Semántica y Consistencia de Datos: Una Trayectoria de Investigación

Leopoldo Bertossi*

Resumen

En este artículo describo algunos de los temas de investigación y los problemas en los que he trabajado en los últimos quince años de mi vida académica. Estos se refieren a la amplia área de manejo de datos, y a la semántica y la calidad de datos, en particular. También describo algunas actividades actuales y sus proyecciones futuras.

1. Bases de Datos y Modelos de la Realidad

Para fijar ideas, identificaré "bases de datos'' con "bases de datos relacionales''. El modelo de datos relacional y la idea de sistemas administradores de bases de datos (SABDs) relacionales fueron propuestos por E. Codd a principios de los años 70 [1]. Su propuesta fue revolucionaria y no carente de controversia. Sin embargo, el modelo se impuso, es un referente natural cuando se propone nuevas formas de bases de datos, y se convirtió en un éxito tecnológico y comercial. [1]

En términos simples, Codd proponía un modelo basado en la lógica de predicados, la que ya había sido estudiada por décadas en el área de lógica matemática. El modelo era entonces "lógico'', y el usuario de la base de datos se enfrentaba con un modelo esencialmente lógico, sin tener que entender o preocuparse de la forma en que los datos se estructuraban, almacenaban y accesaban internamente en el computador, facilitando así la interacción con la base de datos.

Desde esta perspectiva, una base de datos pasa a ser un modelo lógico de una realidad externa, y como tal, pasa a ser, como todo modelo, una representación simplificada de esta última. El lenguaje de representación (o descripción) del modelo es el de la teoría de conjuntos, extendida a las relaciones conjuntistas, y la lógica de predicados. Un simple ejemplo de base de datos universitaria se encuentra en la Figura 1. Como las (extensiones de las) relaciones en una base de datos son finitas, ellas se representan comunmente como tablas.

Un usuario puede almacenar información en estas tablas, las puede actualizar, y puede hacer consultas. Estas últimas pueden combinar información de varias tablas. Los lenguajes de la lógica de predicados resultan ser particularmente apropiados para formular consultas: (a) Son simbólicos, y luego, manejables en principios por un computador; (b) Tienen una

*Profesor, Carleton University, School of Computer Science, Ottawa, Canada. Email: bertossi@scs.carleton.ca.

[1] Por sus trabajos en el tema, Codd recibió el Turing Award, el más importante reconocimiento entregado por la comunidad de computación.

Estudiantes

NumEst	*NomEstu*
101	john bell
102	mary stein
104	claire stevens
105	sue cash
107	pat norton

Inscritos

NumEst	*Curso*
104	comp150
101	comp100
101	comp200
105	comp120

Figura 1: Una instancia de base de datos

sintaxis precisa, definida por una gramática; y (c) Tienen una semántica precisa, y ya definida y estudiada en lógica matemática.

En el ejemplo anterior, si queremos obtener los nombres de los estudiantes que están inscritos en el curso *comp100*, podemos entregar al SABD la consulta:

$$\mathcal{Q}(x)\colon\ \exists u(Estudiantes(u,x) \wedge Inscritos(u, comp100)),$$

donde la variable libre x es usada para recibir las respuestas a través de los valores que hacen verdadera la consulta en conjunto con el resto de la base de datos. Esta es una simple "consulta conjuntiva'', i.e. que es una conjunción de (dos) aseveraciones atómicas, con un cuantificador existencial al frente, que está ahí para indicar que debe haber algún número de alumno, pero no nos interesa cuál es.

Notar que la consulta es simbólica y "declarativa'': Le decimos al SABD lo que queremos obtener, pero sin especificar cómo obtenerlo; ésa es tarea interna del SABD, que usa mecanismos de evaluación de consultas basados en un álgebra de relaciones.

Una base de datos relacional, como modelo de una realidad externa, debe ser un buen modelo de ésta. El significado o semántica de (los datos en) la base de datos debe estar en correspondencia con, o capturar, la realidad externa. Por ejemplo, se espera que un número de alumno no esté nunca asociado a más de un nombre de alumno. ¿Cómo especificamos eso? ¿Cómo nos aseguramos de que esta condición sea satisfecha cuando la base de datos está en evolución debido a actualizaciones (nuevos alumnos, cambios de nombre de éstos, etc.)?

Nuevamente la lógica de predicados nos ofrece lenguajes para especificar declarativamente estas condiciones, las llamadas "restricciones de integridad'' (también, restricciones de consistencia o restricciones semánticas). Para el ejemplo recién dado, la restricción puede ser escrita simbólicamente así:

$$\forall x \forall y \forall z(Estudiantes(x,y) \wedge Estudiantes(x,z) \rightarrow y = z), \tag{1}$$

indicando que cuando un número de alumno (el primer argumento del predicado) tiene asociados dos nombres de alumnos (los segundos argumentos), entonces estos últimos deben ser iguales. Este tipo de restricción se llama "dependencia funcional'', ya que solicita que el segundo atributo (*NomEstu*) sea una función del primer atributo (*NumEst*).

Las restricciones de integridad imponen condiciones para que la base de datos (BD), a través de su satisfacción, sea un buen modelo de la realidad. Varias clases de restricciones de integridad pueden ser declaradas junto con la creación de la base de datos en el SABD, y este último cuenta con mecanismos internos que aseguran que la consistencia se mantenga, típicamente a través del rechazo de actualizaciones que puedan violarlas. Otros tipos de

restricciones de integridad pueden ser manejadas (en términos de su satisfacción) por medio de procedimientos internos (triggers) que son creados y almacenados en la BD por el usuario. Estos procedimientos reaccionan automáticamente cuando una violación de la restricción de integridad se va a producir, ejecutando alguna acción preprogramada por el usuario, por ejemplo, una actualización que compense aquella que está produciendo la violación. Otra forma de "mantención de restricciones de integridad'' se da a nivel de "aplicaciones'', es decir, de los programas o transacciones que interactúan con la BD. Estos pueden chequear posibles violaciones y ejecutar acciones que restituyan la consistencia.

Como indiqué al comienzo, las bases de datos relacionales tienen su base y raíz en la lógica matemática. Habiendo hecho mi doctorado en un tema de lógica matemática, no es extraño que me haya sentido atraído por las bases de datos como un área de aplicación de la lógica matemática. De hecho, aunque las bases de datos puedan parecer muy simples, hay interesantes problemas en torno a ellas. Ellos van desde los aspectos filosóficos, pasando por problemas matemáticos y algorítmicos difíciles, hasta llegar, por supuesto, a problemas técnicos de implementación y uso de SABDs.

He trabajado en diversos temas de investigación en manejo de datos. En el resto de este artículo, me referiré sólo a uno de ellos, en el que he trabajado por varios años. Más aún, este tema fue abierto por mí y mis colaboradores con una publicación de 1999 [2]. Hoy en día este es un tema establecido e importante en la comunidad de investigadores en manejo de datos. A continuación describiré el problema, la forma inicial en que lo abordamos, resultados obtenidos en el camino, y aplicaciones a otros problemas de manejo de datos.

2. Enfrentando la Inconsistencia

A pesar de todas las precauciones que uno puede tomar al formular restricciones de integridad y forzar su satisfacción, las bases de datos pueden ser o convertirse en inconsistentes, es decir, pueden no satisfacer las restricciones de integridad deseadas. Esto puede ocurrir en diversas situaciones, entre ellas: (a) Simples bases de datos debido a la interacción con aplicaciones mal diseñadas. (b) Sistemas "legados'' de bases de datos, que han sido heredados de sistemas antiguos y cuyos datos se quiere volver a usar. Posiblemente se ha perdido la semántica original y subentendida de los datos, y se los quiere someter a nuevos requerimientos semánticos. (c) Integración de diversas fuentes de datos, posiblemente localmente consistentes, en una nueva base de datos sometida a nuevas restricciones de integridad globales. Etc.

En una serie de artículos consideramos el siguiente problema.[2] Si ya tenemos una base de datos inconsistente, muy probablemente la mayor parte de los datos en ella es "consistente''. Dado que la noción de consistencia es holísticamente aplicada a la base de datos completa, es necesario caracterizar en términos precisos cuál es la información consistente (una noción local) dentro de una base de datos inconsistente (la noción global aludida más arriba)? Teniendo una respuesta precisa a esa pregunta, la siguiente pregunta es: ¿Cómo podemos extraer esa información consistente? Y más aún, al hacer una consulta a la base de datos inconsistente: ¿Cuáles son las respuestas consistentes y cómo las obtenemos?

En [2] propusimos respuestas a esas preguntas, y algunos algoritmos para calcular respuestas consistentes; al menos para algunas clases relevantes de consultas y restricciones de

[2]Muchos de mis trabajos tienen coautores, en particular, ex-alumnos. Por eso me expreso en plural en referencia a trabajos de investigación.

integridad. Este trabajo ha sido enormemente citado,[3] y ha sido seminal, dando lugar al área (y problema) de "consistent query answering'' (CQA). En ella han trabajado muchos investigadores de primer nivel internacional. Y el tema sigue fuertemente vigente, y muchos problemas siguen aún abiertos.

La intuición de la definición dada en [2] es la siguiente: Los datos consistentes en una base de datos inconsistente D (es decir, que viola el conjunto RI de restricciones de integridad) son aquellos que son invariantes bajo todas las formas "razonables'' de restituir la consistencia de D. Más precisamente, aquellos datos que aparecen en todas las versiones alternativas D' de D, llamadas "las reparaciones'' de D, que: (a) Son consistentes, es decir, satisfacen las restricciones de integridad en RI; y (b) difieren minimalmente de D en términos de inclusión de conjuntos de tuplas que son insertadas o eliminadas de D (para producir D'). En otras palabras, los datos consistentes persisten a través de todas las versiones minimalmente reparadas de D.

Ejemplo: Consideremos la instancia de base datos D de abajo, y la dependencia funcional que requiere que el *Salario* dependa funcionalmente del *Nombre*, denotada por DF: $Nombre \rightarrow Salario$. La instancia dada es inconsistente: el empleado *page* tiene dos salarios.

$Empleado$	$Nombre$	$Salario$
	$page$	5M
	$page$	8M
	$smith$	3M
	$stowe$	7M

Hay sólo dos reparaciones minimales si sólo eliminaciones/inserciones de tuplas son admitidas: D_1, resp. D_2 mostradas abajo.

$Empleado$	$Nombre$	$Salario$
	$page$	5M
	$smith$	3M
	$stowe$	7M

$Empleado$	$Nombre$	$Salario$
	$page$	8M
	$smith$	3M
	$stowe$	7M

Aquí la tupla $(stowe, 7M)$ persiste ***en todas*** las reparaciones: es información consistente. Sin embargo, la tupla $(page, 8M)$ no persiste. De hecho, ella participa en la violación de DF. □

De acuerdo con lo anterior, una ***respuesta consistente*** a una consulta $\mathcal{Q}$ en una base de datos D es una respuesta que puede ser obtenida, como respuesta usual, a la consulta $\mathcal{Q}$ desde toda posible reparación D' de D con respecto a RI.

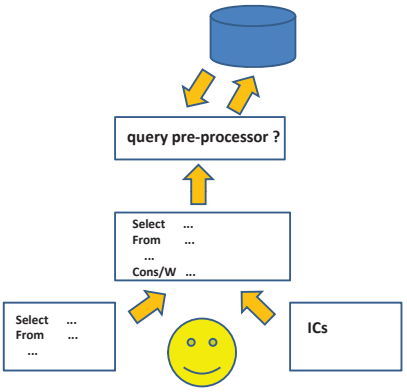
En el ejemplo anterior, para la consulta $\mathcal{Q}_1 : Empleado(x, y)$, que pregunta por todas las tuplas de la tabla, las respuestas consistentes son: $\langle smith, 3M\rangle, \langle stowe, 7M\rangle$. Ahora, para la consulta $\mathcal{Q}_2 : \exists y Empleado(x, y)$, que pregunta sólo por los (nombres de los) empleados, las respuestas consistentes son: $\langle page\rangle, \langle smith\rangle, (stowe\rangle$.

Podemos ver de inmediato de la segunda consulta que CQA es diferente de limpieza de datos. Probablemente la limpieza de datos eliminaría ambas tuplas en conflicto. En nuestro caso, persiste la información de que existe un empleado *page*, aunque haya un conflicto de salarios. Sin embargo, CQA es relevante para el tema de calidad de datos, la que es una necesidad creciente en inteligencia de negocios. CQA también provee conceptos y mecanismos para limpieza de datos.

En nuestra visión, los próximos SABDs deberían proveer mecanismos más flexibles, poderosos y amistosos con el usuario, en particular en lo que se refiere al manejo de restricciones

[3] A la fecha, más de 500 citaciones según Google Scholar.

semánticas. Ellos deberían permitir el solicitar (y entregar) respuestas consistentes a las consultas, en tiempo de consulta. La siguiente figura muestra esta idea.



¿Por qué no un SQL extendido,[4] con cláusulas que permitan indicar las restricciones de integridad de interés?

Las restricciones de integridad (ICs en la figura) pasan a ser un nuevo parámetro de la consulta.

SELECT	Nombre, Salario
FROM	Empleado
CONS/W	DF: Nombre -> Salario;

(Aquí la DF no es mantenida por el SABD.)

Nótese el cambio de paradigma: Las restricciones de integridad pasan a ser restricciones sobre las respuestas a las consultas, y no sobre los estados de la BD!

La complejidad computacional de obtener respuestas consistentes a consultas, usualmente conjuntivas, depende de la clase de consultas y la clase de restricciones de integridad. En [2] propusimos el primer algoritmo para calcular respuestas consistentes sin tener que computar, materializar y consultar todas las reparaciones, como podría sugerir la definición de respuesta consistente. (El número de reparaciones puede ser exponencial en el tamaño de la base de datos original [3].)

El algoritmo fue implementado [4], y se basa en *reescritura de consultas*: la consulta original $\mathcal{Q}$ es reescrita como una nueva consulta $\mathcal{Q}'$, y esta última, al ser hecha a la base de datos (inconsistente), obtiene como respuestas usuales, las respuestas consistentes para $\mathcal{Q}$.

Por ejemplo, las respuestas consistente a la consulta $\mathcal{Q}(x, y)\colon\ Empleado(x, y)$, i.e. sobre todas las tuplas consistentes en la tabla, se obtienen poniendo a la misma BD la consulta reescrita:

$$\mathcal{Q}'(x, y)\colon\ Empleado(x, y) \wedge \neg\exists z(Empleado(x, z) \wedge z \neq y).$$

Esta última es una consulta en lógica de predicados de primer orden que puede ser reformulada en SQL, y respondida en tiempo polinomial (en el tamaño de la BD), es decir, eficientemente.

Reescrituras como la de arriba hacen que CQA sea eficiente, o *tratable*, para ciertas clases de consultas conjuntivas y restricciones de integridad. Sin embargo, éste no es un método de aplicación universal. Algunos casos de CQA son intrínsicamente más complejos, i.e. intratables [3]. Resultados de este tipo requieren de una demostración matemática, como es de esperar.

Partiendo con nuestro trabajo sobre la complejidad de responder consistentemente consultas con agregación [3], donde los primeros resultados de intratabilidad fueron obtenidos, diversos autores comenzaron a identificar casos tratables e intratables. Para los primeros se desarrolló algoritmos eficientes (detalles pueden ser consultados en mi monografía sobre el área [5] o mi survey [6]).

[4]SQL es el lenguaje estándar de consulta y de interacción con bases de datos relacionales.

3. Programas en Lógica y Reparación

Aquellas consultas que hacen al problema de CQA intrísecamente más complejo no permiten una reescritura en lógica de predicados de primer orden (los que las haría tratables). Por consiguiente, era natural preguntarse por otros lenguajes de consulta más expresivos que sí permiten la reescritura.

En ese espíritu iniciamos una nueva línea de investigación, partiendo por [7]. El lenguaje propuesto fue el de los "programas en lógica disyuntivos con semántica de modelos estables". Una serie de trabajos fueron publicados por nosotros [8, 9, 10, 11], e independientemente, por otros autores. El trabajo que describe en mayor detalle esta manera de abordar el problema de CQA es [11], donde, además, se informa sobre una implementación computacional del método. Esta es la manera más general de abordar el problema de CQA.

Ejemplo: Consideremos una instancia de base de datos alternativa a la que dimos al comienzo:

Estudiantes

NumEst	*NomEstu*
101	john bell
101	joe logan
104	claire stevens
107	pat norton

Inscritos

NumEst	*Course*
104	comp150
101	comp100
105	comp200
101	comp200

Figura 2: Otra instancia

Además de la DF en (1), tenemos la "restricción de integridad referencial'' siguiente:

$$\forall x \forall y (Inscritos(x, y) \longrightarrow \exists z Estudiantes(x, z)), \tag{2}$$

la que requiere que todo (número de) alumno inscrito en un curso aparezca en la tabla oficial de estudiantes (con algún nombre asociado). La instancia en la Figura 1 es consistente, pero la de la Figura 2, no, ya que viola ambas restricciones.

El programa en lógica propuesto representa de manera compacta todas las posibles reparaciones de la instancia en la Figura 2 con respecto a las restricciones en (1) y (2). Los *hechos* (facts) del programa son las tuplas en la instancia original. Además, el programa contiene las siguientes reglas. Ellas usan constantes de *anotación* $\mathbf{f}, \mathbf{t}^\star, ...$

1. Una regla para forzar (1):

$$Estudiantes\_(x, y, \mathbf{f}) \vee Estudiantes\_(x, z, \mathbf{f}) \leftarrow Estudiantes\_(x, y, \mathbf{t}^\star), Estudiantes\_(x, z, \mathbf{t}^\star), \\ y \neq z,\ x \neq null,\ y \neq null,\ z \neq null.$$

El lado derecho de la regla verifica si hay dos tuplas (que eran parte de la instancia original o que han sido insertadas en el proceso de reparación, indicado con $\mathbf{t}^\star$) que juntas violan (1). En ese caso, como está indicado en el lado izquierdo de la regla, se resuelve la inconsistencia eliminando (indicado con $\mathbf{f}$) una de las dos tuplas. Notar que se puede producir una violación de (1) en una base de datos con valor nulo ($null$) sólo si x, y, z no son iguales a $null$.

2. Una regla para forzar (2):

$Aux(x) \leftarrow Estudiantes\_(x, z, \mathbf{t}^\star),\ not\ Estudiantes\_(x, z, \mathbf{f}),\ x \neq null,\ z \neq null.$
$Inscritos\_(x, y, \mathbf{f}) \vee Estudiantes\_(x, null, \mathbf{t}) \leftarrow Inscritos\_(x, y, \mathbf{t}),\ not\ Aux(x), x \neq null.$

La primera regla alimenta un predicado auxiliar, almacenando en él todos los números de estudiante de la tabla *Estudiantes* que no son eliminados en el proceso de reparación. La segunda regla verifica a través de su lado derecho si para cada alumno inscrito (indicado con $\mathbf{t}$), su número aparece en la tabla *Estudiantes*. Si no es el caso, hay una violación, y se resuelve la inconsistencia según lo indicado al lado derecho: se fuerza la remoción de la tupla de *Inscritos* o la inserción de una tupla en *Estudiantes* con una valor nulo para el nombre (indicado con $\mathbf{t}$).

3. La semántica de las anotaciones se especifica con reglas:

$$\left.\begin{array}{l} Estudiantes\_(x, y, \mathbf{t}^\star) \leftarrow Estudiantes(x, y). \\ Estudiantes\_(x, y, \mathbf{t}^\star) \leftarrow Estudiantes\_(x, y, \mathbf{t}). \\ Estudiantes\_(x, y, \mathbf{t}^{\star\star}) \leftarrow Estudiantes\_(x, y, \mathbf{t}^\star),\ not\ Estudiantes\_(x, y, \mathbf{f}). \\ \leftarrow Estudiantes\_(x, y, \mathbf{t}), Estudiantes\_(x, y, \mathbf{f}). \end{array}\right\} \text{Similar para } \textit{Inscritos}\_$$

Las dos primeras reglas consiguen que la tuplas anotadas con $\mathbf{t}^\star$ sean aquellas en la instancia original o que han sido insertadas en el proceso de reparación. La tercera regla recolecta todas las tuplas que son verdaderas en la instancia reparación final (i.e. están en la reparación). La última regla es una *restricción de programa*, y descarta aquellas reparaciones (modelos del programa) donde una tupla es a la vez insertada e eliminada.

Se puede dar una demostración matemática [9, 10] de que hay una correspondencia uno-a-uno entre los modelos estables del programa en lógica y las reparaciones de la instancia original: Una reparación se forma a partir de un modelo del programa descartando todas las tuplas que no están anotadas con $\mathbf{t}^{\star\star}$.

Las respuestas consistentes a una consulta se pueden obtener razonando directamente con -o sobre- el programa reparador, usando la *semántica escéptica* para el programa, es decir, aceptando como verdadero con respecto al programa todo lo que es verdadero en todos los modelos estables del programa. Por ejemplo, si queremos responder consistentemente las consultas $\mathcal{Q}_1(x, y) : Estudiantes(x, y)$ y $\mathcal{Q}_2(x) : \exists y Estudiantes(x, y)$, basta, en cada caso, agregar al programa las regla $Ans_{\mathcal{Q}_1}(x, y) \leftarrow Estudiantes\_(x, y, \mathbf{t}^{\star\star})$ y, respectivamente, $Ans_{\mathcal{Q}_2}(x) \leftarrow Estudiantes\_(x, y, \mathbf{t}^{\star\star})$. La evaluación del programa extendido con la consulta entrega las respuestas consistentes a estas consultas. □

En [11] se presenta diversas técnicas para optimizar tanto el programa de reparación como su evaluación a la luz de la consulta que se quiere responder. La idea es concentrarse en la parte del programa que es *relevante* para la consulta de interés.

Ultimamente ha habido interés en la comunidad de manejo de datos en el problema de computar una sola reparación que sea buena en algún sentido predeterminado. Esto acerca el tema al de limpieza de datos, donde la idea es limpiar una base de datos de distintas formas de deficiencias y errores, llegando a producir una única nueva instancia limpia. Desde este punto de vista, una reparación con respecto a restricciones de integridad se puede ver como un proceso de limpieza de errores de naturaleza semántica.

4. Inconsistencia e Integración

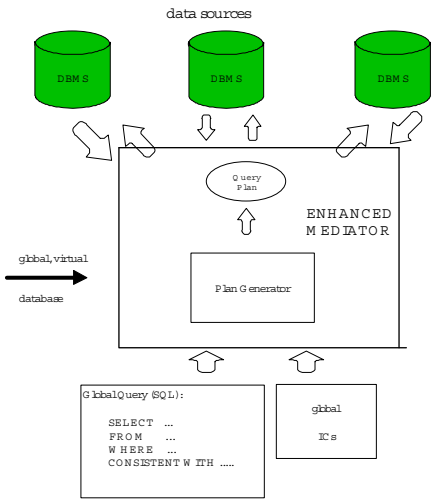
Los programas de reparación presentados anteriormente han permitido atacar otros problemas importantes en manejo de datos. Uno de ellos es el de integración de bases (o fuentes) de

datos.

Cuando existen diversas fuentes de datos, éstas se pueden integrar en un solo y nuevo repositorio físico de datos, con el cual se sigue trabajando, dejando posiblemente de lado las fuentes originales. El amplio tema de *data warehousing* tiene que ver con esto. De hecho, hay problemas semánticos que resolver en ese terreno, y hemos hecho investigación en reparaciones de "data warehouses'' y, más generalmente, de *bases de datos multidimensionales* [12, 13, 14]. Aquí nos concentraremos en *integración virtual* de datos.

Conceptualmente la idea es fácil de entender, aunque no necesariamente fácil de llevar a la práctica: En lugar de producir un nuevo repositorio físico global, las fuentes de datos se dejan como están, y se crea una sistema de software llamado *mediador*. Este integra virtualmente las fuentes existentes, y ofrece a los usuarios una interfaz de consulta basada en un esquema de datos global y unificado. En consecuencia, un usuario hace una consulta en el lenguaje unificado, y el mediador, internamente, se encarga de enviar consultas apropiadas y ad hoc a las fuentes (en sus propios lenguajes de consulta), recibe las respuestas de las fuentes, y las combina para entregar la respuesta final al usuario.

Este paradigma está ilustrado en la figura adyacente. Ahí se muestra que el mediador contiene un generador de planes de consulta, el cual, para trabajar, necesita información sobre las fuentes, en particular, sobre la clase de datos que contienen. Esto se hace almacenando en el mediador las "correspondencias'' lógicas (o mappings) entre los esquemas de las fuentes y el esquema global único.



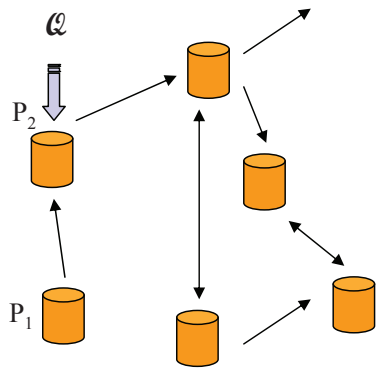
La *semántica del sistema* se expresa a través de la colección de instancia globales subentendidas, o legales; las que sería aceptable materializar (si se quisiera) sobre la base de las fuentes y correspondencias del sistema.

El principal problema en la práctica (y también en la investigación teórica al respecto) tiene que ver con la forma de imponer *restricciones de integridad globales* sobre el sistema, es decir, expresadas en el lenguaje global del mediador. Formularlas no es un problema, pero sí lo es el imponerlas o forzar su satisfacción, ya que las fuentes son autónomas y, en general, no aceptan actualizaciones a través del mediador, sino sólo consultas.

En nuestra visión, la única manera de imponer restricciones globales es en tiempo de consulta, cuando se da respuesta a la consulta. Sobre esta base, la solución que propusimos [15, 16, 17] captura precisamente esta idea, en dos pasos. Más precisamente, propusimos un primer programa en lógica que especifica las instancias globales legales. (Esta es la solución más general que se ha propuesto para este problema en particular [15].) Estas instancias globales probablemente no van a satisfacer las restricciones de integridad globales. Consequentemente, en un segundo paso, acoplamos el programa que especifica las instancias globales con un programa de reparación como el que presentamos en la sección anterior. La combinación de los dos programas entrega instancias globales que sí son consistentes. Todo esto puede ser hecho en tiempo de consulta.

La figura del sistema indica, en su parte inferior, que la información entregada por el usuario al hacer una consulta podría ser la consulta misma, pero acompañada de las restricciones de integridad globales que las respuestas deben satisfacer.

Otra forma de integración de bases de datos es a través de *sistemas de pares de manejo de datos* (peer data management systems). En este caso, hay una colección de pares (o agentes) poseedores de fuentes de datos, conectados algunos con otros, a través de correspondencias lógicas (mappings) entre los correspondientes esquemas de datos locales (que pueden ser todos distintos) de los distintos pares. La figura ilustra esta situación.



Los pares intercambian datos en tiempo de consulta, de acuerdo con las correspondencias con sus pares vecinos inmediatos. Esto puede dar lugar a un proceso transitivo, iterativo, que no necesariamente cuenta con un control global. Las consultas son hechas por un usuario a uno de los pares, quien envía a su vez consultas a sus vecinos. De hecho, los pares importan datos de sus pares vecinos a través de consultas, las que son tratadas como si fueran consultas de usuarios externos. Cada par importa datos de sus vecinos e, indirectamente, de sus conexiones transitivas.

Los problemas que emergen son, entre otros: (a) proponer un modelo lógico-matemático para este tipo de sistema de pares, (b) proveer a través de él una *semántica* a tales sistemas, y (c) proponer mecanismos computacionales que realicen el intercambio de datos.

Nuestra visión y forma de formular el problema es que *el traspaso de datos entre dos vecinos está gatillado y dirigido por la inconsistencia* [18]. Este proceso puede ser modulado por relaciones de confianza (trust) entre pares vecinos.

Por ejemplo, en la figura anterior se está haciendo una consulta $\mathcal{Q}$ al par $\mathbf{P}_2$, quien puede usar sus propios datos, y también, dado que hay una correspondencia desde el par $\mathbf{P}_1$, los datos de este último, importándolos de acuerdo con la consulta. A su vez, $\mathbf{P}_1$ puede recibir datos de otros pares (que no los hay en este caso), etc.

Más específicamente, supongamos que los pares $\mathbf{P}_1, \mathbf{P}_2$ tienen los siguientes esquemas relacionales de datos: $\mathcal{S}_1 = \{R_1(A, B)\}$ y $\mathcal{S}_2 = \{R_2(A)\}$, respectivamente. Además, se tiene la correspondencia lógica $\forall xy(R_1(x, y) \rightarrow R_2(x))$ que pertenece a $\mathbf{P}_2$ (hacia donde apunta la flecha). Supongamos, además, que los pares tienen las siguientes instancias de datos: $I_1 = \{R_1(a, b)\}$ y $I_2 = \{R_2(c)\}$, respectivamente.

De inmediato vemos que la correspondencia no es satisfecha por las instancias dadas, es decir, hay una inconsistencia. Y es esta última la que hará que se traspase datos de $\mathbf{P}_1$ a $\mathbf{P}_2$, para restituir la consistencia. Por ejemplo, si la consulta a $\mathbf{P}_2$ es $\mathcal{Q}(x) : \ R_2(x)$, es decir, por las tuplas en la relación R_2, las respuestas serán $\langle c \rangle$, directamente de la instancia local, más $\langle a \rangle$, que proviene de I_1 a través de la correspondencia.

También puede haber restricciones de integridad locales para cada par, las cuales también deben ser tomadas en cuenta. Y como mencionamos antes, también relaciones de confianza (un par estará dispuesto a modificar sus datos locales al importar datos de otro par en la medida que confíe en él).

En [19] propusimos y desarrollamos este modelo, dando una semántica precisa a las respuestas a consultas, e introdujimos programas en lógica locales a los pares que implementan el intercambio de datos guiado por la inconsistencia.

5. Conclusiones y Temas en Desarrollo

Mi investigación en el tema de manejo de inconsistencias en bases de datos ha sido un largo y fructífero recorrido, en términos de los resultados obtenidos, los trabajos publicados, y el reconocimiento recibido de la comunidad. Y también ha sido intelectualmente desafiante. Los problemas van desde problemas conceptuales (como elegir y formular conceptos y modelos), pasando por problemas lógicos (como la posibilidad de reescribir consultas en lenguage lógicos de limitado poder expresivo), por problemas matemáticos y algorítmicos (como descubrir y demostrar que un problema de CQA es (in)tratable), hasta el proponer posibles implementaciones computacionales. Y quedan muchos problemas abiertos en el área de CQA, y hay muchos investigadores en el mundo atacándolos actualmente.

Yo sigo trabajando en el tema. Sin embargo, en los últimos años he hecho investigación y me he interesado en otros problemas, no distantes de CQA. En el futuro cercano me concentraré más en ellos. A continuación me refiero brevemente a algunos de ellos.

1. **Limpieza de datos y resolución de duplicados:** Una base de datos puede contener diferentes representaciones de la misma entidad externa. Por ejemplo, dos tuplas, $\langle juan\ perez, 56\ 82345567\rangle$ y $\langle juan\ peres, 56\ 9\ 82345567\rangle$, que probablemente se refieren a la misma persona. Se esperaría que hubiera sólo una, pero la base de datos las tratará como representaciones de dos personas distintas. Más generalmente, el problema es identificar duplicados en una base de datos y resolverlos (en nuestro caso, probablemente dejando sólo la tupla $\langle juan\ perez, 56982345567\rangle$).

   Este es un problema clásico y difícil de limpieza de datos, para el cual existen muchas soluciones ad hoc, verticales, rígidas, y dependientes del dominio de aplicación. Nuestro interés está en destilar, modelar e investigar "la lógica'' de estos procesos, proponer soluciones generales, flexibles, declarativas, y con una semántica clara. Diversos trabajos nuestros dan cuenta de estos esfuerzos [20, 21, 22].

2. **Contextos y calidad de datos:** *La calidad de los datos en una base de datos depende del contexto*. Esta simple y aceptada observación no ha recibido el tratamiento serio que merece. Tomarla en serio significa el proponer y modelar contextos para evaluación de datos; relacionar lógicamente un contexto con una base de datos, para la evaluación de la última; y obtener información limpia a través de la interacción lógica y operacional del contexto y la base de datos. Además, para que un contexto sea adecuado para esta tarea, es necesario que capture elementos muy típicamente contextuales, como las nociones de *punto de vista* y *dimensión*. En trabajos recientes hemos atacado estos problemas [23, 24].

Referencias

[1] Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM*, 1970, 13(6):377-387.

[2] Arenas, M., Bertossi, L. y Chomicki, J. Consistent Query Answers in Inconsistent Databases. In Proc. ACM PODS'99, ACM Press, 1999, pp. 68-79.

[3] Arenas, M., Bertossi, L., Chomicki,J., He, X., Raghavan, V. y Spinrad, J. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science,* Volume 296, Issue 3, 2003, pp. 405-434.

[4] Celle, A. y Bertossi, L. Querying Inconsistent Databases: Algorithms and Implementation. In 'Computational Logic - CL 2000', J. Lloyd et al. (eds.). Springer LNAI 1861, 2000, pp. 942-956.

[5] Bertossi, L. *Database Repairing and Consistent Query Answering*. Synthesis Lectures in Data Management, Morgan & Claypool Publishers, 2011.

[6] Bertossi, L. Consistent Query Answering in Databases. *ACM Sigmod Record*, June 2006, 35(2):68-76.

[7] Arenas, M., Bertossi, L., Chomicki, J. Answer Sets for Consistent Query Answers. *Theory and Practice of Logic Programming*, 3, 4&5, 2003, pp. 393-424.

[8] Barcelo, P. y Bertossi, L. Logic Programs for Querying Inconsistent Databases. Proc. Practical Aspects of Declarative Languages (PADL'03). V. Dahl y Ph. Wadler (eds.), Springer LNCS 2562, 2003, pp. 208-222.

[9] Barcelo, P., Bertossi, L. y Bravo, L. Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. Chapter in 'Semantics of Databases', Springer LNCS 2582, 2003, pp. 1-27.

[10] Bravo, L. y Bertossi, L. Semantically Correct Query Answers in the Presence of Null Values. Proc. EDBT WS on Inconsistency and Incompleteness in Databases (IIDB 06), J. Chomicki y J. Wijsen (eds.), Springer LNCS 4254, 2006, pp. 336-357.

[11] Caniupan, M. y Bertossi, L. The Consistency Extractor System: Answer Set Programs for Consistent Query Answering in Databases. *Data & Knowledge Engineering*, 2010, 69(6):545-572.

[12] Bertossi, L., Bravo, L. y Caniupan, M. Consistent Query Answering in Data Warehouses. Proc. The Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2009). CEUR-WS, Vol-450, 12 pp.

[13] Ariyan, S. y Bertossi, L. Structural Repairs of Multidimensional Databases. Proc. Alberto Mendelzon International WS of Foundations of Data Management (AMW 2011). CEUR Workshop Proceedings, Vol. 749, 2011.

[14] Yaghmaie, M., Bertossi, L. y Ariyan, S. Repair-Oriented Relational Schemas for Multidimensional Databases. Proc. EDBT 2012.

[15] Bravo, L., Bertossi, L. Logic Programs for Consistently Querying Data Integration Systems. Proc. International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann, pp. 10-15.

[16] Bertossi, L. y Bravo, L. Consistent Query Answers in Virtual Data Integration Systems. In 'Inconsistency Tolerance', Springer LNCS 3300, 2004, pp. 42-83.

[17] Bravo, L. y Bertossi, L. Disjunctive Deductive Databases for Computing Certain and Consistent Answers to Queries from Mediated Data Integration Systems. *Journal of Applied Logic*, 2005, 3(2):329-367.

[18] Bertossi, L. y Bravo, L. Query Answering in Peer-to-Peer Data Exchange Systems. Proc. International (EDBT) Workshop on Peer-to-Peer Computing & DataBases (P2P&DB 2004). In 'Current Trends in Database Technology'. Springer LNCS 3268, 2004, pp. 478-485.

[19] Bertossi, L. y Bravo, L. The Semantics of Consistency and Trust in Peer Data Exchange Systems. Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 07), 2007, Springer LNCS 4790, pp. 107-122.

[20] Bertossi, L., Kolahi, S. y Lakshmanan, L. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory of Computer Systems* (Springer journal), DOI: 10.1007/s00224-012-9402-7, 2012.

[21] Gardezi, J., Bertossi, L. y Kiringa, I. Matching Dependencies: Semantics and Query Answering. *Frontiers of Computer Science* (Springer journal), 2012, 6(3):278-292.

[22] Bahmani, Z., Bertossi, L., Kolahi, S. y Lakshmanan, L. Declarative Entity Resolution via Matching Dependencies and Answer Set Programs. Proc. KR 2012, AAAI Press, pp. 380-390.

[23] Bertossi, L., Rizzolo, F. y Lei, J. Data Quality is Context Dependent. Proc. WS on Enabling Real-Time Business Intelligence (BIRTE 2010). Collocated with VLDB 2010. Springer LNBIP 48, 2011, pp. 52-67.

[24] Malaki, A., Bertossi, L. y Rizzolo, F. Multidimensional Contexts for Data Quality Assessment. Proc. AMW 2012. CEUR Workshop Proceedings 866.