Teoría de Bases de Datos

Jorge Pérez

Profesor Asistente Departamento de Ciencias de la Computación, Universidad de Chile. Doctor en Ciencias de la Ingeniería (2011), Magíster en Ciencias de la Ingeniería (2004) e Ingeniero Civil en Computación (2003), Pontificia Universidad Católica de Chile. Líneas de Investigación: Bases de Datos - Datos Web, Lógica en Ciencia de la Computación.

jperez@dcc.uchile.cl

El área de Bases de Datos se preocupa de almacenar, consultar y actualizar grandes volúmenes de información. Las bases de datos están presentes hoy en innumerables aplicaciones y contextos, y muchos profesionales y autodidactas no necesariamente interesados en la computación, las usan a diario. Lo que muchos posiblemente desconocen es que el área de Bases de Datos posee una muy rica teoría que la soporta. De hecho, representan uno de los mejores ejemplos de la aplicación exitosa de teoría en la práctica. En este documento presentaremos de manera introductoria, algunos de los componentes teóricos de las bases de datos, los problemas y preguntas fundamentales que surgen, y cómo las respuestas y soluciones a estos problemas han implicado el desarrollo de un área que hoy es considerada una de las más importantes en Ciencia de la Computación. Nos centraremos en la teoría de las bases de datos relacionales, pero también incluiremos brevemente otros tipos de bases de datos.

Una base de datos relacional es un conjunto de *tablas* o *relaciones* (de ahí el nombre de "relacional"). Cada tabla o relación tiene un nombre, además de nombres para cada columna llamados atributos. Por ejemplo, la siguiente base de datos de bebedores tiene dos relaciones: **frecuenta** y **sirve**, la primera con atributos **bebedor** y **bar**, y la segunda con atributos **bar** y **cerveza**[1].

frecuenta	bebedor	bar
	Juan	Constitución
	Pablo	Liguria
	Marcelo	Loreto
	...	...

sirve	bar	cerveza
	Constitución	Delirium
	Liguria	Chimay
	Liguria	Kriek
	Loreto	Chimay
	Loreto	Delirium
	...	...

El uso habitual de una base de datos es el de consultar los datos. Por ejemplo, ser capaz de obtener desde la base de datos todos los nombres de bebedores que frecuentan bares que sirven Chimay y Kriek. En una base de datos de vuelos, podría ser obtener todos los pasajeros que deben conectar entre los vuelos 1380 y 960, o en una base de datos de estudiantes y cursos, decidir si existe un alumno tomando simultáneamente Cálculo y Computación.

LENGUAJES DE CONSULTA: LÓGICA EN ACCIÓN

Una de las ventajas cruciales de las bases de datos relacionales son los llamados *lenguajes de consulta*. Tomemos por ejemplo el caso de la consulta:

C1: "Obtener los nombres de bebedores que frecuentan bares que sirven Kriek"

¿Cómo obtenemos esta información? Una posibilidad es usar un programa imperativo con la siguiente estrategia: avanzar una por una en las filas de la tabla **frecuenta** y para cada tupla **(bebedor, bar)** en esa tabla, buscar en la tabla **sirve** el nombre del bar y chequear que ese bar tiene datos para la cerveza Kriek. Si para cada posible consulta que necesitemos tuviéramos que diseñar una estrategia como la anterior, las bases de datos no serían tan usadas hoy en día.

En vez de diseñar un procedimiento para cada posible consulta, lo que se hace en una base de datos es expresar lo que queremos en un *lenguaje declarativo*, es decir, un lenguaje mucho más cercano a lo que esperamos obtener que al pro-

[1] El ejemplo de los bebedores, es un ejemplo clásico de la literatura de Bases de Datos.

cedimiento necesario para obtenerlo. El lenguaje paradigmático es la Lógica de Predicados de Primer Orden. La anterior consulta puede simplemente expresarse como:

$$C1: \exists Y(\text{frecuenta}(X,Y) \text{ AND } \text{sirve}(Y, \text{“Kriek”}))$$

Ésta es una fórmula que esencialmente obtiene todos los X (bebedores) para los que existe un valor Y tal que (X,Y) es una fila en la tabla **frecuenta** y (Y,“Kriek”) es una fila en la tabla **sirve**. Similarmente se puede hacer fácilmente una consulta, por ejemplo, para obtener los bebedores que frecuentan bares que sirven al menos dos cervezas:

$$C2: \exists Y \exists U \exists V (\text{frecuenta}(X,Y) \text{ AND } \text{sirve}(Y, U) \text{ AND } \text{sirve}(Y,V) \text{ AND } U \neq V)$$

Edgard F. Codd en 1970 [1] demostró que toda fórmula de Lógica de Primer Orden podía traducirse en una expresión de lo que él llamó Álgebra Relacional, un álgebra muy simple de operaciones sobre tablas. Así por ejemplo, la consulta C1 se puede escribir como:

$$C1': \pi_{bebedor}(\sigma_{cerveza} = \text{“Kriek”} (\text{frecuenta} \bowtie \text{sirve}))$$

El álgebra relacional se basa en un par de operaciones básicas: proyección (π) usada para quedarse con alguna de las columnas de una relación (en el ejemplo, con la columna correspondiente al bebedor), *selección* (**σ**) usada para seleccionar las filas que cumplen con cierta condición (en el ejemplo las filas tales que el atributo cerveza es “Kriek”), y producto o join (⋈) que se usa para combinar dos tablas que comparten un mismo atributo (en el ejemplo, las tablas frecuenta y sirve combinadas por su atributo común bar). Adicionalmente cuenta con las operaciones de conjuntos, unión (U) y diferencia (-), y una operación auxiliar para cambiar (dentro de una expresión) el nombre de una tabla. Codd demostró también que toda expresión del álgebra relacional podía traducirse en una fórmula de lógica de primer orden y por lo tanto el álgebra y la lógica definían exactamente las mismas consultas. El resultado de Codd es crucial porque para responder cualquier consulta en la lógica, sólo se necesitan procedimientos que implemente cada una de las operaciones básicas del álgebra. Dado que estas operaciones pueden implementarse por separado, ellas pueden optimizarse de manera específica. Esto hace que hoy las consultas a una base de datos relacional puedan responderse eficientemente.

El trabajo de Codd significó el inicio de lo que hoy conocemos como las bases de datos relacionales y es en gran parte el responsable de que las bases de datos sean tan ampliamente usadas hoy. Por un lado la lógica nos da un lenguaje simple donde expresar lo que queremos obtener. Por otro, el álgebra nos permite implementar estas consultas de manera muy eficiente. En el momento que Codd propuso su modelo basado en relaciones, álgebra y lógica, habían otros sistemas de bases de datos pero estos eran *ad hoc*, difíciles de usar y con poco soporte teórico. Por su trabajo de 1970, Codd recibió en 1981 el “ACM Turing Award”, considerado el premio Nobel de computación. Murió en 2003 a la edad de 79 años. Cabe destacar que tanto el álgebra relacional como la lógica de primer orden están hoy plasmadas en el lenguaje SQL, el estándar para consultar bases de datos relacionales.

EXPRESIVIDAD VERSUS COMPLEJIDAD

El uso de lógica en el corazón de las bases de datos nos permite estudiarlas teóricamente y responder formalmente preguntas muy interesantes. Por ejemplo ¿qué tipo de consultas son las que puede responder una base de datos relacional? Suponga que se tiene una base de datos de conexiones de vuelos directos, por ejemplo, que hay un vuelo directo entre Santiago y Nueva York, y otro entre Nueva York y Londres, etc. Suponga ahora que se quiere responder la siguiente consulta:

C3: “¿Es posible volar desde Santiago a Moscú?”

La consulta implica que no nos importa tener escalas, sólo si es que será posible completar el viaje. Ciertamente la información necesaria para responder **C3** se encuentra en la base de datos, pero se puede demostrar formalmente que esta consulta no se puede *expresar* como una fórmula de la lógica de primer orden y, por lo tanto, (por el resultado de Codd) no existe una consulta en el álgebra relacional que permita responder **C3** [2]. Esto implica que por ejemplo, una consulta de SQL tampoco podrá usarse para responder **C3**. Esencialmente, la lógica de primer orden no se puede utilizar para consultar en general por caminos de *largo arbitrario*. Y esto se extrapola a diferentes escenarios. Por ejemplo, suponga que tiene una base de datos que mantiene las dependencias de cursos en una carrera universitaria, es decir, qué curso es requisito de qué otro. Naturalmente se necesita que este conjunto de dependencias no contenga ciclos ya que implicaría que

los estudiantes no podrían completar la carrera. Es decir queremos impedir que exista una secuencia de cursos A1, A2, ..., Ak, tal que A1 es requisito de A2, A2 es requisito de A3, etc. y que finalmente Ak sea requisito de A1. Pues bien, se puede demostrar que no existe una fórmula en lógica de primer orden que me permita chequear si hay un ciclo en las dependencias de cursos; el álgebra relacional no tiene el poder expresivo suficiente para determinar la existencia de ciclos en los datos. No es difícil notar que los ciclos y los caminos de largo arbitrario están íntimamente relacionados.

Lo profundo del resultado es indicarnos las limitantes del álgebra en cuanto a su poder para responder consultas. Pero la teoría también nos ayuda a determinar qué es lo que le falta a la lógica para ser capaz de responder **C3**. En este caso lo que le falta a la lógica es recursión. No es importante la definición formal para efectos de este artículo, pero lo interesante es que dado que **C3** es una consulta natural que nos gustaría responder en un sistema de bases de datos, necesitamos agregar poder expresivo a nuestro lenguaje. En la práctica esta observación teórica se tradujo en una adición al estándar inicial de SQL (el estándar SQL 3 incluye la posibilidad de hacer recursión).

A priori suena simple y hasta atractivo lo de agregar expresividad a un lenguaje, pero ¿no estaremos perdiendo algo? La respuesta es sí, lo que se pierde es *eficiencia:* cada nueva funcionalidad agregada a un lenguaje implica la implementación de una nueva operación la que puede ser más costosa de ejecutar en un computador. Esto nos lleva al tema de complejidad.

En complejidad computacional se intenta determinar qué tantos recursos computacionales se necesitarán para resolver un problema particular. Generalmente se considera el tiempo como el más importante de estos recursos. Para el caso de las bases de datos el problema es, dada una consulta fija sobre una base de datos de tamaño N ¿cuánto tiempo tardará en ejecutar la consulta como función de N? Si ahora consideramos todas las posibles consultas en un lenguaje específico y tomamos el máximo de estas funciones obtenemos la *complejidad del lenguaje*.

Se puede demostrar que la complejidad de la lógica de primer orden es sumamente baja: la complejidad está en la clase AC^0 [3] que contiene problemas masivamente paralelizables. Esta clase está propiamente contenida en la clase de problemas solubles en tiempo polinomial, PTIME. En cambio, DATALOG que es el lenguaje estándar que incluye recursión, es completo para PTIME y, por lo tanto, considerablemente más complejo de evaluar que la lógica de primer orden. A pesar de ser más complejo que la lógica de primer orden, DATALOG sigue teniendo un procedimiento de evaluación eficiente (de hecho, de tiempo polinomial) por lo que se puede utilizar en la práctica. Sin embargo uno podría agregar mucha expresividad a un lenguaje y sacrificar la complejidad a un nivel que lo vuelva impráctico. Un ejemplo es la lógica de segundo orden que permite expresar consultas sumamente complejas. Considere nuevamente la base de datos de vuelos directos, y suponga que además se cuenta con una tabla **países** de nombres de países por los que un viajero quiere pasar. La lógica de segundo orden permite expresar consultas como la siguiente:

C4: "¿Existe un itinerario de vuelo que pase exactamente una vez por cada país en la tabla países?"

Usar lógica de segundo orden es poco razonable ya que su complejidad es más alta que la de los problemas NP-Completos y por lo tanto será poco usable en la práctica ya que la ejecución de una consulta tardará un tiempo demasiado alto.

En general se pueden utilizar diversos lenguajes de consulta en bases de datos relacionales y siempre existirá este "tira y afloja" entre expresividad y complejidad. El ideal es llegar a un equilibrio entre ambas medidas; encontrar un lenguaje suficientemente expresivo para las aplicaciones que quiero modelar y a la vez suficientemente eficiente y simple de ejecutar para que las consultas se puedan responder en la práctica. La lógica de primer orden es un muy buen ejemplo en el caso de las bases de datos relacionales.

MÁS ALLÁ DE SÓLO CONSULTAR

Hay muchos otros problemas en teoría de bases de datos más allá de sólo consultar. Uno de ellos es el problema de equivalencia de consultas: dadas dos consultas distintas en álgebra relacional (o lógica de primer orden), ¿será el caso que para todas las posibles bases de datos las consultas entregan exactamente el mismo resultado? Alguien podría pensar que esto no puede ocurrir si las consultas son distintas, pero considere la siguiente consulta:

$$C5: \pi_{bebedor}\ (frecuenta \bowtie (\sigma_{cerveza\ =\ \text{“Kriek”}}\ sirve))$$

No es difícil notar que, sea cual sea la base de datos donde la ejecutemos, las consultas **C1'** y **C5** entregarán siempre los mismos resultados. También se puede argumentar que la consulta **C5** se puede

ejecutar más eficientemente que **C1'**. Si un sistema de bases de datos pudiera entonces establecer la equivalencia entre consultas, podría notar que da lo mismo responder **C1'** o **C5,** y si alguien consulta por **C1'**, el sistema podría en vez ejecutar **C5**, optimizando el tiempo de respuesta. Note que establecer la equivalencia entre consultas no parece ser un problema trivial para nada. De hecho ¿cómo chequeamos en general que dos consultas entregan el mismo resultado para todas las posibles bases de datos? La mala noticia es que el problema de equivalencia para el álgebra relacional es *indecidible:* **no existe un algoritmo que, dadas dos consultas del álgebra relacional, determine si éstas son equivalentes.** Una noticia negativa de parte de la teoría para la práctica.

Lo típico que se hace en investigación teórica en computación una vez que se determina que un problema no puede ser resuelto algorítmicamente, es buscar restricciones bajo las cuales el problema se vuelve decidible. Una interesante restricción del álgebra relacional es el fragmento que considera expresiones donde sólo las operaciones de proyección (π), selección (σ) y join ($\bowtie$) son permitidas.

Este fragmento se llama *fragmento conjuntivo* del álgebra relacional. El fragmento conjuntivo contiene la mayor parte de las consultas de la forma SELECT-FROM-WHERE de SQL y, por lo tanto, tiene gran interés práctico. Se puede demostrar [4] que para este fragmento el problema de determinar equivalencia de consultas es NP-Completo. Si bien el problema no tiene una solución eficiente (NP-Completo significa que será difícil de resolver en la práctica), al menos sabemos que el problema es decidible. Otro problema de interés teórico que ha tenido implicaciones prácticas es el de *responder consultas usando vistas* [5]. Suponga que un sistema de bases de datos tiene un conjunto de consultas **V1, V2, V3,...** etc., *precomputadas*, es decir, antes las consultó y el resultado de cada una está almacenado. A este conjunto de consultas se les llama vistas. Cuando llega una nueva consulta, digamos **Q**, al sistema de base de datos, se podría antes de ejecutar **Q** desde cero intentar computarla como una combinación de las vistas **V1, V2, V3,...,** y así ahorrar considerable tiempo. Una solución a este problema permitiría a un sistema de bases datos mantener un caché de consultas más solicitadas, o suficientemente generales, de manera tal que puedan ser utilizadas para responder diversas otras consultas. En [5], los autores plantearon este problema y propusieron un algoritmo para el caso en que las vistas son expresadas en el fragmento conjuntivo del álgebra relacional.

El problema de responder consultas usando vistas se ha usado (tanto su teoría como sus algoritmos) en diversos otros problemas de interés práctico, como por ejemplo, el de integración de datos. En un sistema de integración de datos, ya no es sólo una base de datos la que se quiere consultar sino varias posiblemente distribuidas. El usuario no tiene acceso a cada una de las bases de datos por separado sino más bien tiene acceso a una única interfaz global. Se puede pensar por ejemplo en un sistema de bibliotecas distribuidas en distintas universidades. Cada universidad almacena información de libros de manera independiente y no necesariamente usando los mismos esquemas (tablas, nombres de tablas y atributos). Para simplificar el acceso a todas estas posibles bases de datos, se le presenta al usuario una base de datos virtual, o sea nombres y atributos de tablas pero sin datos. Cada vez que el usuario hace una consulta sobre la base de datos virtual, el sistema traduce estas consultas en diversas consultas sobre las distintas bases de datos de bibliotecas, en donde se computan las respuestas y luego se entrega una visión unificada al usuario. Una forma de modelar y resolver este problema, es pensar que cada posible base de datos distribuida es una vista precomputada de la base de datos global (virtual). El sistema entonces no tiene acceso a los datos de la base de datos global y cada nueva consulta debe ser respondida usando sólo la información de las vistas, que en este caso corresponderían a cada una de las bases de datos de bibliotecas independientes [6, 7].

NUEVOS MODELOS DE DATOS

Hemos hecho una revisión de algunos problemas teóricos importantes en bases de datos relacionales. Obviamente la lista visitada está lejos de ser exhaustiva. Una buena referencia para el lector interesado puede ser [8, 9]. Muchos de estos problemas teóricos que han sido estudiados por décadas, hoy deben ser revisitados ante el surgimiento de nuevos modelos de datos principalmente motivados por el manejo e intercambio de información en la Web.

Uno de estos modelos es XML, que ha demostrado ser una interesante fuente de desafíos teóricos donde los resultados para bases de datos relacionales no siempre se pueden adaptar. Otro modelo muy en boga hoy es el modelo de datos de grafos. En este modelo, los datos están representados como puntos y las relaciones entre ellos como links. Un típico ejemplo son las redes sociales tipo Facebook o Twitter. En estas nuevas bases de datos, los datos atómicos son los identificadores de objetos, sean estos usuarios, empresas, ciudades, etc., y las relaciones entre ellos vienen dadas por las relaciones de

amistad (Facebook), por qué usuario sigue a qué otro (Twitter), etc. La misma Web es un espacio gigante de almacenamiento de datos estructurado como grafo. En estos nuevos modelos de datos, las consultas que se quiere responder son principalmente consultas de navegación, por ejemplo, saber cuántos usuarios se pueden alcanzar desde uno en particular siguiendo las "relaciones de amistad". El problema de complejidad se vuelve aún más desafiante dado el gran volumen de datos que deben ser consultados y nuestra discusión acerca de la relación entre expresividad y complejidad toma un nuevo color. En estas aplicaciones de datos a gran escala no tenemos otra opción si no sacrificar expresividad si queremos obtener respuestas en tiempos razonables. Los mismos problemas de equivalencia, responder consultas usando vistas, o integración de datos toman también un nuevo aire.

Hay mucho por hacer hoy en el estudio teórico de Bases de Datos y como ha sido la tónica en los últimos cuarenta años, es de esperar que esta investigación teórica pueda ser exitosamente traspasada a la práctica. BITS

Referencias

[1] E. F. Codd. A relational model of data for large shared data banks. Communications of the ACM 13 (6): 377, 1970.

[2] A. V. Aho and J. D. Ullman. Universality of data retrieval languages. In POPL 1979, pages 110-117.

[3] L. Libkin. Elements of Finite Model Theory. Springer 2004.

[4] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In STOC 1977, pages 77-90.

[5] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. In PODS 1995, pages 95-104.

[6] J. D. Ullman. Information Integration Using Logical Views. Theor. Comput. Sci. 239(2): 189-210, 2000.

[7] M. Lenzerini. Data Integration: A Theoretical Perspective. In PODS 2002, pages 233-246.

[8] D. Maier. The Theory of Relational Databases. Computer Science Press, 1983.

[9] S. Abiteboul, R. Hull, V. Vianu. Foundations of Databases. Addison-Wesley, 1995.