CC42A: Bases de Datos
Auxiliar: Optimizacion Consultas SQL

Rodrigo Paredes Moraleda

23 de junio de 2002

1. Caso de estudio

Considere el siguiente esquema:

CLIENTE{CLI_ID, CLI_NOMB, CLI_RENTA_ANUAL, CLI_TIPO}
EMBARQUE{EMB_ID, EMB_ID_CLI, EMB_PESO, EMB_ID_CAMION, EMB_DESTINO, EMB_FECHA}
Clave Fornea: EMB_CLI_ID referencia a CLI_ID en CLIENTE
Clave Fornea: EMB_DESTINO referencia a CIU_NOMBRE en CIUDAD
Clave Fornea: EMB_ID_CAMION referencia a CAM_ID en CAMION
CAMION{CAM_ID, CAM_NOMBRE_CHOFER}
CIUDAD{CIU_NOMBRE, CIU_POBLACION}

Considere además los siguientes tamaños de las tablas:

- CLIENTE: 200 registros
- EMBARQUE: 7000 registros
- CAMION: 30 registros
- CIUDAD: 20 registros

1.1. Optimize la consulta: Nombre del cliente 100

En SQL la consulta es:

```
SELECT CLI_NOMB
  FROM CLIENTE
 WHERE CLI_ID = 100
```

Pasando esto al álgebra relacional

- `SELECT`: Proyección (π)
- `FROM`: Producto cartesiano ($\times$), que usamos para hacer los joins
- `WHERE`: Selección (σ)

Con esto la consulta resultante es:

$$\pi_{\texttt{CLI\_NOMBRE}} \left(\sigma_{\texttt{CLI\_ID}=100} \left(\texttt{CLIENTE}\right)\right)$$

Para resolver esto, sólo necesitamos el identificador del cliente y el nombre, así que aprovechamos esto para hacer la siguiente optimización:

$$\pi_{\texttt{CLI\_NOMBRE}} \left(\sigma_{\texttt{CLI\_ID}=100} \left(\pi_{\texttt{CLI\_NOMBRE,CLI\_ID}} \left(\texttt{CLIENTE}\right)\right)\right)$$

1.2. Optimize la consulta: Cómo se llaman los clientes que han enviado paquetes a Rancagua

En SQL la consulta es:

```
SELECT CLI_NOMB
  FROM CLIENTE,
       EMBARQUE
 WHERE CLI_ID = EMB_ID_CLI
   AND EMB_DESTINO = 'Rancagua'
```

Con esto la consulta resultante es:

$$\pi_{\texttt{CLI\_NOMBRE}} \left(\sigma_{\substack{\texttt{CLI\_ID} = \texttt{EMB\_ID\_CLI} \\ \texttt{EMB\_CIUDAD} = \texttt{'Rancagua'}}} \left(\texttt{CLIENTE} \times \texttt{EMBARQUE}\right)\right)$$

Para hacer esto, se deben inspeccionar $|\texttt{CLIENTE}| \times |\texttt{EMBARQUE}| = 1,400,000$ registros, aunque no necesariamente almacenamos esta cantidad de registros pues al hacer el join se eliminan muchas combinaciones.

Suponiendo que los embarques a Rancagua son aproximadamente 1000, tiene sentido inspeccionar tantas tuplas??

Vamos a considerar las siguientes reglas de optimización:

1. Hacer las selecciones y proyecciones lo antes posible.
2. Preprocesar los datos antes de hacer un join (o producto cartesiano), para esto se puede ordenar o usar/construir índices.

3. Buscar sub-expresiones comunes, lo que permite "Factorizar" o mantener resultados parciales intermedios que se utilizarán muchas veces en memoria (si son pequeños) o disco.

4. Hacer selecciones y proyecciones en cascada.

Debemos recordar que las selecciones reducen el rango de las tuplas que debemos inspeccionar, y las proyecciones reducen el tamaño de la tupla al tamaño de las componentes que nos interesan.

También hay que acordarse de la conmutatividad de los operadores de join, y de producto cartesiano.

Por último debemos recordar que la consulta optimizada no necesariamente es la "óptima", pero al menos necesita menos trabajo.

Volviendo a la consulta de interés:

$$\pi_{\texttt{CLI\_NOMBRE}} \left(\sigma_{\substack{\texttt{CLI\_ID} = \texttt{EMB\_ID\_CLI} \\ \texttt{EMB\_CIUDAD} = \texttt{'Rancagua'}}} (\texttt{CLIENTE} \times \texttt{EMBARQUE}) \right)$$

Podemos pasar la selección de $\texttt{EMB\_CIUDAD} = \texttt{'Rancagua'}$ para que opere sobre la tabla EMBARQUE y no sobre el producto cartesiano.

Además basta con considerar las componentes EMB_ID_CLI y EMB_CIUDAD de EMBARQUE, y luego de esto hacer la selección $\texttt{EMB\_CIUDAD} = \texttt{'Rancagua'}$, con esto nos queda lo siguiente:

$$\pi_{\texttt{CLI\_NOMBRE}} (\sigma_{\texttt{CLI\_ID}=\texttt{EMB\_ID\_CLI}} (\texttt{CLIENTE} \times \sigma_{\texttt{EMB\_CIUDAD}=\texttt{'Rancagua'}} (\pi_{\texttt{EMB\_ID\_CLI},\texttt{EMB\_CIUDAD}} (\texttt{EMBARQUE}))))$$

Para hacer el join, necesitamos CLI_ID y EMB_ID_CLI. Además necesitamos el atributo CLI_NOMBRE para la proyección de CLI_NOMBRE más externa:

$$\pi_{\texttt{CLI\_NOMBRE}} \left(\sigma_{\texttt{CLI\_ID}=\texttt{EMB\_ID\_CLI}} \left(\pi_{\substack{\texttt{CLI\_ID} \\ \texttt{CLI\_NOMBRE}}} (\texttt{CLIENTE}) \times \pi_{\texttt{EMB\_ID\_CLI}} \left(\sigma_{\texttt{EMB\_CIUDAD}=\texttt{'Rancagua'}} \left(\pi_{\substack{\texttt{EMB\_ID\_CLI} \\ \texttt{EMB\_CIUDAD}}} (\texttt{EMBARQUE}) \right) \right) \right) \right)$$

Noten que con estas proyecciones y selecciones, reducimos la cantidad de tuplas y atributos a inspeccionar, lo que nos permite hacer la consulta con un menor uso de memoria.

Cómo manejamos el join:

- Fuerza bruta.
- Join Merge. Ordenamos los dos conjuntos según el atributo con el que se hace el join, y luego verificamos en orden cuando se tenga el calce. Con esto pagamos el costo de las ordenaciones de ambos conjuntos, más el recorrido secuencial por ambos conjuntos ordenados.

- Uso de índices. Si tenemos un conjunto indexado y el otro no, recorremos el conjunto NO indexado, y para cada valor, utilizamos el índice para encontrar el correspondiente en el otro conjunto, con esto hacemos un recorrido secuencial del conjunto no indexado, y en el indexado accedemos a los valores gracias al índice. Si los dos conjuntos están indexados, hacemos lo mismo (recorrimos secuencialmente un conjunto y con el índice del otro conjunto recuperamos el valor correspondiente)

1.3. Optimize la consulta: Quiénes son los choferes que han conducido empaques de clientes que tienen renta anual sobre los $ 20,000,000 a ciudades con población por encima del millón

En SQL la consulta es:

```
SELECT CAM_NOMBRE_CHOFER
  FROM CLIENTE,
       EMBARQUE,
       CAMION,
       CIUDAD
 WHERE CLI_RENTA_ANUAL > 20,000,000
   AND CIU_POBLACION > 1,000,000
   AND CLI_ID = EMB_ID_CLI
   AND CAM_ID = EMB_ID_CAMION
   AND CIU_NOMBRE = EMB_DESTINO
```

Con esto la consulta resultante es:

$$\pi_{\texttt{CAM\_NOMBRE\_CHOFER}} \left(\sigma_{\substack{(1)\texttt{CLI\_RENTA\_ANUAL} > 20,000,000 \\ (2)\texttt{CIU\_POBLACION} > 1,000,000 \\ (3)\texttt{CLI\_ID} = \texttt{EMB\_ID\_CLI} \\ (4)\texttt{CAM\_ID} = \texttt{EMB\_ID\_CAMION} \\ (5)\texttt{CIU\_NOMBRE} = \texttt{EMB\_DESTINO}} (\texttt{CLIENTE} \times \texttt{EMBARQUE} \times \texttt{CAMION} \times \texttt{CIUDAD}) \right)$$

Lo primero que haremos será seleccionar un orden para la ejecución de los productos cartesianos, consideremos este orden:

(i) $\texttt{EMBARQUE} \times \texttt{CIUDAD}$

(ii) $(\texttt{EMBARQUE} \times \texttt{CIUDAD}) \times \texttt{CAMION}$

(iii) $((\texttt{EMBARQUE} \times \texttt{CIUDAD}) \times \texttt{CAMION}) \times \texttt{CLIENTE}$

Esto nos permite hacer el join de una tabla grande, contra una tabla pequeña, lo que va ha disminuir los accesos a disco (por qué, piénselo).

En segundo lugar, hay que mover las selecciones sobre una tabla lo más abajo posible, de modo que operen lo más cerca de las tablas que están inspeccionando. En nuestro caso la condición de selección (1) opera sólo sobre la tabla `CLIENTES`, luego la desplazamos de modo que actúe sobre `CLIENTES` (fijense en lo que hicimos con la selección de `EMB_CIUDAD` = `'Rancagua'` en el ejemplo anterior), supongan que esto deja 40 clientes de los 200 iniciales. Lo mismo se puede hacer con la selección (2), la que sólo opera sobre la tabla `CIUDAD`, supongan que esto deja 4 ciudades de las iniciales. Noten que con esto, dejo de hacer joins con tuplas que se que NO interesan.

En tercer lugar, movemos las selecciones sobre 2 tablas lo más abajo posible, esto es inmediatamente despues del producto cruz que involucra ambas tablas, de este modo, la condición de selección (5) se traslada sobre el producto cruz (i) (pues en este nivel ya estoy considerando las tablas involucradas: `EMBARQUE` y `CIUDAD`), la condición de selección (4) se traslada sobre el producto cruz (ii) (pues en este nivel ya estoy considerando las tablas involucradas: `EMBARQUE` y `CAMION`), la condición de selección (3) se traslada sobre el producto cruz (iii) (pues en este nivel ya estoy considerando las tablas involucradas: `EMBARQUE` y `CLIENTE`). Noten que este orden es dependiente de la forma como voy a ejecutar los productos cruz.

En cuarto lugar, vamos a hacer uso de la proyección: Para hacer la selección (1), solo necesito el atributo `CLI_RENTA_ANUAL`, y para la condición (5) el atributo `CLI_ID`. Luego previo a hacer el select (1), hago la proyección de `CLI_ID` y `CLI_RENTA_ANUAL` a la tabla `CLIENTES`, y luego de hacer la selección (1), proyecto `CLI_ID`, que es el atributo que necesito para la condición (5). Hago lo mismo para `CIUDAD`, donde para el select (5) sólo necesito `CIU_NOMBRE` y para la condición (2) `CIU_POBLACION`, luego hago la proyección de `CIU_NOMBRE` y `CIU_POBLACION`, hago la selección (2) y luego proyecto `CIU_NOMBRE`. Para `EMBARQUE`, donde para los select (3, 4 y 5) sólo necesito `EMB_CLI_ID`, `EMB_CAMION` y `EMB_DESTINO`, proyecto estos atributos. Para `CAMION`, para el select (4) sólo necesito `CAM_ID`, y para el resultado final necesito `CAM_NOMBRE_CHOFER`, luego proyecto estos atributos. Noten que con esto dejo de inspeccionar atributos que se que NO interesan.

En quinto lugar reviso si tienen sentido hacer estas proyecciones, por ejemplo la proyección sobre `CAMION` no tiene sentido, pues estoy proyectando todos los atributos de la tabla. Así mismo, la proyección de `CIU_NOMBRE` y `CIU_POBLACION`, previa al select (2) tampoco tiene sentido, pues son los dos atributos de la tabla. Las otras están bien, pues filtran información reduciendo el tamaño de las tuplas a inspeccionar (y esto nos permite inspeccionar "más" tuplas a la vez).

En sexto lugar, uso proyecciones luego de las condiciones de selección de los productos cruz (que es lo que usamos para implementar los joins). Luego del join (i) (que lo implementamos con la selección (5)), sólo me interesan los atributos `EMB_CLI_ID` y `EMB_CAMION`, que serán usados en los joins (iii) y (ii) respectivamente (esto porque ya ocupe el atributo `EMB_DESTINO` en el join (i), y este valor no me importa para el resultado). Luego del join (ii) (selección (4)), sólo me interesan los atributos `EMB_CLI_ID` (para el join (iii)) y `CAM_NOMBRE_CHOFER` para el resultado final. Y por último, luego del join (iii), proyecto `CAM_NOMBRE_CHOFER` y tengo el resultado final.

Les dejo propuesto hacer el dibujo de cada etapa, y la estimación de los tamaños de tuplas a estimar tras cada join.

Recuerden que para hacer el join (producto cruz + selección) hay tres estrategias: Fuerza bruta, Join Merge y uso de índices.

Otra estrategia para optimizar es hacer estimaciones de los resultados de las tablas durante el proceso de optimización, y de este modo seleccionar la operación a realizar dependiendo de que es más eficiente. Tablas más pequeñas requieren menos accesos a disco y esto significa menos tiempo. Revisen si esto les sirve o no para optimizar consultas.