Universidad Nacional Autónoma de México

Facultad de Ciencias

Descomposición en valores singulares usando LAPACK integrado al SMBD PostgreSQL

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

AUGUSTO JULIO CÉSAR VEGA GUTIÉRREZ

DIRECTOR DE TESIS:

DR. JAVIER GARCÍA GARCÍA

FACULTAD DE CIENCIAS, UNAM

Ciudad Universitaria, D.F.

2014

Hoja de datos del jurado

1. Datos del alumno

Apellido Paterno	Vega
Apellido Materno	Gutiérrez
Nombre(s)	Augusto Julio César
Teléfono	62790387
Universidad	Universidad Nacional Autónoma de México
Facultad	Facultad de Ciencias
Carrera	Ciencias de la Computación
Número de cuenta	307333180

2 Datos del tutor

Grado	Dr.
Nombre(s)	Javier
Apellido Paterno	García
Apellido Materno	García

3 Datos del sinodal 1

Grado	Dra.
Nombre(s)	Amparo
Apellido Paterno	López
Apellido Materno	Gaona

4 Datos del sinodal 2

Grado	M. en C.
Nombre(s)	Rogelio
Apellido Paterno	Montero
Apellido Materno	Campos

5 Datos del sinodal 3

Grado	Dra.
Nombre(s)	María de Luz
Apellido Paterno	Gasca
Apellido Materno	Soto

6 Datos del sinodal 4

Grado	M. en I.
Nombre(s)	Gerardo
Apellido Paterno	Avilés
Apellido Materno	Rosas

7. Datos del trabajo escrito

Título	Descomposición en valores singulares usando LAPACK integrado al SMBD PostgreSQL
Número de páginas	83
Año	2015

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La minería de datos ha atraído una gran atención para la industria de Tecnologías de la Información y la sociedad ya que tenemos una gran cantidad de información, puede haber conocimiento que no haya sido descubierto aún pero se encuentre de manera implícita. El conocimiento obtenido puede ser relevante para algunas áreas como análisis de mercado, detección de fraudes, etcétera.
La minería de datos puede ser vista como un resultado de la evolución de las Tecnologías de la Información. La industria de las bases de datos ha sido testigo de un camino de evolución en el desarrollo de las siguientes funcionalidades

- Colección de datos y creación de las bases de datos, incluyendo almacenamiento y recuperación, además de procesamiento de transacciones.

- Análisis avanzado de datos, incluyendo almacenes y minería de datos.

En la Figura 1.1 se observa el avance de las funcionalidades en el manejo de datos y tecnología de los sistemas de bases de datos.

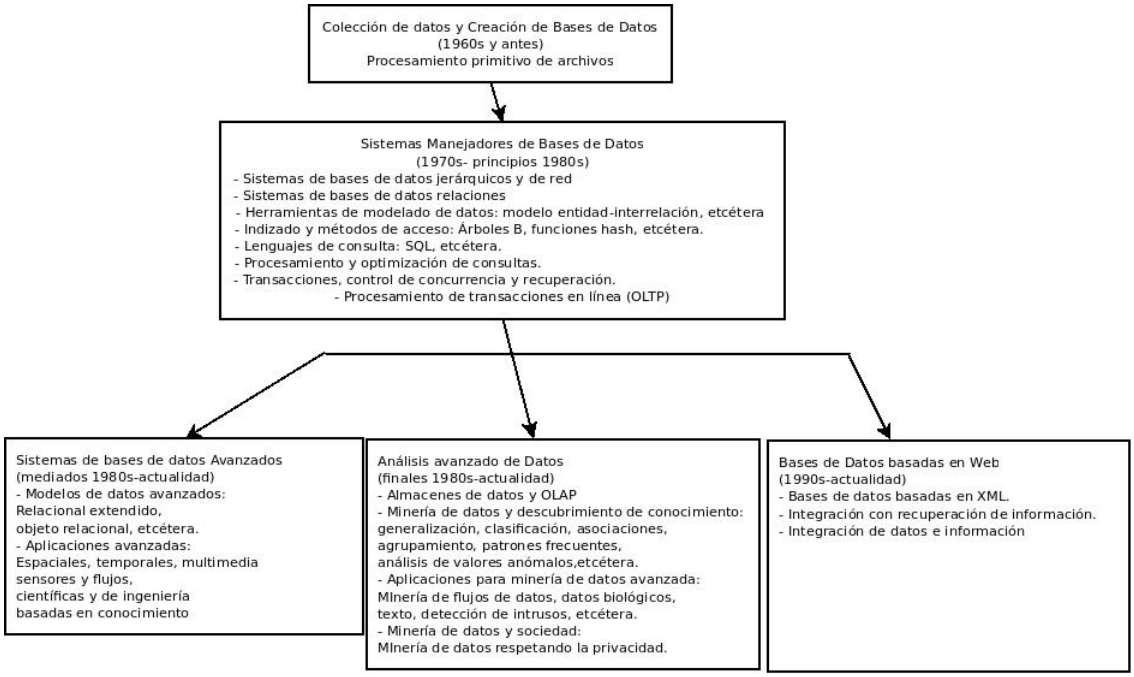

Figura 1.1: Evolución de las funcionalidades para el manejo de datos y la tecnología de los sistemas de bases de datos. Imagen traducida del libro cuyo autor es Han [14].

Un sistema manejador de bases de datos relacional (SMBDR) se encuentra limitado para realizar cálculos con matrices y vectores lo cual es de suma importancia en el área de minería de datos, ya que existen muchos métodos estadísticos, utilizados en esta área, que involucran el manejo de matrices y vectores. El motivo por el que se encuentra limitado es debido a que la estructura básica del modelo relacional es la relación. Por lo cual resolver el problema de calcular operaciones matriciales y vectoriales dentro de un sistema manejador de bases de datos de manera eficiente es relevante.

Objetivo

El objetivo de la presente tesis es proponer una estrategia para realizar cálculos de álgebra lineal, como multiplicación de matrices, mismos que son ampliamente utilizados en modelos para realizar minado de datos de manera eficiente dentro de un sistema manejador de bases de datos(SMBD). Como ejemplo se puede mencionar la descomposición en valores singulares, explicada con mayor detalle en el capítulo siguiente, utiliza estos cálculos y nos ayuda al proceso de descubrimiento de conocimiento KDD [1].

Para lograr el objetivo se utilizará el SMBD PostgreSQL ligado a la biblioteca de funciones de álgebra lineal LAPACK al SMBD mencionado y se ejemplificará la estrategia mediante la descomposición en valores singulares.

En el trabajo se verificará la eficiencia de esta estrategia, logrando demos-

[1] Knowledge Discovery from Data

trar así que al evitar extraer los datos del SMBD y llevarlos a un programa externo, no se exponen los datos a ambientes inseguros ya que dentro del SMBD se encuentran protegidos y al sacarlos se puede perder la consistencia, integridad y otras propiedades que nos proporciona el SMBD.

En el capítulo 2 se explica de manera más detallada el software que puede realizar estos cálculos.

En resumen, los principales beneficios con esta estrategia son:

- Mejorar el tiempo de procesamiento de los datos para realizar cálculos matriciales.
- Evitar exponer los datos a ambientes inseguros.
- Trabajar con una versión reciente de los datos.

Cabe resaltar que este análisis de los datos se puede realizar de diversas formas por ejemplo mediante almacenes de datos o con el paradigma tradicional de extraer los datos y llevarlos a un software especializado para este fin. La propuesta presentada en este escrito tiene ciertas ventajas y desventajas sobre cada una de estas formas. Éstas serán analizadas en el Capítulo 2 Marco Teórico.

En la minería de datos se manejan grandes volúmenes de datos y el objetivo principal es obtener conocimiento. Por ejemplo, descubrimiento de patrones, obtención de conocimiento, etcétera a partir de los datos que se tienen. Como ya se mencionó, existen métodos en la minería de datos que requieren de estos cálculos para lograr su objetivo. Debido a esto el pro-

blema de realizar cálculos con matrices y vectores de manera eficiente es relevante, ya que en grandes volúmenes de datos la eficiencia es un factor muy relevante.

Para mostrar la eficiencia de la estrategia nos enfocaremos en una técnica, muy utilizada en minería de datos, la descomposición en valores singulares (SVD [2]).

Trabajos Relacionados

Este trabajo está basado en el artículo **Vector and matrix operations programmed with udfs in a relational dbms** [12]. Una función definida por el usuario UDF[3] es un mecanismo que proveen los SMBD para extender la funcionalidad del mismo. La estrategia que propone el artículo es realizar los cálculos matriciales y vectoriales mediante funciones definidas por el usuario escritas en SQL. Para hacer experimentación con el fin de verificar la eficiencia de la estrategia, el SMBD utilizado fue **Teradata**.

Los autores del artículo **Fast udfs to compute sufficient statistics on large data sets exploiting caching and sampling** proponen una optimización para poder calcular, mediante operaciones de álgebra lineal, un conjunto equivalente a los datos originales, llamado *estadísticas suficientes* [13]. El objetivo es acelerar este cálculo sobre grandes conjuntos de datos con UDF explotando la técnica de cacheo[4] ,cuando la tabla cabe en

[2]Singular Value Decomposition
[3]User Define Function
[4]Se usa el término cacheo como traducción de caching

memoria, y muestreo (obtener un conjunto representativo de los datos). Esta estrategia nos puede ser útil para no trabajar con todos los datos originales sino sólo con un conjunto representativo.

En el artículo **Efficient olap with udfs** se trata el tema de procesamiento analítico en línea OLAP [5] [22]. OLAP consiste de un conjunto de técnicas con funcionalidades como resumen (sumarización) y agregación así como la habilidad de ver la información de diversos ángulos. La estrategia que presenta el artículo es hacer el procesamiento mediante UDF dentro de un SMBD en lugar de las implementaciones que lo hacen fuera del SMBD, ya sea con servidores OLAP o directamente en la computadora del cliente.

Para el siguiente artículo se explicará de manera breve qué es el análisis de componentes principales. El análisis de componentes principales PCA[6] es una de las técnicas más comunes de reducción de dimensiones. Esta técnica nos permite identificar la relevancia de cada dimensión, eliminando dimensiones que podrían ser no relevantes o de bajo impacto si son eliminadas.

En el artículo [18] se propone una solución para calcular PCA sin bibliotecas externas que combina el resumen (sumarización) del conjunto de datos con la matriz de covarianza o correlación y resolver PCA mediante la descomposición en valores singulares (SVD, por sus siglas en inglés).
Además, en [21] se hace el análisis de conjuntos de microarreglos de datos dentro del SMBD en lugar de extraer los datos fuera de éste.

[5] On-Line Analytical Processing
[6] Principal Component Analysis

En el presente trabajo, a diferencia de los trabajos presentados y como una aportación, tomamos como SMBD a PostgreSQL, un SMBD que tiene un gran impacto en el ambiente científico y ligamos a este SMBD las funciones de LAPACK mediante el uso de Funciones Definidas por el Usuario (UDF), un mecanismo que provee este SMBD para extender la funcionalidad del mismo.
Asimismo, experimentamos esta solución con un algoritmo muy común para preprocesamiento de datos que es PCA y en particular, descomposición en valores singulares, que requiere de operaciones matriciales.

Como ya fue mencionado, este trabajo está basado en el artículo [12] el cual es un trabajo en conjunto por parte de mi tutor y el Dr. Carlos Ordonez. Esta idea ha tenido seguimiento con otros artículos publicados en foros y revistas de mucho reconocimiento. La intención de esta tesis es continuar con aportaciones en este tema.

Descripción de capítulos

La presente tesis está estructurada de la siguiente manera:

- Capítulo 2 Marco Teórico. Proporciona la teoría necesaria para la comprensión de la tesis. Manejando los siguientes conceptos del área de bases de datos: SQL, SMBD, Funciones de Agregación, PostgreSQL, UDF, Descomposición en valores singulares, Reducción de dimensiones y PCA, así como algunas aplicaciones. Además se presenta el software que fue utilizado para la experimentación. También se presentan aplicaciones de una de las funciones más utilizadas en minería de datos.

- Capítulo 3 Propuesta y experimentación. En este capítulo se presenta la propuesta para resolver el problema de realizar el cálculo eficiente de operaciones matriciales y vectoriales. Además encontraremos el código utilizado para realizar los experimentos, los cuales nos permiten corroborar o refutar la hipótesis que al realizar la integración de *LAPACK* dentro de *PostgreSQL* esto se hace de manera eficiente. Además, algunos detalles técnicos que fueron supuestos por simplicidad y que son fácilmente modificables en caso de ser necesario para algún experimento posterior. Además se muestran los resultados de la experimentación, imágenes tanto de los registros de las tablas, ejemplificando la estructura que tienen éstas, además de presentar ejemplos de las ejecuciones de las UDFs. También se discuten los resultados obtenidos de la experimentación.

- Capítulo 4. Conclusión, se da un análisis en base a los resultados obtenidos y la propuesta que se hizo.

Capítulo 2

Marco Teórico

En este capítulo definiremos los siguientes términos necesarios para una mayor comprensión del material a desarrollar en este trabajo:

- SQL, ya que es el lenguaje para la manipulación, definición y control dentro del SMBD.
- Sistema Manejador de Bases de Datos (SMBD), se describen sus características de un SMBD, PostgreSQL, que es el SMBD utilizado para la experimentación y dentro del cual se realizará la integración.
- Minería de datos, reducción de dimensiones, descomposición en valores singulares (SVD) así como aplicaciones de ésta y del análisis de componentes principales. El concepto de SVD es fundamental ya que la integración se hará con esta factorización.
- LAPACK, una biblioteca que contiene funciones de álgebra lineal así como las implementaciones usadas en el presente trabajo.

Por último, se menciona cómo se realizó la inhibición del caché.

2.1. SQL

SQL es un lenguaje de consulta estructurado el cual fue desarrollado originalmente por IBM. Esta versión de SQL, llamada Sequel, se desarrolló en los 70's. En 1986, el Instituto de Estándares Nacionales Americanos (ANSI, por sus siglas en inglés) y la Organización Internacional para Estandarizar (ISO, por sus siglas en inglés) publicaron un estándar de SQL llamado *SQL-86*. Posteriormente, se liberaron las siguientes versiones: *SQL-89*, *SQL-92*, *SQL:1999*, *SQL:2003*, *SQL:2006*, *SQL:2008*.
Los conceptos necesarios acerca del lenguaje SQL son: [19]

- *Lenguaje de Definición de Datos (DDL, por sus siglas en inglés)*. Provee comandos para la definición de los esquemas, borrar relaciones y modificar los esquemas de las relaciones.
- *Lenguaje de Manipulación de Datos (DML, por sus siglas en inglés)*. Provee la habilidad de obtener información de la base de datos e insertar tuplas, borrarlas y modificarlas en la base de datos.
- *Control de Transacciones*. SQL incluye comandos para especificar el inicio y término de una transacción.
- *SQL Embebido* y *SQL Dinámico*. Define cómo las sentencias SQL pueden ser embebidos con lenguajes de propósito general, como C++ o Java.
- *Control*. Incluye comandos para indicar los permisos de acceso a relaciones y vistas.

2.2. Sistemas manejadores de bases de datos

Un sistema manejador de base de datos (SMBD) es un software que facilita el proceso de creación y modificación de la base de datos. Permite definir, construir, manipular y compartir una base de datos a varios usuarios y aplicaciones. Además, cuando se crea la base de datos es necesario definir algunos puntos como:

- Tipos de datos
- Estructuras
- Reglas sobre los datos

La definición de la base de datos (BD) o información descriptiva también es guardada por el SMBD como un diccionario o catálogo de la BD; llamado metadatos. Construir la BD es el proceso de guardar los datos en algún medio de almacenamiento controlado por el SMBD. La manipulación de las bases de datos incluye funciones como hacer una consulta a la BD para obtener información específica, actualizar la base de datos y generando reportes de los datos. Compartir la BD permite tener múltiples usuarios y programas accediendo a la base de datos.

Silberschatz define a un SMBD como una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. [19] El objetivo principal de un SMBD es proveer una camino de guardar y obtener información de la base de datos; esto es, de manera conveniente y eficiente.

Una función importante del SMBD es la protección y mantenimiento de la base de datos. En la Figura 2.1 se observan los componentes que tiene

un SMBD, así como la forma de comunicación entre ellos cuando se realiza un cambio o consulta a una base de datos. Por ejemplo, si se tiene una consulta, observamos que los pasos para obtener el resultado es pasar por el compilador de consultas, se obtiene un plan de ejecución posteriormente se ejecuta en el módulo de ejecución (execute engine). Para asegurar la consistencia también se tienen las bitácoras y un módulo que se encarga de hacer la recuperación. Además se tiene el manejador de *buffer* que se encarga de dividir la memoria en buffers, que son regiones del tamaño de una página donde se pueden transferir datos al disco duro.

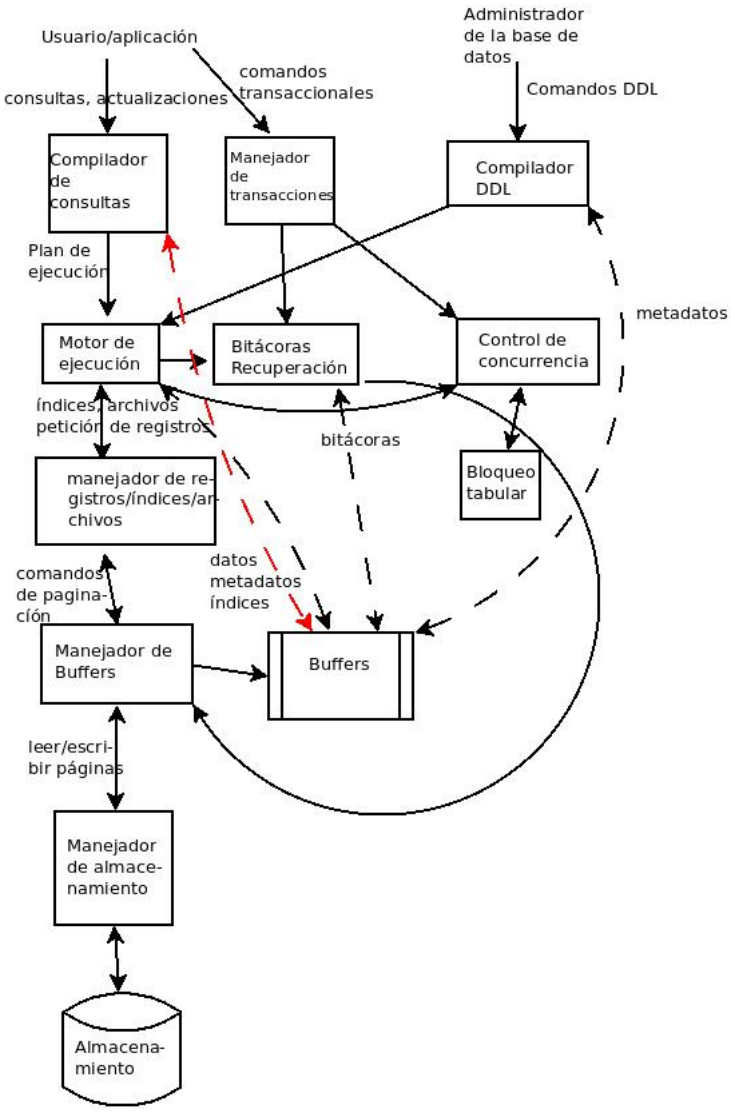

Figura 2.1: Componentes de un SMBD [15]

2.2.1. Propósito de un Sistema Manejador de Bases de Datos

Los Sistemas Manejadores de Bases de Datos (SMBD) surgieron como una respuesta a los primeros métodos computarizados de manejo de datos comerciales. Para ejemplificar estos métodos, típicos de los años 60's, consideremos parte de la organización de una universidad que guarda, además de otros datos, información sobre todos los instructores, estudiantes, departamentos y cursos ofrecidos. Una manera de guardarlo es manteniendo archivos dentro del sistema operativo.
Mantener esta forma de organización tiene una gran cantidad de desventajas, entre las principales están:

- Redundancia e inconsistencia. La misma información puede aparecer en varios archivos. Por ejemplo, si un alumno tiene dos carreras (digamos música y matemáticas) sus datos van a aparecer en el departamento de matemáticas y música. Esta redundancia nos lleva a un almacenamiento mayor y accesos más costosos. Además puede causar inconsistencias en los datos, esto es, puede que no todas las copias coincidan. Por ejemplo, se puede actualizar en el departamento de música pero no en el de matemáticas.

- Dificultades en el acceso a los datos. Supongamos que la universidad necesita los nombres de todos los estudiantes con cierto código postal. Dado que no estaba pensada esta consulta, no existe aplicación que lo haga. Sólo se tiene una aplicación para extraer a todos los estudiantes, por lo cual se tienen dos soluciones: Obtener todos y filtrarlos manualmente o pedirle a un programador escriba una aplicación. Am-

bas soluciones son insatisfactorias dado que si se cambia la consulta, se tendría que escribir un nuevo programa.

- Aislamiento de los datos. Dado que la información se encuentra repartida en varios archivos y los archivos pueden estar en formatos diferentes, escribir nuevos programas para obtener la información.

- Problemas de integridad. Los valores almacenados en la base de datos deben cumplir restricciones de consistencia. Supongamos que la universidad mantiene un balance por cada departamento pero este balance nunca se encuentre por debajo de cero. Los desarrolladores deben agregar código para cumplir con esta restricción. Sin embargo, si se desean agregar restricciones nuevas, es difícil cambiar los programas para que se cumplan.

- Problemas de atomicidad. Un sistema, como cualquier otro dispositivo, es propenso a fallos. En muchas aplicaciones, es crucial que si una falla ocurre, los datos deben ser restaurados a el estado de consistencia que existía antes del fallo. Consideremos un programa para transferir \$500 de la cuenta del departamento A a la cuenta del departamento B. Si ocurre un fallo durante la ejecución del programa, es posible que los \$500 hayan sido retirados de la cuenta del departamento A pero no fue transferido a la cuenta del departamento B, quedando la base de datos en un estado inconsistente. Por esto la transferencia debe ser atómica, esto es, debe suceder completa o no se debe hacer.

- Anomalías de acceso concurrente. Para mejorar el rendimiento del sistema y una respuesta más rápida, muchos sistemas permiten a múltiples usuarios actualizar los datos simultáneamente.

- Problemas de seguridad. No todos los usuarios deben ser capaces de acceder a todos los datos. Por ejemplo, en una universidad la gente de pagaduría necesita sólo acceso a la información financiera. No necesitan acceso a la información académica.

Dentro del SMBD se resuelven los conflictos mencionados previamente. Esto es una ventaja de mantener los datos dentro de SMBD

2.3. Funciones de Agregación

Las *funciones de agregación* son funciones que provee el SMBD donde se toma un conjunto de valores como entrada, los cuales son agrupados bajo cierto criterio para regresar un valor. SQL provee varias funciones, entre las principales podemos mencionar las siguientes:

- *Promedio (avg).*
- *Mínimo (min).*
- *Máximo (min).*
- *Suma (sum).*
- *Conteo (count).*

Las funciones *sum* y *avg* reciben como parámetros de entrada campos numéricos mientras que las demás reciben cualquier tipo como parámetro de entrada. Estas funciones pueden manejar valores nulos. Por ejemplo, avg obtiene el promedio de los valores que no sean nulos o regresa nulo si todos los valores son nulos.

2.4. PostgreSQL

El SMBD *PostgreSQL* [6], originalmente llamado Postgres, fue creado por un profesor de ciencias de la computación de la Universidad de California en Berkley llamado Michael Stonebraker, quien se convirtió en CTO (Chief Technology Officer) en la corporación *Informix*. Postgres se inició en 1986 como proyecto de seguimiento de su predecesor *Ingres*. Ingres, desarrollado desde 1977 a 1985, fue un ejercicio por crear un sistema manejador de base de datos de acuerdo a la teoría clásica de las SMBDR. Postgres, desarrollado entre 1986 - 1994, fue un proyecto para crear nuevas bases en conceptos de bases de datos como exploración de tecnologías "objeto-relacionales".
Durante esos ocho años, Postgres introdujo reglas, procedimientos, tipos extensibles con índices y conceptos objeto relacionales. Postgres fue comercializado posteriormente para convertirse en *Illustra*, posteriormente fue comprado por Informix e integrado a su Servidor Universal (Universal Server).
En 1995, dos estudiantes de doctorado del laboratorio de Stonebraker, Andrew Yu y Jolly Chen, reemplazaron el lenguaje de consulta de Postgres *POSTQUEL* con un subconjunto extendido de SQL. Fue renombrado como *Postgres95*.
En 1996, Postgres95 partió de la academia y empezó una vida en el mundo del código abierto cuando un grupo de desarrolladores vió futuro en este sistema y continuaron con su desarrollo. Este grupo transformó radicalmente Postgres aportando sus habilidades y mucho tiempo. Durante los siguientes ocho años, dieron consistencia y uniformidad al código base, crearon detalladas pruebas de regresión para asegurar la calidad, colocaron listas de correo para reporte de errores (bugs), arreglaron innumerables errores (bugs), agregar funcionalidades y llenando varios espacios como documen-

tación para desarrolladores y usuarios.
El fruto de esta labor fue un nuevo SMBD que obtuvo una reputación muy buena. Con el inicio de esta nueva vida en el mundo del código abierto, con nuevas funcionalidades y mejoras, tomó su nombre actual: *PostgreSQL* ("Postgres" se sigue usando como un sobrenombre fácil de pronunciar).
PostgreSQL empezó en la versión 6, dando crédito a sus años previos de desarrollo. Con la ayuda de cientos de desarrolladores alrededor del mundo, el sistema cambió y mejoró en casi todas las áreas. Durante los siguientes cuatro años, muchas mejoras y nuevas funcionalidades fueron hechas como:

- Control de concurrencia Multiversión (MVCC).
- Importantes mejoras SQL.
- Tipos mejorados (built-in). Nuevos tipos básicos fueron incluidos incluyendo un rango de tipos tiempo/dato y tipos geométricos adicionales.
- Velocidad. Velocidad mayor e incrementar en un orden de 20-40 %, y un inicio den backend y el tiempo de inicio fue decrementado por 80 %.

Ejecuta funciones definidas por el usuario (*UDF*) en muchos lenguajes como Java, Perl, Python, Ruby, C/C++ y su propio lenguaje PL/pgSQL. Incluye algunas funciones que van desde matemática básica y operaciones con cadenas hasta criptografía y compatibilidad con Oracle©. Los disparadores (*triggers*) y las UDF pueden ser escritos en C, cargados a la base de datos como una biblioteca, permitiendo extender la funcionalidad de PostgreSQL. También permite crear tipos de datos propios, así como operadores y funciones para definir el comportamiento [1].

2.4.1. UDF

Una *función definida por el usuario* (*UDF* por sus siglas en inglés) es una forma de poder extender la funcionalidad del Sistema Manejador de Bases de Datos. *PostgreSQL* provee cuatro tipos de funciones:

- Funciones en lenguaje de consulta. Ejecutan una lista arbitraria de sentencias SQL, devolviendo el resultado de la última consulta.

- Funciones en lenguajes procedurales. PostgreSQL permite definir UDF escritas en otros lenguajes distintos a C y SQL, generalmente llamados lenguajes procedurales (PLs). Los lenguajes procedurales no son construidos dentro del servidor de PostgreSQL, son ofrecidos como módulos que pueden ser cargados. Como ejemplo podemos mencionar (**PL/pgSQL PL/Tcl PL/Perl PL/Python**).

- Funciones internas. Son funciones escritas en C que son ligadas de manera estática en el servidor de PostgreSQL.

- Funciones en lenguaje C. Estas funciones pueden ser escritas en C (o un lenguaje compatible con C como C++). Estas funciones son compiladas en objetos dinámicos que pueden ser cargados.

Cada una de estas funciones puede recibir tipos simples, compuestos o una combinación de estos como parámetros. Además, cada tipo de función puede devolver un tipo simple o un tipo compuesto. También se pueden devolver conjuntos de tipo básico o compuesto.

2.4.1.1. Ejemplo UDF

Veamos el ejemplo de una UDF escrita en el lenguaje de consulta SQL. Consideremos la tabla *bank* con los atributos *balance* y *accountno* que corresponde al saldo actual de la cuenta y el número de cuenta respectivamente. Podemos definir la función *tf1* la cual actualizará el saldo de la cuenta, cuyo número es indicado en el primer parámetro, retirando la cantidad indicada en el segundo parámetro. La UDF se definiría de la siguiente manera

```
CREATE FUNCTION tf1 (integer, numeric) RETURNS numeric AS $$
    UPDATE bank
        SET balance = balance - $2
        WHERE accountno = $1;
    RETURNING balance;
$$ LANGUAGE SQL;
```

Figura 2.2: Ejemplo de una UDF [5]

2.4.2. Agregaciones definidas por el usuario

Las agregaciones definidas por el usuario (UDA), al igual que las UDF, son mecanismos para extender la funcionalidad. En el caso de PostgreSQL, las UDA son expresadas en términos de *valores de estado* y *funciones de transición de estado*. Esto es, un operador de agregación usa un valor de estado que es actualizado cada vez que una fila es procesada. Para definir una función de agregación nueva, se debe seleccionar un tipo de datos para el estado inicial, un valor inicial para el estado y una función de transición de estado. La función de transición de estado es una función ordinaria que puede ser usada fuera del contexto de la agregación. En términos simples podemos decir que el valor de estado es el valor actual acumulado de las

filas procesadas, la función de transición de estado es la que se encarga de procesar el valor actual de la fila y el valor de estado.

2.4.2.1. Ejemplo UDA

En la Figura 2.4.2.1 observamos un ejemplo de la creación de una UDA [2] Suponemos que tenemos el tipo *complex* ya definido. Este tipo correspondería a un número complejo, es decir, tiene un campo r y un campo i que corresponde a la parte real y a la parte imaginaria de un número complejo, respectivamente. Además tenemos definida la función *complex_add* la cual suma dos números de tipo complex de la forma usual, es decir, si se tiene un número **(a,b)** y otro **(c,d)** devuelve otro número complex **(a+c,b+d)**. Definiremos la función sum que será una UDA, la cual obtendrá la suma de todos los números complejos almacenadas en una columna de la tabla.
Supongamos que la tabla *test_complex* contiene los siguientes valores, cada uno como un registro de la tabla, (1,1),(20,13.6),(5,20),(3,9),(5,10.3).

```
CREATE AGGREGATE sum (complex)
(
    sfunc = complex_add,
    stype = complex,
    initcond = '(0,0)'
);
```

La función *sum* toma la columna y mediante la función *complex_add* suma el acumulado de la función, cuyo valor inicial es *initcond*, con el valor actual de *val*.

Consideremos la tabla *test_complex* con una única columna *val* de tipo

complex.

```
SELECT sum(val) FROM test_complex;

   sum
-----------
 (34,53.9)
```

Figura 2.3: Ejemplo de una UDA [2].

En este ejemplo al ejecutar *sum* sobre la columna *val*, se suman todos los valores de la columna y la condición inicial. Por lo general la condición inicial será un valor neutro que no afecte el acumulado al aplicar la función de transición. Cabe mencionar que si todos los valores son nulos regresa la condición inicial, es decir, (0,0).

2.5. Preprocesamiento y Minería de Datos

Han [14] define la minería de datos como extraer o "minar" conocimiento de grandes cantidades de datos. Algunas personas manejan la minería de datos como un paso en el proceso de Descubrimiento de Conocimiento.
El proceso de Descubrimiento de Conocimiento se puede resumir en siete pasos que se hacen de manera iterativa. Estos pasos son:

- *Limpieza de datos*. Los datos del mundo real tienden a estar incompletos, con ruido (fuera de rango) o inconsistentes, en este paso se definen rutinas que llenen los valores faltantes, eliminen el ruido identificando valores fuera de rango y corrijan inconsistencias en la base de datos.
- *Integración de los datos*. La integración de datos consiste en combinar

datos de múltiples fuentes en un almacenamiento coherente de datos, como un almacén de datos. Estas fuentes pueden incluir múltiples bases de datos o cubos de datos.

- *Selección de los datos.* Trabajar sobre todo el conjunto de datos puede ser muy costoso, debido a la cantidad de accesos a disco y el tamaño de la tabla. Por lo cual se obtienen datos relevantes de la base de datos para el análisis

- *Transformación de los datos.* En este paso necesitamos transformar o consolidar los datos a formas apropiadas para minarlos. Por ejemplo, realizando agregaciones u operaciones de sumarización.

- *Minería de datos.* Un proceso esencial donde son aplicados métodos para extraer patrones útiles y novedosos de los datos.

- *Evaluación de patrones.* Identificar los patrones interesantes y útiles que representan conocimiento basados en medidas de interés.

- *Presentación de los datos.* Mediante técnicas de representación y visualización el conocimiento minado es presentado al usuario.

Este proceso puede ser cíclico debido a que es posible en algún paso regresar a un paso previo. Por ejemplo, al realizar la integración podría ser necesaria hacer una limpieza de datos. En el sector empresarial y algunos otras áreas, se populariza el término minería de datos como el proceso completo.
Un punto muy importante es suponer que los datos nos llegan “listos” para realizar la reducción de dimensiones. Es decir, nos estamos enfocando en el paso de selección de datos. La idea del presente trabajo es hacer eficiente el manejo de matrices y vectores necesarios para el paso de selección de datos.

Uno de los métodos más utilizados para la selección de datos es el análisis de componentes principales, cuyo apoyo es la descomposición singular de valores, la cual explicaremos en la siguiente sección.

2.5.1. Descomposición en valores singulares (SVD)

2.5.1.1. Definición

La descomposición en valores singulares (SVD, por sus siglas en inglés) proporciona una gran cantidad de información sobre las matrices. Además "da un orden" a la información contenida en la matriz, por lo cual la "parte relevante" se hace notoria.

Teorema. SVD. Sea A una matriz arbitraria de $m \times n$ con $m \geq n$. Entonces puede ser escrita $A = U\Sigma V^T$, donde U es una matriz de $m \times m$ y satisface $U^T U = I$, donde I es la matriz identidad, V es una matriz de $n \times n$ que satisface

- $V^T V = I$

- $\Sigma = diag(\sigma_1, ..., \sigma_n)$ donde $\sigma_1 \geq ... \geq \sigma_n \geq 0$. Es decir, Σ es una matriz diagonal.

Las columnas $u_1, ..., u_n$ de U son llamados *vectores singulares izquierdos.*
Las columnas $v_1, ..., v_n$ de V son llamados *vectores singulares derechos.*
Los valores σ_i son llamados *valores singulares.*
Nota: Si $m \leq n$ en el teorema anterior, SVD se define considerando A^T

Una interpretación geométrica de este teorema es la siguiente:

Dada cualquier matriz A de $m \times n$, pensada como un mapeo de un vector $x \in \mathbb{R}^n$ a un vector $y = Ax \in \mathbb{R}^m$. Podemos elegir un sistema de coordenadas ortogonales para $\mathbb{R}^n$ (las unidades de los ejes son las columnas de V) y otro sistema de coordenados ortogonales para $\mathbb{R}^m$ (las unidades de los ejes son las columnas de U) tal que A es diagonal (Σ); es decir, mapea un vector $x = \sum_{i=1}^{n} \beta_i v_i$ al vector $y = Ax = \sum_{i=1}^{n} \sigma_i \beta_i u_i$

2.5.1.2. Ejemplo SVD en GNU Octave

Para ilustrar SVD usaremos *GNU Octave*, una versión libre de MATLAB. Además haremos uso de la función *svd* incluida dentro de este software. Consideremos la matriz

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$$

Ejecutamos la función *svd* de *GNU Octave*, la cual nos dará una descomposición en valores singulares, es decir los valores de U,S,V tal que $A = U * S * V^t$

octave:3>[U,S,V]=svd(A)

$$U = \begin{pmatrix} -0{,}219529 & 0{,}807346 & 0{,}023607 & 0{,}547214 \\ -0{,}383342 & 0{,}391214 & -0{,}439345 & -0{,}712023 \\ -0{,}547155 & -0{,}024917 & 0{,}807869 & -0{,}217595 \\ -0{,}710969 & -0{,}441048 & -0{,}392131 & 0{,}382405 \end{pmatrix}$$

$$S = \begin{pmatrix} 5{,}77938 & 0 \\ 0 & 0{,}77381 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$V = \begin{pmatrix} -0{,}32201 & 0{,}94674 \\ -0{,}94674 & -0{,}32201 \end{pmatrix}$$

Ahora podemos corroborar que estas matrices, U,S,V cumplen con A=$U*S*V^T$

octave:5>U*S*(V.')

$$ans = \begin{pmatrix} 1{,}00000 & 1{,}00000 \\ 1{,}00000 & 2{,}00000 \\ 1{,}00000 & 3{,}00000 \\ 1{,}00000 & 4{,}00000 \end{pmatrix}$$

Nota: V.' es la notación de Octave para V^T

El procedimiento para obtener las matrices está fuera del alcance de la pre-

sente tesis. El tema puede ser consultado en literatura del área de algebra lineal [20].

En la siguiente sección presentaremos dos aplicaciones de la descomposición en valores singulares, una de ellas la Matriz Inversa Moore-Penrose, la cual es muy utilizada en álgebra lineal. La segunda aplicación es la compresión de imágenes, bastante útil debido a que actualmente se tienen imágenes de gran tamaño, por lo cual se desea representar la imagen original sin que se pierda lo que se ve en la imagen, aunque la calidad de la imagen no sea la misma y algunos detalles se pueden perder pero no la imagen por completo. El objetivo de mostrar estas aplicaciones es observar la descomposición en valores singulares en diversas áreas como el álgebra lineal o la visión computacional. Es decir, es un método no sólo utilizado en minería de datos o en un ambiente matemático sino en otras áreas.

2.5.2. Aplicaciones

2.5.2.1. *Matriz PseudoInversa Moore-Penrose*

Dada una matriz B de $m \times n$, la matriz pseudoinversa[1] Moore-Penrose [11] B^+ es una matriz que satisface:

- $BB^+B = B$
- $B^+BB^+ = B^+$
- $(BB^+)^H = BB^+$

[1]Se denomina pseudoinversa debido a que está definida sobre los números complejos y no cumple, de manera estricta, con la definición de una matriz inversa.

- $(B^+B)^H = B^+B$

Donde B^H se define como la matriz conjugada transpuesta.

También se cumple que $z = B^+c$ es la longitud más corta al problema de mínimos cuadrados $Bz = c$. Si queremos ver la solución de esta ecuación observemos lo siguiente. Si la inversa de B^HB existe entonces se cumple que $B^+ = (B^HB)^{-1}B^H$. Podemos premultiplicar ambos lados de la ecuación anterior por B^H obtenemos $B^HBz = B^Hc$. Entonces, si premultiplicamos ambos lados de la ecuación por $(B^HB)^{-1}$ tenemos que $z = (B^HB)^{-1}B^Hc = B^+c$

El cálculo de la matriz B^+ se puede hacer mediante el uso de SVD. Si tenemos una descomposición en valores singulares $M = U\Sigma V^*$ entonces $M^+ = V\Sigma^+U^*$, donde Σ^+ es la pseudoinversa de Σ, reemplazando por el recíproco cada entrada distinta de cero y obteniendo la transpuesta de la matriz resultante

2.5.2.2. Aproximación de bajo rango

Consideremos una matriz $A \in R^{m\times n}$, con SVD dado por el siguiente resultado [10]:

$$A = USV^T, S = diag(\sigma_1, ..., \sigma_r, 0, ..., 0)$$

donde los valores singulares están ordenados en orden decreciente,
$\sigma_1 \geq ... \geq \sigma_r > 0$. En muchas aplicaciones, nos puede ser muy útil aproximar A con una matriz de bajo rango. En la siguiente sección muestro una aplicación a esta aproximación de bajo rango.

Ejemplo: Asumamos que A contiene la bitácora resultante de n aseveraciones sobre m periodos de tiempo, entonces cada columna de A es una serie de tiempo para cada aseveración particular. Aproximando A por una matriz de rango uno de la forma pq^T, con $p \in R^m$ y $q \in R^n$ sostiene al modelo de movimientos de las aseveraciones como todos siguen el mismo patrón dado por el perfil de tiempo p, cada movimiento de la aseveración siendo escalado por los componentes en q. Entonces, el componente (t, i) componente de A, que es la bitácora de regreso de la aseveración i al tiempo t, expresado como $p(t)q(i)$.

Consideramos el problema de aproximación de bajo rango como:
$min_x||A - X||_F : rank(X) = k$ para una $k(1 \leq k < r = rank(A))$ dada

Aproximación de bajo rango
Una mejor aproximación de rango k A_k está dada haciendo cero los $r - k$ valores singulares de A, esto es;

$$A_k = US_kV^T, S_k = diag(\sigma_1, ..., \sigma_k, 0, ..., 0)$$

El error mínimo está dado por la norma Euclidiana de los valores singulares que se han hecho cero en el proceso:

$$||A - A_k||_F = \sqrt{\sigma_{k+1}^2 + ... + \sigma_r^2}$$

2.5.2.3. Compresión de Imágenes

Podemos representar las imágenes como matrices. Usando SVD podemos representar la imagen con menor calidad, es decir con una matriz de menor dimensión, teniendo así que aún se distingue la misma imagen. Consideremos una imagen con $n \times m$ pixeles. Para el caso de una imagen en blanco y negro, sólo necesitamos un bit por pixel. Para el caso en color, necesitamos tres números para cada color: rojo, verde y azul (RGB por los nombres en inglés red, blue, green). Cada color puede ser como una matriz de $n \times m$ y podemos representar todos los colores como una matriz de $n \times 3m$, donde guardamos cada color en una matriz columna sobre cada entrada, por ejemplo A=[A_rojo,A_verde,A_azul].

2.5.2.4. Ejemplo de compresión de imágenes

Para ejemplificar la compresión de imágenes, veremos cómo se realiza en *GNU Octave* [9]. Supondremos que el archivo es llamado *baboon-grayscale.png*
$>>$ A = imread(baboon-grayscale.png); % carga la imagen en la matriz
$>>$ A = double(A); % transforma a una matriz real
$>>$ A = A(:,:,1); % para imágenes en escala de grises, sólo consideramos la primera matriz del arreglo.
$>>$ [U,S,V] = svd(A);
$>>$ Ak = U(:,1:k)*S(1:k,1:k)*V(:,1:k)';

En la Figura 2.4 se observa la imagen original con un rango $k = 298$. Ahora, si observamos la Figura 2.5 notamos que el tamaño $k = 50$ aproxima de una buena manera a la imagen original, cuyo rango es $r = m = n = 298$.

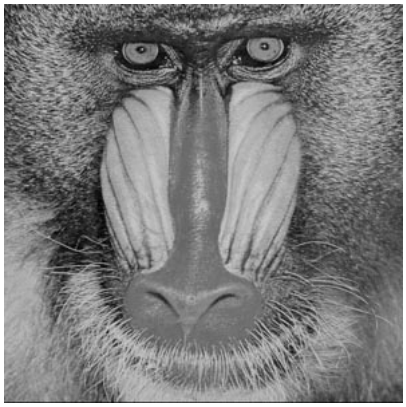

Figura 2.4: Esta imagen puede ser representada por una matriz de 298x298

2.5.2.5. Relación con Análisis de Componentes Principales

Para el Análisis de Componentes Principales se utiliza la matriz de covarianza de los datos, que es proporcional a AA^T y los conjuntos de direcciones principales serán los eigenvectores de la matriz (simétrica). Como se vió previamente, los eigenvectores AA^T son simplemente los vectores singulares por la izquierda de A. Entonces, ambos métodos, la aproximación de bajo rango y PCA, recaen sobre la misma herramienta, SVD. Éste último es una aproximación más completa dado que, además, provee los eigenvectores de A^TA, que pueden ser útiles si queremos analizar los datos en términos de las filas en lugar de las columnas.

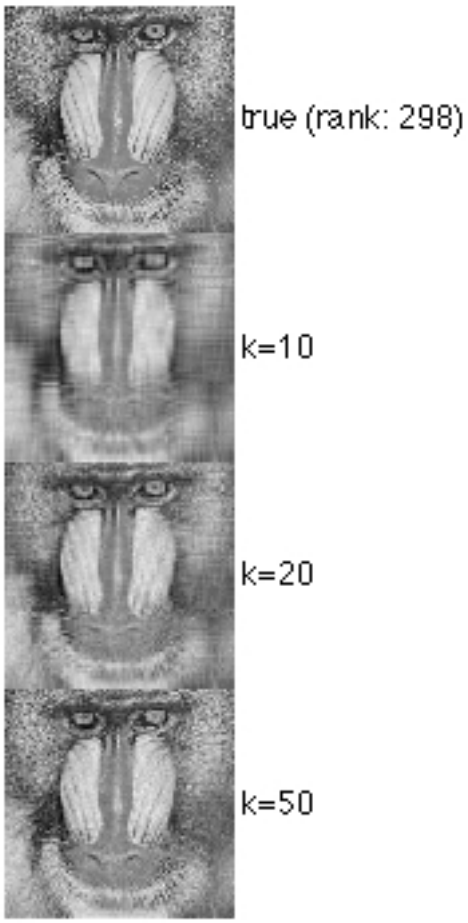

Figura 2.5: La imagen original y las imágenes aproximadas varían de rango.

2.5.3. Reducción de Dimensiones

La idea principal de la reducción de dimensiones es codificar los datos o transformarlos para obtener una reducción o una representación "comprimida" de los datos originales. Si el conjunto original de datos puede ser reconstruido con los datos comprimidos sin pérdida de información, entonces la reducción es llamada *sin pérdida*. A continuación presentaremos *Análisis de componentes principales*.

2.5.3.1. Análisis de componentes principales

El análisis de componentes principales [14] es una técnica muy utilizada para reducción de columnas en minería de datos y estadística.
Supongamos que los datos a reducir son tuplas o vectores de datos descritos por n atributos o dimensiones. El *análisis de componentes principales* (*PCA*, por sus siglas en inglés) busca k vectores n-dimensionales ortogonales que puedan ser usados para representar los datos de una buena manera, donde $k \leq n$. A diferencia de la selección de atributos, que reduce el conjunto de atributos a un subconjunto del conjunto inicial, *PCA* "combina" la esencia de los atributos creando un conjunto alternativo de variables más pequeño. Los datos iniciales pueden ser proyectados a este nuevo conjunto. *PCA* muchas veces revela relaciones que no se sospechaban y, por lo tanto, permite interpretaciones que podrían no resultar de manera inmediata.
El procedimiento básico es:

- La entrada es normalizada, para que cada atributo caiga en el mismo rango. Este paso ayuda a asegurar que los atributos con un dominio numérico grande no ganarán sobre los que están en dominios pequeños.

- *PCA* calcula k vectores ortonormales que proveen una base ortonormal para la entrada normalizada. Estos vectores se denominan como *componentes principales*. Los datos de entrada son una combinación de los *componentes principales*.

- Los componentes principales son ordenados de forma decreciente, respecto a la "relevancia". Los componentes principales esencialmente sirven como un nuevo sistema de ejes para los datos, dando información importante sobre la variación. Esto significa que, los ejes coordenados ordenados son tal que el primer eje coordenado muestra la mayor variación sobre los datos, el segundo eje muestra la siguiente mayor variación y así sucesivamente. Por ejemplo, en la Figura 2.6 muestra dos componentes principales, $Y1$ y $Y2$, para el conjunto de datos originalmente mapeado a los ejes $X1$ y $X2$. Esta información ayuda a identificar grupos o patrones con los datos.

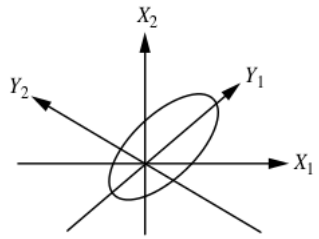

Figura 2.6: Rotación de ejes en dos componentes principales $Y1, Y2$ respecto al conjunto original $X1, X2$

- Debido a que los componentes se encuentran ordenados en un orden decreciente de "relevancia", el tamaño de los datos puede ser reducido eliminando los componentes más débiles, esto es, aquellos con la varia-

ción más baja. Usando los componentes principales más fuertes, debe ser posible reconstruir una buena aproximación de los datos originales.

Existen diversos campos donde se puede aplicar la descomposición en valores singulares, por ejemplo:

- Compresión de datos.
- Procesamiento de imágenes.
- Reconocimiento de patrones.
- Agrupación (Clustering).
- Clasificación.

PCA puede ser aplicado a atributos ordenados o desordenados y puede manejar datos dispersos o datos cercanos.

2.6. Bibliotecas numéricas

Existen diversas bibliotecas para realizar cálculos matriciales. La decisión sobre la mejor biblioteca a utilizar depende de diversos factores, por ejemplo:

- ¿Qué lenguaje se utilizará?
- ¿Qué tipo de cálculos se realizarán?
- ¿Sabemos algo sobre los datos?

Estos factores nos permiten elegir la biblioteca adecuada para nuestra fin. Por ejemplo, si una matriz tiene casi todos sus valores distintos de cero, llamada matriz densa, nos convendría utilizar una biblioteca optimizada para esto.

Algunos ejemplos de bibliotecas numéricas se mencionan a continuación según el lenguaje que se esté utilizando:

- Multilenguaje
  - *ALGLIB* es una biblioteca de análisis numérico de código abierto que puede ser usada desde C++, C#, FreePascal, Delphi, VBA.
  - *IMSL* son bibliotecas de análisis numérico implementadas en lenguajes estándar de programación como C, Java, C#, .NET, Fortran y Python.
  - *NAG* es una colección de rutinas matemáticas y estadísticas para múltiples lenguajes de programación (C, C++,Fortran, Visual Basic, Java y C#) y paquetes (Matlab, Excel, R, LabVIEW).
  - *GNU Octave* es lenguaje de programación de alto nivel y bibliotecas de código abierto, incluyendo una línea de comandos, API's, funciones y bibliotecas pueden ser llamados desde diversas plataformas.
- C
  - *AmgX* es una biblioteca comercial de rutinas diseñada para correr en GPU's Nvidia usando CUDA.
  - *BLOPEX Solucionadores de eigenvalores precondicionados localmente óptimos por bloques* es una biblioteca de código abierto

para la solución escalable (paralela) de los problemas de eigenvalores. Su diseño orientado a objetos permite fácil portabilidad.

- *FFTW (Fastest Fourier Transform in the West)* es una biblioteca para calcular transformadas de Fourier y relacionadas.
- *GNU Scientific Library* una popular y libre biblioteca de análisis numérico implementada en C.
- *GNU Multi-Precision* es una biblioteca para realizar aritmética de precisión arbitraria.
- *LIS* es una biblioteca paralela escalable para resolver sistemas de ecuaciones y problemas estándar de eigenvalores con matrices reales ralas(sparse) usando métodos iterativos.
- *PETSc Caja de Herramientas portable y extensible para cálculos científicos* es un conjunto de estructuras de datos y rutinas para la solución de aplicaciones científicas modeladas por ecuaciones diferenciales parciales.
- *SLEPc. Biblioteca Escalable para cálculos del problema de eigenvalores* es una biblioteca de código abierto basada en PETSc para la solución escalable de problemas de eigenvalores.

- C++
  - *Armadillo* en una biblioteca de álgebra lineal, que mantiene un buen balance entre velocidad y facilidad de uso. Utiliza clases plantilla y tiene la opción de ligar a BLAS y LAPACK.
  - *BLITZ++* es una biblioteca de alto rendimiento para matemáticas vectoriales en C++
  - *Boost* bibliotecas de C++ para cálculos numéricos.

- *Eigen* es una biblioteca de matemáticas vectoriales.
- *GNU Scientific Library (GSL)* es una biblioteca numérica para C y C++. Provee un gran rango de rutinas matemáticas tales como generadores de números aleatorios.
- *LAPACK++* es un wrapper para LAPACK y BLAS.
- *MTL4* es un conjunto de plantillas que proveen funcionalidad para BLAS para ralas y densas.

- .NET Frameworks
  - *FinMath* es una biblioteca para cálculos científicos y financieros en el framework .NET. Su funcionalidad incluye álgebra matricial y vectorial, estadística, análisis de datos, funciones avanzadas, procesamiento de señales, optimizaciones numéricas entre otras. Usa Intel Math Kernel Library e Intel Integrated Performance Primitives para hacer la mayoría de los cálculos de bajo nivel mientras provee una interfaz orientado a objetos.

- Fortran
  - *BLAS Subrutinas Básicas de Álgebra Lineal* es una interfaz de programación estándar para publicar bibliotecas para realizar operaciones de álgebra lineal básica como multiplicación de vectores o matrices.
  - *EISPACK* es una biblioteca para hacer cálculos numéricos de eigenvalores y eigenvectores de matrices, escrito en FORTRAN. Contiene subrutinas para calcular los eigenvalores de nueve clases de matrices: complejas generales, complejas hermitianas, real

general, real simétrica, real simétrica banda (banded) , simétricas reales tridiagonales, real tridiagonal especial, generalizadas reales y generalizadas reales simétricas.

- *LAPACK Biblioteca de Álgebra Lineal (LAPACK, por sus siglas en inglés)* es una biblioteca para cálculos numéricos originalmente escrito en Fortran 77 y ahora escrita en Fortran 90.
- *LINPACK* es una biblioteca para realizar álgebra lineal numérica en computadoras digitales. Fue escrito por Jack Dongarra, Jim Bunch, Cleve Moler y Pete Stewart y fue pensado para ser usada en supercomputadoras de 1970's y posteriormente 1980's. Ha sido superada por LAPACK, que es más eficiente en arquitecturas modernas.
- *MINPACK* es una biblioteca de subrutinas de FORTRAN para resolver sistemas de ecuaciones no lineales o el problema de mínimos cuadrados del residuo de un conjunto de ecuaciones lineales o no.

- Java
    - *EJML Biblioteca Eficiente de Matrices de Java* es una biblioteca de álgebra lineal de código abierto para manipulación de matrices densas.
    - *Jblas*: Álgebra lineal para Java, una biblioteca de álgebra lineal que es un wrapper de *BLAS* y *LAPACK*
    - *Biblioteca Universal de Matrices para Java (UJM, por sus siglas en inglés)* Provee implementaciones para matrices densas y ralas, así como cálculos de álgebra lineal como descomposición de matrices, inversas, multiplicación, media, correlación,desviación estándar, etc.

En la siguiente sección se presentará *LAPACK*. El motivo de utilizar *LAPACK* fue ser una biblioteca muy utilizada como base en lenguajes estadísticos, por ejemplo R, además que su objetivo es realizar cálculos matriciales de manera eficiente y contiene rutinas para diversos tipos de matrices, lo cual hace que sea una buena opción para poder ampliar el conjunto de operaciones que se podrán calcular con la integración deseada.

2.7. LAPACK

La biblioteca *LAPACK* está escrita en Fortran y provee rutinas para resolver diversos problemas como sistema de ecuaciones lineales, eigenvalores y valores singulares, entre otros problemas relacionados con matrices. Las rutinas están escritas para llamar a otros programas, los cuales son englobados como *BLAS* [8].

LAPACK contiene muchas implementaciones para diversos fines. Por ejemplo, *SCALAPACK* está optimizada para máquinas paralelas con memoria distribuida. Para estos experimentos se utilizaran dos implementaciones: *CLAPACK* e *Intel ® MKL*

A continuación se presentan las bibliotecas CLAPACK e Intel® MKL. Se decidió usar estas dos implementaciones debido a que las UDF de PostgreSQL permiten ciertos lenguajes particulares, por lo cual tenía que haber una consistencia entre estos dos lenguajes. Ambas implementaciones están escritas en C, por lo cual es conveniente para la estrategia aquí presentada.

2.7.1. CLAPACK

Esta implementación consiste en transformar el código original de *LAPACK*, escrita en *FORTRAN* en *ANSI C*. La transformación es realizada por el traductor Fortran-to-C *f2c*, el cual fue escrito por David Gray en los Laboratorios Bell. Adicionalmente, se hace una limpieza para hacerlo más fácil de comprender. El código se encuentra disponible en la página oficial [8].

2.7.2. Intel® MKL

Intel® Math Kernel Library incluye un conjunto de rutinas optimizadas por Intel® para acelerar el rendimiento de las aplicaciones. Intel® MKL incluye álgebra lineal, Transformadas Rápidas de Fourier, operaciones sobre vectores y funciones estadísticas. El programa se encuentra disponible en la página oficial [7]

En la siguiente sección se presenta con mayor detalle una parte extra de la estrategia presentada. El objetivo de introducir este factor fue verificar la utilización de la memoria caché por parte de *LAPACK* en sus dos implementaciones utilizadas para esta experimentación. Además, este es un factor que afecta en el rendimiento de nuestros experimentos.

2.8. Inhibición de caché

Para poder realizar la inhibición de caché, se utilizó el documento [17] Las arquitecturas Intel 64 y IA-32 proveen registros de control y bits para uso de la activación o restricción del cacheo para varias páginas o regiones de memoria. Entre ellos los dos principales y los que fueron usados para el propósito de inhibir el caché.

- **bandera CD, bit 30 del control de registro CR0**. Si esta bandera está apagada, el cacheo está activado para toda la memoria, pero puede estar restringido para páginas individuales o regiones de memoria individuales por otros mecanismos de control de caché. Cuando la bandera está encendida, el cacheo es restringido en los procesadores de los cachés. Los caches deben ser explícitamente liberados (*flushed*) para asegurar la consistencia de la memoria. Para un rendimiento mayor de los procesadores, ambas banderas (CD yNW) en el registro de control CR0 deben ser apagados.
- **bandera NW, bit 29 del registro de control CR0**. Controla la política de escritura para las localidades de memoria. Si las banderas NW y CD están apagadas, write-back está activada para toda la memoria, pero puede ser restringido para páginas individuales o regiones

de memoria por otros mecanismos de control de caché.

Algunos notas son:

- Para los procesadores Pentium IV e Intel Xeon, la bandera *NW* es una bandera que no importa; esto es, cuando la bandera *CD* está activa, el procesador usa el modo de no-llenado (no-fill) del caché en lugar de las opciones de la bandera *NW*.
- Para los procesadores Intel Atom, la bandera *NW* es una bandera que no importa.
- Para los procesadores Pentium, cuando la caché L1 es deshabilitado (las banderas *CD* y *NW* en el registro de control CR0 están activas).

Del estudio realizado, se llegó a la conclusión que para inhibir el caché se tiene que realizar en varios pasos:

- Deshabilitar el bit 30 que indica si existe memoria caché disponible.
- Para evitar la pérdida de información, se debe vaciar el caché
- También se debe deshabilitar otro bit (29) para inhabilitar la escritura.

En el siguiente capítulo se hará una discusión más profunda sobre la propuesta de esta tesis. También se especifican las condiciones en que se desarrolló la experimentación así como capturas de pantalla de algunos resultados obtenidos.

Capítulo 3

Propuesta y experimentación

En el presente capítulo se muestran los códigos desarrollados para la experimentación así como los resultados obtenidos de la misma, además de mostrar los pasos necesarios para crear el ambiente necesario para los experimentos.

3.1. Cálculo de la descomposición en valores singulares

Un SMBD provee mecanismos para extender la funcionalidad que ya viene incluida dentro de éste. Las UDFs son un ejemplo de este mecanismo. Recordemos que el objetivo de esta tesis es proponer una estrategia para realizar cálculos de álgebra lineal, como multiplicación de matrices, cálculos

que son ampliamente utilizados en modelos, como PCA, SVD, Stocastich Search Variable Selection(SSVS) , para realizar minado de datos, de manera eficiente dentro de un sistema manejador de bases de datos (SMBD). SQL se encuentra limitado en este sentido ya que realizar operaciones matriciales no se encuentra dentro de las funcionalidades originales pensadas para este lenguaje debido a que su paradigma es relacional.

Para este fin, en nuestra propuesta se aprovecha la biblioteca de álgebra lineal *LAPACK*, la cual se encuentra optimizada para realizar este tipo de operaciones. Además, se explota el mecanismo de UDF para poder hacer llamadas a rutinas externas.

La propuesta para cumplir con una implementación de nuestro objetivo, es incorporar las rutinas de LAPACK dentro del SMBD PostgreSQL mediante UDF.

Observemos que este mecanismo es útil dado que estamos proponiendo utilizar una biblioteca de álgebra lineal para resolver el problema planteado, así no es necesario desarrollar nuevas rutinas para resolverlo, sino que delegamos el trabajo a rutinas ya optimizadas. Nuestro objetivo no es implementar nuevas rutinas sino proveer una manera que nos permita ejecutar las rutinas ya existentes dentro del SMBD.

Esta propuesta tiene las siguiente ventajas respecto a otros métodos:

- Ahorramos cierto tiempo del proceso de extracción, transformación y carga (ETL, por sus siglas en inglés) debido a que no necesitamos construir todas las tablas que requeriría todo el proceso ETL debido a que lo estamos manejando directamente.

- Conservar las propiedades que provee el SMBD como consistencia, integridad, entre otras, dado que no exponemos los datos fuera del

SMBD.

- Esta propuesta nos podría facilitar un módulo de análisis que algunos sistemas ya proveen dentro del SMBD pero, estos módulos, tienen un alto costo.
- Evitamos tiempo al no tener que hacer un respaldo de las bases de datos para explotarlo en un programa externo.

Es importante mencionar también las desventajas de esta propuesta.

- Se está aumentando la carga de trabajo del SMBD lo cual implica un mayor poder de procesamiento y un mayor uso de recursos por parte del SMBD.
- Podría afectar el rendimiento de otras transacciones dentro del SMBD si el análisis se ejecuta en forma simultánea con esas transacciones.

3.2. Experimentos

En esta sección se presentarán dos experimentos:

- Integración de *LAPACK*, en su versión *CLAPACK* e *Intel* ® *MKL*, en PostgreSQL.
- Inhibición de cache para comprobar optimizaciones.

El equipo donde se realizaron los experimentos cuenta con las siguientes características:

- Sistema operativo **Ubuntu 11.10**.

- Sistema manejador de bases de datos **PostgreSQL versión 9.1**.
- Disco Duro con capacidad de **120 GB**.
- Memoria RAM con capacidad de **2 GB**.
- **Memoria caché L1 y L2**.
- **Arquitectura de 32 bits**. También es posible hacerlo en una máquina de 64 bits ya que los componentes son soportados en ambas arquitecturas.

El software necesario para realizar las pruebas es:

- *CLAPACK versión 3.2.1.* Es necesario hacer un procedimiento previo para generar las bibliotecas. En la siguiente sección se muestra este procedimiento.
- *compilador gcc versión 4.3.2*
- *Intel® MKL*
- *compilador de Intel ® icc.* Aunque lo recomendable es tener *Intel® Parallel Studio XE 2013 SP1*, éste ya incluye MKL y el compilador icc

3.2.1. Compilación de bibliotecas a partir de CLAPACK

El procedimiento para compilar las bibliotecas es el siguiente: se descarga el archivo de *clapack.tgz*, disponible en el portal de Netlib [3], después se descomprime y modificamos el archivo *make.inc.example* de la siguiente manera:

- Renombramos *make.inc.example* como *make.inc*
- Modificamos, dentro del archivo *make.inc*, la opción **cc= gcc -fPIC** y **LOADER= gcc -fPIC**
- Desde la ruta donde está el archivo *make.inc* y *Makefile*, se ejecuta make all.

La opción **-fPIC** (Position Independent Code) genera código independiente de la posición, adecuado para el ligado dinámico y evitando cualquier límite en el tamaño de la tabla de offset global (*global offset table*)

Creamos una tabla de n columnas, las cuales deben cumplir con:

- llamada *experimenton* (sustituyendo la última n por el número de columnas)
- las columnas son llamadas $x1$,$x2$,...,xn (se puede sustituir x por algún otro nombre pero deben cumplir con tener el sufijo 1,2,...,n)

La función siguiente se encarga de convertir la matriz a_{ij} a una cadena de la forma
$\{\{a[0][0], a[0][1], ..., a[0][n-1]\}, \{a[1][0], a[1][1], ..., a[1][n-1]\}, ...,$
$\{a[n-1][0], a[n-1][1], ..., a[n-1][n-1]\}\}$

execq.c

```
1 #include "postgres.h"
2 #include "executor/spi.h"
3 #include "utils/builtins.h"
```

```
4  #ifdef PG_MODULE_MAGIC
5  PG_MODULE_MAGIC;
6  #endif
7  Datum execq(text *sql, int cnt);
8  Datum execq(text *sql, int cnt)
9  {
10     StringInfo  result,result1;
11     char *command;
12     int ret;
13     int proc;
14     command = text_to_cstring(sql);
15     SPI_connect();
16     //0 indica que se toman todos (de acuerdo a la
          documentacion)
17     ret = SPI_exec(command, 0);
18     proc = SPI_processed;
19     text* array;
20     if (ret > 0 && SPI_tuptable != NULL)
21     {
22         TupleDesc tupdesc = SPI_tuptable->tupdesc;
23         SPITupleTable *tuptable = SPI_tuptable;
24         int i, j,x,y;
25                 float val[3];
26                 float corr[cnt*cnt];
27                 result = makeStringInfo();
28                 appendStringInfo(result, "{");
29         for (j = 0; j < proc; j++)
30         {
31             HeapTuple tuple = tuptable->vals[j];
32             for (i = 1; i <= tupdesc->natts; i++){
33                                 val[i-1]=atof(
```

```
                                        SPI_getvalue(
                                        tuple, tupdesc,
                                        i));
34                          appendStringInfo(result, "
                               %s%s",
35                  SPI_getvalue(tuple, tupdesc, i),(j
                       >= proc) ? "" : ",");
36                  }//end for i
37              x=(int)val[0];
38              y=(int)val[1];
39              corr[calculaPos(x,y,cnt)]=val[2];
40              corr[calculaPos(y,x,cnt)]=val[2];
41          }//end for j
42                  result1=makeStringInfo();
43                  appendStringInfo(result1, "{");
44                  for(i=0;i<cnt*cnt;i++){
45                          if(i==cnt*cnt-1){
46                                  appendStringInfo(
                                        result1,"%lf",
                                        corr[i]);
47                          }else{
48                                  appendStringInfo(
                                        result1,"%lf,",
                                        corr[i]);
49                          }
50                  }//end for i
51          appendStringInfo(result1, "}");
52          appendStringInfo(result, "}");
53          array=(text *) cstring_to_text_with_len(
                result1->data, result1->len);
54      }//end if
```

```
55      SPI_finish();
56      pfree(command);
57      PG_RETURN_TEXT_P(array);
58  }
59
60  int calculaPos(int i, int j, int n) {
61          //esta formula se obtiene de transformar
                una matriz de nxn a un vector de nxn
62          return ((i-1)*n+(j-1));
63  }
```

Los parámetros de entrada de la función **execq** son:

- text *sql. Este parámetro recibe la consulta para obtener las filas de la tabla de correlación. Se decidió recibir de esta forma debido al nombre de la tabla. Es importante mencionar que la tabla de correlación tiene las columnas i,j,$corr(i,j)$ y en ese orden. No es necesario que en la tabla de correlación se encuentren todos los valores ya que, al ser una matriz simétrica, se pueden obtener mediante la entrada j,i suponiendo se tiene la entrada i,j. Es decir, se puede tener los valores de la matriz triangular superior o inferior y a partir de ahí obtener los valores de las entradas inferiores o superiores respectivamente.

- int cnt. El número de filas de la matriz de correlación. Dado que suponemos que es una matriz cuadrada, el número de columnas es igual al de las filas.

La salida de esta función es una cadena que representa la matriz de correlación. Supongamos que la matriz de correlación se llama correlación y posee entradas.

En la línea 17 se ejecuta el comando, obteniendo los datos de la matriz. En el ciclo que inicia en la línea 29 leemos los valores de las columnas, i, j, $corr(i, j)$, en ese orden. Además estos valores se concatenan a la cadena que se está construyendo.
En la línea 37 y 38 se leen los valores de i ($val[0]$) y j ($val[1]$) de la tabla de correlación así como el valor de $corr(i, j)$ ($val[2]$) y posteriormente se guarda en el arreglo $corr$ tanto en el valor i,j como en el j,i. Este arreglo corresponde a un vector de $n \times n$.
Posteriormente, construimos la cadena final con los valores de $corr[i]$ en el ciclo que inicia en la línea 44 y finaliza en la línea 50.
En la línea 53 convertimos la cadena en C a algo de tipo text para devolverla a PostgreSQL.

La siguiente función se encarga de calcular la matriz de correlación, se decidió guardarlo en una tabla con entradas i,j,$corr(i, j)$ debido a la limitación de columnas que se presenta, si cada entrada fuera una columna.

get_corr_table.sql

```
1  create or replace function get_corr_table(
2  column_name text, table_name text,
3  num_columns int) returns void AS $$
4    DECLARE
5      x integer;
6      y integer;
7      name_table_corr text;
8    begin
```

```
9       name_table_corr=concat('corr_array',num_columns
            );
10      execute 'create table if not exists '||
            name_table_corr||' (
11        i int,
12        j int,
13        corr double precision
14      );';
15      execute 'delete from '||name_table_corr;
16      for x in 1..num_columns loop
17        for y in x..num_columns loop
18                execute concat('insert into ',
                      name_table_corr,' values(',x,',',y
                      ,',
19                (select corr(',column_name,x,',',
                      column_name,y,') from ',table_name
                      ,'
20                ));');
21              end loop;
22      end loop;
23  end
24  $$ language plpgsql;
```

3.2.3. Integración de LAPACK en PostgreSQL

En esta sección se presentará el código utilizado para poder hacer llamada a la función que calcula SVD dentro de la biblioteca LAPACK dentro del SMBD.

Se utilizaron dos paquetes:

- CLAPACK (f2c'ed version of LAPACK) versión 3.2.1
- Intel ® Parallel Studio XE Suites.

La integración se logró mediante el uso de UDF en PostgreSQL. Los códigos que se presentan en las siguientes secciones corresponden a las UDF que permitieron hacer la llamada a una función de LAPACK que se encarga de hacer el cálculo de SVD. LAPACK originalmente está escrito en Fortran pero posee una biblioteca **f2c** la cual nos permite hacer llamadas desde el lenguaje C y dado que este es uno de los lenguajes soportados por PostgreSQL para construir UDF, se decidió desarrollar la UDF en este lenguaje. A continuación se presenta el código de la UDF además del procedimiento para compilar la UDF y cargarla al SMBD PostgreSQL.

3.2.3.1. CLAPACK

La UDF se encarga, mediante el arreglo de entrada, calcula los valores singulares usando de la rutina `dgesvd`, que calcula estos valores, de *LAPACK*. Posteriormente, regresa la suma de estos valores singulares.
El código utilizado para realizar la integración se presenta a continuación. El archivo es llamado **array_sum.c**

```
1 #include "stdio.h"
2 #include "stdlib.h"
3 #include "f2c.h"
4 #include "clapack.h"
5 #include "postgres.h"
6 #include "fmgr.h"
7 #include "utils/array.h"
```

```
8
9  #ifdef PG_MODULE_MAGIC
10 PG_MODULE_MAGIC;
11 #endif
12
13 PG_FUNCTION_INFO_V1(suma_arreglo);
14 Datum suma_arreglo(PG_FUNCTION_ARGS)
15 {
16               ArrayType  *input;
17         Datum      *i_data;
18         bool       *nulls;
19         int         ndims, *dims, *lbs;
20         Oid         i_eltype, s_eltype, o_eltype;
21         int16       i_typlen, o_typlen;
22         bool        i_typbyval, o_typbyval;
23         char        i_typalign, o_typalign;
24         int         i,j, n;
25               integer M;
26               integer N;
27               /* return null on null input */
28         if (PG_ARGISNULL(0) || PG_ARGISNULL(1)){
29                 PG_RETURN_NULL();
30         }
31         /* get input args */
32         input = PG_GETARG_ARRAYTYPE_P(0);
33             M=PG_GETARG_INT32(1);
34             N=PG_GETARG_INT32(2);
35
36         /* get input array element type */
37         i_eltype = ARR_ELEMTYPE(input);
```

```
38          get_typlenbyvalalign(i_eltype, &i_typlen, &
                i_typbyval, &i_typalign);
39
40              /* get src data */
41          deconstruct_array(input,i_eltype, i_typlen,
                 i_typbyval, i_typalign,
42          &i_data, &nulls, &n);
43          char JOBU;
44          char JOBVT;
45          integer LDA = M;
46          integer LDU = M;
47          integer LDVT = N;
48          //suggested by
49          //http://www.math.utah.edu/software/lapack/
                lapack-d/dgesvd.html
50          integer LWORK=5*M-4;
51          integer INF;
52          integer mn = min( M, N );
53          integer MN = max( M, N );
54              double s[M];
55          double wk[LWORK];
56          double uu[M*M];
57          double vt[M*M];
58          JOBU = 'A';
59          JOBVT = 'A';
60              double a[n];
61              for(i=0; i<n; i++){
62                a[i] = DatumGetFloat8(i_data[i]);
63              }
64              dgesvd_(&JOBU,&JOBVT,&M,&N,a,&LDA,s,uu
                    ,&LDU,vt,&LDVT,wk,&LWORK,&INF);
```

```
65              double res=0;
66              for(j=0; j<M; j++){
67                          res = res + s[j];
68              }
69              PG_RETURN_FLOAT8(res);
70  }
```

En este caso no se especifica explícitamente los parámetros de entrada sino que, revisando el código, en las líneas 32-34 observamos que la entrada tiene como parámetros un arreglo y dos enteros (corresponden a m y n respectivamente).
En la línea 41 obtenemos el arreglo que es guardado en *i_data* En la línea 64 es donde se hace la llamada a la función **dgesvd_** de CLAPACK. Esta función contiene codificado en su nombre algunas propiedades sobre la matriz que está recibiendo.
d significa que corresponde a un vector de tipo double.
ge es una matriz general,es decir, una matriz no simétrica o rectangular.
svd las tres siglas de lo que se desea calcular (svd para este caso).
Para mayor referencia se puede consultar el portal de Netlib [4]

Para crear el archivo compilado es necesario haber compilado previamente todas las bibliotecas de CLAPACK. A partir de este punto, suponemos que las bibliotecas ya se han creado. El archivo requerido para la UDF tiene extensión .so para lo cual la compilación lleva dos pasos:

- `gcc -c -fPIC -I /usr/include/postgresql/9.1/server -I CLAPACK-3.2.1/INCLUDE/ array_sum.c`.
  Suponemos una instalación por defecto donde las cabeceras y otros

archivos de PostgreSQL se encuentran en la primer ruta. La segunda ruta corresponde a donde se encuentre la carpeta INCLUDE que viene incluida en los archivos descargados de la página de CLAPACK

- `gcc -shared -o array_sum.so array_sum.o CLAPACK-3.2.1/lapack_LINUX.a CLAPACK-3.2.1/blas_LINUX.a CLAPACK-3.2.1/F2CLIB/libf2c.a`

Este último genera el archivo `array_sum.so`, el cual se debe copiar a la ruta donde se encuentran los archivos .so de PostgreSQL, es decir, las bibliotecas de PostgreSQL. Para nuestro caso, por defecto la ruta es `/usr/lib/postgresql/9.1/lib`. Posteriormente, crearemos la UDF en PostgreSQL con el siguiente código:

```
1 CREATE FUNCTION array_sum(double precision[],int,
    int) RETURNS
2 double precision AS 'array_sum', 'suma_arreglo'
3 LANGUAGE C STRICT;
```

Debemos tener una tabla de n columnas, que cumpla con las reglas antes mencionadas. También se deben ejecutar **get_corr_table**.
Teniendo estas dos condiciones, además de todas las UDF presentadas en las secciones previas, se puede ejecutar la llamada a la UDF automatizada **exec_query**
Por ejemplo:
select exec_query('array_sum','clapack with cache',30); // llamada para array_sum con una tabla de 30 columnas y el mensaje que es el segundo parámetro es por control, para saber cuál experimento fue realizado.

3.2.3.2. Intel® Math Kernel Library

Para el caso de Intel® Math Kernel Library, tiene su propia función **LAPACKE_dgesvd** para hacer el cálculo de SVD. El archivo para esta sección es llamado `array_sum_mkl.c`.

```
1  #include "stdio.h"
2  #include "stdlib.h"
3  #include "mkl_lapacke.h"
4  #include "postgres.h"
5  #include "fmgr.h"
6  #include "utils/array.h"
7
8  #ifdef PG_MODULE_MAGIC
9  PG_MODULE_MAGIC;
10 #endif
11
12 PG_FUNCTION_INFO_V1(suma_arreglo);
13
14 int min(int a,int b);
15
16 Datum
17 suma_arreglo(PG_FUNCTION_ARGS)
18 {
19                 ArrayType  *input;
20         Datum      *i_data;
21         bool       *nulls;
22         int         ndims, *dims, *lbs;
23         Oid         i_eltype, s_eltype, o_eltype;
24         int16       i_typlen, o_typlen;
25         bool        i_typbyval, o_typbyval;
```

```
26          char         i_typalign, o_typalign;
27          int          i,j, n,M,N,LDA,LDU,LDVT;
28                  /* return null on null input */
29          if (PG_ARGISNULL(0) || PG_ARGISNULL(1)||
               PG_ARGISNULL(2)){
30                  PG_RETURN_NULL();
31          }
32
33          /* get input args */
34          input = PG_GETARG_ARRAYTYPE_P(0);
35                  M=PG_GETARG_INT32(1);
36                  N=PG_GETARG_INT32(2);
37                  LDA=M;
38                  LDU=M;
39                  LDVT=N;
40          /* get input array element type */
41          i_eltype = ARR_ELEMTYPE(input);
42          get_typlenbyvalalign(i_eltype, &i_typlen, &
               i_typbyval, &i_typalign);
43
44                  /* get src data */
45          deconstruct_array(input, i_eltype, i_typlen
               , i_typbyval, i_typalign,
46          &i_data, &nulls, &n);
47                  /* Locals */
48                  MKL_INT m = M, nn = N, lda = LDA,
                       ldu = LDU, ldvt = LDVT, info;
49                  double superb[min(M,N)-1];
50                  /* Local arrays */
51                  double s[N], u[M*M], vt[M*M];
52                  double a[n] ;
```

```
53                  for(i=0; i<n; i++){
54                          a[i] = DatumGetFloat8(
                               i_data[i]);
55                  }
56                  /* Compute SVD */
57                  info = LAPACKE_dgesvd(
                       LAPACK_COL_MAJOR, 'A', 'A', m,
                       nn, a, lda,
58                          s, u, ldu, vt, ldvt, superb
                             );
59
60                  double res=0;
61                  for(j=0; j<N; j++){
62                          printf("s[j]:%f \n",s[j]);
63                          printf("res :%f \n",res);
64                          res = res + s[j];
65                  }
66
67                  PG_RETURN_FLOAT8(res);
68
69  }
70
71  int min(int a,int b){
72  if(a>b){
73  return b;
74  }
75  return a;
76  }
```

Básicamente, en comparación con el código desarrollado para CLAPACK, sólo se modificaron detalles de tipos de datos y nombres de funciones. Ade-

más, se incluyó la función min ya que sin ésta se generaba un error que no permitía crear la UDF.
En la línea 57 donde se hace la llamada a la función LAPACKE_dgesvd que corresponde a la versión de Intel® MKL

Las líneas de compilación para esta sección requieren de la instalación del compilador de Intel® icc (recomendable).
La compilación requiere de 2 pasos

- `icc -c -fPIC -I /opt/intel/mkl/include/ -I /usr/include/postgresql/9.1/server/ array_sum_mkl.c`
- La segunda línea de compilación depende de la arquitectura debido a los nombres de las carpetas.

- (64 bits) icc -shared -fPIC -o array_sum_mkl.so array_sum_mkl.o -static-intel -Wl,–start-group /opt/intel/mkl/lib/intel64/libmkl_intel_lp64.a /opt/intel/mkl/lib/intel64/libmkl_sequential.a /opt/intel/mkl/lib/intel64/libmkl_core.a -Wl,–end-group
- (32 bits) icc -shared -fPIC -o array_sum_mkl.so array_sum_mkl.o -static-intel -Wl,–start-group /opt/intel/mkl/lib/ia32/libmkl_intel.a /opt/intel/mkl/lib/ia32/libmkl_sequential.a /opt/intel/mkl/lib/ia32/libmkl_core.a -Wl,–end-group

Los siguientes pasos son análogos a los pasos anteriores.

Este último genera el archivo `array_sum_mkl.so`, el cual se debe copiar a `/usr/lib/postgresql/9.1/lib`. Posteriormente, crearemos la UDF en PostgreSQL con el siguiente código:

```
1 CREATE FUNCTION array_sum_mkl(double precision[],
     int,int)
2 RETURNS double precision
3 AS 'array_sum_mkl', 'suma_arreglo'
4 LANGUAGE C STRICT;
```

A partir de este punto se procede de una manera igual que en la UDF de CLAPACK. Es decir, se ejecuta la UDF principal exec_query.
Por ejemplo:
select exec_query('array_sum_mkl','mkl with cache',30); // llamada para array_sum_mkl con una tabla de 30 columnas.

Esta función se encarga de hacer la llamada a la UDF correspondiente a los parámetros.

- El primer parámetro es el nombre de la UDF, suponiendo que la UDF que se recibe de parámetro recibe como primer parámetro el arreglo, segundo y tercer parámetro es el número de filas y columnas de la tabla de datos. Se tienen los parámetros necesarios ya definidos.
- El segundo es un mensaje para saber cuáles fueron las condiciones en que se desarrolló el experimento. Para este caso se tuvieron 4 mensajes:
  - CLAPACK with cache
  - CLAPACK without cache
  - MKL with cache
  - MKL without cache

Estos mensajes son opcionales y podrían ser cambiados, pero en este caso por simplicidad, se decidieron esos.

```
1  #include "postgres.h"
2  #include "executor/spi.h"
3  #include "utils/builtins.h"
4  #include <string.h>
5  #ifdef PG_MODULE_MAGIC
6  PG_MODULE_MAGIC;
7  #endif
8
9  int exec_query(text* mkl_or_clapack,text *
       kind_of_test, int cnt);
10
11 int
12 exec_query(text* mkl_or_clapack,text *kind_of_test,
       int cnt)
13 {
14     char command[500];
15     char cmd_insert[500];
16     int ret;
17     int proc;
18     /* Convert given text object to a C string */
19     sprintf(command,"select %s(cast(execq('select *
           from corr_array%d',%d )
20     as double precision[]),%d,%d);",text_to_cstring
           (mkl_or_clapack),
21     cnt,cnt,cnt,cnt);
22
23     sprintf(cmd_insert, "insert into
           experiments_time(actual_date,
24     kind_of_test,size)
25
26     values (timeofday(),'%s inicial',%d);",
```

```
        text_to_cstring(kind_of_test),cnt);
27      SPI_connect();
28      ret=SPI_exec(cmd_insert, 0);
29      //0 indica que se toman todos (de acuerdo a la
            doc)
30      ret = SPI_exec(command, 0);
31      proc = SPI_processed;
32      sprintf(cmd_insert, "insert into
            experiments_time(actual_date,
33      kind_of_test,size)
34      values (timeofday(),'%s final',%d);",
            text_to_cstring(kind_of_test),cnt);
35      SPI_exec(cmd_insert, 0);
36      SPI_finish();
37      PG_RETURN_INT32(proc);
38  }
```

3.2.4. Inhibición de cache.

Una de las pruebas que fueron realizadas, para lo cual se desarrollaron los siguiente módulos, es la inhibición de caché, dado que Intel® MKL aprovecha la memoria caché para las operaciones. Los módulos fueron creados para que de manera automática se pudiera inhibir el caché o volverlo a activar. Estos códigos fueron inspirados en la tesis [16]. Se desarrollaron varios archivos para facilitar la obtención de resultados.

Para los subsecuentes módulos se escribió, para cada uno, un `Makefile` con el siguiente contenido:

```
obj-m += nombre\_archivo.o
KVERSION = $(shell uname -r)
all:
make  -C /lib/modules/$(KVERSION)/build M=$(PWD) modules

clean:
make  -C /lib/modules/$(KVERSION)/build M=$(PWD) clean
```

Se desarrolló un módulo de kernel para inhabilitar el cache

disable_cache.c

```
1
2  #include <linux/module.h>        /* Needed by all
       modules */
3  #include <linux/kernel.h>        /* Needed for
       KERN_INFO */
4  #include <linux/init.h>          /* Needed for the
       macros */
5  #include <asm/io.h>
6  #include <asm/bugs.h>
7  #include <asm/setup.h>
8  #include <asm/sections.h>
9  #include <asm/cacheflush.h>
10
11 #ifdef CONFIG_X86_LOCAL_APIC
12 #include <asm/smp.h>
13 #endif
14
15 static void __init disable_cache(void)
16 {
17         asm("push %eax");//              ; save eax
```

```
18          asm("cli");//                    ; disable
               interrupts while we do this
19
20          asm("mov %cr0, %eax");//          ; read CR0
21
22          asm("or  $0x40000000, %eax");//   ; set CD
               but not NW bit of CR0
23
24          asm("mov %eax, %cr0");//          ; cache is
                now disabled
25          asm("wbinvd"); //flush
26
27          asm("or $0x20000000, %eax");//       ; now
               set the NW bit
28          asm("mov %eax, %cr0");      //     ; turn off
                the cache entirely
29          asm("pop %eax");//             ; restore eax
30
31  }
32
33  static void __exit unload_module(void)
34  {
35          printk(KERN_INFO "Unloading module kernel(
               disable cache).\n");
36  }
37
38  module_init(disable_cache);
39  module_exit(unload_module);
```

También, se creó un módulo de kernel para activar el caché.

enable_cache.c

```
1  #include <linux/module.h>        /* Needed by all
       modules */
2  #include <linux/kernel.h>        /* Needed for
       KERN_INFO */
3  #include <linux/init.h>          /* Needed for the
       macros */
4  #include <asm/io.h>
5  #include <asm/bugs.h>
6  #include <asm/setup.h>
7  #include <asm/sections.h>
8  #include <asm/cacheflush.h>
9
10 #ifdef CONFIG_X86_LOCAL_APIC
11 #include <asm/smp.h>
12 #endif
13
14 static void __init enable_cache(void)
15 {
16
17         asm("push %eax");//              ; save eax
18         asm("cli");// ; disable interrupts while we
               do this
19
20         asm("mov %cr0, %eax");//          ; read CR0
21
22         asm("and  $0xB1111111, %eax");//      ; set
               CD but not NW bit of CR0
```

```
23
24          asm("mov %eax, %cr0");//          ; cache is
                now disabled
25          asm("wbinvd"); //flush
26
27          asm("and $0xD1111111, %eax");//       ; now
               set the NW bit
28          asm("mov %eax, %cr0");      //     ; turn off
                the cache entirely
29          asm("pop %eax");//            ; restore eax
30  }
31
32  static void __exit unload_module(void)
33  {
34          printk(KERN_INFO "Unloading module kernel(
               enable cache).\n");
35  }
36
37  module_init(enable_cache);
38  module_exit(unload_module);
```

3.3. Resultados

En esta sección presentaremos pantallas de la experimentación realizada. En particular, se toma el caso para $n = 5$ ejemplificando las salidas obtenidas para cada tamaño.

Como ya se mencionó, primero se deben tener todas las UDF necesarias. En la Figura 3.1 observamos la definición de las UDF cargadas dentro del SMBD

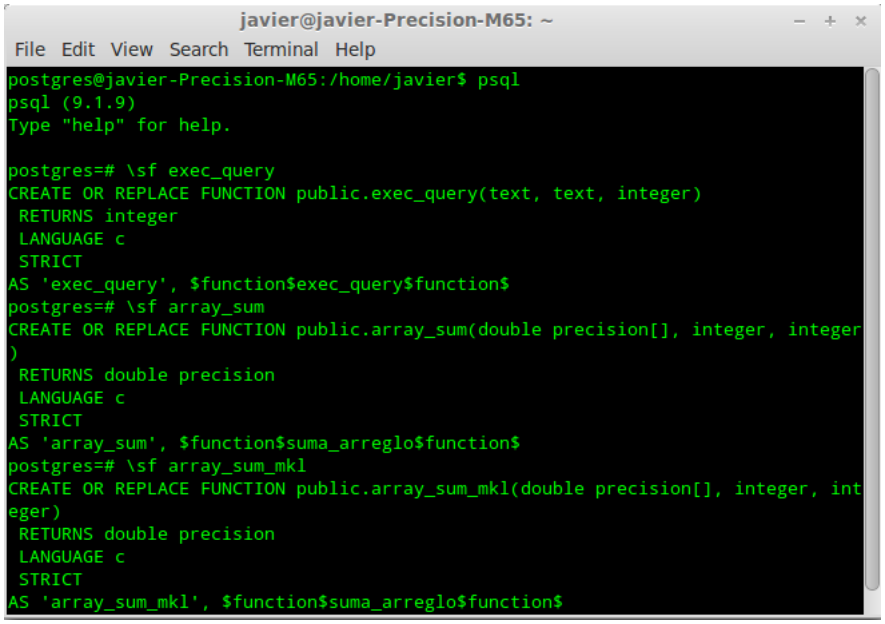

Figura 3.1: UDFs necesarias para la experimentación

También son importantes la tabla con los datos con las reglas mencionadas previamente. Las tablas *experiments_time* y *substract* son donde se guardan los datos de tiempos y tamaños. En la Figura 3.2 observamos las tablas que se encuentran dentro del SMBD, cada una indica el número de columnas dentro del nombre de la tabla.

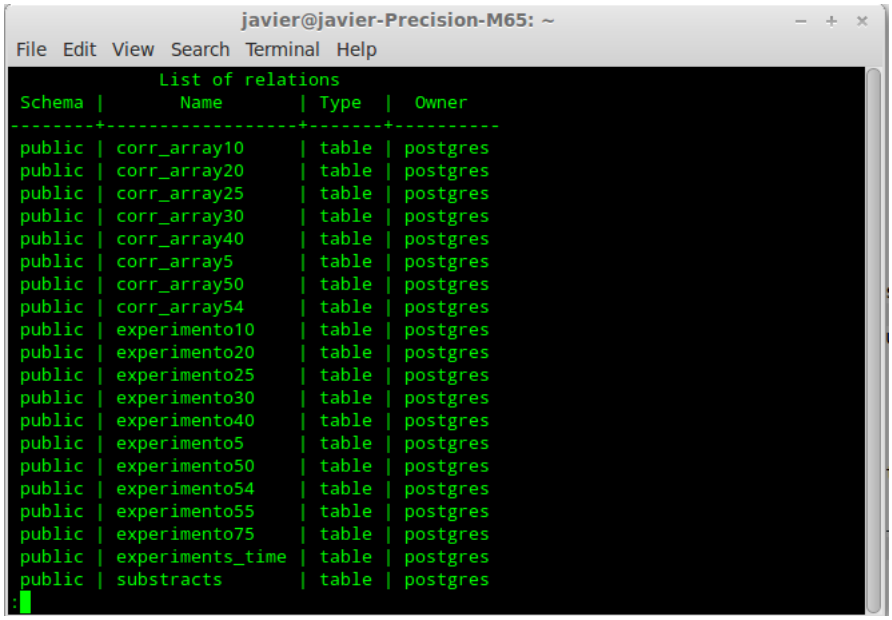

Figura 3.2: Tabla con datos cargados para experimentos

En la Figura 3.3 se observa la llamada a la UDF, como se puede observar ahí no aparece alguna medida de tiempo, pero internamente se guardan etiquetas de tiempo dentro de la tabla *experiments_time*.

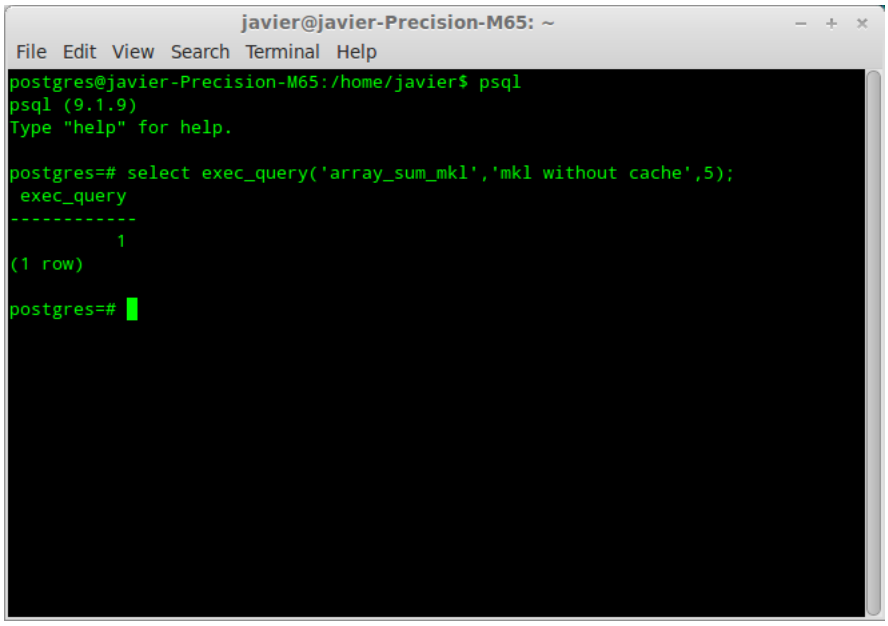

Figura 3.3: Ejemplo llamada a UDF

En la Figura 3.4 se muestra parte del contenido de la tabla para $n = 5$ (5 columnas), para el caso particular de los experimentos. Este tamaño sólo fue tomado como ejemplo para mostrar la estructura de la tabla.

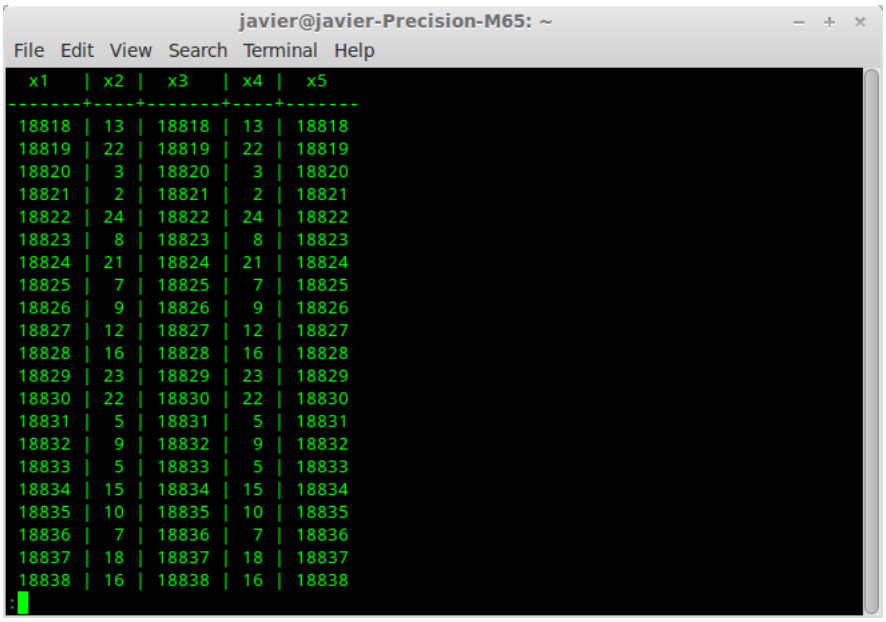

Figura 3.4: Ejemplo datos cargados en una tabla para n=5

Recordemos que también es necesaria la matriz de correlación que se encuentra guardada en la tabla corr_array5 (para el caso $n = 5$). Dado que es una matriz simétrica, sólo se guardan las entradas donde $i <= j$ y posteriormente cuando se construya el arreglo que será la entrada para la UDF, estas entradas a_{ji} se tienen en la entrada a_{ij} cuando $i > j$. En la Figura 3.5 observamos el ejemplo de la matriz de correlación para 5 columnas.

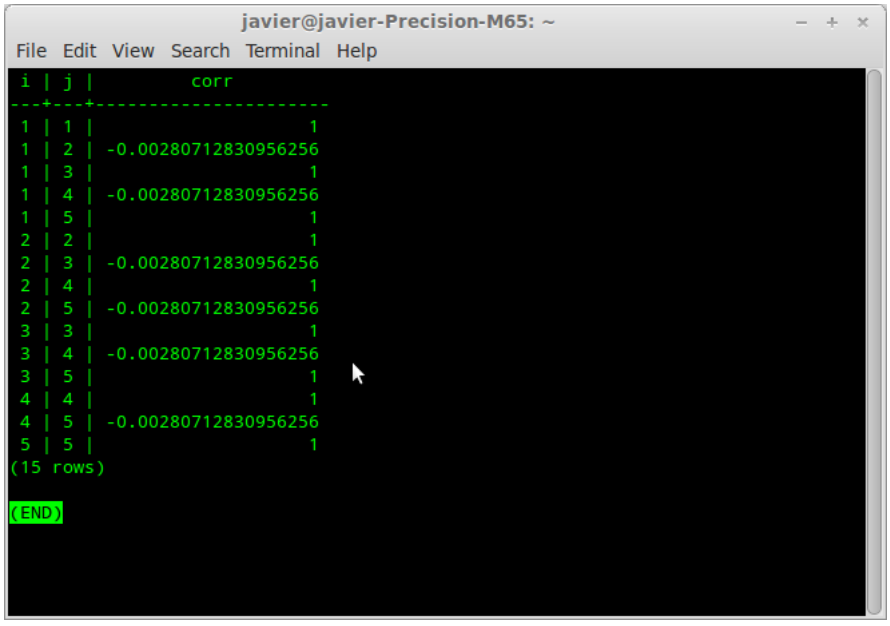

Figura 3.5: ejemplo de tabla que guarda las entradas de la matriz de correlación

En la Figura 3.6 observamos el formato en que se guardan los tiempos dentro de la tabla *experiments_time*.

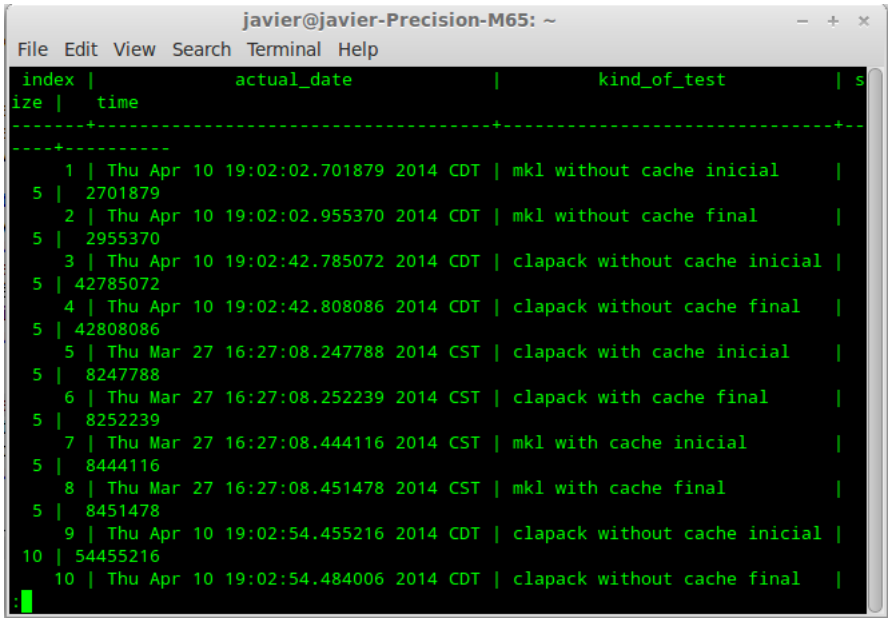

Figura 3.6: Ejemplo de la tabla que almacena los tiempos de ejecución

La Figura 3.7 muestra los tiempos obtenidos al realizar el promedio de los tiempos obtenidos dado que por cada tamaño de la tabla se hicieron 5 experimentos.

substract	kind_of_test	size
4451	clapack with cache inicial	5
23014	clapack without cache inicial	5
7362	mkl with cache inicial	5
253491	mkl without cache inicial	5
5689	clapack with cache inicial	10
28790	clapack without cache inicial	10
11818	mkl with cache inicial	10
9193	mkl without cache inicial	10
100861	clapack with cache inicial	20
51447	clapack without cache inicial	20
4993	mkl with cache inicial	20
245506	mkl without cache inicial	20
11669	clapack without cache inicial	25
29929	mkl without cache inicial	25
34356	clapack with cache inicial	30
14590	clapack without cache inicial	30
323548	mkl with cache inicial	30
62081	clapack with cache inicial	40
23286	clapack without cache inicial	40
41762	mkl with cache inicial	40
81855	mkl without cache inicial	40
84102	clapack with cache inicial	50
34584	clapack without cache inicial	50
22554	mkl with cache inicial	50
60344	mkl without cache inicial	50
67181	clapack with cache inicial	54
41094	clapack without cache inicial	54
22875	mkl with cache inicial	54
117838	mkl without cache inicial	54

(29 rows)

(END)

Figura 3.7: Tabla con los tiempos tomados de psql de la manera mencionada previamente

Cuadro 3.1: Tabla de tiempos

Número de columnas	con caché			sin caché		
n	CLAPACK	MKL	convencional	CLAPACK	MKL	convencional
05	4.451	7.362	5.031	23.014	25.3491	8.52
10	5.689	11.818	10.1548	28.790	9.193	15.154
20	100.861	4.993	120.861	51.447	245.506	53.415
30	34.356	32.3548	37.145	14.590	32.251	37.148
40	62.081	41.762	65.457	23.286	81.855	68.785
50	84.102	22.554	90.254	34.584	60.344	96.148
54	67.181	22.875	74.897	41.094	83.356	117.838

Los tiempos son mostrados en milisegundos. El método convencional es extraer los datos del SMBD

En este caso el método convencional se refiere a realizar una copia de los datos y llevarlos a un programa externo para su análisis. En el cuadro 3.1 observamos los tiempos obtenidos de la experimentación, muestra qué sucede cuando n aumenta. En un inicio los tiempos son muy parecidos pero mientras n crece, podemos comparar qué sucede con los tiempos en cada uno de los métodos que fueron experimentados.

Capítulo 4

Conclusión

Mediante esta experimentación se comprobó la hipótesis que teníamos sobre la mejora proporcionada por la estrategia propuesta de integrar las funciones de LAPACK dentro del SMBD PostgreSQL. Es importante notar la validez del resultado a partir de ciertos tamaños, principalmente para cuando el tamaño se incrementa y dado que, la hipótesis está planteada en el contexto de grandes volúmenes de datos, que son los que se manejan en minería de datos, se comprueba la eficiencia de la propuesta. Además, como se mencionó previamente, tenemos otros factores para no sacar los datos del SMBD, entre los cuales se pueden mencionar:

- Los datos se encuentran en un ambiente protegido que es proporcionado por el SMBD
- Al ejecutar el procedimiento se puede trabajar con una versión actualizada de los datos, mientras que en el método convencional, si se trabaja con esta versión se tendría que extraer los datos del SMBD,

lo cual puede ser muy costoso ya que estamos hablando de tablas de gran cantidad de registros, aunque no se evita el tener que procesar los datos para hacerles un preprocesamiento para tener datos de calidad.

- Se evita el tiempo de hacer una copia de la base de datos.

Algunos de los problemas que fueron encontrados fue hacer un ligado dinámico para evitar que creciera el tamaño debido a un ligado estático (incluir la biblioteca dentro del archivo objeto generado). También se tuvo un problema con el tamaño de las tablas que sólo aceptaba 50 columnas debido a una limitación en la memoria.

Un punto que cabe resaltar sobre esta tesis es que aquí trabajamos sobre SVD pero los resultados obtenidos se pueden extender hacia otros procedimientos que involucre manejo de matrices debido a que se tiene una gran variedad de rutinas para poder calcular operaciones matriciales.

Se debe notar que la estrategia propuesta es una alternativa hacia el paradigma tradicional de extraer los datos o construir un almacén de datos, las ventajas que tiene esta propuesta es que los datos conservan las propiedades que nos da un SMBD como la consistencia, se evita el tiempo de exportación de datos así como la construcción de algunas tablas adicionales ya que los datos se manejan directamente. Las desventajas de la propuesta es la carga de trabajo que se le da al sistema transaccional ya que, como se mencionó, el trabajo se hace sobre los datos almacenados en el SMBD. Entonces si se tiene un sistema con una alta carga transaccional este método no es efectivo ya que el rendimiento se vería afectado debido a su alta carga de trabajo. Otra desventaja sería tener que construir una UDF para cada función de LAPACK ya que en el presente trabajo sólo se trabajó con una de las funciones pero esta biblioteca incluye muchas funciones.

Bibliografía

[1] Acerca de postgresql. http://www.postgresql.org/about/.

[2] Agregaciones definidas por el usuario de postgresql. http://www.postgresql.org/docs/9.1/static/xaggr.html (consultada el 15 de abril de 2014).

[3] Clapack. www.netlib.org/clapack/clapack.tgz (consultada el 15 de abril de 2014).

[4] Esquema de nombres netlib. http://www.netlib.org/lapack/lug/node24.html (consultada el 15 de abril de 2014).

[5] Funciones definidas por el usuario de postgresql. http://www.postgresql.org/docs/9.1/static/xfunc-sql.html (consultada el 15 de abril de 2014).

[6] Historia de postgresql. http://www.postgresql.org/about/history/ (consultada el 15 de abril de 2014).

[7] Intel® mkl. https://software.intel.com/en-us/intel-mkl (consultada el 15 de abril de 2014).

[8] Netlib. `http://www.netlib.org/lapack/` (consultada el 15 de abril de 2014).

[9] Svd application. `https://inst.eecs.berkeley.edu/~ee127a/book/login/l_svd_apps_image.html` (consultada el 15 de abril de 2014).

[10] Svd low rank. `https://inst.eecs.berkeley.edu/~ee127a/book/login/l_svd_low_rank.html` (consultada el 15 de abril de 2014).

[11] Wolfram pseudoinverse. `http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html` (consultada el 15 de abril de 2014).

[12] Ordonez Carlos and García-García Javier. Vector and matrix operations programmed with udfs in a relational dbms. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 503–512. ACM, 2006.

[13] Ordonez Carlos and Pitchaimalai Sasi K. Fast udfs to compute sufficient statistics on large data sets exploiting caching and sampling. *Data & Knowledge Engineering*, 69(4):383–398, 2010.

[14] Kamber Micheline Han Jiawei and Pei Jian. *Data Mining: Concepts and Techniques*. Morgan kaufmann, 2006.

[15] Garcia-Molina Hector. *Database systems: the complete book*. Pearson Education India, 2008.

[16] Zuñiga J. I. Estudio y experimentación de los factores que inciden en el desempeño de operaciones complejas dentro de un sistema manejador de bases de datos. Tesis de maestría. Universidad Nacional Autónoma de México, 2014.

[17] Intel Intel. and ia-32 architectures software developer's manual. *Volume 3A: System Programming Guide, Part*, 1, 64.

[18] Navas Mario and Ordonez Carlos. Efficient computation of pca with svd in sql. In *Proceedings of the 2nd Workshop on Data Mining using Matrices and Tensors*, page 5. ACM, 2009.

[19] Korth Henry F. Silberschatz Abraham and Sudarshan S. *Database system concepts*, volume 6. McGraw-Hill New York, 2011.

[20] Demmel James W. *Applied numerical linear algebra*. Siam, 1997.

[21] Rinsurongkawong Waree and Ordonez Carlos. Microarray data analysis with pca in a dbms. In *Proceedings of the 2nd international workshop on Data and text mining in bioinformatics*, pages 13–20. ACM, 2008.

[22] Chen Zhibo and Ordonez Carlos. Efficient olap with udfs. In *Proceedings of the ACM 11th international workshop on Data warehousing and OLAP*, pages 41–48. ACM, 2008.