www.unemi.edu.ec

Formato
para **Actividades de Aprendizaje**

TAREA

UNEMI
UNIVERSIDAD ESTATAL DE MILAGRO

UNEMI
UNIVERSIDAD ESTATAL DE MILAGRO

FACULTAD CIENCIAS DE LA INGENIERIA INGENIERIA DE SOFTWARE

MODALIDAD:
Presencial

ASIGNATURA:
Base De Datos

DOCENTE:
RICHARD IVAN RAMIREZ ANORMALIZA

ALUMNOS:
JOEL ALEXANDER ARMIJOS QUEZADA ANDY CHAFLA GUAMAN
BRYAN MIGUEL MARTINEZ MOSQUERA

TEMA:
¨TEMA: Sistema de Gestión Académica¨

CURSO:
4 SEMESTRE -- B-1

PERÍODO LECTIVO
2024-2025

Ejercicios

1. Control de Matriculación

Crear un trigger que verifique el cupo disponible antes de permitir una nueva matrícula.

```
CREATE TRIGGER after_insert_matricula
ON Matricula
AFTER INSERT
AS
BEGIN
   DECLARE @cupo_maximo INT;
   DECLARE @cantidad_actual INT;
   DECLARE @seccion_id INT;
   -- ID de la sección de la nueva matrícula
   SELECT @seccion_id = seccion_id FROM inserted;
   -- Obtener el cupo máximo de la sección
   SELECT @cupo_maximo = cupo_maximo
   FROM Seccion
   WHERE seccion_id = @seccion_id;
   -- Contar la cantidad de matrículas existentes en esa sección
   SELECT @cantidad_actual = COUNT(*)
   FROM Matricula
   WHERE seccion_id = @seccion_id;
   -- Verifica si hay cupo disponible
   IF @cantidad_actual > @cupo_maximo
   BEGIN
      RAISERROR('Cupo lleno, no se puede realizar la matrícula.', 16, 1);
      ROLLBACK TRANSACTION;
   END
END;
```

```
INSERT INTO Matricula (matricula_id, estudiante_id, seccion_id, fecha_matricula, estado)
VALUES (4, 1, 3, '2024-11-13', 'A');
-- ver el restante de cupos
SELECT
    s.seccion_id,
    c.nombre AS curso_nombre,
    p.nombre AS profesor_nombre,
    s.cupo_maximo,
    COUNT(m.matricula_id) AS estudiantes_matriculados,
    (s.cupo_maximo - COUNT(m.matricula_id)) AS cupo_restante
FROM
    Seccion s
JOIN
    Curso c ON s.curso_id = c.curso_id
JOIN
    Profesor p ON s.profesor_id = p.profesor_id
LEFT JOIN
    Matricula m ON s.seccion_id = m.seccion_id
GROUP BY
    s.seccion_id, c.nombre, p.nombre, s.cupo_maximo
ORDER BY
    s.seccion_id;
```

2. Actualización de Calificaciones

Crear un procedimiento almacenado que calcule y actualice la nota final basada en las notas parciales.

```
CREATE PROCEDURE ActualizarNotaFinal
AS
BEGIN
  --  nota final para cada estudiante en la tabla
  UPDATE Calificacion
  SET nota_final = (ISNULL(nota_parcial1, 0) + ISNULL(nota_parcial2, 0)) / 2  -- Promedio de las notas parciales
  WHERE nota_final IS NULL;  -- Solo actualiza si la nota final aún no ha sido calculada
END;

EXEC ActualizarNotaFinal;

SELECT matricula_id, nota_parcial1, nota_parcial2, nota_final
FROM Calificacion;

SELECT * FROM Estudiante e
LEFT JOIN Historico_Academico ha ON e.estudiante_id = ha.estudiante_id;
```

3. Reporte de Rendimiento Académico

Crear una vista que muestre el promedio de calificaciones por estudiante y carrera.

```
CREATE VIEW reporte_rendimiento_academico AS
SELECT
   e.estudiante_id,
   e.nombre AS estudiante_nombre,
   e.apellido AS estudiante_apellido,
   c.nombre AS carrera_nombre,
   AVG(ha.nota_final) AS promedio_calificaciones
FROM
   Estudiante e
JOIN
   Carrera c ON e.carrera_id = c.carrera_id
JOIN
   Historico_Academico ha ON e.estudiante_id = ha.estudiante_id
GROUP BY
   e.estudiante_id, e.nombre, e.apellido, c.carrera_id, c.nombre;

SELECT * FROM reporte_rendimiento_academico;
```

4. Control de Asistencia

Crear un procedimiento almacenado para registrar asistencia masiva.

```
CREATE PROCEDURE RegistrarAsistenciaMasiva
   @fecha_asistencia DATE,
   @asistencia_data NVARCHAR(MAX)
AS
BEGIN
   DECLARE @index INT = 0;
   DECLARE @num_estudiantes INT;
   DECLARE @estudiante_id INT;
   DECLARE @matricula_id INT;
   DECLARE @estado CHAR(1);
   DECLARE @observacion NVARCHAR(200);

   -- Obtener el número de elementos en el JSON (contar cuántos objetos hay)
   SET @num_estudiantes = (SELECT COUNT(*)
                           FROM OPENJSON(@asistencia_data));

   WHILE @index < @num_estudiantes
   BEGIN
      -- Extraer los datos del JSON para cada estudiante
      SET @estudiante_id = JSON_VALUE(@asistencia_data, CONCAT('$[', @index, '].estudiante_id'));
      SET @matricula_id = JSON_VALUE(@asistencia_data, CONCAT('$[', @index, '].matricula_id'));
      SET @estado = JSON_VALUE(@asistencia_data, CONCAT('$[', @index, '].estado'));
      SET @observacion = JSON_VALUE(@asistencia_data, CONCAT('$[', @index, '].observacion'));

      INSERT INTO Asistencia (matricula_id, fecha, estado, observacion)
      VALUES (@matricula_id, @fecha_asistencia, @estado, @observacion);

      -- Incrementar el índice
      SET @index = @index + 1;
   END
END;
```

```
INSERT INTO Asistencia (asistencia_id, matricula_id, fecha, estado, observacion)
VALUES (4, 1, '2024-11-10', 'P', 'Ninguna');  -- Ejemplo insertando explícitamente el 'asistencia_id'

SELECT * FROM Asistencia WHERE fecha = '2024-11-10';

SELECT E.nombre AS estudiante, E.apellido, A.estado AS asistencia, A.observacion
FROM Asistencia A
JOIN Matricula M ON A.matricula_id = M.matricula_id
JOIN Estudiante E ON M.estudiante_id = E.estudiante_id
WHERE A.fecha = '2024-11-10';
```

5. Histórico Académico

Crear un trigger que actualice automáticamente el histórico académico cuando se completa un curso.

```
CREATE TRIGGER ActualizarHistoricoAcademico
ON Calificacion
AFTER INSERT, UPDATE
AS
BEGIN
   -- Obtener los datos de la calificación insertada o actualizada
   DECLARE @matricula_id INT, @nota_final DECIMAL(4,2), @seccion_id INT;

   SELECT @matricula_id = matricula_id, @nota_final = (COALESCE(nota_parcial1, 0) +
COALESCE(nota_parcial2, 0) + COALESCE(nota_final, 0)) / 3
   FROM inserted;

   -- Obtener el seccion_id correspondiente a la matrícula
   SELECT @seccion_id = seccion_id
   FROM Matricula
   WHERE matricula_id = @matricula_id;

   -- Obtener el curso_id a partir del seccion_id
   DECLARE @curso_id INT;
   SELECT @curso_id = curso_id
   FROM Seccion
   WHERE seccion_id = @seccion_id;
```

```
 --semestre y año de la matrícula--
  DECLARE @semestre INT, @año INT;
  SELECT @semestre = semestre, @año = año
  FROM Seccion
  WHERE seccion_id = @seccion_id;

  -- Insertar o actualizar el histórico académico con la nueva calificación
  MERGE INTO Historico_Academico AS target
  USING (SELECT @matricula_id AS matricula_id, @curso_id AS curso_id, @nota_final AS nota_final,
@semestre AS semestre, @año AS año) AS source
  ON target.estudiante_id = (SELECT estudiante_id FROM Matricula WHERE matricula_id =
@matricula_id)
    AND target.curso_id = @curso_id
    AND target.semestre = @semestre
    AND target.año = @año
  WHEN MATCHED THEN
    UPDATE SET target.nota_final = @nota_final
  WHEN NOT MATCHED THEN
    INSERT (estudiante_id, curso_id, nota_final, semestre, año, estado)
    VALUES ((SELECT estudiante_id FROM Matricula WHERE matricula_id = @matricula_id), @curso_id,
@nota_final, @semestre, @año, 'A');
END;


EXEC sp_help 'Historico_Academico';
```