Base de datos 1

Alumno: Santiago Vietto

Docente: Federico Luis Garofalo

DNI: 42654882

Institución: UCC

Año: 2021

Introducción

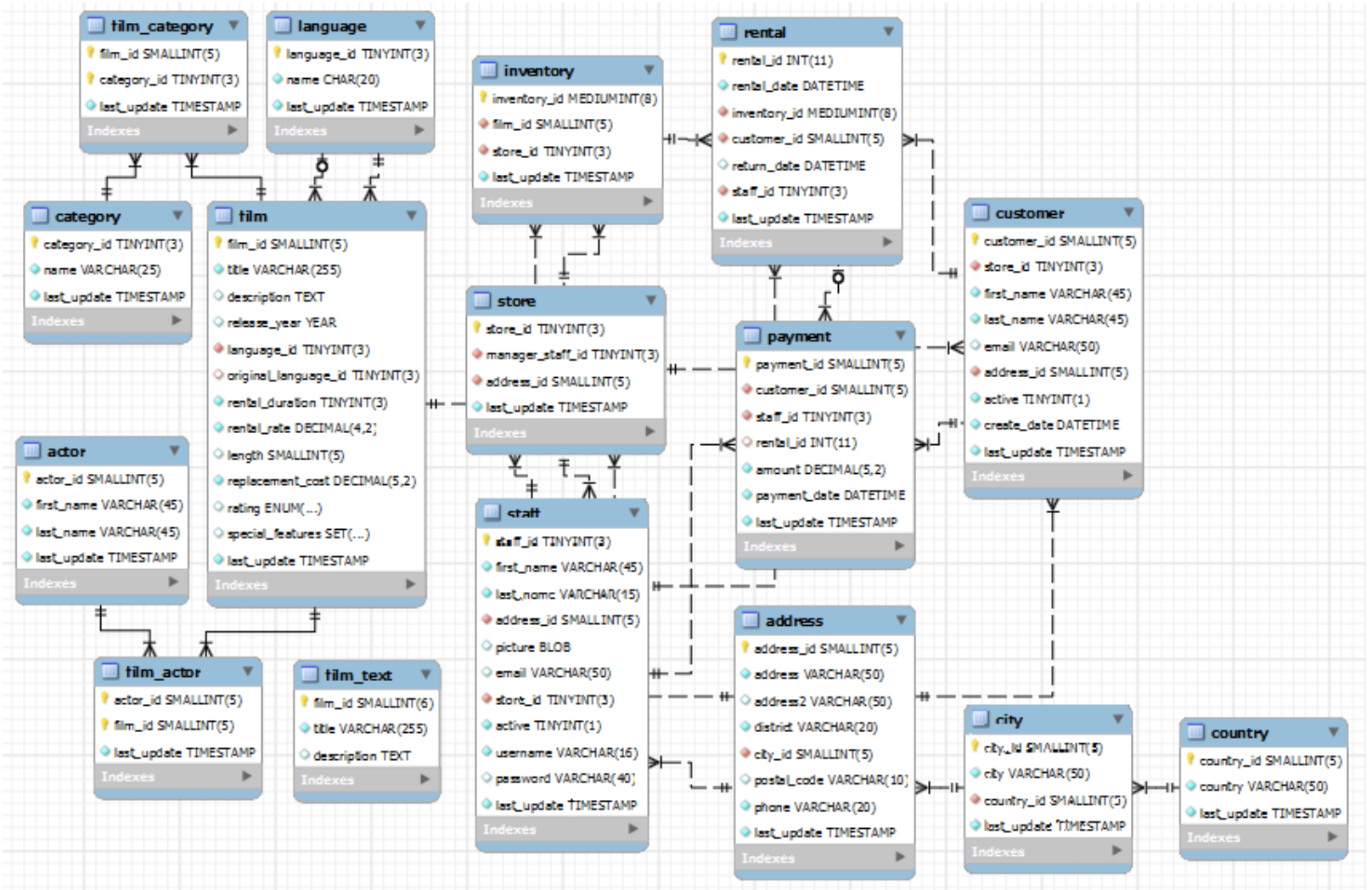
_ Antes vimos como diseñar la base de datos para que las preguntas que le hagamos a la base de datos sean lo más eficiente posibles. Lo que hacemos ahora es trabajar con los datos. Ahora utilizamos un motor de base de datos, donde lo llenamos de datos y trabajamos con las consultas.

_ Al motor de base de datos hay muchas formas de instalarlo, una de las formas más fáciles es con una herramienta que se llama XAMPP que básicamente es un servidor Apache que se instala localmente en nuestra computadora, y dentro de este ya tenemos una instancia de MySQL corriendo, configurada, con un usuario root con el que se pueden hacer consultas, instalar bases nuevas, crear o quitar tablas, etc, sin tener que renegar con los permisos. Adentro de este además viene Perl, PHP, entre otras, que no usaremos, pero básicamente es un entorno de desarrollo para desarrolladores backend. En el panel de control de XAMPP tenemos que levantar Apache y MySQL, donde ambos tienen que estar en verde, y al terminar debemos darle stop.

_ En el buscador de Google colocamos el enlace localhost/phpmyadmin, que es un administración visual, este nos lleva a un panel que básicamente es un entorno visual agradable a un usuario, por el que vamos a poder ver todas las bases de datos que tengamos instaladas, y además vamos a poder crear tablas nuevas, alterar tablas, ver las tuplas de una tabla, tirar consultas, entre otras cosas.

_ Para crear la tabla en el phpMyAdmin, ponemos New, le agregamos un nombre, y le damos crear. Una vez creada vamos a importar, le damos a “browse” y buscamos los archivos SQL correspondientes y le damos a Go o Continuar y asi con todos.

Base de dato para el practico



_ La base de datos trata de un video club. Viendo esta tabla por arriba, vemos que tenemos actores con sus datos, en donde si un actor trabaja en muchas películas y una película tiene muchos actores, necesitamos una tabla pasarela para una relación muchos a muchos. A su vez, una película puede tener muchas categorías, y puede haber muchas categorías de películas, por ende tenemos una tabla pasarela entre estas dos. La película va a tener un lenguaje. Vemos que hay un negocio o store con sus atributos, donde se maneja el personal o staff con sus respetivos atributos también, donde estos tienen una dirección que se relaciona con una ciudad, y el país. El negocio o store tiene un inventario de películas. Cuando viene un cliente o customer (con sus atributos), este puede en un video club hacer más de una renta. Para rentar una película es necesario que la película este en el inventario, la entidad renta tiene sus atributos.

Ejemplo 1: listar todos los actores que hay:

SELECT *
FROM actor

Ejemplo 2: listar todos los actores que son argentinos. Pero en este caso no podemos porque no está la relación entre actores y país, o no existe en la entidad actor un atributo de nacionalidad.

Ejemplo 3: quiero ver el nombre del actor con ID igual a 16:

SELECT first_name
FROM actor
WHERE actor_id = 16

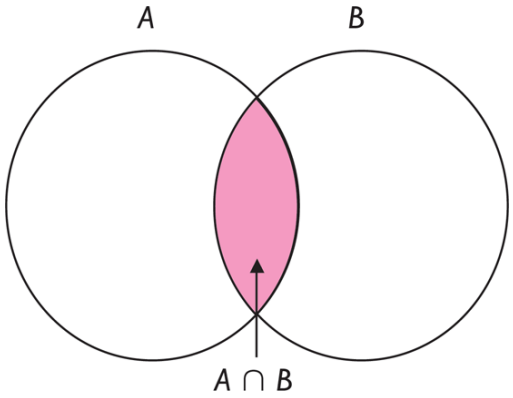
Consulta a la base de datos

_ Las respuestas a una consulta siempre son tablas con elementos. Cuando hacemos el producto cartesiano entre una tabla y otro, lo que hacemos es combinar tablas. Y lo que podemos corroborar en la tabla resultante es que si lo que nosotros preguntamos era correcto o no. En el administrador phpMyAdmin, en la sección de SQL, por defecto tenemos en los datos una consulta que es por ejemplo:

SELECT *
FROM 'actor'
WHERE 1

_ Las comillas no son necesarias. Cuando editemos código y corramos, para volver a recuperar lo que escribimos apretamos el botón "Mostrar ventana de consultas SQL". Una consulta mínima y básica para correr necesita de dos sentencias, el SELECT y el FROM. Para concatenar condiciones en un WHERE usamos un AND, o un OR, dependiendo el ejercicio. En el gráfico de la base de datos, vemos un triángulo rojo, que significa que es una clave foránea, es decir, un ID que apunta a otra tabla.

_ Debemos tener en cuenta las intersecciones de queries (consultas):



_ En los ejercicios de SQL hay mucho de teoría de conjuntos. Las consultas anidadas se pueden hacer de otra forma más simple por ejemplo con los JOIN.

_ El SELECT son todos los datos que queremos mostrar, FROM son las tablas donde vamos a buscar la información, WHERE son las restricciones, podemos agrupar por con un GROUP BY, tenemos el HAVING, podemos ordenar por un criterio con el ORDER BY. Se puede hacer un LIMIT para limitar los resultados. Con el COUNT siempre contamos la cantidad de veces que aparece el ID de una entidad. Si agrupamos, con un GROUP BY, agrupamos por el ID de una entidad. El DISTINCT y el GROUP BY cumplen una función similar, ya que el DISTINCT cada vez que encuentra el mismo ID lo ignora, y siempre cuando agrupamos por algo, cada grupo es una sola tupla. Las funciones de agregación se aplican sobre los grupos disponibles.

_ Hay una regla de MySQL Server (Microsoft) en la que debemos saber, que es que todo lo que uno pone en el SELECT tiene que ponerlo en el GROUP BY, siempre y cuando el GROUP BY este puesto en la consulta. Donde si nosotros agrupamos primero por lo más atómicamente posible (ID), podemos seguir aplicando todos los GROUP BY que queramos sin cambiar el resultado, porque ya llegamos a lo más chico que se puede. Para el parcial no es obligatorio.

_ Como las funciones de agregación se aplican en un grupo, por ejemplo:

```
SELECT film.*, COUNT(film.film_id) AS stock
…
HAVING stock = 2
```

_ Esto es lo mismo que hacer:

```
SELECT film.*
…
HAVING COUNT(film.film_id) = 2
```

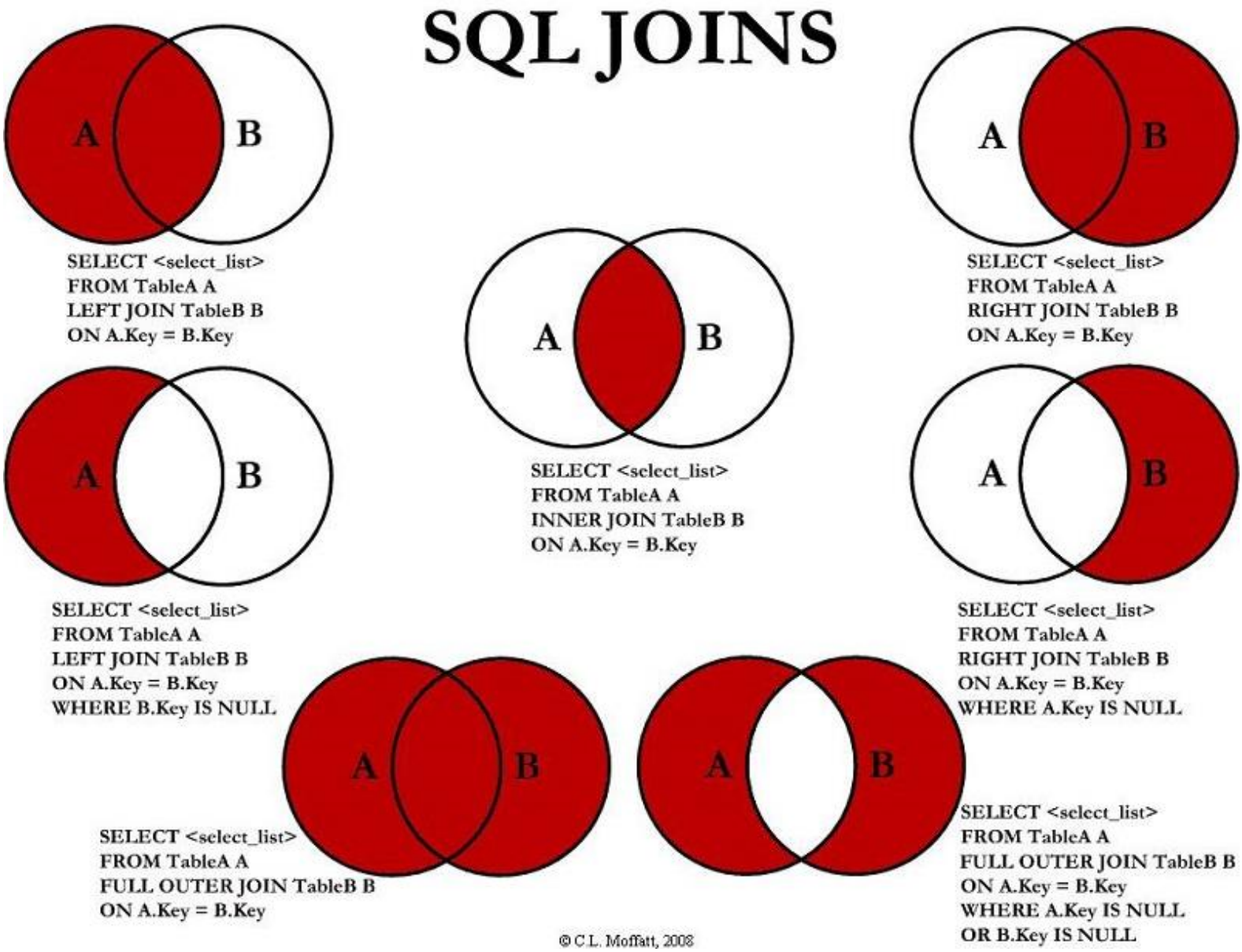
_ Donde el HAVING se ejecuta sobre el grupo de ID. Pero por regla general todas las funciones de agregación las mandamos al SELECT.

SQL Joins

_ Recordamos que para hacer una unión entre dos o varias tablas, lo que solemos hacer es un producto cartesiano entre una tabla y la tabla consecutiva. Lo que sucede con esto es que, si o si cuando hacemos un producto cartesiano nunca vamos a querer todos los resultados que nos devuelve la nueva tabla, porque por ejemplo si tenemos 10 películas y 10 cosas en el inventario, un producto cartesiano nos va a dar 100 tuplas y probablemente no las necesitemos porque quizás nos interese saber por ejemplo aquellos productos en inventario que corresponden a la película que buscamos, donde generalmente para evitar esto es en el WHERE para decir donde se cumple una determinada condición.

_ JOIN es una forma de unir dos tablas de la misma forma en la que la haríamos con un producto cartesiano y un WHERE, pero el JOIN ya tiene incluida la condición de unión, y esto es bueno ya que una vez que hicimos todos los JOINS a todas las tablas ya tenemos todos los WHERE lógicos de vinculación de ID y clave foráneas ya aplicados y solamente hacemos los WHERE que tengan que ver con la consigna en sí y no con la unión de las tablas. Esto queda más prolijo, es más performante, ya que solo aplicamos el WHERE sobre las tuplas que ya están unidas.

_ A continuación analizamos el grafico:



_ El LEFT y RIGHT JOIN son lo mismo, ya que todo depende de qué lado pongamos la tabla a analizar con otra.

Consulta a la base de datos - parte 2

_ Nosotros en un SELECT además de pedir campos que estén presentes en la tabla resultante, podemos inventar campos, es decir, crear columnas con un dato string. Para agregar una fila debajo de lo que ya tenemos y encima es otra consulta distinta, usamos la palabra reservada UNION que nos permite unir el resultado de una consulta con otra siempre y cuando ambas tengan la misma cantidad de columnas.

_ Asi como tenemos el IN y el NOT IN, tenemos el ALL y el ANY. Donde por ejemplo si decimos que queremos los datos donde la ganancia sea mayor a todos los promedios usamos el ALL, y si nos aclara que debe ser mayor a algunos promedios usamos el ANY.