



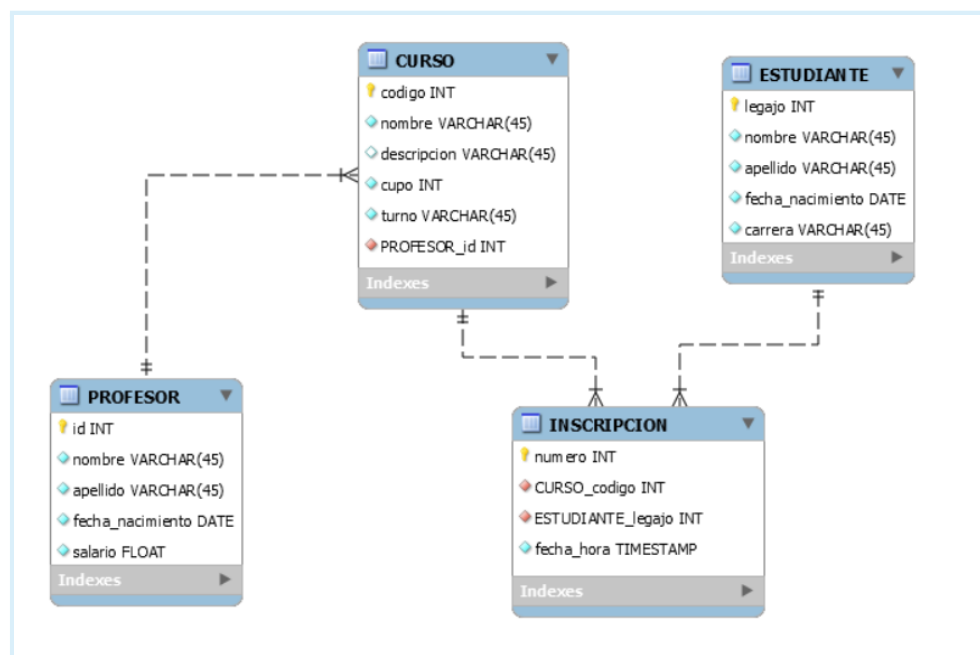
Tema 2: Multiple tables

SELECT

La sentencia SELECT, utilizada para una tabla o múltiples tablas, sirve para consultar datos. Es posible consultar o bien todas las columnas de la tabla (o las tablas en cuestión), o bien solo un subconjunto de ellas.

Tomemos como ejemplo el siguiente modelo relacional de cursos:

Figura 3: Modelo relacional de cursos



Fuente: [Imagen sin título sobre modelo relacional]. (s.f.).

Con este caso podemos hacer lo siguiente:

- Consultar todas las columnas de la tabla PROFESOR ejecutando
`SELECT * FROM PROFESOR;`
- Consultar solamente los nombres de los profesores ejecutando
`SELECT nombre FROM PROFESOR;`



- Consultar nombre, apellido y cursos que dicta cada profesor ejecutando
SELECT p.nombre, p.apellido, c.nombre
FROM PROFESOR p INNER JOIN CURSO c ON p.id =
c.PROFESOR_id;
- Consultar todas las columnas resultantes del JOIN entre PROFESOR y CURSO
SELECT *
FROM PROFESOR p INNER JOIN CURSO c ON p.id =
c.PROFESOR_id;

AS

La palabra reservada “AS” sirve para darle un alias a una tabla o a una columna de una tabla.

El AS es útil, por ejemplo, cuando la consulta obtiene datos de más de una tabla.

Siguiendo con el ejemplo del apartado anterior del modelo relacional de cursos, la consulta

```
SELECT p.nombre, p.apellido, c.nombre  
FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id;
```

fue simplificada con el uso de alias. “p” es un alias para la tabla PROFESOR y “c” es un alias para la tabla CURSO. En ninguno de los dos casos se utilizó “AS” para usar el alias, pero su uso no hubiera afectado el funcionamiento. Sabiendo esto, la consulta anterior puede escribirse de la siguiente manera:

```
SELECT p.nombre, p.apellido, c.nombre  
FROM PROFESOR as p INNER JOIN CURSO as c ON p.id =  
c.PROFESOR_id;
```

Utilizar AS también es útil cuando queremos que, en el resultado de la consulta, una columna se muestre con otro nombre:

```
SELECT p.nombre as “Nombre”, p.apellido as “Apellido”, c.nombre  
as “Curso”  
FROM PROFESOR as p INNER JOIN CURSO as c ON p.id =  
c.PROFESOR_id;
```

DISTINCT



Utilizamos SELECT DISTINCT cuando queremos que la consulta devuelva solo registros con valores diferentes.

Tabla 19: Tabla CURSO

	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
▶	101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1
	102	Matemática Discreta	NULL	20	Tarde	2
	103	Programación Java	POO en Java	35	Noche	4
	104	Programación Web	NULL	35	Noche	5
	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

A partir de la tabla CURSO, el DISTINCT puede utilizarse con una sola columna:

```
SELECT DISTINCT turno  
FROM CURSO;
```

Resultado:

Tabla 20. Resultado

	turno
▶	Mañana
	Tarde
	Noche

Fuente: elaboración propia.

Y también puede utilizarse con varias columnas:

```
SELECT DISTINCT turno, cupo  
FROM CURSO;
```

La última consulta retornará solamente combinaciones diferentes de turnos y cupos de la tabla CURSO.

Tabla 21. CURSO



	turno	CUPO
►	Mañana	20
	Tarde	20
	Noche	35
	Noche	30

Fuente: elaboración propia.

WHERE

Se trata de una sección de una consulta que permite agregar filtros o condiciones al resultado. El WHERE puede utilizarse independientemente de la cantidad de tablas que estén involucradas.

Tomemos como ejemplos las tablas PROFESOR y CURSO:

Tabla 22: Tablas PROFESOR y CURSO

	id	nombre	apellido	fecha_nacimiento	salario
►	1	Juan	Pérez	1990-06-06	55000
	2	María Emilia	Paz	1984-07-15	72000
	3	Martín	Correa	1987-12-07	63000
	4	Lucía	Díaz	1991-02-24	45000
	5	Raúl	Martínez	1980-10-15	85000
	6	Mabel	Ríos	1982-06-12	83000

	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
►	101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1
	102	Matemática Discreta	NULL	20	Tarde	2
	103	Programación Java	POO en Java	35	Noche	4
	104	Programación Web	NULL	35	Noche	5
	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

Aquí, el WHERE se puede agregar en una consulta para una sola tabla:

```
SELECT * FROM PROFESOR WHERE id = 1;
```



Resultado:

Tabla 23. Resultado

	id	nombre	apellido	fecha_nacimiento	salario
►	1	Juan	Pérez	1990-06-06	55000

Fuente: elaboración propia.

Se puede, también, utilizar en una consulta con múltiples tablas, haciendo referencia a cada una con su alias:

```
SELECT * FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id  
WHERE c.cupo = 35;
```

Resultado:

Tabla 24. Resultado

	id	nombre	apellido	fecha_nacimiento	salario	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
►	4	Lucía	Díaz	1991-02-24	45000	103	Programación Java	POO en Java	35	Noche	4
	5	Raúl	Martínez	1980-10-15	85000	104	Programación Web	NULL	35	Noche	5

Fuente: elaboración propia.

Los operadores más comunes para poner filtros en la sección WHERE son “=”, “<” y “>”, aunque existen otros que se verán en los apartados siguientes.

Para todos los casos, los ejemplos se realizarán a partir de las tablas PROFESOR y CURSO.

LIKE

LIKE permite filtrar “valores similares” en un campo texto. Es útil cuando solo conocemos una parte del valor que queremos filtrar, pero no sabemos cómo es en su totalidad, es decir, que hay valores desconocidos.

Por ejemplo, si queremos obtener todos los cursos que tengan en su nombre la palabra “Programación”, podemos ejecutar la siguiente sintaxis:

```
select * from CURSO  
where nombre LIKE '%Programación%'
```



Con los porcentajes, especificamos que puede haber 0 o más caracteres desconocidos antes (cuando lo agregamos antes de la palabra) o al final (cuando lo agregamos después de la palabra). El resultado sería el siguiente:

Tabla 25. Resultado

	codigo	nombre	descripcion	cupos	turno	PROFESOR_id
▶	103	Programación Java	POO en Java	35	Noche	4
	104	Programación Web	NULL	35	Noche	5
	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

Además de los porcentajes, es posible utilizar guiones bajos. El guion bajo se utiliza cuando sabemos exactamente la cantidad de caracteres desconocidos que hay adelante (o al final) de la parte conocida de la palabra. Supongamos que ejecutamos la siguiente consulta:

```
select * from CURSO
where nombre LIKE '%Programación _ _ _ _'
```

Resultado:

Tabla 26. Resultado

	codigo	nombre	descripcion	cupos	turno	PROFESOR_id
▶	103	Programación Java	POO en Java	35	Noche	4

Fuente: elaboración propia.

En ese caso, solo obtenemos el curso de programación en Java porque especificamos cuatro guiones bajos después de “Programación”.

BETWEEN

BETWEEN sirve para agregar un rango de valores como filtro. Por ejemplo, si queremos todos los cursos cuyo cupo sea entre 20 y 30, debemos usar la siguiente sintaxis:

```
SELECT * FROM CURSO WHERE cupo BETWEEN 20 AND 30;
```

Resultado:



Tabla 27. Resultado

	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
▶	101	Algoritmos	Algoritmos y estructuras de datos	20	Mañana	1
	102	Matemática Discreta	NULL	20	Tarde	2
	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

Se puede utilizar con consultas a una sola tabla o a varias, pero siempre referenciando a qué tabla corresponde el campo que se quiere utilizar como filtro.

AND

El operador lógico AND sirve para unir dos o más condiciones en una misma consulta SQL. Para que un registro sea incluido en el resultado, **todas** las condiciones que estén enlazadas con AND deben ser verdaderas.

```
SELECT *  
FROM tabla  
WHERE condicion1  
AND condicion2  
AND ....
```

Ejemplo:

```
SELECT *  
FROM PROFESOR  
WHERE fecha_nacimiento between '1980-01-01' and  
'1989-12-31'  
AND Salario > 60000;
```

Resultado:

Tabla 28. Resultado

	id	nombre	apellido	fecha_nacimiento	salario
▶	2	María Emilia	Paz	1984-07-15	72000
	3	Martín	Correa	1987-12-07	63000
	5	Raúl	Martínez	1980-10-15	85000
	6	Mabel	Ríos	1982-06-12	83000

Fuente: elaboración propia.



Asimismo, es posible agregar condiciones que hagan referencia a tablas diferentes:

```
SELECT *  
FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id  
WHERE c.Cupo >= 30  
AND p.Salario > 60000;
```

Resultado:

Tabla 29. Resultado

	id	nombre	apellido	fecha_nacimiento	salario	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
▶	5	Raúl	Martínez	1980-10-15	85000	104	Programación Web	HTML	35	Noche	5
	6	Mabel	Ríos	1982-06-12	83000	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

OR

Al igual que AND, OR es un operador lógico para unir condiciones, pero la diferencia yace en que, para que un registro se incluya en el resultado, al menos una de las condiciones enlazadas con OR debe ser verdadera.

Si tomamos el ejemplo anterior y cambiamos AND por OR, obtenemos la siguiente sintaxis:

```
SELECT *  
FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id  
WHERE c.Cupo >= 30  
OR p.Salario > 60000;
```

Resultado:

Tabla 30. Resultado

	id	nombre	apellido	fecha_nacimiento	salario	codigo	nombre	descripcion	cupo	turno	PROFESOR_id
▶	2	María Emilia	Paz	1984-07-15	72000	102	Matemática Discreta	NULL	20	Tarde	2
	4	Lucía	Díaz	1991-02-24	45000	103	Programación Java	POO en Java	35	Noche	4
	5	Raúl	Martínez	1980-10-15	85000	104	Programación Web	HTML	35	Noche	5
	6	Mabel	Ríos	1982-06-12	83000	105	Programación C#	.NET, Visual Studio 2019	30	Noche	6

Fuente: elaboración propia.

IN



Cuando tenemos una lista de posibles valores para un campo, el operador IN es de mucha utilidad. IN nos permite especificar una lista de valores posibles en la sección del WHERE.

```
SELECT * FROM table
WHERE campo1 IN (valor1, valor2, ...);
```

Ejemplo:

```
SELECT * FROM curso
WHERE turno IN ("Mañana", "Tarde");
```

Con esa consulta, estamos pidiendo que todos los cursos se dicten en el turno mañana o en el turno tarde. Es una forma corta de escribir condiciones anidadas con OR.

Utilizando OR, esa consulta se escribiría de la siguiente manera:

```
SELECT * FROM curso
WHERE turno = "Mañana"
OR turno = "Tarde";
```

Con el uso del IN, podemos hacer la consulta más compacta.

ORDER BY

Para ordenar el resultado de una consulta en orden ascendente o descendente (teniendo en cuenta uno o más campos), utilizamos ORDER BY, el cual nos permite ordenar columnas de consultas simples (a una tabla) o compuestas (muchas tablas).

Por defecto, ORDER BY ordena los datos de manera ascendente.

Ejemplo:

```
SELECT p.nombre, p.apellido, c.nombre as 'curso'
FROM PROFESOR p INNER JOIN CURSO c ON p.id =
c.PROFESOR_id
ORDER BY c.nombre;
```

Resultado:

Tabla 31. Resultado



	nombre	apellido	curso
▶	Juan	Pérez	Algoritmos
	María Emilia	Paz	Matemática Discreta
	Mabel	Ríos	Programación C#
	Lucía	Díaz	Programación Java
	Raúl	Martínez	Programación Web

Fuente: elaboración propia.

Si quisiéramos hacerlo en forma descendente, esta sería la sintaxis:

```
SELECT p.nombre, p.apellido, c.nombre  
FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id  
ORDER BY c.nombre DESC;
```

Y si quisiéramos realizar el orden utilizando varios campos de diferentes tablas, la sintaxis sería esta:

```
SELECT p.nombre as "Nombre", p.apellido as "Apellido", c.nombre  
as "Curso"  
FROM PROFESOR p INNER JOIN CURSO c ON p.id =  
c.PROFESOR_id  
ORDER BY c.nombre DESC;
```

Resultado:

Tabla 32. Resultado

	nombre	apellido	curso
▶	Raúl	Martínez	Programación Web
	Lucía	Díaz	Programación Java
	Mabel	Ríos	Programación C#
	María Emilia	Paz	Matemática Discreta
	Juan	Pérez	Algoritmos

Fuente: elaboración propia.

Esa consulta devolverá los nombres de los cursos que dicta cada profesor, ordenados descendientemente.



LIMIT

La palabra reservada LIMIT sirve para especificar la cantidad máxima de registros que queremos en el resultado de una consulta.

```
SELECT *  
FROM tabla  
WHERE condiciones  
LIMIT n; → Donde n es un número entero positivo
```

Es muy útil cuando se tiene una tabla con un gran volumen de datos y solo queremos ver los primeros registros.

Ejemplo:

```
SELECT p.nombre, p.apellido, c.nombre  
FROM PROFESOR as p INNER JOIN CURSO as c ON p.id =  
c.PROFESOR_id  
LIMIT 2;
```

Devolverá solamente dos registros, independientemente de la cantidad resultante del cruce entre PROFESOR y CURSO.

CASE

El CASE va incluido en la parte del SELECT de una consulta. Sirve para evaluar varias condiciones y retornar un valor si estas son verdaderas. Evalúa las condiciones según el orden en el que aparecen. Si ninguna condición se cumple, se devolverá el valor que aparece después del ELSE.

```
SELECT CASE  
  
    WHEN condicion1 THEN resultado1  
    WHEN condicion2 THEN resultado2  
    WHEN condicionN THEN resultadoN  
  
    ELSE result  
  
END nombre_columna  
FROM tabla;
```

Ejemplo:



```
SELECT codigo,  
       nombre,  
       CASE  
           WHEN descripcion is null THEN 'El curso no tiene  
descripción'  
           ELSE descripcion  
       END 'Descripción'  
FROM curso;
```

Esto retornaría lo siguiente:

Tabla 33. SELECT CASE

	codigo	nombre	Descripción
▶	101	Algoritmos	Algoritmos y estructuras de datos
	102	Matemática Discreta	El curso no tiene descripción
	103	Programación Java	POO en Java
	104	Programación Web	El curso no tiene descripción
	105	Programación C#	.NET, Visual Studio 2019

Fuente: elaboración propia.

UNION

El operador UNION sirve para unir (combinar) el resultado de dos o más consultas. Para que eso sea posible, las consultas que se quieren unir deben devolver la misma cantidad de columnas.

Ejemplo:

```
select nombre, apellido  
from estudiante  
union  
select nombre, apellido  
from profesor;
```

Resultado:

Tabla 34. Resultado



	nombre	apellido
►	Romina	Nieva
	Brenda	Medrano
	Ramiro	Ríos
	Cristian	Gómez
	María	Velazquez
	Alexis	Reinoso
	Gabriel	Morales
	Lourdes	Martinez
	Juan	Pérez
	María Emilia	Paz
	Martín	Correa
	Lucía	Díaz
	Raúl	Martínez
	Mabel	Ríos

Fuente: elaboración propia.

Supongamos que intentáramos ejecutar la siguiente consulta para mostrar, en el caso de los estudiantes, la carrera que siguen:

```
select nombre, apellido, carrera
from alkemy.estudiante
union
select nombre, apellido
from alkemy.profesor;
```

En ese caso, obtendríamos error:

Error Code: 1222. The used SELECT statements have a different number of columns

Este error se debe a que en el primer SELECT queremos obtener tres campos (nombre, apellido y carrera) y en el segundo SELECT, solamente dos campos (nombre y apellido). Para hacer un UNION, se requiere que las consultas que se van a unir tengan la misma cantidad de campos y, además, que tengan tipos de datos compatibles

Otro detalle para tener en cuenta con este operador es que solo devuelve registros distintos.

Ejemplo:

Agregamos un estudiante con el mismo nombre y apellido de un profesor:

```
INSERT INTO estudiante ('legajo', 'nombre', 'apellido',
'fecha_nacimiento', 'carrera') VALUES ('30495', 'Lucía', 'Díaz',
'2000-05-12', 'Mecánica');
```



Como resultado, tendremos nueve estudiantes:

Tabla 35. Estudiantes

	legajo	nombre	apellido	fecha_nacimiento	carrera
►	30495	Lucía	Díaz	2000-05-12	Mecánica
	36485	Romina	Nieva	1999-11-26	Mecánica
	39685	Brenda	Medrano	2000-09-25	Sistemas
	41258	Ramiro	Ríos	1994-12-06	Sistemas
	43651	Cristian	Gómez	1995-03-19	Mecánica
	47521	María	Velazquez	1998-01-02	Sistemas
	47961	Alexis	Reinoso	1994-12-17	Sistemas
	48952	Gabriel	Morales	1996-10-03	Sistemas
	51200	Lourdes	Martinez	2001-12-13	Sistemas

Fuente: elaboración propia.

Y seis profesores:

Tabla 36. Profesores

	id	nombre	apellido	fecha_nacimiento	salario
►	1	Juan	Pérez	1990-06-06	55000
	2	María Emilia	Paz	1984-07-15	72000
	3	Martín	Correa	1987-12-07	63000
	4	Lucía	Díaz	1991-02-24	45000
	5	Raúl	Martínez	1980-10-15	85000
	6	Mabel	Ríos	1982-06-12	83000

Fuente: elaboración propia.

Si ejecutamos la consulta para ver el nombre y apellido de profesores y estudiantes, obtenemos 14 registros:

```
select nombre, apellido
from estudiante
union
select nombre, apellido
from profesor;
```



Resultado:

Tabla 37. Resultado

	nombre	apellido
►	Lucía	Díaz
	Romina	Nieva
	Brenda	Medrano
	Ramiro	Ríos
	Cristian	Gómez
	María	Velazquez
	Alexis	Reinoso
	Gabriel	Morales
	Lourdes	Martinez
	Juan	Pérez
	María E...	Paz
	Martín	Correa
	Raúl	Martínez
	Mabel	Ríos

Fuente: elaboración propia.

Si quisiéramos obtener todos los registros, sin eliminación de repetidos, tenemos que utilizar UNION ALL:

```
select nombre, apellido
from estudiante
unión all
select nombre, apellido
from profesor;
```

Resultado:

Tabla 38. Resultado



	nombre	apellido
▶	Lucía	Díaz
	Romina	Nieva
	Brenda	Medrano
	Ramiro	Ríos
	Cristian	Gómez
	María	Velazquez
	Alexis	Reinoso
	Gabriel	Morales
	Lourdes	Martinez
	Juan	Pérez
	María E...	Paz
	Martín	Correa
	Lucía	Díaz
	Raúl	Martínez
	Mabel	Ríos

Fuente: elaboración propia.

Documentación

Teoría y ejercicios SQL:


Fuente: W 3 Schools (s. f. d). SQL Tutorial. Recuperado de <https://www.w3schools.com/sql/default.asp>

Resumen consultas SQL:

Fuente: Universidad de Sevilla (2013). Consultas SQL. Recuperado de <https://www.cs.us.es/blogs/bd2013/files/2013/09/Consultas-SQL.pdf>

Videos

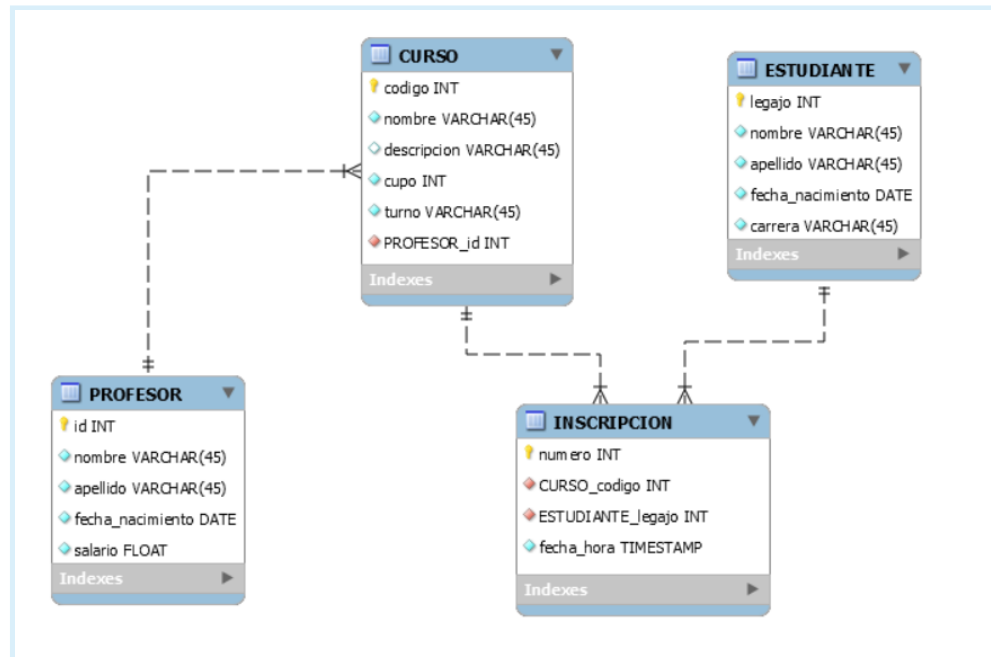
Consultas con más de una tabla:

Fuente: SEO - Blogger y Más [SEO - Blogger y Más], (13 de noviembre de 2013). Crear consultas con MAS de una tabla en SQL SERVER  [YouTube]. Recuperado de https://www.youtube.com/watch?v=SVP4nFnD7_w



Ejercitación 2

Figura 4: Modelo relacional



Fuente: [Imagen sin título sobre modelo relacional]. (s.f.).

Para este modelo relacional, escribe la siguiente información:

- 1) Nombre, apellido y cursos que realiza cada estudiante
- 2) Nombre, apellido y cursos que realiza cada estudiante, ordenados por el nombre del curso
- 3) Nombre, apellido y cursos que dicta cada profesor
- 4) Nombre, apellido y cursos que dicta cada profesor, ordenados por el nombre del curso
- 5) Cupo disponible para cada curso (si el cupo es de 35 estudiantes y hay 5 inscriptos, el cupo disponible será 30)
- 6) Cursos cuyo cupo disponible sea menor que 10

Formato de entrega: la actividad se entrega a través de una URL correspondiente al repositorio sobre el que se haya trabajado.