Lab1 - Creación de esquema relacional y datos

Pepe Calderón, Fernando Sola, Daniel Ayala, Inma Hernández, Margarita Cruz, Carlos Arévalo, David Ruiz

Escuela Técnica Superior de
Ingeniería Informática

Índice

1

Objetivo

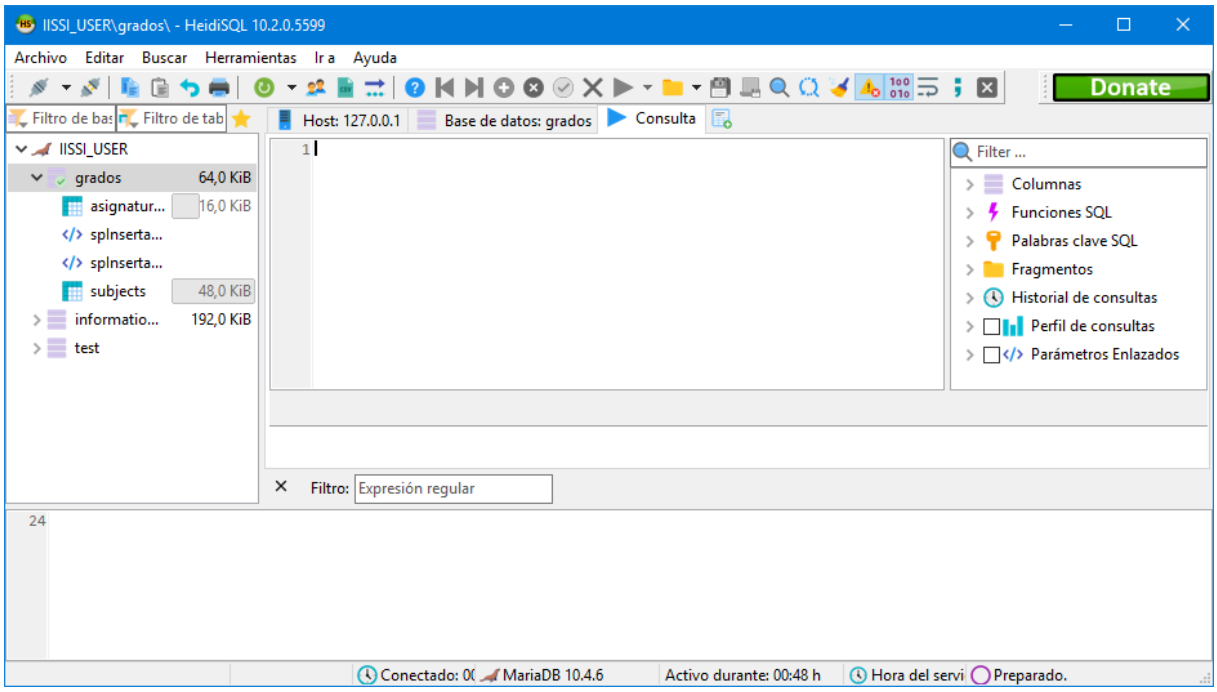
El objetivo de esta práctica es aprender las instrucciones SQL para crear el esquema relacional de una base de datos con una carga de datos inicial. El alumno aprenderá a:

- Crear tablas mediante instrucciones SQL.
- Definir atributos con diferentes tipos de datos.
- Definir claves primarias.
- Definir claves ajenas e implementar relaciones 1:N y N:M.
- Definir campos con autoincremento.
- Definir valores por defecto para campos.
- Insertar valores iniciales

2 Preparación del entorno

Si se ha instalado MariaDB siguiendo el procedimiento mostrado en el anexo de este documento, el SGBD ha quedado registrado como un servicio Windows, por lo que debería iniciarse automáticamente con el sistema operativo.

Iniciamos HeidiSQL y nos conectamos con el usuario iissi_user mediante la conexión creada en el laboratorio anterior. Seleccionamos la base de datos grados y abrimos la pestaña de Consulta. En esa pestaña podemos escribir código SQL para que se ejecute en la base de datos seleccionada. El contenido de la pestaña puede escribirse manualmente o ser un archivo .sql cargado. En este laboratorio trabajaremos con los archivos `createdb.sql` y `populatedb.sql`.


IISSI_USER\grados\ - HeidiSQL 10.2.0.5599
Archivo
Editar
Buscar
Herramientas
Ir a
Ayuda
Donate
Filtro de bas
Filtro de tab
Host: 127.0.0.1
Base de datos: grados
Consulta
IISSI_USER
grados 64,0 KiB
asignatur... 16,0 KiB
spInserta...
spInserta...
subjects 48,0 KiB
informatio... 192,0 KiB
test
Filter ...
Columnas
Funciones SQL
Palabras clave SQL
Fragmentos
Historial de consultas
Perfil de consultas
Parámetros Enlazados
Filtro:
Expresión regular
Conectado: 0(
MariaDB 10.4.6
Activo durante: 00:48 h
Hora del servi
Preparado.

3 Control de versiones (GradesDB)

Para realizar el trabajo, crearemos un repositorio llamado `GradesDB` y haremos commits tras cada paso importante.

1. En tu carpeta de trabajo añade los dos scripts del laboratorio, inicialmente en blanco:
   - `createdb.sql`
   - `populatedb.sql`
2. Inicializa el repositorio y haz un primer commit:
   - `git init`
   - `git add createdb.sql populatedb.sql` o `git add -A`
   - `git commit -m "Primer commit, inicializa scripts de base de datos"`
3. A partir de aquí, tras completar cada tabla, haz commit:
   - `git add -A`
     `git commit -m "Añadida tabla *****"`
4. Al finalizar, crea el repositorio remoto `GradesDB` en tu plataforma GitHub (github.com p github.eii.us.es) y haz el push:
   - `git remote add origin <URL_DEL_REPO>`
   - `git branch -M main`
   - `git push -u origin main`

4 Consideraciones de estilo

SQL no distingue entre mayúsculas y minúsculas, por lo que es posible que código de diferentes fuentes las usen de forma diferente. Nosotros seguiremos las siguientes normas para unificar el estilo del código:

- Las palabras reservadas del lenguaje SQL (es decir, todo lo que no son nombres definidos por nosotros) se escribirán enteramente en mayúsculas. Por ejemplo: `INSERT, UPDATE, DELETE`.
- Los nombres de tablas se escribirán en minúsculas, en `snake_case` [1] si están formados por varias palabras, y en plural. Por ejemplo: `degrees, people, professors, subjects, teaching_loads, ...`.
- Los nombres de atributos, constraints, y otros elementos que no sean tablas, se escribirán completamente en minúsculas. Por ejemplo: `degree_id. group_name, with_honors, ...`
- El código estará escrito en inglés.

[1] El estilo `snake_case` es una convención para nombrar variables, funciones y otros identificadores en programación. Se caracteriza por: todas las letras en minúsculas, palabras separadas por guiones bajos (_), sin espacios ni caracteres especiales.

5

Intensión relacional

Aplicado las transformaciones MC->MR vistas en clase una posible intensión relacional sería la siguiente:

```
Personas(personaId, dni, nombre, apellidos, edad, email)
    PK(personaId)

Alumnos(alumnoId, métodoAcceso)
    PK(alumnoId)
    FK(alumnoId) / Personas

Profesores(profesorId, categoría)
    PK(profesorId)
    FK(profesorId) / Personas

Grados(gradoId, nombre, años)
    PK(gradoId)

Asignaturas(asignaturaId, gradoId, nombre, acrónimo, créditos, curso, tipo)
    PK(asignaturaId)
    FK(gradoId) / Grados

Grupos(grupoId, asignaturaId, nombre, actividad, añoAcadémico)
    PK(grupoId)
    FK(asignaturaId) / Asignaturas

Notas(notaId, alumnoId, grupoId, valor, convocatoria, esMH)
    PK(notaId)
    FK(alumnoId) / Alumnos
```

```
    FK(grupoId) / Grupos

GruposAlumnos(grupoAlumnoId, alumnoId, grupoId)
    PK(grupoAlumnoId)
    FK(alumnoId) / Alumnos
    FK(grupoId) / Grupos

Cargas(cargaId, profesorId, grupoId, créditos)
    PK(cargaId)
    FK(profesorId) / Profesores
    FK(grupoId) / Grupos

Matrículas(matrículaId, alumnoId, asignaturaId)
    PK(matrículaId)
    FK(alumnoId) / Alumnos
    FK(asignaturaId) / Asignaturas
```

6

Extesión relacional

```
Grados = {
    (1, "Ingeniería del Software", 4),
    (2, "Ingeniería de Computadores", 4),
    (3, "Tecnologías Informáticas", 4)
}

Asignaturas = {
    (1, 3, "Fundamentos de Programación", "FP", 12, 1, "Formación Básica"),
    (2, 3, "Cálculo Infinitesimal y Numérico", "CIN", 6, 1, "Formación
Básica"),
    (10, 3, "Análisis y Diseño de Datos y Algoritmos", "ADDA", 12, 2,
"Obligatoria"),
    (11, 3, "Introducción a la Ingeniería del Software y los Sistemas de
Información I", "IISSI-1", 6, 2, "Obligatoria"),
    (15, 3, "Introducción a la Ingeniería del Software y los Sistemas de
Información II", "IISSI-2", 6, 2, "Obligatoria")
}

Grupos = {
    (1, 11, "T1", "Teoría", 2024),
    (2, 11, "L1", "Laboratorio", 2024),
    (3, 11, "L2", "Laboratorio", 2024)
}

Personas = {
    (1, "00000001A", "David", "Ruiz", 50, "druiz@us.es"),
    (2, "00000002B", "Inma", "Hernández", 40, "inmahernandez@us.es"),
```

```
    (3, "00000003C", "Fernando", "Sola", 28, "fsola@us.es"),
    (6, "10000006F", "David", "Romero", 22, "david.romero@alum.us.es"),
    (7, "10000007G", "Lucía", "Molina", 21, "lucia.molina@alum.us.es"),
    (8, "10000008H", "Hugo", "Paredes", 20, "hugo.paredes@alum.us.es"),
    (16, "10000016R", "Pablo", "Vidal", 22, "pablo.vidal@alum.us.es"),
    (25, "10000025A", "Paula", "Bermejo", 21, "paula.bermejo@alum.us.es")
}

Profesores = {
    (1, "Catedrático"),
    (2, "Titular"),
    (3, "AyudanteDoctor")
}

Alumnos = {
    (6, "Selectividad"),
    (7, "Selectividad"),
    (8, "Selectividad"),
    (16, "Selectividad"),
    (25, "Selectividad")
}

Matrículas = {
    (1, 6, 11),
    (2, 7, 11),
    (3, 8, 11),
    (4, 16, 11),
    (5, 25, 11)
}

GruposAlumnos = {
    (1, 6, 1),
    (2, 7, 1),
    (3, 8, 1),
    (4, 16, 1),
    (5, 25, 1),
    (6, 6, 2),
    (7, 7, 2),
    (8, 8, 2),
    (9, 16, 3),
    (10, 25, 3)
}

Cargas = {
    (1, 1, 1, 3.0),
    (2, 2, 1, 3.0),
    (3, 3, 2, 3.0)
}

Notas = {
```

```
    (1, 6, 1, 9.8, "Primera", 1),
    (2, 7, 1, 9.2, "Primera", 1),
    (3, 8, 1, 5.4, "Primera", 0),
    (11, 16, 1, 4.2, "Primera", 0),
    (20, 25, 1, 4.7, "Primera", 0),
    (21, 16, 1, 6.2, "Segunda", 0),
    (30, 25, 1, 4.3, "Segunda", 0),
    (35, 25, 1, 5.7, "Tercera", 0)
}
```

La extensión relacional representa varios escenarios que ilustran el funcionamiento del sistema de gestión académica:

Escenario 1: Organización académica

- El grado de "Tecnologías Informáticas" (id=3) contiene asignaturas de primer curso (FP, CIN) y segundo curso (IISSI-1, IISSI-2, ADDA)
- La asignatura IISSI-1 está organizada en 3 grupos: 1 de teoría (T1) y 2 de laboratorio (L1, L2)

Escenario 2: Jerarquía de personas

- Se definen 8 personas en el sistema, diferenciadas entre profesores (ids 1-3) y alumnos (ids 6-8, 16, 25)
- Los profesores tienen diferentes categorías: Catedrático (David Ruiz), Titular (Inma Hernández) y AyudanteDoctor (Fernando Sola)
- Todos los alumnos accedieron al grado por Selectividad

Escenario 3: Asignación docente

- El grupo de teoría T1 tiene docencia compartida entre dos profesores (Catedrático y Titular), con 3 créditos cada uno
- El grupo de laboratorio L1 es impartido por un único profesor (AyudanteDoctor) con 3 créditos
- Cumple la restricción RN03: ningún grupo tiene más de 2 profesores

Escenario 4: Matrículas y pertenencia a grupos

- Los 5 alumnos están matriculados en IISSI-1 y pertenecen al grupo de teoría T1
- Los alumnos se distribuyen en laboratorios: 3 en L1 (ids 6, 7, 8) y 2 en L2 (ids 16, 25)
- Cumple RN04 y RN07: cada alumno pertenece a un solo grupo de teoría y un solo grupo de laboratorio de la asignatura en la que está matriculado

Escenario 5: Evaluación y convocatorias

- **Primera convocatoria**: 2 alumnos obtienen matrícula de honor (9.8 y 9.2), 1 aprobado (5.4) y 2 suspensos (4.2 y 4.7)
- **Segunda convocatoria**: El alumno 16 mejora su nota de 4.2 a 6.2 (aprueba), mientras que el alumno 25 se mantiene suspenso (4.3)
- **Tercera convocatoria**: El alumno 25 finalmente aprueba con un 5.7
- Cumple RN01: cada alumno tiene una única nota por convocatoria

- Cumple RN05: los cambios de nota entre convocatorias no superan 4 puntos (ej: alumno 16 pasa de 4.2 a 6.2, diferencia de 2 puntos)
- Cumple RN001: las matrículas de honor tienen valores ≥ 9

7

Creación de tabla `people` (Personas)

Para ello escribimos el siguiente código en `createdb.sql`:

```
DROP TABLE IF EXISTS people;

CREATE TABLE people (
    person_id INT AUTO_INCREMENT,
    dni CHAR(9) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(150) NOT NULL,
    age TINYINT NOT NULL,
    email VARCHAR(255) NOT NULL,
    PRIMARY KEY (person_id)
);
```

Observe lo siguiente:

- La primera línea, `DROP TABLE IF EXISTS people;`, se encarga de que se borre la tabla si ya existía. Sin esta línea, se produciría un error si intentamos crear la tabla cuando ya existe (por ejemplo, porque haciendo pruebas queramos ejecutar el script varias veces). Es conveniente que, en un script de creación de tablas, lo primero que se haga sea eliminar todas las tablas si existen.
- El contenido dentro de `CREATE TABLE people` contiene la definición de la tabla. Separamos con comas la definición de atributos o restricciones.
- `INT` y `VARCHAR` son tipos de datos. `VARCHAR` corresponde a un String, y el número indicado entre paréntesis es el número máximo de caracteres.

- En la última línea indicamos cuál es la clave primaria. Se indican entre paréntesis los atributos que forman parte de ella. Si una clave primaria estuviera formada por varios atributos, se incluirían todos separados por comas.
- Nos aseguramos de que los nombres de los IDs sean siempre diferentes (por ejemplo: degree_id, subject_id). Darle distintos nombres a los ID de todas las tablas evita problemas en caso de despistes.
- A los atributos marcados como `AUTO_INCREMENT` se les dará valor automáticamente siguiendo una secuencia. Los atributos marcados con `AUTO_INCREMENT` deben ser una clave, ya sea primaria o alternativa.

Para insertar datos en esta tabla escribiremos el siguiente código SQL en el archivo `populatedb.sql`:

```
 -- Primero eliminados todos los datos de tabla en el caso de que los
hubiese y después insertamos los datos iniciales.
DELETE FROM people;

INSERT INTO people (person_id, dni, first_name, last_name, age, email)
VALUES
  (1, '00000001A', 'David', 'Ruiz', 50, 'druiz@us.es'),
  (2, '00000002B', 'Inma', 'Hernández', 40, 'inmahernandez@us.es'),
  (3, '00000003C', 'Fernando', 'Sola', 28, 'fsola@us.es'),
  (4, '00000004D', 'Daniel', 'Ayala', 32, 'dayala1@us.es'),
  (5, '00000005E', 'Pepe', 'Calderón', 43, 'pepecalderon@us.es'),
  (6, '10000006F', 'David', 'Romero', 22, 'david.romero@alum.us.es'),
  (7, '10000007G', 'Lucía', 'Molina', 21, 'lucia.molina@alum.us.es'),
  (8, '10000008H', 'Hugo', 'Paredes', 20, 'hugo.paredes@alum.us.es'),
  ...
  (25, '10000025A', 'Paula', 'Bermejo', 21, 'paula.bermejo@alum.us.es');
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla people"`). {: .notice–info}

8

Creación de tabla `professors` (Profesores)

Ahora escribiremos en `createdb.sql` el siguiente código:

```
SET FOREIGN_KEY_CHECKS = 0;
DROP TABLE IF EXISTS people;
DROP TABLE IF EXISTS professors;
SET FOREIGN_KEY_CHECKS = 1;

-- Definción tabla people

CREATE TABLE professors (
    professor_id INT,
    category VARCHAR(30) NOT NULL,
    PRIMARY KEY (professor_id),
    FOREIGN KEY (professor_id) REFERENCES people(person_id)
);
```

Añada la eliminación de la tabla si existe antes de crearla, es importante poner la variable `FOREING_KEY_CHECKS` a falso mientras borramos todas las tablas. Observe lo siguiente:

- Para definir una clave ajena, incluimos un atributo **del mismo tipo** que la clave que queremos referenciar, y usamos la sentencia `FOREIGN KEY`. La sentencia recibe primero los atributos que forman parte de la clave ajena, la tabla a la que referencian, y los atributos de la tabla que se referencian. Estos últimos deben ser una clave primaria o alternativa.

- El nombre de la clave ajena no tiene que ser igual que el de la clave primaria referenciada, aunque en ocasiones puede ser así.
- Si una clave ajena está formada por varios atributos, se escribirían entre paréntesis y separados por comas.
- En este caso hemos implementado una relación de herencia.
- La existencia de claves ajenas puede crear problemas, por ejemplo, si se intenta borrar o modificar una fila que es referenciada por otra mediante una clave ajena. Por defecto, no se permiten eliminar filas referenciadas en otras tablas.

Para la inserción de datos en la tabla `professors` escribiremos el siguiente código SQL en `populatedb.sql`:

```
SET FOREIGN_KEY_CHECKS = 0;
DELETE FROM people;
DELETE FROM professors;
SET FOREIGN_KEY_CHECKS = 1;

INSERT INTO professors (professor_id, category) VALUES
        (1, 'Catedrático'),
        (2, 'Titular'),
        (3, 'AyudanteDoctor'),
    ... ;
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla professors"`). {: .notice–info}

9

Creación de tabla `students` (Alumnos)

Continuemos con el archivo `createdb.sql`:

```
-- Recuerde hacer el DROP de esta tabla al principio del script

-- definición de people

-- definición de professors

CREATE TABLE students (
    student_id INT,
    access_method VARCHAR(20) NOT NULL,
    PRIMARY KEY (student_id),
    FOREIGN KEY (student_id) REFERENCES people(person_id)
);
```

Fíjese que aún no estamos definiendo ningún tipo de restricción sobre los datos, eso lo dejaremos para el siguiente laboratorio. Para añadir datos a esta tabla, escribiremos en `populatedb.sql` los siguiente:

```
-- Recuerde hacer primero el DELETE de los datos.

INSERT INTO students (student_id, access_method) VALUES
        (6, 'Selectividad'),
        (7, 'Selectividad'),
        (8, 'Selectividad'),
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla students"`). {: .notice–info}

10

Creación de la tabla `degrees` y `subjects` (Grados y Asignaturas)

Ahora añadiremos la definición de ambas tablas, como siempre el esquema estará en `createdb.sql` y los datos en `populatedb.sql`. En aras de la simplicidad mostraremos el código que hay que escribir en ambos archivos en el siguiente fragmento de código SQL:

```
CREATE TABLE degrees (
    degree_id INT AUTO_INCREMENT,
    degree_name VARCHAR(80) NOT NULL,
    duration_years TINYINT NOT NULL,
    PRIMARY KEY (degree_id)
);

INSERT INTO degrees (degree_id, degree_name, duration_years) VALUES
        (1, 'Ingeniería del Software', 4),
        (2, 'Ingeniería de Computadores', 4),
        (3, 'Tecnologías Informáticas', 4);

CREATE TABLE subjects (
    subject_id INT AUTO_INCREMENT,
    degree_id INT NOT NULL,
    subject_name VARCHAR(120) NOT NULL,
    acronym VARCHAR(12) NOT NULL,
    credits TINYINT NOT NULL,
    course TINYINT NOT NULL,
    subject_type VARCHAR(30) NOT NULL,
```

```
    PRIMARY KEY (subject_id),
    FOREIGN KEY (degree_id) REFERENCES degrees(degree_id)
);

INSERT INTO subjects (subject_id, degree_id, subject_name, acronym,
credits, course, subject_type) VALUES
        -- Primer curso (Tecnologías Informáticas)
        (1, 3, 'Fundamentos de Programación', 'FP', 12, 1, 'Formación
Básica'),
        (2, 3, 'Cálculo Infinitesimal y Numérico', 'CIN', 6, 1, 'Formación
Básica'),
        (3, 3, 'Circuitos Electrónicos Digitales', 'CED', 6, 1, 'Formación
Básica'),
        ...
        -- Segundo curso (Tecnologías Informáticas)
        (10, 3, 'Análisis y Diseño de Datos y Algoritmos', 'ADDA', 12, 2,
'Obligatoria'),
        (11, 3, 'Introducción a la Ingeniería del Software y los Sistemas
de Información I', 'IISSI-1', 6, 2, 'Obligatoria'),
        (12, 3, 'Matemática Discreta', 'MD', 6, 2, 'Obligatoria'),
        ...
    ;
```

Tenga cuenta las siguiente consideraciones:

- La relación en `people` y `professors` o `students` es una relación 1:1 que implementamos haciendo que la clave primaria de `professors` o `students` también sea clave ajena que apunta a `people`, de estaa forma conseguimos que los profesores y estudiantes sean personas desde el punto de vista del modelo relacional.
- La asociación entre `degrees` y `subjects` es 1:N, es decir, un grado tiene varias asignaturas, pero uuna asignatura pertenece a un único grado, en este caso la tabla que juega el rol N debe contener como clave ajena la clave primaria de la tabla que juega el rol 1.

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadidas tablas degrees y subjects"`). {: .notice–info}

11

Creación de tabla `groups` y `groups_enrollments` (Grupos y AlumnoGrupo)

En este caso en createdb.sql debemos escribir el siguiente código SQL:

```
-- recuerde siempre hacer el DROP de las tablas al priincipio del script

CREATE TABLE groups (
    group_id INT AUTO_INCREMENT,
    subject_id INT NOT NULL,
    group_name VARCHAR(40) NOT NULL,
    activity VARCHAR(15) NOT NULL,
    academic_year YEAR NOT NULL,
    PRIMARY KEY (group_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);

CREATE TABLE group_enrollments (
    student_id INT,
    group_id INT,
    PRIMARY KEY (student_id, group_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (group_id) REFERENCES groups(group_id)
);
```

Fíjese ahora que tenemos una relación N:M, de forma que un alumno pertenece a varios grupos y en un grupo hay varios alumnos, además también existe una relación 1:N entre asignatura y grupo, por lo que es necesatio disponer de un atributo `subject_id` que sea clave ajena de `subjects`.

Para implementar las relaciones N:M siempre es necesario crear una tabla intermedia que relacione las tablas, en este caso la tabla es `group_enrollments` y debe tener como claves ajenas a `student_id` y a `group_id`, además, la pareje de claves ajenas también es clave primaria, de esta forma se consigue que un alumno no pueda pertenecer al mismo grupo, y que un grupo no tenga alumnos repetidos.

Para la inserción de datos escribiremos el siguiente fragmento de código:

```
-- IISSI-1 de TI tiene un grupo de teoría y dos grupos de laboratorio
INSERT INTO groups (group_id, subject_id, group_name, activity,
academic_year) VALUES
        (1, 11, 'T1', 'Teoría', 2024),
        (2, 11, 'L1', 'Laboratorio', 2024),
        (3, 11, 'L2', 'Laboratorio', 2024);

-- 20 alumnos están en el grupo de teoría
-- los 10 primeros están el L1
-- los 10 restantes en el L2
INSERT INTO group_enrollments (student_id, group_id) VALUES
        -- Teoría
        (6, 1), (7, 1), (8, 1), (9, 1), (10, 1),
        ...
        -- Laboratorio L1
        (6, 2), (7, 2), (8, 2), (9, 2), (10, 2),
        ...
        -- Laboratorio L2
        ...
        (21, 3), (22, 3), (23, 3), (24, 3), (25, 3);
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadidas tablas groups y group_enrollments"`). {: .notice–info}

12

Creación de la tabla `grades` (Notas)

Escribimos el siguiente código SQL en `createdb.sql`:

```
CREATE TABLE grades (
    grade_id INT AUTO_INCREMENT,
    student_id INT NOT NULL,
    group_id INT NOT NULL,
    grade_value DECIMAL(4,2) NOT NULL,
    exam_call VARCHAR(20) NOT NULL,
    with_honors BOOLEAN NOT NULL DEFAULT 0,
    PRIMARY KEY (grade_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (group_id) REFERENCES groups(group_id)
);
```

Observe ahora que `student_id` y `group_id` son claves ajenas de `students` y `groups`, respectivamente.

Un ejemplo de datos inciales puede ser el siguiente:

```
INSERT INTO grades (grade_id, student_id, group_id, grade_value, exam_call,
with_honors) VALUES
        -- Primera convocatoria:
        (1, 6, 1, 9.8, 'Primera', 1),
        (2, 7, 1, 9.2, 'Primera', 1),
        (3, 8, 1, 5.4, 'Primera', 0),
        (4, 9, 1, 8.0, 'Primera', 0),
    ...
```

```
        -- Segunda convocatoria
        (21, 16, 1, 6.2, 'Segunda', 0),
        (22, 17, 1, 5.8, 'Segunda', 0),
    ...
        -- Tercera convocatoria
        (31, 21, 1, 5.6, 'Tercera', 0),
        ...
        (35, 25, 1, 5.7, 'Tercera', 0);
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla grades"`). {: .notice–info}

13

Creación de tabla `subject_enrollments`

Para implementar la relación N:_M entre alomnos y las asignaturas en las que se matriculan, escribiremos la siguiente tabla:

```
CREATE TABLE subject_enrollments (
    student_id INT,
    subject_id INT,
    PRIMARY KEY (student_id, subject_id),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (subject_id) REFERENCES subjects(subject_id)
);
```

En aras de la simplicidad vamos a introducir datos iniciales sólo para alumnos de IISSI-1 de TI:

```
INSERT INTO subject_enrollments (student_id, subject_id) VALUES
        (6, 11), (7, 11), (8, 11), (9, 11), (10, 11),
        (11, 11), (12, 11), (13, 11), (14, 11), (15, 11),
        (16, 11), (17, 11), (18, 11), (19, 11), (20, 11),
        (21, 11), (22, 11), (23, 11), (24, 11), (25, 11);
```

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla subject_enrollments"`). {: .notice–info}

14

Creación de la tabla `teaching_loads`

Las clases asociación se implemetan con la misma estrategía que las relaciones N:M, en este caso esta tabla debe tenemoos como claves ajenas la claves primarias de profesor y grupo y disponer de un atributo `credits` cuyo valor caracteriza la asocición entre el profesor y el grupo.

En este caso el fragmento SQL quedaría:

```
CREATE TABLE teaching_loads (
    professor_id INT,
    group_id INT,
    credits DECIMAL(4,1) NOT NULL,
    PRIMARY KEY (professor_id, group_id),
    FOREIGN KEY (professor_id) REFERENCES professors(professor_id),
    FOREIGN KEY (group_id) REFERENCES groups(group_id)
);

INSERT INTO teaching_loads (professor_id, group_id, credits) VALUES
  (1, 1, 3.0),
  (2, 1, 3.0),
  (3, 2, 3.0),
  (5, 3, 1.5),
  (4, 3, 1.5);
```

Fijese que en este caso también es necesario que la pareja de claves ajenas sea también clave primaria para evitar asignar carga docente repetida, es decir, el mismo profesor con distintas cargas para un mismo grupo.

En este caso lo datos reflejan que el profesores 1 y 2 impareten 3 créditos en el grupo de teoría, el profesor 3 imparte 3 créditos en el grupo 1 de laboratorio, y los profesores 4 y 5 imparten 1.5 créditos en el grupo 2 de laboratorio.

Recuerda: al finalizar este apartado, haz commit (por ejemplo: `git add -A && git commit -m "Añadida tabla teaching_loads"`). {: .notice–info}

> **Info**
>
> Versión PDF disponible