fee Collection Analysis
SELECT
    c.CourseName,
    d.Semester,
    d.Year,
    SUM(ff.FeeAmount) AS TotalFeeCollected
FROM
    FeeFact ff
JOIN
    CourseDim c ON ff.CourseID = c.CourseID
JOIN
    DateDim d ON ff.FeeDateKey = d.DateKey
GROUP BY
    c.CourseName, d.Semester, d.Year
ORDER BY
    TotalFeeCollected DESC;
```

- **Admission Status**

```sql
-- Admission Status
SELECT
    p.ProgramName,
    a.Status,
    COUNT(a.StudentID) AS Count
FROM
    AdmissionFact a
JOIN
    ProgramDim p ON a.ProgramID = p.ProgramID
GROUP BY
    p.ProgramName, a.Status
ORDER BY
    p.ProgramName, a.Status;
```

- **Student Graduation Trends**

```sql
-- Student Graduation Trends
SELECT
    YEAR(sd.GraduationDate) AS GraduationYear,
    COUNT(sd.StudentID) AS NumberOfGraduates
FROM
    StudentDim sd
WHERE
    sd.GraduationDate IS NOT NULL
GROUP BY
    YEAR(sd.GraduationDate)
ORDER BY
    GraduationYear DESC;
```

- **Course Enrollment**

```sql
-- Course Enrollment
SELECT
    c.CourseName,
    COUNT(e.StudentID) AS EnrollmentCount
FROM
    EnrollmentFact e
JOIN
    CourseDim c ON e.CourseID = c.CourseID
GROUP BY
    c.CourseName
ORDER BY
    EnrollmentCount DESC;
```

11.Challenges and Solutions

Data Quality Issues

- **Challenge:** Ensuring data consistency and accuracy across different sources and entities within the data model.
- **Solution:** Implemented thorough data validation rules and constraints within the data model to ensure data integrity. Developed comprehensive data dictionaries to standardize data definitions and formats.

Performance Optimization

- **Challenge:** Optimizing query performance for large datasets within the data warehouse.
- **Solution:** Designed indexing strategies and optimized schema design to improve query efficiency. Applied best practices in dimensional modeling, such as star and snowflake schemas, to facilitate faster data retrieval.

Scalability Concerns

- **Challenge:** Designing a data model that can efficiently handle increasing volumes of data over time.
- **Solution**: Adopted a scalable schema design with consideration for future data growth. Implemented partitioning strategies and designed the data model to support potential expansion in data volume and complexity.

12. Conclusion and Future Work

Summary of Achievements

- **Data Warehouse Design**: Successfully designed a comprehensive data warehouse for Utah State University, including well-structured dimensional models and a robust schema for data analysis.
- **Dimensional Modeling**: Developed detailed star and snowflake schemas to support efficient analytical queries, facilitating enhanced reporting and decision-making capabilities.

Lessons Learned

- **Data Modeling Best Practices**: Emphasized the importance of a well-designed data model in ensuring data consistency and integrity. Identified how effective dimensional modeling can significantly improve query performance and ease of use.

- **Performance Optimization**: Learned the value of applying indexing and schema design best practices to optimize performance for complex queries and large datasets.

Recommendations for Future Enhancements

- **Real-Time Data Processing**: Consider exploring the integration of real-time data processing capabilities to enable more dynamic and up-to-date reporting and analysis.
- **Advanced Analytics**: Investigate the potential of incorporating advanced analytics and machine learning techniques to derive predictive insights and further enhance decision-making processes.