MODELO BASE DE DATOS MULTIGRANULAR

Diego Gatica Romero
Departamento de Ingeniería Informática y Ciencias de la Computación
Universidad de Concepción

Profesor Guía: Andrea Rodríguez

28 de agosto de 2018

Resumen

La información puede encontrarse a distintos niveles de granularidad, y aunque actualmente hay diversos modelos de bases de datos enfocados a instancias particulares, ninguno de estos aborda la granularidad de los datos. En esta memoria de título se diseñó un modelo que explota la multigranularidad de los datos y los operadores básicos necesarios para la realización de consultas sobre este modelo. Los resultados mostraron la eficiencia temporal de los operadores implementados sobre el modelo propuestos aunque en término espacial no demostró un ahorro con relación a la instancia inicial de prueba.

1. Introducción

Con la masificación de los sistemas de información se ha incrementado en gran cantidad los datos disponibles a procesar en los últimos años, lo cual ha generado la necesidad de desarrollar maneras eficientes para tanto almacenar y procesar estos datos. Como respuesta a tal necesidad se han desarrollado diversos modelos de bases de datos, cada uno orientado a satisfacer de manera eficientes diversas consultas enfocada en ámbitos específicos, tales modelos varían desde lo más tradicional como el modelo relacional hasta esquema estrella utilizado en *data warehousing*.

Actualmente el tratar con información referenciada a una instancia de tiempo o espacio físico específico es bastante frecuente, por lo que se ha vuelto indispensable el desarrollar una forma eficiente, tanto en términos de espacio utilizado como en velocidad de procesamiento de la información, para responder a consultas sobre datos que posean las características mencionadas anteriormente. Un modelo propuesto para abordar la representación de datos con referencia espacial y temporal a distintos niveles de granularidad es el modelo de datos multigranular.

Un modelo de datos multigranular es uno en el cual sus datos se almacenan en distintos niveles de granularidad [7], es decir, que los valores que puede tomar un atributo se organizan en una estructura de orden parcial que define el detalle de representación. Lo anterior presenta grandes desafíos al momento de establecer restricciones y formas de relacionar datos a distintos niveles de granularidad.

Considere como ejemplo ilustrativo una tabla que muestra las personas nacidas por mes en cada provincia de Chile y otra que contiene las personas nacidas por mes en cada región.

Provincia	Mes 2017	Personas nacidas
Arauco	Agosto 2017	A_1
Biobío	Agosto 2017	A_2
Concepción	Agosto 2017	A_3
Ñuble	Agosto 2017	A_4

Region	Mes 2017	Personas nacidas
O'Higgins	Agosto 2017	B_1
Maule	Agosto 2017	B_2
Biobío	Agosto 2017	B_3
Araucanía	Agosto 2017	B_4

De este caso resulta interesante el poder realizar consultas como el obtener la provincia que posea mayor cantidad de personas nacidas en la región de O'Higgins, o más allá, la región con mayor cantidad de personas nacidas (asumiendo que se posee la información a nivel de provincia).

Aunque existen diversos trabajos que han abordado el tema de la granularidad, el enfoque propuesto se diferencia de los existentes principalmente en que soporta el uso de reglas de *join*, por ejemplo la regla de que una región esté formada por comunas que no se sobreponen entre sí, y trata la información tanto espaciales como temporales como referencias a datos y niveles de granularidad, sin recurrir al manejo de atributos espaciales o temporales.

1.1. Objetivo general

Dado este contexto, el objetivo general de esta memoria de título es implementar un modelo de base de datos y los operadores necesarios para realizar diversos tipos de consultas sobre una base de datos multigranular de manera eficiente tanto espacial como temporal.

1.2. Objetivo específicos

- Revisar el estado del arte en el uso de un modelo multigranular en bases de datos.
- Implementar el modelo de base de datos a utilizar, como una extensión al modelo relacional
- Diseñar los algoritmos para los operadores básicos definidos sobre el modelo de datos.
- Desarrollar un generador de datos de simulación para el modelo implementado.
- Implementar los operadores y el diseño definidos previamente y evaluar el rendimiento tanto en expresividad comparado con el lenguaje SQL como en términos de tiempo de respuesta y capacidad de escalabilidad de los operadores frente a diversas instancias de bases de datos.

2. Discusión bibliográfica

Antes que todo se introducirá brevemente las nociones básicas sobre el modelo de datos multigranular sobre el cual se basará este trabajo. El modelo de datos multigranular tiene como base la noción de granularidad, la cual aparece en diversas áreas de las ciencias de la computación. Esta noción tiene como base el hecho de que los atributos con referencia espacio/temporal en general se presentan o describen a distintos niveles de granularidad según sea necesario. Lo anterior ha sido previamente trabajado en [2] y [3] con distintos enfoque y variando la forma en que describen un gránulo de carácter espacio/temporal.

Los componentes base del modelo son las granularidades y los gránulos que las forman. Informalmente una granularidad es una forma de dividir un dominio en gránulos que lo componen. Las relaciones entre las granularidades representan un orden de subsunción entre ellas formando una topología que representa una jerarquía sobre la información a tratar.

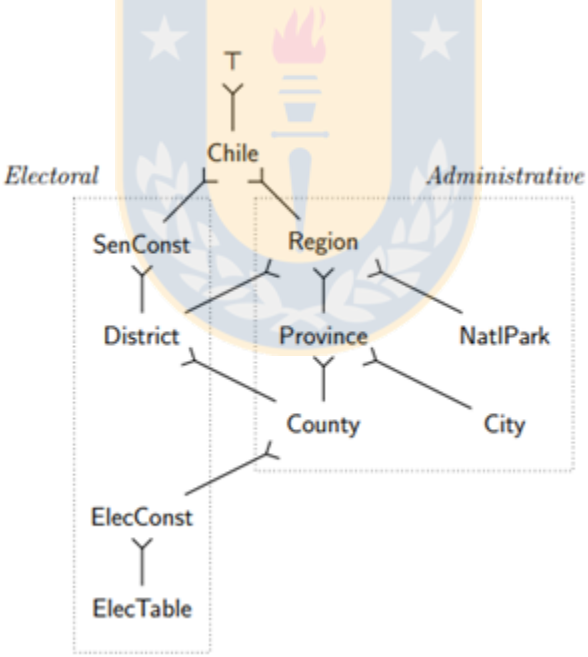

Figura 1: Jerarquía elecciones Chile

En la figura 1 se ilustra una estructura de granularidad asociadas a subdivisiones electorales y administrativas en chile. En este ejemplo una comuna estará formada por un subconjunto de elementos perteneciente a la Circuns-

cripción electoral, siguiendo las definiciones anteriormente mencionadas se puede afirmar que los gránulos que componen a Comuna estarán formados por subconjuntos de gránulos pertenecientes a Circunscripción electoral.

El símbolo 'T' en la figura 1 define una granularidad global, que funciona como límite superior en el conjunto de granularidades, y el símbolo '$\perp$' representa un límite inferior en el conjunto, es decir la granularidad más baja.

Se puede apreciar de la figura 1 que la noción principal para trabajar con gránulos en el dominio espacial es la de división del espacio [4], dado que el trabajar con gránulos de carácter espacial se trata principalmente de un mapeo de información instanciada a un espacio específico.

A nivel de gránulos es necesario el poseer una forma de reducirlos a una misma granularidad para poder manejar información que se encuentre a distintos niveles de granularidad, para ello se propone el concepto de *Coarsen* el cual dado un gránulo lo transforma a uno que se encuentre en una granularidad mayor. Este concepto es fundamental, dado que la idea principal tras el uso de bases de datos multigranulares es poseer la información a distintos niveles de detalle, siendo necesario, para poder trabajar con ellos, un mecanismo capaz de llevar dos gránulos a una granularidad en común.

En [1] se presenta una formalización de un modelo de datos multigranular y la definición de restricciones de integridad como extensión a dependencias funcionales con relaciones de orden, capaz de soportar diversas reglas o restricciones sobre la información. Un ejemplo de regla sería:

$$= \bigsqcup^{\perp} Region_R \mid \mathrm{I} \leq \mathrm{R} \leq \mathrm{XVI}$$

Esta regla nos señala, a través del símbolo $\bigsqcup$, que Chile está formado por el *join* de las regiones desde la I hasta la XV, mientras que el simbolo $\perp$ señala que el *join* es *disjoint*, es decir, no se sobreponen entre si.

Las reglas básicas como la señalada anteriormente incluyen dos conceptos primordiales para el modelo: *subsumption* que hace referencia a cómo los gránulos se subsumen, lo que es fundamental al momento de obtener información a niveles superiores de granularidad y el concepto de *disjointness* que define la no superposición total o parcial entre gránulos, lo cual es importante al momento de realizar operaciones de agregación sobre gránulos que forman una regla de tipo (1).

Se puede pensar en otros modelos similares para responder consultas sobre datos a distinto nivel de detalle, como por ejemplo el usar *data warehouse* [5] para almacenar y responder consultas sobre datos a distintos niveles de

granularidad [6], sin embargo, este enfoque tiene problemas de escalabilidad, dado que tiene un alto costo en términos de espacio ocupado, y además, está enfocado en datos que no varíen o sufran modificaciones una vez insertados.

3. Trabajo desarrollado

Se trabajó en el desarrollo de un modelo que explota la granularidad de la información de manera eficiente, además del desarrollo de los operadores básicos necesarios para poder responder diversas consultas comunes y en un generador de datos para la medición de rendimiento de lo desarrollado.

3.1. Modelado de base

Para modelar la base se considera lo siguiente:

El modelo multigranular contempla dos reglas básicas en términos de gránulos:

- Regla de *subsumption*: posee la forma $g_1 \sqsubseteq_{\mathfrak{S}} g_2$, la cual indica que el dominio que representa el gránulo g_1 esta contenido en el dominio representado por g_2.
- Regla de *disjoint*: posee la forma $\bigsqcap_{\mathfrak{S}} \{g_1, g_2\} = \bot_{\mathfrak{S}}$, la cual indica que el dominio del gránulo g_1 no intersecta con el dominio del granulo g_2

En base a estas dos reglas básicas y sus respectivas negaciones lógicas se pueden construir normas complejas en caso de ser necesario y definir diversos tipos de relaciones.

Para el modelo se considerarán los siguientes tipos de relación entre granularidades:

- *Subtype relation*: $G_1 \trianglelefteq^{\Phi}_{\mathfrak{S}} G_2$. Esta relación indica que $\forall\ g$ en G_1 existe un granulo g' en G_2 tal que el dominio representado por g es el mismo que el de g'.
- *Equality-join relation*: $G_1 \trianglelefteq^{\Phi}_{\mathfrak{S}} G_2$. Esta relación indica que $\forall\ g$ en G_2 está compuesto por la unión (disjunta) de gránulos en G_1
- *Inequality-join relation*: $G_1 \circledcirc^{\Phi}_{\mathfrak{S}} G_2$. Esta relación indica que $\forall\ g$ en G_2 está contenido dentro del dominio formado por la unión (disjunta) de gránulos en G_1.
- *Two-way inequality-join relation*: $G_1 \circledcirc\!\!\circledcirc^{\Phi}_{\mathfrak{S}} G_2$. Esta relación indica que se posee tanto $G_1 \circledcirc^{\Phi}_{\mathfrak{S}} G_2$ como $G_2 \circledcirc^{\Phi}_{\mathfrak{S}} G_1$

Estas relaciones entre granularidades definen el comportamiento entre sus gránulos, lo cual es útil al momento de trabajar con las reglas básicas de *disjoint* y *subsumption*. Estas relaciones adquieren importancia debido a lo siguiente:

- Son de frecuente aparición en instancias reales.
- Las reglas bigranulares implican *disjoint*, dado que los gránulos de una misma granularidad son *disjoint* entre sí.
- *Equality-join* es transitivo, dado que se basa en *subsumption*, la cual es transitiva. En consecuencia solo es necesario representar ese tipo de relación en un par de granularidades que sirvan de base para inferir los demás caso transitivos.
- Tanto *equality-join* como *inequality-join* son *equiresolvable*, es decir el resolver *disjoint* entre cualquier par de gránulos resuelve a la vez *subsumption*

En base a lo anterior se propone el siguiente esquema de modelo que representa la estructura multigranular de los datos:

1.- Granularity(Grty) PK Grty.

Esta tabla guardará un único valor por entrada que será el nombre representativo de cada granularidad en la base.

2.- Grty(Granule) PK Granule.

Por cada granularidad en la tabla Granularity existirá una tabla correspondiente a ella con su mismo nombre que guardará todos sus gránulos.

3.- Joins(grtyhead, grtybody, type, Tname, NDname) PK (grtyhead, grtybody), FK (grtyhead, grtybody) from Granularity.

Esta tabla contendrá los nombre del par de granularidades que tengan relación ya sea de manera explícita en el grafo de granularidades o que ciertos gránulos entre ambas posean una relación necesaria de representar para poseer información completa, esto último debido a la existencia en la práctica de relaciones a nivel de gránulos de manera particular pero no a nivel de granularidad. Además, se guarda el tipo de relación, lo que significa que grtyhead posee la relación type con la granularidad grtybody, y donde type puede ser algunas de las siguientes alternativas:

- *Equality-join*: EQU
- *Sub-type*: STP
- *inequality-join*: INE
- *two-way inequality-join*: TIN
- *Join* particular: JNP

Con JNP se hace referencia a la relación entre gránulos que puede referirse tanto a *subsumption* o *disjoint* de manera particular entre solo un conjunto de los gránulos de dichas granularidades y que es necesaria su instancia para poder poseer información completa en la base, esto es necesario dado que tanto las relaciones bigranulares consideradas no aportan suficiente información sobre *disjointness* como también existen instancias que poseen relaciones particulares que difieren de las planteadas.

En esta tabla Tname hace referencia al nombre de la tabla donde se guardará la relación de *subsumption* entre los gránulos de tales granularidades y NDname hace referencia al nombre de la tabla donde se guardará la relación de *no disjoint* entre los gránulos de tales granularidades.

Tname(Ghead, Gbody) PK Ghead,Gbody.

Tabla con nombre particular compuesto por la unión de los nombres de las granularides a las que hace referencia, cada tupla señala que el gránulo Gbody subsume al granulo Ghead.

NDname(Ghead, Gbody) PK Ghead,Gbody.

Tabla con nombre particular compuesto por la unión de los nombres de las granularides a las que hace referencia, cada tupla señala que el gránulo Gbody es *no disjoint* con respecto al gránulo Ghead. Se eligió el guardar no disyunciones dado que asume que lo que no sea *disjoint* será *no disjoint* y en general es necesario guardar menos gránulos *no disjoint* que *disjoint*. Cabe destacar que solo se guardarán *no disjoint* de INE, TIN y JNP solamente. Lo demás se tratará de inferir.

3.2. Operadores

El modelo anterior propone una estructura que relaciona los valores del dominio de un atributo. Un modelo clásico debería guardar de manera explícita la relación entre los valores de los atributos para cada instancia de base de datos que los usará. En el modelo propuesto, por lo contrario, mantiene las relaciones entre valores de los atributos como parte de la estructura.
Dado que el modelo planteado se basa principalmente en reglas de *subsumption* y *disjoint* es necesario proveer métodos que sean capaces de trabajar con tales reglas, con el fin de lograr responder las mismas consultas que en un modelo clásico. Por ello se proponen los siguientes 7 operadores básicos:

1. Coarsen(Granule g, Granularity G, Granularity G') = Granule $g' \in G'$ | g $\in G \wedge$ g $\sqsubseteq$ g'.
   El operador recibe un gránulo g, la granularidad a la que pertenece y una granularidad G' objetivo y retorna un gránulo perteneciente a G' que contiene a g. Ejemplo: Coarsen(Provincia de Concepción, Province, Region) retornará Bio-Bio.

2. CoarsenMUB(Granularity G_1, Granularity G_2) = S $\subseteq$ Glty($\mathfrak{S}$) | $G_1 \subseteq S \wedge G_2 \subseteq S$.
   Dada dos granularidades G_1 y G_2 retorna un conjunto de granularidades tal que cada una contiene una relación de *subsumption* con G_1 y G_2, y donde Glty($\mathfrak{S}$) hace referencia al conjunto de todas la granularidades del esquema $\mathfrak{S}$. Ejemplo: CoarsenMUB(District,Province) retornara Region,Chile.

3. Refine(Granule g,Granularity G, Granularity G') = conjunto gránulos S $\subseteq G'$ | g $\in G \wedge \forall$ g' $\in$ S g' $\sqsubseteq$ g.
   Dado un gránulo g, la granularidad a la que pertenece y una granularidad G' objetivo retornará un conjunto de gránulos pertenecientes a G' tal que g subsuma a cada uno. Ejemplo: Refine(Region de Concepción,Province) retornará conjunto de 4 gránulos correspondiente a las 4 provincias de la region de Bio-Bio.

4. RefineMLB(Granularity G_1, Granularity G_2) = S $\subseteq$ Glty($\mathfrak{S}$) | $S \leq^{\Phi}_{\mathfrak{S}} G_1 \wedge S \leq^{\Phi}_{\mathfrak{S}} G_2$.
   Dada dos granularidades G_1 y G_2 retorna un conjunto de granularidades tal que cada una contiene una relación de *subsumption* con G_1 y G_2.

Ejemplo: RefineMLB(District,Province) retornará County, Village, City, ElecConst, ElecTable.

5. Disjoint(Granule g, Granule g', Granularity G, Granularity G') = Verdadero si $\bigsqcap_{\mathfrak{S}}\{G_1, G_2\} = \bot_{\mathfrak{S}}$, falso en caso contrario.
   Dado dos gránulos y sus respectivas granularidades retorna verdadero en caso que sean *disjoint*, falso en caso contrario. Ejemplo Disjoint(Region del Bio-Bio, Distrito 2) retornará falso.

6. Coarse(Granule g, Granule g', Granularity G, Granularity G') = Verdadero si $g \sqsubseteq_{\mathfrak{S}} g'$, falso en caso contrario.
   Dado dos gránulos y sus respectivas granularidades retorna verdadero en caso que g' subsuma a g, falso en caso contrario. Ejemplo Coarse(Región del Bio-Bio, Concepción) retornará verdadero.

7. Granules(Granularity G) = conjunto gránulos S $\subseteq$ G | $\forall$ g' $\in$ G, g' $\in$ S.
   Dada una granularidad retornará todos los gránulos pertenecientes a ella. Ejemplo Granules(Región) Retornará conjunto de gránulos que representan las regiones de chile.

3.3. Generador de datos

Se creó un generador de datos y esquema de granularidades dada la inexistencia de uno para bases de datos multigranulares, esto con el objetivo de poder realizar pruebas de escalabilidad y probar el comportamiento de los operadores en diversas instancias de base de datos. El generador funciona de la siguiente manera:

1. Recibe como primer parámetro de entrada la cantidad de datos a existir en la instancia de base. Dado que el generador tiene por objetivo medir escalabilidad y correctitud de los operadores los datos a generar corresponden a tuplas que solamente poseen un valor numérico. Además se especifica si es que desea generar el esquema de granularidad de manera aleatoria o si este será ingresado manualmente. Finalmente se ingresan diversos parámetros para guiar la generación del esquema de granularidades, los que incluyen la cantidad de granularidades, cantidad máxima de relaciones por granularidad y tipos de relaciones permitidas. En caso de no desear generar el esquema de granularidad de manera aleatoria, se ingresa el esquema por entrada estándar y continúa al paso 3.

2. Se genera el esquema de granularidades de manera aleatoria en base a los parámetros otorgados.

3. Por cada granularidad se generan los gránulos pertenecientes a ella.

4. Se crean de manera aleatoria según los parámetros de entrada y el esquema de granularidad las relaciones de *subsumption* entre los gránulos.

5. Se asignan los datos básicos generados en el paso 1 a las granularidades más finas, esto con el objetivo de simplificar la realización de pruebas de escalabilidad.

6. En base a lo generado anteriormente se genera un archivo con la información generada traducida a tablas y datos en lenguaje SQL en base al modelo propuesto.

En el paso 2 para la generación del esquema de granularidades se crea en primera instancia las granularidad $\bot$ correspondiente a la granularidad que posee los gránulos a nivel más fino y $\top$ que corresponde a la granularidad con los gránulos en su mayor grosor. Luego partiendo desde $\bot$ y en base a los parámetros ingresados se van generando las relaciones entre granularidades

de manera aleatoria. Para la selección aleatoria, se ocupan dos colas, una para los nodos sin procesar y otra para los nodos en cola para procesar, de manera que solo se seleccionarán nodos aleatorios de la cola de nodos sin procesar, esto para evitar ciclos en las relaciones.

3.4. Implementación operadores

A continuación se listan los algoritmos implementados para cada uno de los operadores propuestos. Se incluye un análisis de complejidad para cada operador, no se incluyen en el análisis factores relacionados al procesamiento de consultas SQL ni de tiempo de extracción de datos.

Algorithm 1 Operador Coarsen

Input: Granule g, Granularity G and G'
Output: Granule g'

```
Vector Visitados, Queue Analizar
Analizar.push((G,g))
while Analizar.size != empty() do
  Granularity grty ← Analizar.top().first
  Granule grano ← Analizar.top().second
  Analizar.pop()
  for grty' IN Join(grty) do
    if !Visitados[grty'] then
      if grty' == G' then
        return Granulos(grty',grano)
      Analizar.push((grty',Granulos(grty',grano))
return
```

Join(grty) es una consulta SQL que retornará las granularidades con las que posee una relación *grty* en la tabla *Join* definida en el modelo propuesto. Granulos($grty'$, *grano*) es una consulta SQL que retornará el gránulo que subsume a *grano* en $grty'$. El algoritmo propuesto recorrerá cada posible camino hasta la granularidad objetivo, a cada paso expande por anchura, de esta forma asegura que una vez encontrado el camino hasta la G' en el esquema de granularidades este será el más corto.
Cabe destacar que el que sea el camino de menor distancia no necesariamente indica que sea el de menor costo dado que parte del costo se asocia a la cantidad de gránulos a analizar. Sea K el tamaño del camino mínimo desde G hasta G', J la cantidad de tuplas en la tabla de joins, M la cantidad de

gránulos en G y N la cantidad de granularidades posibles a visitar iniciando en G y que estén a una distancia a lo más de K, la complejidad de la consulta SQL Join será O(J) y consecuentemente, la complejidad del operador será O(N*M + J*N).

Algorithm 2 Operador CoarsenMUB

Input: Granularity G and G'
Output: Set of Granularities S'
  return Iguales(RecorrerTop(G),RecorrerTop(G'))

Algorithm 3 Función RecorrerTop

Input: Granularity G
Output: Set of Granularities S
  Vector Visitados, Queue Analizar
  Analizar.push(G)
  while Analizar.size != empty() **do**
    Granularity grty ← Analizar.top()
    Analizar.pop()
    for $grty'$ IN Join(grty) **do**
      if !Visitados[$grty'$] **then**
        Analizar.push($grty'$)
  return

CoarsenMUB se compone de dos funciones, la función principal llamada CoarsenMUB la cual utiliza una consulta SQL definida como Iguales, que dado dos conjuntos de granularidades retorna las granularidades en común, y una función llamada RecorrerTop definida a continuación, la cual dado una granularidad G retorna todas las granularidades mas gruesas que sean *no disjoint* con la granularidad G. El costo del operador estará dado principalmente por el costo de la función RecorrerTop. Sea N la cantidad de granularidades que se encuentras entre G y T en el esquema de granularidades, el costo de la función sería O(N), dado que el costo de determinar los elementos iguales entre dos conjuntos de tamaño N es N*N se concluye que el costo del operador CoarsenMUB es O((N*N)+ N).

A continuación se muestra el algoritmo propuesto para la implementación del operador Refine, el cual es similar al operador Coarsen, difiriendo en que avanza a través de granularidades más finas, además de que en vez de avanzar por cada granularidad avanza a través de los gránulos de cada granularidad, guardando por granularidad los gránulos refinados de g. El operador utiliza una consulta SQL denominada Join que retorna las granularidades y los gránulos en aquellas granularidades por los cuales debe avanzar refine. El algoritmo propuesto recorrerá cada posible camino hasta la granularidad objetivo, a cada paso expande por anchura las siguientes granularidades a las cuales avanzar, de esta forma asegura que una vez encontrado el camino hasta la G' en el esquema de granularidades este será el de menor distancia. Sea K la cantidad de gránulos refinados desde G hasta G', J la cantidad de tuplas en la tabla de Joins, M la cantidad de gránulos en la granularidad de mayor tamaño analizada, la complejidad de la consulta SQL Join será O(J), en consecuencia la complejidad del operador será O(K*M + J*K).

Algorithm 4 Operador Refine

Input: Granule g, Granularity G and G'
Output: Set of Granules g'

```
Vector Visitados[][], GranosGranularity[][],Queue Analizar
Analizar.push((G,g))
while Analizar.size != empty() do
  Granularity grty ← Analizar.top().first
  Granule grano ← Analizar.top().second
  Analizar.pop()
  for grty' IN Join(grty, grano) do
    if !Visitados[grty'.first][grty'.second] then
      Analizar.push((grty'.first, grty'.second))
      GranosGranularity[grty'.first].push(grty'.second)
return GranosGranularity[G']
```

Algorithm 5 Operador RefineMUB función principal

Input: Granularity G and G'
Output: Set of Granularities S'
  return Iguales(RecorrerBottom(G), RecorrerBottom(G'))

Algorithm 6 Función RecorrerBottom

Input: Granularity G
Output: Set of Granularities S
  Vector Visitados, Queue Analizar
  Analizar.push(G)
  while Analizar.size != empty() **do**
    Granularity grty ← Analizar.top()
    Analizar.pop()
    for $grty'$ IN Join(grty) **do**
      if !Visitados[$grty'$] **then**
        Analizar.push($grty'$)
  return

Para el operador RefineMUB se propone el mismo algoritmo utilizado en el operador CoarsenMUB, con el único cambio en la consulta SQL utilizada denominada Join en la función RecorrerBottom, la cual para este caso recorre avanzando a través de granularidades más finas a diferencia del operador CoarsenMUB que avanza a través de granularidades más gruesas. Dado lo anterior su costo es el mismo del operador CoarsenMUB.

Algorithm 7 Operador Disjoint

Input: Granule g, Granularity G, Granule g' and G'
Output: A Boolean state
  if G == G' **then**
    return TRUE
  if Coarsen(g,G,G') == g' **then**
    return FALSE
  if Inst(g,G,g',G') **then**
    return FALSE
  return TRUE

El algoritmo propuesto para la implementación del operador Disjoint distingue los siguientes 3 casos:

1. Cuando los granos g y g' son de la misma granularidad.

2. Cuando g subsume a g' o viceversa.

3. Cuando existe una instancia explícita en la base que indique que son *no disjoint*.

Si ningún caso de los anteriores se cumple se asumirá que es *disjoint*, esto debido a que se asume mundo cerrado. El costo del algoritmo propuesto para el operador estará dado principalmente por el caso 2, dado que el caso 1 es de comprobación directa y el costo del caso 3 es N', siendo N' la cantidad de tuplas en la tabla en caso de existir. El costo del caso 2 será el mismo que el costo del operado Coarse dado que lo utiliza para realizar la comprobación.

Algorithm 8 Operador Coarse

Input: Granule g and g', Granularity G and G'
Output: A Boolean state
  if Coarsen(g,G,G') == g' **then**
    return TRUE
  return FALSE

El algoritmo propuesto para la implementación del operador Coarse hace uso del operador Coarsen para verificar si es que g subsume a g', el costo del operador por lo tanto sera el mismo que el costo del operador Coarsen.

Algorithm 9 Operador Granules

Input: Granularity G
Output: Set of granules S
  return Granulos(G)

El algoritmo propuesto para la implementación del operador Granules retornar de manera directa una función SQL que aprovecha el modelo propuesto para retornar todos los gránulos de la granularidad G de manera directa. El costo del operador es O(N), siendo N la cantidad de tuplas en la tabla.

4. Experimentos y Resultados

Se implementaron los operadores en PLPGSQL como funciones, usando como base de datos la ofrecida por http://isci.inf.udec.cl.

4.1. Escalabilidad

Se realizaron pruebas de escalabilidad para cada operador con el objetivo de comprobar su eficiencia en escenarios de grandes volúmenes de datos. Se usaron instancias de bases de datos de diversos tamaños en términos de datos y diversos esquemas de granularidades. Se forzó el esquema de granularidad de las bases de prueba de cada operador para garantizar que con los algoritmos implementados no tuviera que analizar más de una vez una misma granularidad, esto con el objetivo de poder reflejar el comportamiento de los operadores en base al tamaño de datos y granularidades sin verse afectado por el esquema de granularidades. Cada operador se ejecutó un total de 10 veces asegurando que cada ejecución no se viera afectada por el uso de memoria cache, guardando para cada set de ejecuciones el peor tiempo obtenido.

4.1.1. Coarsen

El operador Coarsen fue implementado de manera iterativa. En base al seudocódigo se puede notar que los aspectos relevante en su desempeño son la cantidad de gránulos a analizar y la cantidad de granularidades a visitar.

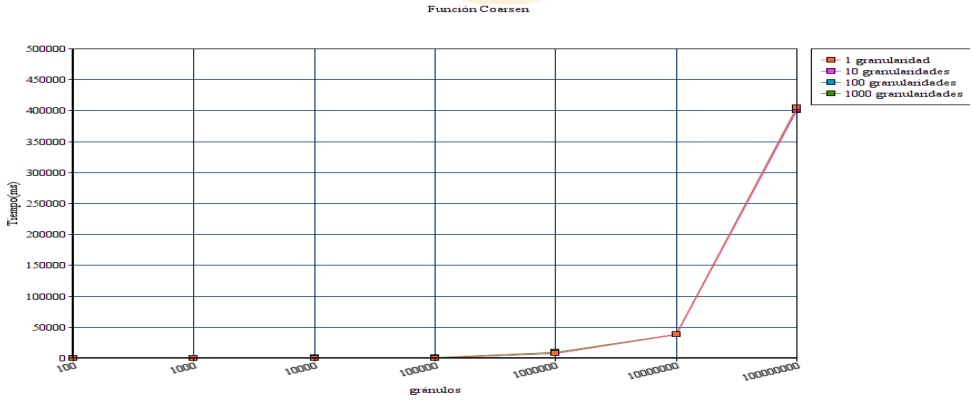

Figura 2: Gráfico desempeño función Coarsen

En la figura 2 se aprecia el desempeño medido en milisegundos del operador coarsen con respecto a la cantidad de gránulos analizados. Cada línea representa una cantidad determinada de granularidades analizadas.

Granos/Granularidades	1	10	100	1000
100	88	112	143	314
1000	110	153	170	353
10000	121	166	211	610
100000	131	352	365	1110
1000000	7759	7790	7668	9633
10000000	37975	38432	38475	38654
100000000	405099	401237	400975	405459

Figura 3: Tabla desempeño función Coarsen (ms)

En la figura 3 se puede apreciar con mayor detalle los tiempos obtenidos. De ella se puede concluir que para un gran números de gránulos a analizar la cantidad de granularidades analizadas no afecta de mayor manera el desempeño del operador.

4.1.2. CoarsenMUB

El operador CoarsenMUB fue implementado de manera iterativa, dado el seudocódigo se puede apreciar que el principal factor que afecta su desempeño en la cantidad de granularidades a analizar.

Granularidades	1	10	100	1000
Tiempo (ms)	3130403	3136969	2829899	1625206

Figura 4: Tabla desempeño función CoarsenMUB

La figura 4 representa el tiempo obtenido en milisegundos con respecto a la cantidad de granularidades analizadas. Se puede apreciar que no presenta una mayor diferencia sin importar la cantidad de granularidades analizadas en la instancia, esto se debe a que sin importar la cantidad de granularidades a retornar, analiza de manera completa la tabla de joins, por lo que su desempeño estará dado según el tamaño de la tabla.

4.1.3. Refine

Implementado de manera iterativa, el principal factor que afecta su desempeño es la cantidad de gránulos a analizar de manera particular.

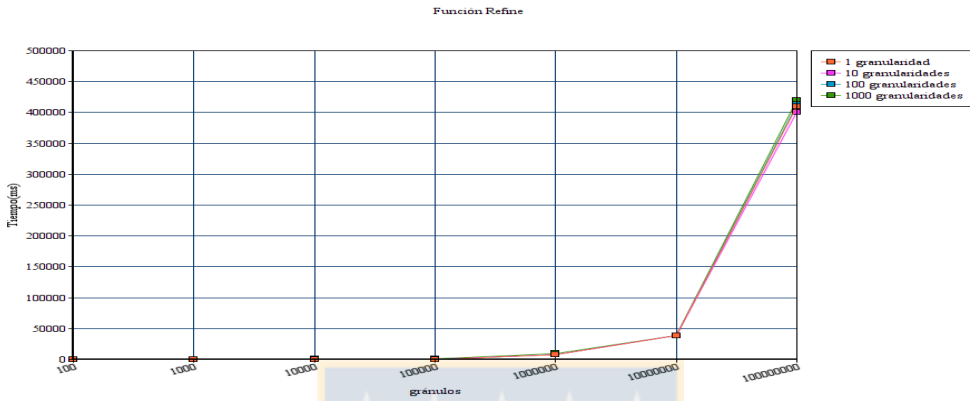

Figura 5: Grafico desempeño función Refine

Granos/Granularidades	1	10	100	1000
100	84	110	146	344
1000	106	156	170	360
10000	117	184	220	610
100000	143	341	366	1111
1000000	7643	7846	7776	9633
10000000	38654	38846	38755	38901
100000000	410010	400795	412605	420046

Figura 6: Tabla desempeño función Refine (ms)

La figura 6 refleja el desempeño en términos de cantidad de gránulos y cantidad de gránulos analizados. Se puede notar un desempeño similar al operador Coarsen considerando la similitud que tiene la cantidad de gránulo analizados con la cantidad de granularidades y dado que se forzó el esquema de granularidades de las instancias de bases de datos de manera que el operador no analizara más de una vez una granularidad.

Granularidades	1	10	100	1000
Tiempo (ms)	2945097	3060743	3100043	2322709

Figura 7: Tabla desempeño función RefineMUB

4.1.4. RefineMUB

Se puede notar resultados similares a CoarsenMUB dado que es el mismo algoritmo pero con retorno distinto solamente.

4.1.5. Disjoint y Coarse

Disjoint y Coarse tienen como función base Coarsen por lo que su desempeño y escalabilidad estará determinada por este último.

4.1.6. Granules

Granularidades	100	1000	10000	100000	1000000	10000000	100000000
Tiempo (ms)	12	34	56	453	47654	171287	3948812

Figura 8: Tabla desempeño función Granules

Se puede notar que el desempeño de esta función se ve afectada solamente por el tamaño de la tabla a retornar, a tamaños muy grandes de retorno se vuelve lenta.

4.2. Comportamiento

Los operadores de mayor relevancia en base a lo expuesto anteriormente son los operadores de Coarsen, CoarsenMUB y Refine. Por ello se realizaron pruebas con diversas instancias de bases de datos creadas con el generador desarrollado, con el objetivo de mostrar el rendimiento de los operadores implementados frente a diversos esquemas de granularidad y detectar los factores que afectan de manera negativa a los operadores. Todas las instancias generadas poseen un total de 100000 datos básicos y para la prueba se seleccionaron parámetros de entrada manualmente en base a los parámetros a medir.

4.2.1. Coarsen

Se realizaron pruebas en diversas instancias, dado que el algoritmo tiene dependencia principalmente de la cantidad de granularidades se analizó la influencia de esta variable.

Cantidad Granularidades	Distancia	Granularidades analizadas	Tiempo(ms)
200	2	20	9.82
2000	2	20	9.80
20000	2	20	9.81
1000	3	50	10.98
1000	3	150	10.95
1000	3	1100	11.06
1000	10	1000	15.47
1000	100	1000	60.64
1000	1000	1000	3305.06

Cuadro 1: Tabla operador Coarsen

4.2.2. CoarsenMUB

Se realizaron pruebas con el objetivo de medir el desempeño del operador en base a la cantidad de granularidades en el esquema de granularidades.

Cantidad Granularidades	Granularidades analizadas	Tiempo(ms)
200	500	808.02
2000	500	807.45
20000	500	807.97
1000	50	9.59
1000	150	86.95
1000	1100	2401.98

Cuadro 2: Tabla operador CoarsenMUB

4.2.3. Refine

Se realizaron pruebas con el objetivo de medir el desempeño del operador en base a la cantidad de granularidades y gránulos analizados.

Cantidad Granularidades	Cantidad gránulos	Granularidades analizadas	Tiempo(ms)
200	500	5	432.78
2000	500	50	434.01
20000	500	500	433.96
500	500	500	431.68
5000	500	500	433.46
50000	500	500	434.97
500	5	5	8.12
5000	50	50	20.64
50000	500	500	437.44

Cuadro 3: Tabla operador Refine

4.3. Expresividad

Se probó si los operadores implementados eran suficientes para responder consultas SQL en el modelo propuesto y se comparó con consuta SQL ejecutadas en un modelo plano. Para la prueba se utilizó la información disponible en el SERVEL con respecto a los resultados de las elecciones primarias 2013 y se montaron dos bases, una replicando la que poseen ellos y otra en base al modelo propuesto

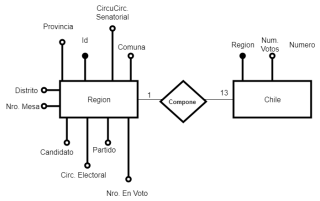

Figura 9: Entidad Relación base usada en el SERVEL

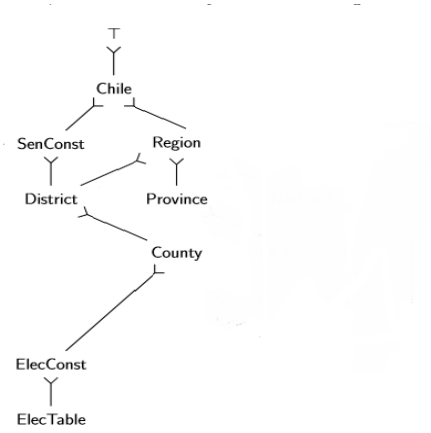

Figura 10: Esquema granularidad para los datos

Se probaron 3 consultas básicas sobre ambos modelos:

1. Candidatos con mayor cantidad de votos
2. Región con más votos emitidos
3. Candidato con mayor cantidad de votos en la comuna de Lebu

El cuadro 4 muestra un resumen de la información obtenida de ambos esquemas

Modelo	Tamaño(mb)	Consulta 1(ms)	Consulta 2(ms)	Consulta 3(ms)
Plano	20	109.6	116.1	22.6
Granular	20	521.0	164.6	90.4

Cuadro 4: Consultas

Estos resultados se deben a que la instancia utilizada era de poco tamaño y el esquema de granularidad asociado era pequeño, por lo que un modelo plano realiza menos accesos a memoria y posee la información de manera directa. En modelos de gran tamaño y que posean asociada una estructura de granularidad mayor o que posean más de una fuente de datos que deban integrar tendría sentido utilizar el modelo propuesto.

5. Conclusiones

Se desarrolló un modelo de base de datos que explota la granularidad de los datos segmentando según sea su nivel de detalle. Además se propusieron los operadores necesarios para lograr responder consultas SQL en el modelo propuesto y sus respectivos algoritmos para su implementación, con el fin de probar que el modelo propuesto es capaz de responder las mismas consultas que una implementación de base de datos plana.
Considerando los resultados, es visible que los operadores propuestos son suficiente para responder las mismas consultas SQL que en un modelo plano. De los resultados obtenidos se puede concluir que los operadores son escalables, y que el principal factor que influye en el desempeño de los operados en el procesamiento de grandes volúmenes de datos es el costo de extraer las tuplas de las diversas tablas.
Como trabajo futuro se plantea implementar un lenguaje de consultas que haga uso de este modelo y de los operadores propuesto. Al mismo tiempo se propone expandir el abarque del modelo, tomando en cuenta posibles $join - rules$, buscando además proponer un nuevo modelo que aporte información suficiente para mejorar la eficiencia de los operadores propuestos.
Adicionalmente se propone incluir la existencia de tuplas incompletas en la base de datos, y expandir el modelo propuesto de manera que sea capaz de trabajar con datos incompletos.

Referencias

[1] Hegner, S. J., Rodríguez, M. A. (2017). A Model for Multigranular Data and Its Integrity. Informatica, 28(1), 45-78.

[2] Alberto Belussi, Carlo Combi, and Gabriele Pozzani. Formal and conceptual modeling of spatio-temporal granularities. In International Database Engineering and Applications Symposium IDEAS, pages 275–283. ACM, 2009.

[3] Elena Camossi, Michela Bertolotto, and Elisa Bertino. A multigranular object-oriented framework supporting spatio-temporal granularity conversions. International Journal of Geographical Information Science, 20(5):511–534, 2006.

[4] Martin Erwig and Markus Schneider. Partition and conquer. In Spatial Information Theory: A Theoretical Basis for GIS, International Conference COSIT '97, Laurel Highlands, Pennsylvania, USA, October 15-18, 1997, Proceedings, volume 1329 of Lecture Notes in Computer Science, pages 389–407. Springer, 1997.

[5] Inmon, William H. Building the data warehouse. John wiley sons, 2005.

[6] Iftikhar, Nadeem, and Torben Bach Pedersen. "Schema design alternatives for multi-granular data warehousing.Ïnternational Conference on Database and Expert Systems Applications. Springer, Berlin, Heidelberg, 2010.

[7] Implicit Representation of Bigranular Rules for Multigranular Data, Stephen J. Hegner and M. Andrea Rodríguez. En Hartmann et al. (Eds.): DEXA 2018, LNCS 11029, pp. 1–18, 2018.