¿Cómo funcionan los sistemas de bases de datos?

Parte I - Ciclo de vida de una consulta y manejo de disco

Si estás leyendo esto es porque probablemente, en algún punto de tu carrera, has tenido que utilizar algún sistema de bases de datos. Quizás has tenido que escribir alguna consulta en SQL, o has tenido que diseñar una base de datos utilizando el modelo E/R. Si eres desarrollador web probablemente has tenido que implementar las operaciones CRUD, quizás utilizando algún ORM que te esconde las operaciones directas que se hacen en una base de datos.

Por lo mismo, quizás más de alguna vez te has preguntado cómo lo hace el sistema de bases de datos para responder una consulta; más aún, quizás a veces te has preguntado por qué el sistema se demora tanto en responder consultas que parecen ser tan sencillas. Para entender todo esto, es importante conocer:

- El ciclo de vida de una consulta SQL.
- Cómo se guardan los datos en el disco duro.
- Cómo se indexan los datos.
- Cómo los algoritmos hacen uso de los índices.

La idea de esta serie de lecturas es responder cada una de estas preguntas, y así, entender de mejor forma cómo trabaja un sistema de bases de datos.

Sobre este material: El contenido que voy a enseñar en estas lecturas es una versión de alto nivel de lo que aprendí en el curso de Implementación de Bases de Datos que dicta el profesor Cristian Riveros, y es (más o menos) lo que enseñaba yo en el curso de Bases de Datos que dicté durante 5 años. Además está complementado con parte de la experiencia adquirida durante todos estos años trabajando en el área de Bases de Datos, tanto a nivel teórico como práctico.

Sobre el autor: Mi nombre es Adrián Soto Suárez y durante mi carrera he estudiado bastante este tema. Para más detalles sobre mí puedes ir a http://adriansoto.cl.

1. Ciclo de vida de una consulta SQL

Para partir, necesitamos conocer el ciclo de vida de una consulta SQL. Como bien sabemos, un sistema de bases de datos (como Postgres, MySQL, etc.) tiene muchas componentes. Por ejemplo, hay una componente que hace cargo del manejo concurrente, otra componente que hace cargo de la interacción entre la memoria principal y secundaria, otra componente que maneja los distintos usuarios, y así. En definitiva, un sistema de bases de datos es una pieza de software **muy** compleja.

Una componente clave en los sistemas de bases de datos es el procesador de consultas. Esta componente se encarga de tomar el *string* de una consulta, parsear este *string* para ejecutarlo en el sistema y así entregar la respuesta. Suena bastante sencillo, pero hay hartas sutilezas en el proceso. Para entender esta parte vamos a partir con un ejemplo. Supongamos que tenemos dos tablas:

```
Personas(pid, nombre)
Mascotas(mid, nombre_mascota, pid)
```

La primera tabla almacena datos de personas, en concreto, su identificador y su nombre. La segunda tabla almacena datos de mascotas; aquí guardamos el identificador de la mascota y su nombre, pero además, el identificador del usuario que corresponde al dueño de esta mascota. Por simplicidad, asumiremos que una mascota tiene un único dueño. Ahora, imaginemos que queremos todos los nombres de las mascotas del usuario con identificador 6. En SQL, podemos pedir esto con la siguiente consulta:

```
SELECT nombre_mascota
FROM Personas AS P, Mascotas AS M
WHERE P.pid = M.pid AND P.pid = 6
```

Ahora, ¿cómo se evalúa esta consulta?. A grandes rasgos tenemos dos opciones:

- Cruzar la información en las tablas `Personas` y `Mascotas`, para luego dejar solamente las filas que corresponden a la persona con identificador igual a 6. Finalmente, dejamos la columna que necesitamos, que sería el nombre de la mascota.
- Primero, filtar la tabla `Personas`, dejando solo la fila que corresponde a la persona con identificador igual a 6. Luego, cruzamos la información con la tabla `Mascotas` y finalmente dejamos la columna del nombre de la mascota.

Intuitivamente, la segunda idea es mucho mejor que la primera porque al cruzar la información, la tabla de `Personas` ya viene filtrada y así esta operación será mucho más "liviana". Pero una pregunta natural aquí es: ¿qué es lo que hace el sistema de bases de datos?. La respuesta es la siguiente:

1. Parseo de la consulta. Primero, el sistema valida la consulta para asegurar que la sintaxis se esté respetando. Luego, se parsea el *string*, convirtiéndolo en una consulta básica de álgebra relacional:

$$\pi_{\text{M.nombre\_mascota}}(\sigma_{\text{P.pid} = \text{M.pid} \,\wedge\, \text{P.pid} = 6}(\texttt{Personas} \times \texttt{Mascotas}))$$

2. Optimización de la consulta. El sistema toma la consulta de álgebra relacional y lo envía al optimizador de consultas para construir una mejor versión de la consulta. Primero vamos a cambiar los productos cruz por *joins* dentro de lo posible:

$$\pi_{\text{M.nombre\_mascota}}(\sigma_{\text{P.pid} = 6}(\texttt{Personas} \bowtie \texttt{Mascotas}))$$

y luego vamos a hacer "*push*" de la selección lo más posible:

$$\pi_{\text{M.nombre\_mascota}}(\sigma_{\text{P.pid} = 6}(\texttt{Personas}) \bowtie \texttt{Mascotas})$$

3. Ejecución de la consulta. Finalmente, el motor escoge un algoritmo para cada operador. Por ejemplo, hay varias formas de ejecutar un *join*, dependiendo si las tablas están indexadas o no (esto lo veremos en la siguiente lectura). Lo mismo pasa con las selecciones y todos los operadores que pueden haber. Así, finalmente obtenemos la respuesta de nuestra consulta.

Un detalle muy importante aquí es la importancia del **álgebra relacional**. No es solo un lenguaje que le gusta a la gente que hace teoría, sino que es parte fundamental en la optimización de consultas. Esto porque, como todo buen álgebra, tiene propiedades que nos permiten reescribir las consultas. Por ejemplo, transforma los productos cruz con selección en los *joins* respectivos, o empuja la selección lo más posible. Así, cada vez que escribimos una consulta y esta se "compila" al álgebra relacional, el motor aplicará reglas que permiten tener una versión más eficiente de la consulta. Pero, ¿siempre encontraremos la versión más eficiente?. Lamentablemente no, porque como podrás suponer, reescribir la consulta no siempre es trivial. Por ejemplo, si tenemos operadores como `EXISTS`, `ALL`, `IN` que involucran consultas

anidadas, se hace más complicado encontrar una reescritura. Sin embargo, para consultas de `SELECT - FROM - WHERE`, el motor no debería tener problemas.

2. Almacenamiento de los datos

Después de entender el ciclo de vida de una consulta, ahora vamos a entrar en otro terreno. Discutiremos cómo se almacenan los datos en el disco duro. Esto es súper importante porque nos permite entender por qué los índices son la clave para la evaluación de consultas.

2.1. El disco duro

Como bien sabemos, para lograr hacer que los datos sean persistentes, estos se deben guardar en memoria secundaria. Esto es, **el disco duro de nuestro computador**. Para ejemplificar el funcionamiento de un disco duro, miremos la figura Figura 1.

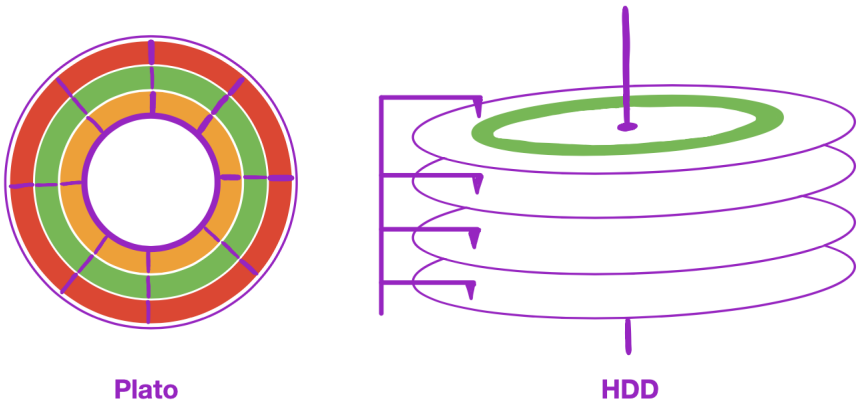

Figura 1: representación gráfica de un plato y un HDD.

La parte básica de los discos duros (HDD) son los platos. A su vez, los platos se componen de distintos *tracks*; por ejemplo, el plato en la figura tiene los *tracks* rojo, verde y naranjo. En estos tracks es donde se van escribiendo los bytes de los distintos archivos que vamos guardando, uno al lado del otro.

A grandes rasgos (omitiendo detalles por simplicidad), el sistema operativo organizará los tracks como un conjunto de varias páginas **páginas** (en la figura, vemos que cada track se divide en varias páginas, separadas por líneas púrpura). Cada página es la unidad mínima que el sistema operativo puede llevar a la memoria RAM, y en general, suelen ser del orden de 4Kb. Para que te hagas una idea, en un *track* puedes encontrar cientos de páginas, y en cada plato puedes encontrar miles de *tracks*.

Así, cuando tú abres un archivo en tu computador, el sistema operativo debe ir a buscar al disco duro todas las páginas respectivas y dejarlas en memoria RAM mientras el archivo este abierto. Hay más detalles relacionados a cómo el sistema de archivos maneja la memoria RAM, pero eso es más bien parte de un curso de Sistemas Operativos.

Lo que a nosotros nos importa, es que cuando prendemos el computador, el disco duro está en constante rotación, y cada vez que el sistema operativo quiere una página del disco duro, este último debe posicionar su cabeza lectora (la aguja que se ve en la Figura 1) en el *track* en donde está la página, y debe esperar que el disco siga rotando hasta que encuentre la página que desea. Luego los bytes de esa página se transmiten a la memoria RAM.

Como podrás esperar, este proceso no es muy rápido; puedes suponer que leer una página del disco duro tarda en promedio 0.1 ms, por lo que si quieres leer un archivo de 1000 páginas, esto se demora aproximadamente un segundo, que es un tiempo considerable en este contexto. Así, el principal cuello de botella para los sistemas de bases de datos es la lectura y escritura (I/O) de páginas en el disco duro, y es lo que vamos a intentar minimizar cada vez que utilicemos nuestro sistema de bases de datos.

2.2. ¿Cómo se guardan los datos de una base de datos?

Dado que ya sabemos cómo funciona el disco duro, ahora podemos entender cómo se guardan las tuplas de una tabla en él. Recordemos que si estamos usando un sistema de bases de datos es razonable pensar que los datos no van a caber en memoria RAM, por lo que debemos hacer uso de la memoria secundaria.

En su versión más simple, el almacenamiento en disco duro es como lo muestra la Figura 2. Esto es, las tuplas se almacenan adyacentes unas a otras.

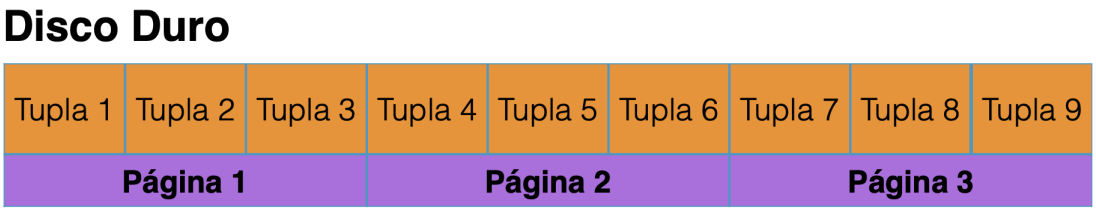

Figura 2: representación del almacenamiento de una tabla en el disco duro.

Ojo que si la tabla es muy grande, existe la posibilidad que no todas las tuplas sean adyacentes unas a las otras, y tengamos que dar saltos. Este fenómeno (en general, y no solamente para las bases de datos) se conoce como fragmentación del disco duro. A medida que vamos utilizando el computador o servidor, y lo vamos llenando de archivos, no necesariamente vamos a tener todas los archivos de un determinado tipo (imágenes, videos, documentos) adyacentes. Si una tabla va creciendo a lo largo del tiempo (por ejemplo, una tabla en la que vamos registrando compras de un *e-commerce*), es posible que no todas las tuplas estén adyacentes, y el sistema de archivos tenga que trabajar bastante para entregar todas las tuplas de una tabla.

Ahora, volviendo a la Figura 2, ¿qué pasa si queremos una tabla `Personas(id, nombre)` y queremos encontrar la persona con identificador igual a 6? Si no tenemos mayor información de la tabla, vamos a tener que necesariamente recorrer **toda la tabla** para responder esta consulta. Como supondrás, para una relación de un tamaño no muy grande esto se vuelve un problema. Considera una relación que utilice 1000 páginas; esto significa que cada vez que queramos una persona en particular de nuestra base de datos, ¡nos vamos a demorar 1 segundo!. Este tiempo **no es razonable** para una consulta tan sencilla y una base de datos de ese tamaño. Si te preguntas el tamaño en tuplas de una tabla de 1000 páginas, esto equivale a $4\text{Kb} \cdot 1000 = 4\text{Mb}$, en donde si suponemos que cada tupla es de 100 bytes, sería una relación de aproximadamente 40000 tuplas.

Entonces, ¿por qué los sistemas de bases de datos funcionan tan rápido?. Esto es porque sus tablas están **indexadas**. Esto es, exsiten estructuras de datos que me permiten encontrar de forma mucho más rápida las páginas de disco duro que estoy buscando. Esto lo vamos a estudiar en la siguiente lectura.

3. Palabras al cierre

Como vimos en esta lectura, un sistema de base de datos es una pieza de software muy compleja. Para poder responder consultas, aunque sean sencillas, vamos a tener que usar técnicas avanzadas.

Como detalle a lo señalado aquí, hay que aclarar que los discos tradicionales (HDD) cada vez son menos frecuentes en los computadores personales, ya que han sido reemplazados por los discos de estado sólido (SSD). Estos discos remueven la cabeza lectora por un mecanísmo mucho más rápido, haciendo que la velocidad de I/O del disco duro sea casi tan rápido como acceder a la memoria RAM. Ahora bien, estos discos siguen siendo bastante caros, y a la hora de almacenar grandes cantidades de información, el uso de estos discos no escala (a no ser que tengamos infinitos recursos).

Lo segundo, es recalcar que los sistemas de bases de datos están pensados para ser usados con datos que no me van a caber en memoria (además de proveer acceso concurrente, persistencia de los datos, entre otros). La forma de trabajar con datos (y archivos en general) que no caben en memoria, es apoyarme en el disco duro para ir leyendo las páginas del disco duro a medida que las voy necesitando. Sin embargo, esto también es más bien parte de un curso de Sistemas Operativos, más que de Bases de Datos.