

INSTITUTO TECNOLÓGICO DE MEXICALI

Carrera:

Ing. en Sistemas Computacionales

Materia:

Fundamentos de base de datos.

Alumno:

Hernandez Garcia Martin 22490354

Actividad:

Examen Final de Unidad 4

Profesor:

Jose Ramón Bogarin Valenzuela.

Día y Hora de entrega:

23 de Mayo del 2025 11:59 pm

Mexicali, Baja California a 23 de mayo del 2025.



Planteamiento del Problema

Una universidad requiere un sistema de gestión de información que permita registrar, consultar y actualizar datos de estudiantes, profesores, cursos, inscripciones, departamentos, programas de estudio, carreras, campus, entre otros. El sistema debe permitir realizar análisis sobre la información académica, tales como la cantidad de estudiantes inscritos por curso, el promedio de calificaciones por programa, o qué profesor imparte más cursos.

Objetivo General:

Diseñar una base de datos relacional y realizar diversas operaciones para gestionar la información de la universidad. Esto incluye la creación y modificación de la estructura de las tablas, la manipulación de los datos y la realización de consultas complejas para obtener información específica.

Solución Propuesta

1. Diseño de la Base de Datos

Entidades Principales:

- Estudiantes
 - estudiantes_id PK
 - nombre_estudiantes varchar
 - apellido_estudiantes varchar
 - fecha_nacimiento date
 - direccion varchar
 - email varchar
- Carreras
 - carreras_id PK
 - nombre_carrera varchar
 - titulo_otorgado varchar
- Campus
 - campus_id serial PK
 - nombre_carrera varchar
 - direccion_campus varchar
- Cursos
 - curso_id PK
 - name_curso varchar
 - descripción varchar



- creditos int
- semestre int
- departamentos_id FK
- Inscripciones
 - inscripciones_id PK
 - estudiantes_id FK
 - curso_id FK
 - fecha_inscripcion date
 - calificacion decimal
- Profesores
 - profesor_id PK
 - nombre_profesor varchar
 - apellido_profesor varchar
 - titulo varchar
 - departamento_id FK
- Departamentos
 - departamentos_id PK
 - nombre_departamento varchar
 - edificio varchar
- Programas de Estudio
 - programa_id PK
 - nombre_programa varchar
 - descripcion_programa varchar
- Horarios
 - horario_id PK
 - curso_id FK
 - fecha_inicio date
 - fecha_final date
 - hora_inicio time
 - hora_fin time

Relaciones:

- Muchos a muchos entre Estudiantes y Carreras.
- Muchos a muchos entre Cursos y Profesores.
- Muchos a muchos entre Programas y Cursos.
- Un curso pertenece a un departamento y a un campus.
- Un estudiante pertenece a una carrera.



2. Creación de Tablas (DDL)

Código SQL completo para crear las tablas mencionadas.

```
create table estudiantes (  
  estudiantes_id serial primary key,  
  nombre_estudiante varchar(50) not null,  
  apellido_estudiante varchar(50) not null,  
  fecha_nacimiento date not null,  
  direccion varchar(50) not null,  
  ciudad varchar(50) not null,  
  email varchar(50) not null  
);  
  
create table departamentos (  
  departamentos_id serial primary key,  
  nombre_departamento varchar(50) not null,  
  edificio varchar(20) not null  
);  
  
create table cursos (  
  curso_id serial primary key,  
  name_curso varchar(50) not null,  
  descripcion varchar(50) not null,  
  creditos int not null check (creditos > 0),  
  semestre int not null check (semestre > 0),  
  departamentos_id int references departamentos (departamentos_id)  
);  
  
create table inscripciones (  
  inscripcion_id serial primary key,  
  estudiantes_id int references estudiantes(estudiantes_id),  
  curso_id int references cursos(curso_id),  
  fecha_inscripcion date not null,  
  calificacion decimal(3,1) not null  
);  
  
create table profesores (  
  profesor_id serial primary key,  
  nombre_profesor varchar(50) not null,  
  apellido_profesor varchar(50) not null,  
  titulo varchar(50) not null,  
  departamento_id int references departamentos(departamentos_id)  
);  
  
create table aulas (  
  aula_id serial primary key,  
  nombre_aula varchar(50) not null,  
  capacidad int not null,
```



```
ubicacion varchar(50) not null
);
create table horarios (
horario_id serial primary key,
curso_id int references cursos(curso_id),
aula_id int references aulas(aula_id),
fecha_inicio date not null,
fecha_fin date not null,
hora_inicio time not null,
hora_fin time not null
);
create table cursos_profesores (
curso_profesor_id serial primary key,
curso_id int references cursos(curso_id),
profesor_id int references profesores(profesor_id)
);
create table programas_estudio (
programa_id serial primary key,
nombre_programa varchar(50) not null,
descripcion_programa varchar(50) not null
);
create table programas_cursos (
programa_curso_id serial primary key,
programas_id int references programas_estudio(programa_id),
curso_id int references cursos(curso_id)
);
```

3. Modificaciones de Tablas

- Agregar tablas:
 - Tabla Campus
 - IDCampus(Clave Principal)
 - NombreCampus
 - DireccionCampus

```
-- Creacion Tabla Campus--
create table campus (
campus_id serial primary key,
nombre_campus varchar(50) not null,
direccion_campus varchar(50) not null
);
```



- Tabla: Carreras
 - IDCarrera
 - NombreCarrera
 - TituloOtorgado

```
-- Creacion Tabla Carreras --  
create table carreras (  
carreras_id serial primary key,  
nombre_carrera varchar(50) not null,  
titulo_otorgado varchar(50) not null  
);
```

- Agregar una relacion de muchos a muchos entre Estudiantes y Carreras

```
-- Creacion relacion Estudiantes/Carrera --  
create table estudiantes_carreras (  
estudiantes_carreras_id serial primary key,  
estudiantes_id int references estudiantes(estudiantes_id),  
carreras_id int references carreras(carreras_id)  
);
```

- Agregada la columna **carreras_id** en la tabla **estudiantes**

```
-- Agregar clave foranea a Estudiantes --  
alter table estudiantes  
add carreras_id int references carreras(carreras_id);
```

Utilizamos la funcion ALTER TABLE ya que es usado para agregar, eliminar o modificar columnas en un tabla ya existente

- Agregada la columna **campus_id** en la tabla **cursos**.

```
-- Agregar clave foranea a Cursos --  
alter table cursos  
add campus_id int references campus(campus_id);
```

Utilizamos la funcion ALTER TABLE ya que es usado para agregar, eliminar o modificar columnas en un tabla ya existente

- Agregada la columna **email** en la tabla **profesores**.

```
-- Agregar columna a Profesores --
```



```
alter table profesores  
add email varchar(50);
```

Utilizamos la funcion ALTER TABLE ya que es usado para agregar, eliminar o modificar columnas en un tabla ya existente

- Eliminada la columna **ciudad** de **estudiantes**.

```
-- Eliminar columna a Estudiantes --  
alter table estudiantes  
drop column ciudad;
```

Utilizamos la funcion ALTER TABLE ya que es usado para agregar, eliminar o modificar columnas en un tabla ya existente

- Eliminada la tabla **aulas**

```
-- Eliminar tabla Aulas --  
alter table horarios  
drop column aula_id;  
drop table aulas;
```

Lo primero que hicimos fue eliminar cualquier relacion que tuviera la tabla **Aulas** con cualquier otra table. En esto caso estaba relacionada con la tabla **Horarios** despues de esto ya eliminamos la tabla **Aulas**

- Reiniciadas las secuencias para empezar desde 1.
Esto es un paso extra que tuve que hacer ya que las secuencias de los ID SERIAL comenzaban desde un numero aleatorio, observe el error en la insercion de datos ya que al llamar el profesor_id = 1 me arrojaba un error.

Al verificar el error y ver las tablas estas empezaban de un numero aleatorio

Se resolvió el problema estableciendo todos los ID que iniciaran en 1 con los siguientes comandos

```
alter sequence estudiantes_estudiantes_id_seq restart with 1;
```

```
alter sequence departamentos_departamentos_id_seq restart with 1;  
alter sequence cursos_curso_id_seq restart with 1;  
alter sequence inscripciones_inscripcion_id_seq restart with 1;  
alter sequence profesores_profesor_id_seq restart with 1;  
alter sequence horarios_horario_id_seq restart with 1;  
alter sequence cursos_profesores_curso_profesor_id_seq restart with 1;  
alter sequence programas_estudio_programa_id_seq restart with 1;  
alter sequence programas_cursos_programa_curso_id_seq restart with 1;
```



```
alter sequence campus_campus_id_seq restart with 1;  
alter sequence carreras_carreras_id_seq restart with 1;  
alter sequence estudiantes_carreras_estudiantes_carreeras_id_seq restart with 1;
```

Alter sequence es un comando que modifica las propiedades de una secuencia
Su sintaxis es

**Alter sequence nombretabla_nombresecuencia_seq restart with
numero_a_inicializar**

4. Inserción de Datos

Se insertaron datos para que las tablas no estuvieran vacías y hacer las consultas pertinentes. En esta inserción de dato ya se tomaron en cuenta los cambios en las tablas anteriores como la eliminacion de la tabla **AULA**, la nueva columna **EMAIL** en la tabla **Profesores** entre otros cambios

-- Inserccion de datos --

```
insert into departamentos (nombre_departamento, edificio) values  
(('Ingeniería', 'A1'),  
(('Ingenieria', 'B1'),  
(('Contabilidad', 'C1'),  
(('Matemáticas', 'A2'),  
(('Contabilidad', 'B2'),  
(('Derecho', 'C2'),  
(('Educación', 'D1');  
insert into campus (nombre_campus, direccion_campus) values  
(('Campus Mexicali', 'Lazaro 1'),  
(('Campus Monterrey', 'Lazaro 2'),  
(('Campus Tijuana', 'Lazaro 3'),  
(('Campus Ensenada', 'Lazaro 4'),  
(('Campus Tecate', 'Lazaro 5'),  
(('Campus Mexico', 'Lazaro 6'),  
(('Campus Jalisco', 'Lazaro 7');  
insert into carreras (nombre_carrera, titulo_otorgado) values  
(('Sistemas Computacionales', 'Ingeniero en Sistemas'),  
(('Contabilidad', 'Licenciado en Contabilidad'),  
(('Derecho', 'Abogado'),  
(('Educación Primaria', 'Licenciado en Educación'),  
(('Matemáticas', 'Licenciado en Matemáticas'),  
(('Psicología', 'Licenciado en Psicología'),  
(('Deportes', 'Licenciado en Deportes');  
insert into programas_estudio (nombre_programa, descripcion_programa) values  
(('Plan Ingenieria', 'Plan ingenieria basica'),
```




```
('Plan Ingeniería', 'Plan especializado en ingeniería'),
('Plan Contabilidad', 'Contabilidad'),
('Plan Educación', 'Docencia y pedagogía'),
('Plan Derecho', 'Derecho Penal'),
('Plan Ciencias', 'Ciencias básicas'),
('Plan Humanidades', 'Deportes');
insert into estudiantes (nombre_estudiante, apellido_estudiante, fecha_nacimiento,
direccion, email, carreras_id) values
('Ana', 'López', '2000-01-15', 'Calle 1', 'ana@gmail.com', 1),
('Luis', 'Pérez', '1999-06-22', 'Calle 2', 'luis@gmail.com', 2),
('María', 'Ramírez', '2001-03-30', 'Calle 3', 'maria@gmail.com', 1),
('Carlos', 'Soto', '2002-11-11', 'Calle 4', 'carlos@gmail.com', 3),
('Lucía', 'Fernández', '2000-09-09', 'Calle 5', 'lucia@gmail.com', 1),
('Pedro', 'Gómez', '1998-12-25', 'Calle 6', 'pedro@gmail.com', 4),
('Sofía', 'Díaz', '2001-05-05', 'Calle 7', 'sofia@gmail.com', 2);
insert into profesores (nombre_profesor, apellido_profesor, titulo, departamento_id,
email) values
('Jose', 'Bogarin', 'Ing.', 1, 'jose@tec.edu'),
('Elena', 'Ruiz', 'Dra.', 2, 'elena@tec.edu'),
('Raúl', 'Cano', 'Ing', 1, 'raul@tec.edu'),
('Laura', 'Jiménez', 'Mtra.', 4, 'laura@tec.edu'),
('Miguel', 'Torres', 'Dr.', 3, 'miguel@tec.edu'),
('Claudia', 'Ríos', 'Ing.', 1, 'claudia@tec.edu'),
('Andrés', 'Luna', 'Mtro', 5, 'andres@tec.edu');
insert into cursos (name_curso, descripcion, credits, semestre, departamentos_id,
campus_id) values
('Bases de Datos', 'Introducción a BD', 3, 2, 1, 1),
('Contabilidad', 'Contabilidad 1', 4, 2, 5, 2),
('Derecho Civil', 'Leyes', 3, 3, 6, 2),
('Matemáticas Discretas', 'Compuertas Logicas', 3, 2, 4, 1),
('Programación I', 'Fundamentos de código', 4, 1, 1, 1),
('Psicología General', 'Teoría básica', 3, 2, 2, 3),
('Didáctica', 'Estrategias educativas', 3, 3, 7, 4);
insert into inscripciones (estudiantes_id, curso_id, fecha_inscripcion, calificacion)
values
(1, 1, '2024-01-10', 8.5),
(2, 2, '2024-01-11', 9.0),
(3, 1, '2024-01-12', 7.0),
(4, 3, '2024-01-13', 6.5),
(5, 4, '2024-01-14', 8.0),
(6, 5, '2024-01-15', 7.5),
(7, 2, '2024-01-16', 9.5);
insert into horarios (curso_id, fecha_inicio, fecha_fin, hora_inicio, hora_fin) values
(1, '2024-02-01', '2024-06-01', '08:00', '10:00'),
```



```
(2, '2024-02-01', '2024-06-01', '10:00', '12:00'),
(3, '2024-02-01', '2024-06-01', '12:00', '14:00'),
(4, '2024-02-01', '2024-06-01', '14:00', '16:00'),
(5, '2024-02-01', '2024-06-01', '08:00', '10:00'),
(6, '2024-02-01', '2024-06-01', '10:00', '12:00'),
(7, '2024-02-01', '2024-06-01', '12:00', '14:00');
insert into cursos_profesores (curso_id, profesor_id) values
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 1),
(6, 5),
(7, 6);
insert into programas_cursos (programas_id, curso_id) values
(1, 1),
(1, 2),
(2, 1),
(3, 2),
(4, 4),
(5, 3),
(6, 6);
insert into estudiantes_carreras (estudiantes_id, carreras_id) values
(1, 1),
(2, 2),
(3, 1),
(4, 3),
(5, 1),
(6, 4),
(7, 2);
```

5. Consultas SQL

Consultas Específicas

1. **Selección Básica:** Muestra todos los nombres y apellidos de los estudiantes.

```
-- 1. Seleccion Basica --
select e.nombre_estudiante, e.apellido_estudiante
from estudiantes e;
```

Utilizamos un Select donde solo priorizamos el nombre de estudiante y su apellido. Le dimos un alias a la tabla Estudiantes e para poder evitar posibles errores.



	A-Z nombre_estudiante ▼	A-Z apellido_estudiante ▼
1	Ana	López
2	Luis	Pérez
3	María	Ramírez
4	Carlos	Soto
5	Lucía	Fernández
6	Pedro	Gómez
7	Sofía	Díaz

2. **Cláusula WHERE:** Encuentra todos los cursos que tienen 3 créditos.

```
-- 2. Clausula WHERE --  
select c.name_curso as nombre_curso, c.creditos  
from cursos c  
where creditos = 3;
```

En este Select solo tomamos en cuenta el nombre del curso y la cantidad de creditos que tiene cada uno. La solicitud fue encontrar los que solo tienen 3 creditos por eso utilizamos la clausula WHERE para filtrar los resultados

	A-Z nombre_curso	123 creditos	
1	Bases de Datos	3	
2	Derecho Civil	3	
3	Matemáticas Discretas	3	
4	Psicología General	3	
5	Didáctica	3	



3. **INNER JOIN:** Obtén una lista que muestre el nombre del estudiante y el nombre del curso en el que está inscrito.

```
-- 3. INNER JOIN --
select e.nombre_estudiante || ' ' || e.apellido_estudiante as nombre_estudiante ,
c.name_curso as nombre_curso
from inscripciones i
inner join estudiantes e on i.estudiantes_id = e.estudiantes_id
inner join cursos c on i.curso_id = c.curso_id;
```

Para resolver este problema utilizamos dos **INNER JOIN** esto debido que esta funcion solo regresa valores que estan conectados ambas tablas, la tabla **INSCRIPCIONES** es nuestra tabla detalla y contiene el nombre del estudiante que se encuentra en la tabla **ESTUDIANTES** y el nombre del curso se encuentra en la tabla **CURSOS**

	A-Z nombre_estudiante	A-Z nombre_curso
1	Ana López	Bases de Datos
2	Luis Pérez	Contabilidad
3	María Ramírez	Bases de Datos
4	Carlos Soto	Derecho Civil
5	Lucía Fernández	Matemáticas Discretas
6	Pedro Gómez	Programación I
7	Sofía Díaz	Contabilidad

4. **LEFT JOIN:** Muestra todos los estudiantes y, si están inscritos en algún curso, el nombre del curso. Si un estudiante no está inscrito en ningún curso, el campo del nombre del curso debe mostrar un valor que lo indique (ej: NULL).

```
-- 4. LEFT JOIN --
select e.nombre_estudiante || ' ' || e.apellido_estudiante as nombre_estudiante ,
c.name_curso as nombre_curso
from estudiantes e
left join inscripciones i on e.estudiantes_id = i.estudiantes_id
left join cursos c on i.curso_id = c.curso_id;
```

Al utilizar el **LEFT JOIN** se toman todos los datos de la tabla **ESTUDIANTES** y tomas las



inscripciones de la tabla **INSCRIPCIONES** en caso que no tengan alguna inscripcion nos arroja un valor NULL. La union con la tabla **CURSOS** es para que se asigne el nombre del curso al estudiante que si tenga inscripcion en caso que no tenga ningun curso asignado el valor ser NULL

En esta base de datos utilizamos la restricción NOT NULL para que no pudiera tener valores NULL

	A-Z nombre_estudiante ▼	A-Z nombre_curso ▼
1	Ana López	Bases de Datos
2	Luis Pérez	Contabilidad
3	María Ramírez	Bases de Datos
4	Carlos Soto	Derecho Civil
5	Lucía Fernández	Matemáticas Discretas
6	Pedro Gómez	Programación I
7	Sofía Díaz	Contabilidad

5. **RIGHT JOIN:** RIGHT JOIN: Lista todos los cursos y, si tienen estudiantes inscritos, el nombre de los estudiantes. Muestra todos los cursos, incluso si no tienen estudiantes inscritos actualmente.

```
6. select c.name_curso as nombre_curso, e.nombre_estudiante || ' ' ||  
   e.apellido_estudiante  
7. from inscripciones i  
8. right join cursos c ON c.curso_id = i.curso_id  
9. left join estudiantes e ON i.estudiantes_id = e.estudiantes_id;
```

Utilizamos un **RIGHT JOIN** esto para que nos muestre todos los cursos incluso los que no tienen inscripciones y el **LEFT JOIN** es para mostrar los estudiantes incluso los que no tengan algun curso asignado

	A-Z nombre_curso ▼	A-Z nombre_estudiante ▼
1	Bases de Datos	Ana López
2	Contabilidad	Luis Pérez
3	Bases de Datos	María Ramírez
4	Derecho Civil	Carlos Soto
5	Matemáticas Discretas	Lucía Fernández
6	Programación I	Pedro Gómez
7	Contabilidad	Sofía Díaz
8	Psicología General	[NULL]
9	Didáctica	[NULL]



6. **GROUP BY y COUNT:** Calcula cuántos estudiantes están inscritos en cada curso. Muestra el nombre del curso y la cantidad de estudiantes.

```
select c.name_curso as nombre_curso, count(i.estudiantes_id) as total_estudiantes
from cursos c
left join inscripciones i on c.curso_id = i.curso_id
group by c.name_curso;
```

La función **COUNT** nos ayuda a contar cuántos estudiantes están inscritos en cada curso, decidí utilizar un **left join** con **INSCRIPCIONES** para que me muestre cursos que no tengan estudiantes. Al utilizar **GROUP BY** para que se agrupen por el nombre del curso

	A-Z nombre_curso ▼	123 total_estudiantes ▼
1	Didáctica	0
2	Psicología General	0
3	Contabilidad	2
4	Derecho Civil	1
5	Programación I	1
6	Bases de Datos	2
7	Matemáticas Discretas	1

7. **BETWEEN:** Encuentra todos los estudiantes que nacieron entre el 1 de enero de 1995 y el 31 de diciembre de 1998.

```
select e.nombre_estudiante || ' ' || e.apellido_estudiante as nombre,
e.fecha_nacimiento
from estudiantes e
where e.fecha_nacimiento between '1995-01-01' and '1998-12-31';
```

Aquí utilizamos la función **WHERE** junto con **BETWEEN** para poder establecer las restricciones de búsqueda



	A-Z nombre ▼	🕒 fecha_nacimiento ▼
1	Pedro Gómez	1998-12-25

	A-Z ?column? ▼	🕒 fecha_nacimiento ▼
1	Pedro Gómez	1998-12-25

8. **ORDER BY:** Muestra todos los cursos ordenados alfabéticamente por su nombre.

```
-- 8. ORDER BY --  
select c.name_curso  
from cursos c  
order by c.name_curso asc;
```

Utilizando el comando **ORDER BY** junto con **ASC** tomamos los nombres de los cursos para que se muestren ordenados alfabéticamente.



	A-Z name_curso
1	Bases de Datos
2	Contabilidad
3	Derecho Civil
4	Didáctica
5	Matemáticas Discretas
6	Programación I
7	Psicología General

9. **CTE:** Crea una tabla de expresión común que liste el número de inscripciones por estudiante. Luego, consulta esta CTE para encontrar los 3 estudiantes con más inscripciones, mostrando el nombre del estudiante y el número de inscripciones.

```
with inscripcion_estudiantes as (  
select e.estudiantes_id, e.nombre_estudiante || ' ' || e.apellido_estudiante AS  
nombre_completo,  
count(i.inscripcion_id) as total_inscripciones  
from estudiantes e  
left join inscripciones i on e.estudiantes_id = i.estudiantes_id  
group by e.estudiantes_id, nombre_completo  
)  
select nombre_completo, total_inscripciones  
from inscripcion_estudiantes  
order by total_inscripciones DESC  
limit 3;
```

En esta consulta con **CTE** iniciamos nombrandola como **INSCRIPCION_ESTUDIANTES**, dentro de esta tabla o espacio de memoria tomamos y nombramos los valores a utilizar en la consulta. Principalmente usamos la funcion **COUNT** para tener el conteo de las inscripciones que tienen los estudiantes. Se utilizo **LEFT JOIN** para que cuente las inscripciones que no tengan estudiantes



Al momento de hacer la consulta usamos **ORDER BY** para poder ordenarlos de mayor a menos con el comando **DESC** y a su vez utilizamos **LIMIT** para que solo nos muestre los 3 mas altos

	A-Z nombre_completo ▼	123 total_inscripciones ▼
1	María Ramírez	1
2	Lucía Fernández	1
3	Pedro Gómez	1

Consultas Avanzadas

1. **Curso más inscrito por departamento:** Para cada departamento, muestra el nombre del departamento y el nombre del curso con la mayor cantidad de estudiantes inscritos.

```
-- Consulta Avanzada 1 --
with conteo_inscripciones as (
select d.nombre_departamento, c.name_curso,
count(i.inscripcion_id) as total_inscritos,
rank() over (partition by d.departamentos_id order by count(i.inscripcion_id)
desc) as posicion
from cursos c
inner join departamentos d on c.departamentos_id = d.departamentos_id
left join inscripciones i on c.curso_id = i.curso_id
group by d.departamentos_id, d.nombre_departamento, c.name_curso
)
select nombre_departamento, name_curso, total_inscritos
from conteo_inscripciones
where posicion = 1;
```

Creamos una tabla **CTE** para que la consulta sea mas facil de usar

Se utilizo la tabla **Cursos** como tabla principal de esta manera **COUNT(inscripciones_id)** esto para contar todos los cursos que cuentan con inscripciones o no cuenten con ninguna ya que se unio a la tabla **INSCRIPCIONES** por medio de un **LEFTJOIN**. Tambien utilice un **INNER JOIN** esto para que solo se muestren los cursos que tienen asignado un departamento

Al investigar acerca de diferentes funciones donde se puedan agrupar valores, la mas recomendada fue **RANK O DENSE_RANK** estas son funciones que le asignan una posicion a los valores que estemos buscando

En este caso nosotros utilizamos **PARTITION BY** por medio del **departamento_id** pero igual funcion con **nombre_departamento** esto nos sirve para separar cada departamento.



Utilizamos **COUNT(incipciones_id) DESC** para que el curso con mas inscripciones tome la posicion 1

	A-Z nombre_dep ▼	A-Z name_curso ▼	123 total_inscrip ▼
1	Ingeniería	Bases de Datos	2
2	Ingenieria	Psicología General	0
3	Matemáticas	Matemáticas Discre	1
4	Contabilidad	Contabilidad	2
5	Derecho	Derecho Civil	1
6	Educación	Didáctica	0

2. **Profesores con más de 2 cursos:** Encuentra a los profesores que imparten más de dos cursos, mostrando su nombre, apellido y la cantidad de cursos que imparten.

```
select p.nombre_profesor || ' ' || p.apellido_profesor as nombrecompleto_profesor,  
count(cp.curso_id) as cantidad_cursos  
from profesores p  
inner join cursos_profesores cp on p.profesor_id = cp.profesor_id  
group by p.profesor_id, nombrecompleto_profesor  
having count(cp.curso_id) > 2;
```

Este problema se resolvió con la ayuda de la función **HAVING** su función es casi igual a la función **WHERE** la diferencia principal es debido a que **WHERE** no puede hacer cálculos y tiene que utilizarse antes del **GROUP BY**

COUNT(cp.curso_id) cuenta los cursos que tiene cada profesor en la tabla **CURSOS_PROFESORES** que es la tabla que establece la relación entre cursos y profesores. Fue mediante un **INNERJOIN** ya que solo nos interesa los profesores que cuenten con cursos.

	A-Z nombrecompleto_profesor ▼	123 cantidad_cursos ▼

3. **Curso con mejor promedio por programa:** Lista los nombres de los programas de estudio y, para cada programa, el nombre del curso con el promedio de



calificación más alto.

```
with promedio_cursos as (  
select pe.nombre_programa, c.name_curso as nombre_curso,  
round(avg(i.calificacion), 2) as promedio,  
rank() over (partition by pe.programa_id order by avg(i.calificacion) desc) as  
posicion  
from programas_estudio pe  
inner join programas_cursos pc on pe.programa_id = pc.programas_id  
inner join cursos c on pc.curso_id = c.curso_id  
inner join inscripciones i on c.curso_id = i.curso_id  
group by pe.programa_id, pe.nombre_programa, c.name_curso  
)  
select nombre_programa, nombre_curso, promedio  
from promedio_cursos  
where posicion = 1;
```

La función **AVG(i.calificacion)** calcula el promedio de calificaciones para cada curso, y **ROUND()** redondea el resultado a dos decimales para mayor claridad.

La tabla **INSCRIPCIONES** se utilizó porque es donde se encuentran las calificaciones de los estudiantes, y se unió con la tabla **CURSOS** usando un **INNER JOIN**. Además, se conectó con la tabla **PROGRAMAS_ESTUDIO** mediante la tabla **PROGRAMAS_CURSOS**, ya que representa la relación entre programas y sus cursos.

	A-Z nombre_pro	A-Z nombre_cur	123 promedio
1	Plan Ingeniería	Contabilidad	9.25
2	Plan Ingeniería	Bases de Datos	7.75
3	Plan Contabilidad	Contabilidad	9.25
4	Plan Educación	Matemáticas Discre	8
5	Plan Derecho	Derecho Civil	6.5