SQL Server

1. Instalación en WSL

1.1 Pasos generales para instalar MS SQL en WSL (Ubuntu/Debian):

1. **Actualizar paquetes:** Abre tu terminal WSL y ejecuta

   sudo apt update && sudo apt upgrade.
2. **Instalar SQL Server:** Ejecuta

   sudo apt-get install -y mssql-server.
3. **Paramos el servicio SQL Server:** Ejecuta

   sudo systemctl stop mssql-server
4. **Configurar SQL Server:** Ejecuta el asistente de configuración con:

   sudo /opt/mssql/bin/mssql-conf setup
5. Selecciona la edición (Developer o Express son gratuitas) y Preferiblemente en ingles.
6. Establece una contraseña segura para el usuario sa.
7. **Verificar servicio:** Comprueba que esté corriendo con :

   systemctl status mssql-server.
8. **Instalar mssql-tools:** Para usar sqlcmd en la terminal:

   sudo apt install -y mssql-tools unixodbc-dev

1.2 Uso basico de SQL Server

Para ingresar :

sqlcmd -S localhost -U sa -C

Para ver todas las bases de datos

select name from sys.databases;

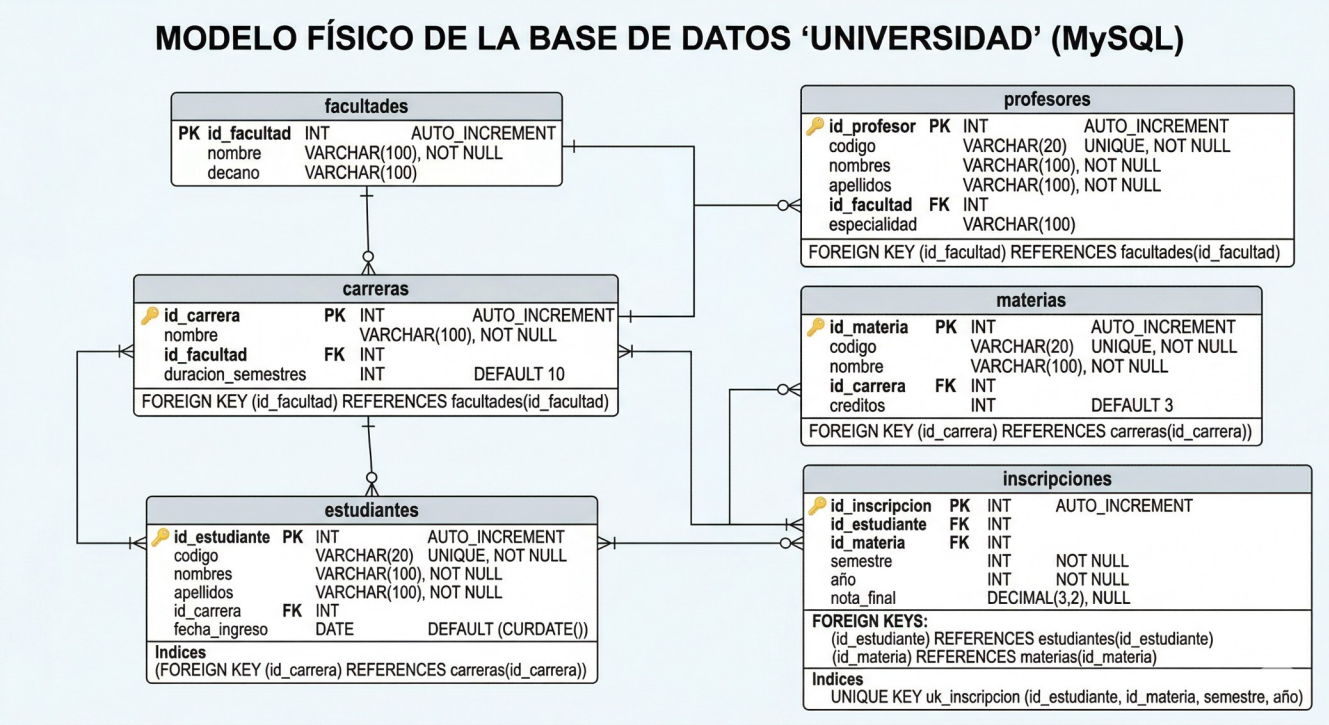
go

Ver que base de datos estoy usando

```
SELECT DB_NAME() ;

go
```

2. DDL Creación de la base de datos



2.1 Creamos la base de datos universidad

```
IF DB_ID('universidad') IS NULL
   CREATE DATABASE universidad;
GO
```

Comprobamos que este creada

```
SELECT name FROM sys.databases;
GO
```

2.2. Creación de las tablas

2.2.1 Creamos la tabla facultades

```
CREATE TABLE facultades (
```

```
    id_facultad INT IDENTITY(1,1) PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    decano VARCHAR(100)
);

GO
```

Para comprobar la creacion de la nueva tabla:

```
SELECT name FROM sys.tables;
```

Para ver la estructura

```
EXEC sp_help 'facultades';
GO
```

ó

```
select * from facultades;
GO
```

2.2.2 Creamos la tabla carreras

```
CREATE TABLE carreras (
    id_carrera INT IDENTITY(1,1) PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    id_facultad INT,
    duracion_semestres INT DEFAULT 10,

    CONSTRAINT FK_carreras_facultad
        FOREIGN KEY (id_facultad)
        REFERENCES facultades(id_facultad)
);

GO
```

2.2.3 Creamos la tabla estudiantes

```
CREATE TABLE estudiantes (
    id_estudiante INT IDENTITY(1,1) PRIMARY KEY,
    codigo VARCHAR(20) NOT NULL,
    nombres VARCHAR(100) NOT NULL,
    apellidos VARCHAR(100) NOT NULL,
    id_carrera INT,
    fecha_ingreso DATE DEFAULT GETDATE(),

    CONSTRAINT UQ_estudiantes_codigo UNIQUE (codigo),
```

```
    CONSTRAINT FK_estudiantes_carrera
        FOREIGN KEY (id_carrera)
        REFERENCES carreras(id_carrera)
);

GO
```

2.2.4 Creamos la tabla profesores

```
CREATE TABLE profesores (
    id_profesor INT IDENTITY(1,1) PRIMARY KEY,
    codigo VARCHAR(20) NOT NULL,
    nombres VARCHAR(100) NOT NULL,
    apellidos VARCHAR(100) NOT NULL,
    id_facultad INT,
    especialidad VARCHAR(100),

    CONSTRAINT UQ_profesores_codigo UNIQUE (codigo),
    CONSTRAINT FK_profesores_facultad
        FOREIGN KEY (id_facultad)
        REFERENCES facultades(id_facultad)
);

GO
```

2.2.5 Creamos la tabla materias

```
CREATE TABLE materias (
    id_materia INT IDENTITY(1,1) PRIMARY KEY,
    codigo VARCHAR(20) NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    id_carrera INT,
    creditos INT DEFAULT 3,

    CONSTRAINT UQ_materias_codigo UNIQUE (codigo),
    CONSTRAINT FK_materias_carrera
        FOREIGN KEY (id_carrera)
        REFERENCES carreras(id_carrera)
);

GO
```

2.2.6 Creamos la tabla inscripciones

```
CREATE TABLE inscripciones (
    id_inscripcion INT IDENTITY(1,1) PRIMARY KEY,
```

```
    id_estudiante INT,
    id_materia INT,
    semestre INT NOT NULL,
    anio INT NOT NULL,          -- 'año' sin tilde para evitar problemas
    nota_final DECIMAL(3,2) NULL, -- 0.00 a 5.00

    CONSTRAINT FK_inscripciones_estudiante
        FOREIGN KEY (id_estudiante)
        REFERENCES estudiantes(id_estudiante),

    CONSTRAINT FK_inscripciones_materia
        FOREIGN KEY (id_materia)
        REFERENCES materias(id_materia),

    -- Evitar inscripción duplicada
    CONSTRAINT UQ_inscripcion
        UNIQUE (id_estudiante, id_materia, semestre, anio)
);

GO
```

2.2.7 Tabla comparativa

MySQL	SQL Server	Cambio
AUTO_INCREMENT	IDENTITY(1,1)	Diferente sintaxis
CREATE DATABASE IF NOT EXISTS	IF DB_ID('x') IS NULL	No existe IF NOT EXISTS
USE	USE + GO	Requiere GO como separador de lotes
UNIQUE KEY inline	CONSTRAINT ... UNIQUE separado	SQL Server prefiere constraints nombrados
DEFAULT (CURDATE())	DEFAULT GETDATE()	Función diferente
año (con tilde)	anio (sin tilde)	SQL Server maneja mejor ASCII
UNIQUE KEY uk_nombre	CONSTRAINT UQ_nombre UNIQUE	Convención de nombres diferente
; como terminador	GO como separador de lotes	Cambio de delimitador de bloques

3. DML - Poblar y consultar

3.1 Poblamos Facultades

INSERT INTO facultades (nombre, decano) VALUES
('Ingeniería', 'Dr. Carlos Martínez'),
('Ciencias Básicas', 'Dra. Ana López'),
('Ciencias Sociales', 'Dr. Pedro Gómez');
GO

3.2 Poblamos Carreras

INSERT INTO carreras (nombre, id_facultad, duracion_semestres) VALUES
('Ingeniería de Sistemas', 1, 10),
('Tecnología en Desarrollo de Software', 1, 6),
('Matemáticas', 2, 10),
('Psicología', 3, 10);
GO

3.3 Poblamos Estudiantes

INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera, fecha_ingreso) VALUES
('202310001', 'Juan Carlos', 'Rodríguez Pérez', 2, '2023-02-01'),
('202310002', 'María Fernanda', 'López García', 2, '2023-02-01'),
('202310003', 'Carlos Alberto', 'Martínez Silva', 1, '2022-02-01'),
('202320001', 'Ana Lucía', 'Gómez Torres', 3, '2023-08-01');
GO

3.4 Poblamos Profesores

INSERT INTO profesores (codigo, nombres, apellidos, id_facultad, especialidad) VALUES
('P001', 'Luis Eduardo', 'Fernández Ruiz', 1, 'Bases de Datos'),
('P002', 'Diana Patricia', 'Castro Mendoza', 1, 'Programación Web'),
('P003', 'Roberto', 'Gutiérrez Vargas', 2, 'Estadística');
GO

3.5 Poblamos Materias

INSERT INTO materias (codigo, nombre, id_carrera, creditos) VALUES
('BD101', 'Base de Datos I', 2, 3),
('BD201', 'Base de Datos II', 2, 3),

('PG101', 'Programación I', 2, 4),
('PG102', 'Programación II', 1, 4),
('MT101', 'Cálculo I', 3, 4);
GO

3.6 Poblamos Inscripciones

INSERT INTO inscripciones (id_estudiante, id_materia, semestre, anio, nota_final) VALUES
(1, 1, 1, 2023, 4.5), -- Juan Carlos en BD I, nota 4.5
(1, 3, 1, 2023, 4.0), -- Juan Carlos en Programación I
(2, 1, 1, 2023, 3.8), -- María en BD I
(2, 3, 1, 2023, 4.2), -- María en Programación I
(3, 4, 4, 2023, 4.7); -- Carlos en Programación II
GO

3.7 Comprobamos

Listamos tablas

SELECT name FROM sys.tables;

Para ver la estructura

EXEC sp_help 'facultades';
GO

ó

*select * from facultades;*
GO

3.9 Consultas básicas:

1. Listar todos los estudiantes

*SELECT * FROM estudiantes;*

2. Obtiene solo apellidos de la tabla estudiantes

SELECT apellidos FROM estudiantes;

3. Obtiene nombre completo (nombre y apellido juntos)

SELECT nombres, apellidos FROM estudiantes;

4. Obtiene todos los identificadores y nombres de la tabla estudiantes

SELECT id_estudiante, nombres FROM estudiantes;

5. Cambia el nombre de la columna mostrada a "Nombre_Estudiante"

SELECT nombres AS Nombre_Estudiante FROM estudiantes;

ó usando espacio en vez de _

SELECT nombres AS [Nombre Estudiante] FROM estudiantes;

6. Múltiples alias

SELECT id_estudiante AS ID, nombres AS Nombre_Estudiante, apellidos AS Apellido_Estudiante FROM estudiantes;

ó

SELECT id_estudiante AS ID, nombres AS [Nombre Estudiante], apellidos AS [Apellido Estudiante] FROM estudiantes;

3.10 Ejercicios de modificación de estructura (ALTER TABLE)

1. Agregar una columna `email` de tipo `VARCHAR(150)` a la tabla `estudiantes`.

ALTER TABLE estudiantes ADD correo_electronico *VARCHAR(150);*

2. Cambia el tipo de dato de correo_electronico a VARCHAR(200) para permitir emails más largos.

ALTER TABLE estudiantes ALTER COLUMN correo_electronico VARCHAR(200);

3. Renombra la columna correo_electronico a email.

EXEC sp_rename 'estudiantes.correo_electronico', 'email', 'COLUMN';

4. Agregar una columna `correo` de tipo `VARCHAR(150)` a la tabla `estudiantes`, para despues borrarlo.

ALTER TABLE estudiantes ADD correo VARCHAR(150);

ALTER TABLE estudiantes DROP COLUMN correo;

Precaución: DROP COLUMN elimina permanentemente los datos. No tiene "deshacer".

Adicionamos datos en email

UPDATE estudiantes SET email='juancarlos@gmail.com'
WHERE id_estudiante=1;

7. Mostrar email, pero si es NULL muestra 'Sin email'.

SELECT nombres, apellidos, ISNULL(email, 'Sin email') FROM estudiantes;

8. Muestra email, pero si es NULL muestra 'Sin email'. Cambiamos el titulo del campo Email a estudiantes sin email.

SELECT nombres, apellidos, ISNULL(email, 'Sin email') AS 'estudiantes sin email' FROM estudiantes;

o

SELECT nombres, apellidos, ISNULL(email, 'Sin email') AS [estudiantes sin email] FROM estudiantes;

9. Solo usuarios llamados Juan Carlos

*SELECT * FROM estudiantes WHERE nombres = 'Juan Carlos';*

10. Calcula años desde la fecha de ingreso (si fecha_ingreso existe)

SELECT nombres, fecha_ingreso, DATEDIFF(YEAR, fecha_ingreso, GETDATE()) AS [años transcurridos] FROM estudiantes WHERE fecha_ingreso IS NOT NULL;

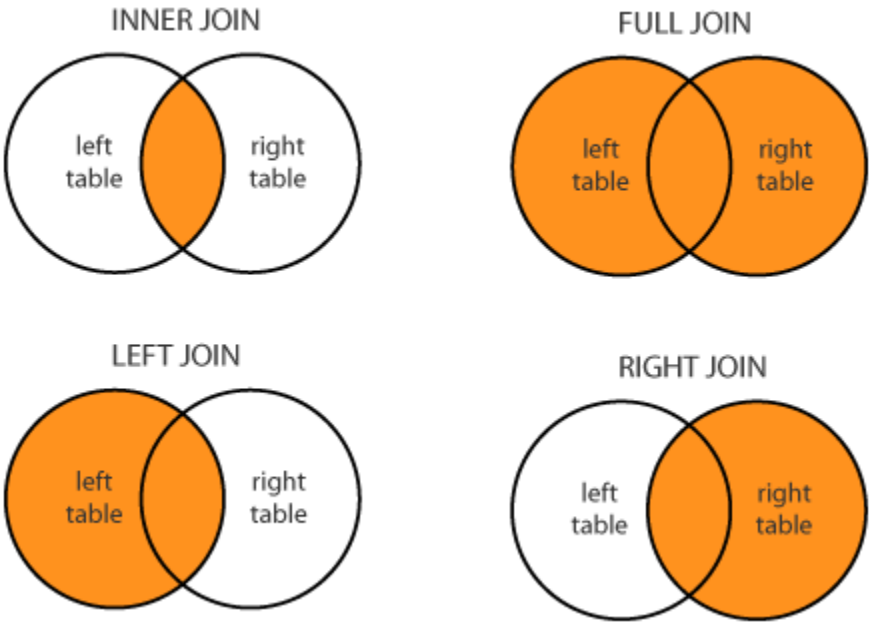
Nota: DATEDIFF(YEAR, ...) en SQL Server calcula la diferencia de años calendario (como restar los años). Si necesitas años completos transcurridos, usa:

DATEDIFF(DAY, fecha_ingreso, GETDATE()) / 365 AS años_transcurridos

4. JOINs - Uniendo tablas

Adicionamos el siguiente registro o tupla:

```
INSERT INTO estudiantes (codigo, nombres, apellidos, id_carrera,
fecha_ingreso) VALUES ('202310005', 'Pedro', 'Gómez', NULL, '2023-02-01');
```



4.1. INNER JOIN

4.1.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e INNER JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM carreras c INNER JOIN estudiantes e ON e.id_carrera =
c.id_carrera;
```

4.1.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
INNER JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM carreras c
INNER JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

4.2. LEFT JOIN

4.2.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e LEFT JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM  carreras c LEFT JOIN estudiantes e ON e.id_carrera =
c.id_carrera;
```

4.2.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM  carreras c
LEFT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

4.3. RIGHT JOIN

4.3.1 Con SELECT * FROM

```
SELECT * FROM estudiantes e RIGHT JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT * FROM carreras c RIGHT JOIN estudiantes e ON e.id_carrera =
c.id_carrera;
```

4.3.2 Seleccionando Campos

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

- El resultado es igual a alguno de los dos ejemplos del LEFT JOIN?
- Comparar el orden en nombres y carrera en el left y right join.
- Cual es el patrón del orden?

4.4. FULL JOIN

4.4.1 Con SELECT * FROM

Versión nativa

```
SELECT * FROM estudiantes e FULL OUTER JOIN carreras c ON e.id_carrera =
c.id_carrera;
```

Versión Union

```
SELECT * FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION
SELECT * FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Cual es la diferencia si invertimos el orden de las tablas en la sentencia?

Versión nativa

```
SELECT * FROM carreras c FULL OUTER JOIN estudiantes e ON e.id_carrera = c.id_carrera
```

Versión Union

```
SELECT * FROM carreras c
LEFT JOIN estudiantes e ON e.id_carrera = c.id_carrera
UNION
SELECT * FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```

Recordar:

Operador	Qué hace	Uso
UNION	Elimina duplicados	FULL JOIN simulado
UNION ALL	Mantiene todos los registros	Cuando sabes que no hay duplicados o los necesitas

4.4.2 Seleccionando Campos

Versión nativa

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera FROM estudiantes e
FULL OUTER JOIN carreras c ON e.id_carrera = c.id_carrera
```

Versión Union

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

Comparar usando UNION ALL

```
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
```

```
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera
UNION ALL
SELECT e.nombres, e.apellidos, c.nombre AS carrera
FROM estudiantes e
RIGHT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

UNION elimina los registros duplicados del resultado final, mientras que UNION ALL incluye todas las filas, duplicados incluidos

5. Pregunta:

Cual es la diferencia entres estas dos consultas:

```
SELECT * FROM estudiantes e
LEFT JOIN carreras c ON e.id_carrera = c.id_carrera;
```

y

```
SELECT * FROM carreras c
RIGHT JOIN estudiantes e ON e.id_carrera = c.id_carrera;
```