

BASE DE DATOS II

FRGP - UTN



Integrantes:

- Berti, Bruno - Registro: 31837
- Burgos Vega, Nicole – Registro 31711
- Gomez Fara, Valentina - Registro: 31932
- Solís, Hernan Abel – Registro 31703

TABLA DE CONTENIDO

BASE DE DATOS: ENTRENOAPP	2
Estructura de la base de datos:.....	2
DATOS SEMILLA.....	2
PROCEDIMIENTOS ALMACENADOS	3
sp_InsertHistorialEjercicio	3
sp_ListarProfesoresPorEspOTit	5
sp_EliminarAlumnoInactivo	6
TRIGGERS	8
trg_alumno_rutina_dia_completado	8
tr_AsignarProfesorYCrearRutina	9
tr_Eliminar_Alumno	10
VISTAS	12
vw_UsuariosConRolesAsignados	12
vw_CargaAlumnosProfesor	12
vw_ListaProfAlu	13
LINKS	15
Link a repositorio:	15
Link a video:.....	15

Esta base de datos está pensada para dar servicio a una aplicación llamada 'ENTRENOAPP', que gestiona profesores, alumnos, rutinas y ejercicios en el marco de un gimnasio.

ESTRUCTURA DE LA BASE DE DATOS:

La base de datos está compuesta por varias tablas:

- **persona:** contiene todos los datos personales (nombre, apellido, email, teléfono).
- **profesor:** extiende a persona, agregando especialidad, título y fecha de ingreso.
- **alumno:** extiende persona con información física y estado activo/inactivo.
- **rutina_alumno:** relaciona un alumno con un profesor y representa su rutina general.
- **alumno_rutina_dia:** divide la rutina en días, como piernas, espalda, empuje, etc.
- **ejercicio_base:** catálogo de ejercicios generales.
- **ejercicio_asignado:** ejercicios asignados dentro de cada día de rutina.
- **historial_ejercicio:** donde queda registrado todo lo que el alumno completó.
- **rol**
- **persona_rol**

Todas las tablas están relacionadas mediante claves primarias y foráneas, lo cual asegura integridad, consistencia y facilita las operaciones automáticas con triggers.

DATOS SEMILLA

Para poder probar la base, cargamos datos semilla que incluyen:

- Profesores de distintas especialidades.
- Alumnos con diferentes fechas de inicio y estados activos/inactivos.
- Ejercicios base como sentadilla, press banca, peso muerto y otros.
- Rutinas de ejemplo ya asignadas a los alumnos.
- Días de rutina completos (piernas, empuje, tirón).
- Ejercicios asignados dentro de esos días.

Estos datos permiten demostrar fácilmente el funcionamiento de los procedimientos y triggers sin tener que cargar manualmente cada entidad.

```
1  CREATE PROCEDURE sp_InsertHistorialEjercicio
2      @alumno_rutina_dia_id INT
3  AS
4  BEGIN
5      SET NOCOUNT ON;
6
7      INSERT INTO historial_ejercicio (
8          alumno_id,
9          ejercicio_base_id,
10         fecha_rutina_dia,
11         series,
12         repeticiones,
13         peso,
14         observaciones,
15         nombre_apellido_profesor,
16         titulo_rutina_alumno
17     )
18     SELECT
19         ra.alumno_id,
20         ea.ejercicio_base_id,
21         ard.fecha,
22         ea.series,
23         ea.repeticiones,
24         ea.peso,
25         ard.observaciones,
26         CONCAT(p_prof.nombre, ' ', p_prof.apellido),
27         ra.titulo
28     FROM alumno_rutina_dia ard
29     INNER JOIN rutina_alumno ra ON ra.id = ard.rutinaid
30     INNER JOIN profesor prof ON prof.id = ra.profesor_id
31     INNER JOIN persona p_prof ON p_prof.id = prof.persona_id
32     INNER JOIN ejercicio_asignado ea ON ea.rutina_dia_id = ard.id
33     WHERE ard.id = @alumno_rutina_dia_id;
34 END
35 GO
```

Su función es **copiar automáticamente** los ejercicios realizados por un alumno cuando completa un día de su rutina.

Qué hace:

- Recibe el ID del día de rutina (@alumno_rutina_dia_id).
- Obtiene el alumno, el profesor, la fecha y los ejercicios asignados.

- Inserta en historial_ejercicio:
 - Alumno
 - Ejercicio
 - Series
 - Repeticiones
 - Peso
 - Observaciones
 - Fecha
 - Profesor responsable
 - Título de la rutina

En definitiva, este procedimiento construye el **historial** que después sirve para evaluar el progreso del alumno.

```

1  CREATE PROCEDURE ListarProfesoresPorEspOTit
2      @Especialidad VARCHAR(30) = NULL,
3      @Titulo VARCHAR(255) = NULL
4  AS
5  BEGIN
6      SELECT
7          p.id AS ProfesorID,
8          pr.nombre AS Nombre,
9          pr.apellido AS Apellido,
10         pr.email AS Email,
11         pr.telefono AS Teléfono,
12         p.especialidad AS Especialidad,
13         p.titulo AS Título,
14         p.inicio_actividades AS InicioActividades
15     FROM
16         profesor p
17     INNER JOIN
18         persona pr ON p.persona_id = pr.id
19     WHERE
20         -- Filtra por Especialidad si el parámetro NO es NULL
21         (@Especialidad IS NULL OR p.especialidad = @Especialidad)
22     AND
23         -- Filtra por Título si el parámetro NO es NULL
24         (@Titulo IS NULL OR p.titulo LIKE '%' + @Titulo + '%')
25     ORDER BY
26         pr.apellido, pr.nombre;
27 END
28 GO

```

Este procedimiento se ejecuta para obtener un listado de profesores filtrado por su especialidad y/o por su título profesional.

Qué hace:

- Recibe dos parámetros opcionales: @Especialidad y @Titulo. Busca en las tablas profesor y persona los registros que coincidan con los filtros proporcionados. Si el parámetro es NULL, se ignora el filtro correspondiente.
- Recibe los parámetros:
- @Especialidad VARCHAR(30): Filtra por la especialidad exacta del profesor (por ejemplo, 'fuerza', 'hipertrofia', 'funcional').
- @Titulo VARCHAR(255): Filtra por una cadena de texto contenida en el título del profesor (utiliza LIKE).
- Devuelve un conjunto de resultados con los datos personales (Nombre, Apellido, Email, Especialidad, Título) de los profesores que cumplen los criterios. Permite generar un informe parametrizado que facilita la búsqueda y gestión del profesor.

```

1  CREATE PROCEDURE sp_EliminarAlumnoInactivo
2      @AlumnoID INT
3  AS
4  BEGIN
5
6      DECLARE @TieneHistorial BIT;
7      DECLARE @CompletoRutinaUltimos30Dias BIT;
8      DECLARE @PersonaID INT;
9
10     -- Verificar si el alumno tiene algún registro en historial_ejercicio
11     IF EXISTS (SELECT 1 FROM historial_ejercicio WHERE alumno_id = @AlumnoID)
12         SET @TieneHistorial = 1;
13     ELSE
14         SET @TieneHistorial = 0;
15
16     -- Verificar si el alumno completó alguna rutina_dia en los últimos 30 días
17     IF EXISTS (
18         SELECT 1
19         FROM alumno_rutina_dia ard
20         INNER JOIN rutina_alumno ra ON ard.rutinaid = ra.id
21         WHERE ra.alumno_ID = @AlumnoID
22         AND ard.completado = 1
23         AND ard.fecha >= DATEADD(day, -30, GETDATE())
24     )
25         SET @CompletoRutinaUltimos30Dias = 1;
26     ELSE
27         SET @CompletoRutinaUltimos30Dias = 0;
28
29
30
31     SELECT @PersonaID = persona_id FROM alumno WHERE id = @AlumnoID;
32
33
34     -- Eliminación solo si NO tiene Historial o NO completó rutina en los últimos 30 días
35     IF @TieneHistorial = 0 OR @CompletoRutinaUltimos30Dias = 0
36     BEGIN
37
38         IF @PersonaID IS NULL
39         BEGIN
40             SELECT 'Fallo: El AlumnoID proporcionado no existe en la tabla alumno.' AS Resultado;
41             RETURN;
42         END
43

```

```

44
45
46      -- Eliminar registros dependientes de rutina_alumno y alumno_rutina_dia
47      DELETE FROM ejercicio_asignado
48      WHERE rutina_dia_id IN (
49          SELECT id FROM alumno_rutina_dia
50          WHERE rutinaid IN (SELECT id FROM rutina_alumno WHERE alumno_ID = @AlumnoID)
51      );
52
53      DELETE FROM alumno_rutina_dia
54      WHERE rutinaid IN (SELECT id FROM rutina_alumno WHERE alumno_ID = @AlumnoID);
55
56      DELETE FROM rutina_alumno
57      WHERE alumno_ID = @AlumnoID;
58
59      --. Eliminar el usuario asociado (si existe)
60      DELETE FROM usuario_rol
61      WHERE usuario_id IN (SELECT id FROM usuario WHERE persona_id = @PersonaID);
62
63      DELETE FROM usuario
64      WHERE persona_id = @PersonaID;
65
66      --. Eliminar el registro de alumno
67      DELETE FROM alumno
68      WHERE id = @AlumnoID;
69
70      -- Eliminar a la persona (SOLO si no es también profesor)
71      IF NOT EXISTS(SELECT 1 FROM profesor WHERE persona_id = @PersonaID)
72      BEGIN
73          DELETE FROM persona
74          WHERE id = @PersonaID;
75      END
76
77
78      SELECT 'Éxito: El alumno y sus datos asociados fueron eliminados debido a inactividad o falta de historial.' AS Resultado;
79  END
80  ELSE
81  BEGIN
82
83      SELECT 'Fallo: El alumno NO puede ser eliminado. Tiene historial Y completó rutinas en los últimos 30 días.' AS Resultado;
84  END
85  END
86  GO

```

Este procedimiento se ejecuta para realizar la eliminación física de un alumno que cumpla con los criterios de inactividad por 30 días o mas, o falta de historial de actividad.

- Qué hace: Recibe el @AlumnoID del alumno a evaluar. Comprueba si el alumno tiene algún registro en la tabla historial_ejercicio Comprueba si el alumno ha completado alguna rutina (- Videcompletado = 1) en la tabla alumno_rutina_dia durante los últimos 30 días Si el alumno NO tiene historial O NO ha completado rutinas en los últimos 30 días, procede a la eliminación física Elimina en el orden correcto (en cascada): ejercicio_asignado, alumno_rutina_dia, rutina_alumno, usuario_rol, usuario, alumno, y finalmente persona (si esta no es también un profesor).

-Devuelve un mensaje indicando si la eliminación fue exitosa o si el alumno no pudo ser eliminado.

```
1 CREATE TRIGGER trg_alumno_rutina_dia_completado
2 ON alumno_rutina_dia
3 AFTER UPDATE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     DECLARE @id INT;
9
10    SELECT @id = i.id
11    FROM inserted i
12    INNER JOIN deleted d ON i.id = d.id
13    WHERE d.completado = 0 AND i.completado = 1;
14
15    IF @id IS NOT NULL
16    BEGIN
17        EXEC dbo.sp_InsertHistorialEjercicio @alumno_rutina_dia_id = @id;
18    END
19 END
20 GO
```

Este trigger es el que hace que el procedimiento SP_INSERTHISTORIALEJERCICIO se ejecute automáticamente.

Cuándo se dispara:

Después de un UPDATE en la tabla alumno_rutina_dia.

Que detecta:

Que el campo completado cambió de 0 a 1, es decir, que el alumno marcó su día de entrenamiento como finalizado.

Qué hace:

- Toma el ID del día recién actualizado.
- Llama automáticamente al procedimiento sp_InsertHistorialEjercicio.
- Genera los registros correspondientes en historial_ejercicio.

Este trigger conecta el uso real de la aplicación con el guardado automático del historial.

```

1  CREATE TRIGGER tr_AsignarProfesorYCrearRutina
2  ON alumno
3  AFTER INSERT
4  AS
5  BEGIN
6
7      DECLARE @ProfesorAsignadoID INT;
8
9      SELECT TOP 1 @ProfesorAsignadoID = P.id
10     FROM profesor P
11     LEFT JOIN rutina_alumno RA ON P.id = RA.profesor_ID
12     GROUP BY P.id
13     ORDER BY COUNT(RA.alumno_ID) ASC, P.id ASC;
14
15
16     INSERT INTO rutina_alumno (alumno_ID, profesor_ID, titulo, descripcion, fecha_creacion, status)
17     SELECT
18         I.id,
19         @ProfesorAsignadoID,
20         'Rutina Inicial',
21         'Rutina generada automáticamente tras la alta.',
22         GETDATE(),
23         0 -- Status = 0
24     FROM
25         inserted I;
26 END
27 GO

```

- Cuándo se ejecuta:

Después de un INSERT en la tabla alumno.

- Qué detecta: La creacion de un nuevo alumno.
- Qué hace: Identifica al profesor que tiene la menor cantidad de alumnos asignados (en caso de empate, identifica al profesor mas antiguo) Inserta una fila en la tabla rutina_alumno, donde el alumno_id es obtenido de la tabla inserted, el profesor_id es obtenido en el paso anterior, y establece titulo, descripcion, fecha (GETDATE) y status (en 0) por defecto.
- Resultado: Cada alumnmo nuevo es asignado automaticamente a un profesor (de forma equitativa segun la carga de trabajo) y se le crea una rutina inicial con datos establecidos por defecto.

```

1  CREATE TRIGGER tr_Eliminar_Alumno on alumno
2  INSTEAD OF DELETE
3  AS
4  BEGIN
5      -- 1. Soft Delete del alumno.
6      UPDATE alumno
7      SET active = 0
8      WHERE id IN (SELECT id FROM deleted);
9
10     -- 2. Limpiar los Ejercicios Asignados asociados
11     DELETE ejercicio_asignado
12
13     WHERE rutina_dia_id IN (
14         SELECT ard.id FROM alumno_rutina_dia ard
15         WHERE ard.rutinaid IN (
16             SELECT ra.id FROM rutina_alumno ra
17             WHERE ra.alumno_id IN (SELECT id FROM deleted)
18         )
19     );
20
21     -- 3. Limpiar los Días de Rutina asociados
22     DELETE alumno_rutina_dia
23     WHERE rutinaid IN (
24         SELECT ra.id FROM rutina_alumno ra
25         WHERE ra.alumno_id IN (SELECT id FROM deleted)
26     );
27
28     -- 4. Limpiar la tabla de Rutina asociados
29     DELETE rutina_alumno
30     WHERE alumno_id IN (SELECT id FROM deleted);
31
32 END

```

- Cuándo se ejecuta: En lugar de un DELETE en la tabla alumno (INSTEAD OF DELETE).

Qué detecta: Un intento de eliminar una o más filas de la tabla alumno.

- Qué hace: Implementa una eliminación suave (Soft Delete) en la tabla alumno y realiza una eliminación en cascada de todos los datos asociados al alumno en las tablas de rutina. Soft Delete en alumno: Actualiza el campo active a 0 para el/los alumno(s) que se intentaban eliminar.

Limpieza de ejercicio_asignado: Elimina todos los ejercicios asignados que estén asociados indirectamente a las rutinas de los alumnos dados de baja.

Limpieza de alumno_rutina_dia: Elimina los días de rutina asociados a las rutinas de los alumnos.

Limpieza de rutina_alumno: Elimina las rutinas creadas para los alumnos.

- Resultado:
- En lugar de eliminar físicamente al alumno, se desactiva (se marca como active = 0) para preservar su registro. Al mismo tiempo, se eliminan físicamente todos sus datos transaccionales relacionados con rutinas y ejercicios asignados para mantener la base de datos limpia.

VW_USUARIOSCONROLESASIGNADOS

```

1  CREATE VIEW vw_UsuariosConRolesAsignados AS
2  SELECT
3      U.id AS UsuarioID,
4      U.nombre_usuario AS NombreUsuario,
5      R.nombre_rol AS RolAsignado
6  FROM
7      usuario U
8  INNER JOIN
9      usuario_rol UR ON U.id = UR.usuario_id
10 INNER JOIN
11     rol R ON UR.rol_id = R.id;
12 GO
13

```

- Qué hace: Esta vista cuenta cuántos alumnos tiene asignado cada profesor, utilizando la tabla rutina_alumno. Se utilizan las tablas profesor, persona y rutina_alumno.

El LEFT JOIN con rutina_alumno garantiza que si un profesor no tiene datos cargados en la tabla de rutinas (es decir, no tiene alumnos asignados), seguirá apareciendo en la lista, y la función COUNT le asignará un 0. Cuenta el número de filas en rutina_alumno que están asociadas a cada profesor. Agrupa los resultados por los datos del profesor para que la función COUNT pueda calcular la suma para cada grupo individual.

VW_CARGAALUMNOSPROFESOR

```

1  CREATE VIEW vw_CargaAlumnosProfesor AS
2  SELECT
3      P.id AS ProfesorID,
4      PE.nombre AS NombreProfesor,
5      PE.apellido AS ApellidoProfesor,
6      P.especialidad AS Especialidad,
7      COUNT(RA.alumno_id) AS CantidadAlumnosAsignados
8  FROM
9      profesor P
10 INNER JOIN
11     persona PE ON P.persona_id = PE.id
12 LEFT JOIN
13     rutina_alumno RA ON P.id = RA.profesor_id
14 GROUP BY
15     P.id, PE.nombre, PE.apellido, P.especialidad;
16 GO

```

- Qué hace: Esta vista muestra el usuario con su respectivo rol de permiso asignado.

Muestra el nombre de usuario y el rol asignado (ej. 'admin', 'profesor', 'alumno') a cada usuario simplificando la gestión de accesos.

Utiliza las tablas usuario, la tabla intermedia usuario_rol y la tabla rol.

La vista conecta estas tres tablas, asegurando que solo se muestren usuarios que tienen al menos un rol asignado.

Conecta el usuario_id con el rol_id a través de la tabla usuario_rol.

Convierte el id del rol en su nombre (nombre_rol) para facilitar la lectura

VW_LISTAPROFALU

```
1  CREATE VIEW VW_LISTAPROFALU AS
2  SELECT
3      -- COLUMNAS PROFESOR
4      perPro.nombre AS ProfNombre,
5      perPro.apellido AS ProfApellido,
6      p.titulo AS ProfTitulo,
7      perPro.telefono AS ProfTelefono,
8      perPro.email AS ProfEmail,
9
10     -- COLUMNAS ALUMNO
11     perAlu.nombre AS AluNombre,
12     perAlu.apellido AS AluApellido,
13     DATEDIFF(YEAR, perAlu.fecha_nac, GETDATE()) AS AluEdad,
14     perAlu.telefono AS AluTelefono,
15     perAlu.email AS AluEmail,
16     a.fecha_fin_suscripcion as AluFinSus
17
18     --TABLA PUENTE
19 FROM rutina_alumno ra
20
21     -- ENLASA EL PUENTE CON EL PROFE MAS DATOS PERSONALES DEL PROF
22 INNER JOIN profesor p
23     ON ra.profesor_id = p.id
24 INNER JOIN persona perPro
25     ON p.persona_id = perPro.id
26
27     -- ENLASA EL PUENTE CON EL ALU MAS DATOS PERSONALES DEL ALU
28 INNER JOIN alumno a
29     ON ra.alumno_id = a.id
30 INNER JOIN persona perAlu
31     ON a.persona_id = perAlu.id
32
33     --FILTRO: SOLO SI ALU ESTA ACTIVO
34 WHERE a.active = 1;
```

Qué hace: Esta vista muestra una lista detallada que relaciona cada profesor con sus alumnos activos. Devuelve información personal tanto del profesor como del alumno.

Qué incluye:

Datos del profesor: nombre, apellido, título, teléfono y email.

Datos del alumno: nombre, apellido, edad (calculada con DATEDIFF), teléfono, email y fecha de fin de suscripción.

Solo se incluyen alumnos con active = 1 (alumnos activos).

Tablas utilizadas:

rutina_alumno (relación principal entre profesor y alumno)

profesor

persona (para profesor)

alumno

persona (para alumno)

Propósito: Permite obtener rápidamente un listado de profesores con sus respectivos alumnos activos, útil para reportes, paneles administrativos o asignación de seguimiento.

LINKS

LINK A REPOSITORIO:

https://github.com/nikiburgos/tp_base_de_datos_grupo_70

LINK A VIDEO:

Compartimos dos links. **Es el mismo video**, pero entregamos los dos links en caso de que alguno no funcione correctamente.

https://www.canva.com/design/DAG49XcfGJY/zwICrM4j9ybOBBnE_3rluQ/watch?utm_content=DAG49XcfGJY&utm_campaign=designshare&utm_medium=link2&utm_source=uniqueLinks&utlId=h2a89faeadb

o:

<https://drive.google.com/file/d/1939vZgBuf5ZLwh8U6LMYYAJypxJrmHqa/view?usp=sharing>