UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

PRACTICAS DE LABORATORIO PARA LOS CURSOS DE SISTEMAS DE BASES DE DATOS Y SISTEMAS MANEJADORES DE BASES DE DATOS

T E S I S

QUE PARA OBTENER EL TITULO DE:

LICENCIADO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

RENE ALEJANDRO VILLEDA RUZ

FACULTAD DE CIENCIAS
UNAM

DIRECTOR DE TESIS: M. en C. JAVIER GARCIA GARCIA

2005

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito: "Prácticas de Laboratorio para los cursos de Sistemas de Bases de Datos y Sistemas Manejadores de Bases de Datos"

realizado por René Alejandro Villeda Ruz

con número de cuenta . 092003206 , quien cubrió los créditos de la carrera de:

Lic. en Ciencias de la Computación.

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	M. en C. Javier García García
Propietario	Dra. Amparo López Gaona
Propietario	Dra. Hanna Oktaba
Suplente	Lic. en C.C. Egar Arturo García Cárdenas
Suplente	Lic. en C.C. Manuel Alberto Sugawara Muro

Consejo Departamental de Matemáticas.

Dr. Francisco Hernández Quintana

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMATICAS

A mi padre.
Enrique Villeda Navarro
por el apoyo y cariño brindado
incondicionalmente.

A mi hermano.
Enrique Villeda Ruz
por encontrarse siempre a mi lado.

Agradecimientos

El presente trabajo no es sino el reflejo de la experiencia personal con familiares, amigos y profesores, quienes en un grado u otro han sido importantes para mi en distintos aspectos y han contribuido a la realización de esta meta.

Primeramente quiero agradecer a mi padre: **Enrique Villeda Navarro**, por estar presente en todos los aspectos de mi vida, enseñarme desde las cosas más simples hasta las más complejas, encaminarme en la máxima casa de estudios y mostrarme el cumplir como persona ante las adversidades más fuertes. Eres único y te admiro en todos los sentidos. A mi hermano **Enrique**, por apoyarme e impulsarme para ser siempre una mejor persona y haber despertado en mi el deseo de superación; eres mi segundo padre y siempre apreciare lo que haces por todos.

A la **UNAM**, por brindarme la oportunidad de pertenecer a ella desde Iniciación Universitaria en la **ENP 2**, ser grande en todos los sentidos e inculcarme la generación de conocimiento. Porque el conocimiento bien encaminado es el alimento de nuestro espíritu.

A mi familia paterna, en especial a mi primo René (otro René...) por ser tan abierto conmigo y considerarnos como hermanos. A mi abuelo Aquilino, por transmitir durante generaciones su conocimiento, demostrar que la sabiduría no se aprende en la escuela y brindarme todo el apoyo en momentos difíciles. A mis primas Rosario, Elideth y Dánae por ser tan amables y el apoyo prestado a pesar de la distancia. A mis primos Christian y Salvador, por todos esos días largoooos de vacaciones que terminaban hasta entrada la noche y interminable capacidad de sorprenderme. A mi abuela Isabel y a mis tías y tíos por ser tan buenas personas, animarme y apoyarme siempre que los necesite, particularmente mi tío René y a mi tía Edith por encontrarse presente en momentos importantes de nuestra familia y por el cariño y confianza que tenemos.

A mis amigas y amigos, pero de manera muy, muy especial a mi querida novia Verónica Cruces Ramírez por ser tan buena persona conmigo, compartir tantos detalles y experiencias pero sobre todo por apoyarme incondicionalmente y aguantarme tantas

y tantas veces aunque no lo mereciera. Sin su apoyo hubiese tardado más en terminar esta meta; a Karla Ramírez Pulido, por ser una excelente persona en todos los aspectos y siempre una fuente de admiración. Sus consejos y observaciones siempre tan a la mano han sido una gran influencia; a Pedro Abundes Jiménez ('Chino') por compartir tantas anécdotas desde la 'H. Prepa 2' y hasta ahora; a Karla Jiménez Álvarez por ayudarme a 'madurar' rápidamente desde Iniciación Universitaria, sus charlas tan amenas y llenas de sabiduría y sus consejos que siempre tengo presentes; a Elia A. Calderón Ríos por ser tan tierna y linda persona conmigo, demostrarme que los problemas no son obstáculos infranqueables y ser siempre tan detallista en todos los sentidos; a Maricarmen García González ('Mega') por saber que le importaba a alguien, sorprenderme con su gran talento y esas salidas de la Prepa 2 donde siempre terminábamos divirtiéndonos como niños. A Beatriz Rodríguez Escalera, por mostrarme que un año no es nada en la vida y ser un ejemplo de constancia y dedicación; a Ilitia Sauer Vera, por abrirme los ojos en muchas situaciones y otras tantas aguantar mis sollozos, gracias por tu inmensa comprensión.

A Selene Calva Estrada, por mostrarme el tener siempre ánimos y seguir adelante ante cualquier condición; a Juan Pablo Vivaldo Martínez ('Bull'), por enseñarme a reír de la vida y conservar siempre los ideales; a Belén Reyes Alcaraz por ser tan linda persona, enseñarme a bailar (aún siendo yo tan pésimo aprendiz ☺) y a no desesperarme a cada momento; a Carolina Chávez Cortés ('Caro'), Guadalupe Alvarado Arias ('Lupis') y Ana María Pitalua Leyva ('Anita') por ser excelentes amigas, compartir tantos momentos agradables y ejemplificar lo que es la superación constante; a Tomiko Saisho García (¡salud!) por todas esas fiestas y consejos prestados tan oportunamente. A Perla González Bastida ('Perliux'), Ariadna Cadena Almaraz ('Ari'), Flor González Jaramillo ('Florecita'), María González Vázquez, Lorena Montes de Oca Muñoz y Lorena Valadez Jaimes por brindarme siempre su gran amistad; y a Yolanda Rocha Rodríguez ('Mayo') por compartir tantos momentos simpáticos en '*JI*', sueños profesionales y mostrarme como afrontar situaciones difíciles de la manera más sencilla.

Además, a Liliana González Suárez ('Liz') por ser tan linda y escucharme en toda ocasión que ameritaba (aunque nunca me dio 'achicalada' ☹); a Julieta López Melendez ('Gina') por enseñarme a no ser obsesivo (... bueno, el intento lo hizo) y el verdadero significado de la palabra 'dedicación', a Eudave (ok, ok... Lorena Eudave Loera) por mostrarme que las cosas siempre tienen otro punto de vista y siempre vale la pena conocerlos; a Minerva L. Luna Nava (☾) por enseñarme a portarme mal, que la constancia siempre debe estar bien encaminada, brindarme su amistad incondicional y a estar siempre lleno de ánimos. A Javier Sánchez Aguilar y Alfonso Ayala Rangel, por enseñarme que no siempre hay que correr más sino saber *por donde* correr; y a Rodolfo Conde Martínez por la gran ayuda prestada durante todos los semestres y mostrarme siempre el lado bueno del opensource (aunque internamente seas Matemático).

También a Vanessa B. Ortiz Cuellar y Mónica G. Morales Camacho por ser fuente de motivación y superación en momentos importantes de mi vida; a Adriana Ramírez Vigueras por empujarme a dar el paso dentro del ámbito académico y ser tan entusiasta

y constante; a Virginia Teodosio Procopio ('Vicky') por esas conversaciones siempre tan amenas e interesantes y pasar momentos agradables durante la carrera; y a Yasley Mora Campos ('Yas') por animarme a vivir la vida de otra manera y apoyarme para bailar un poco mejor.

Y en general a todas(os) mis amigas(os) de la Prepa 2 y Facultad de Ciencias (si no los he nombrado, saben que es por mi pésima memoria) por compartir algo de su vida conmigo.

A mis profesores, por que cada uno a dejado algo en mi y han contribuido en distinta proporción a que logre mis objetivos. Muy en especial al Maestro Javier García García excelente profesor a quien considero mi amigo, me dio la confianza de iniciarme en este ambiente tan gratificante que es la enseñanza, es un ejemplo de trabajo y sin quien este último trabajo hubiese demorado más; al Maestro José Galaviz Casas, quien literalmente rescato un grupo de ICC1 y con ello marco el camino de una generación de 'Computólogos' (G4), pues es una persona muy agradable y genial profesor quien demuestra que la docencia de excelencia no se encuentra peleada con lo ameno e interesante; y a la Doctora Hanna Oktaba, por su confianza y mostrarme que aún los detalles mínimos son siempre importantes.

Por último, a quienes han sido mis alumnos (en especial a mi primer grupo de SMBD), porque me han demostrado que también uno aprende de ellos.

Y en general agradezco nuevamente a todos por su amistad y apoyo, me siento orgulloso y afortunado de conocer a personas tan valiosas (en muchos sentidos) como lo son ustedes GRACIAS.

René A. Villeda Ruz.

"How is education supposed to make me feel smarter?
Besides, every time I learn something new, it pushes some old stuff out of my brain."
— Homer Simpson

Índice general

Introducción I

Convenciones III

Índice de tablas XI

Índice de figuras XIII

1. Sistemas Manejadores de Bases de Datos 5
- 1.1. Meta . . . 5
- 1.2. Objetivos . . . 5
- 1.3. Desarrollo . . . 5
  - 1.3.1. PostgreSQL . . . 6
  - 1.3.2. Instalación . . . 6
  - 1.3.3. postmaster . . . 7
  - 1.3.4. psql . . . 8
- 1.4. Ejercicios . . . 10

2. Modelado de Bases de Datos 13
- 2.1. Meta . . . 13
- 2.2. Objetivos . . . 13
- 2.3. Desarrollo . . . 13
  - 2.3.1. Diagramas Entidad-Relación . . . 14
  - 2.3.2. Dia . . . 14
  - 2.3.3. UML . . . 16
- 2.4. Ejercicios . . . 19

3. SQL
Lenguaje de Definición de Datos **23**
3.1. Meta . . . 23
3.2. Objetivos . . . 23
3.3. Desarrollo . . . 23
3.3.1. SQL . . . 24
3.3.2. Creación de tablas en la base de datos . . . 25
3.3.3. Eliminación de tablas . . . 27
3.3.4. Modificación de tablas . . . 28
3.3.5. Proceso por lote . . . 29
3.4. Ejercicios . . . 31

4. SQL
Lenguaje de Manipulación de Datos **35**
4.1. Meta . . . 35
4.2. Objetivos . . . 35
4.3. Desarrollo . . . 35
4.3.1. Inserción de datos . . . 36
4.3.2. Eliminación de datos . . . 38
4.3.3. Actualización de datos . . . 39
4.4. Ejercicios . . . 41

5. SQL
Lenguaje de Manipulación de Datos (II) **45**
5.1. Meta . . . 45
5.2. Objetivos . . . 45
5.3. Desarrollo . . . 45
5.3.1. Consultas . . . 46
5.4. Ejercicios . . . 55

6. SQL
Lenguaje de Manipulación de Datos (III) **59**
6.1. Meta . . . 59
6.2. Objetivos . . . 59
6.3. Desarrollo . . . 59
6.3.1. Consultas . . . 60
6.3.2. Vistas . . . 68
6.4. Ejercicios . . . 70

7. Integridad **75**
7.1. Meta . . . 75
7.2. Objetivos . . . 75
7.3. Desarrollo . . . 75

7.3.1. La Integridad 76
7.4. Ejercicios 82

8. Seguridad **85**
8.1. Meta 85
8.2. Objetivos 85
8.3. Desarrollo 85
8.3.1. Seguridad dentro de PostgreSQL 86
8.4. Ejercicios 95

9. Normalización **99**
9.1. Meta 99
9.2. Objetivos 99
9.3. Desarrollo 99
9.3.1. Un ejemplo práctico 100
9.4. Ejercicios 107

10. Conexión a Bases de Datos **111**
10.1. Meta 111
10.2. Objetivos 111
10.3. Desarrollo 111
10.3.1. ODBC 112
10.3.2. JDBC 112
10.3.3. Proceso general de conexión 115
10.4. Ejercicios 121

11. Acceso a Memoria **127**
11.1. Meta 127
11.2. Objetivos 127
11.3. Desarrollo 127
11.3.1. La jerarquía de memoria 128
11.3.2. Perl 129
11.4. Ejercicios 134

12. Niveles RAID **137**
12.1. Meta 137
12.2. Objetivos 137
12.3. Desarrollo 137
12.3.1. RAID 138
12.3.2. Hardware RAID 140
12.3.3. Software RAID 140
12.4. Ejercicios 144

13.Sistemas de archivos **147**
13.1. Meta 147
13.2. Objetivos 147
13.3. Desarrollo 147
13.3.1. Sistemas de archivo sin Journaling 150
13.3.2. Sistemas de archivo con Journaling 152
13.4. Ejercicios 155

14.Manejo de índices **159**
14.1. Meta 159
14.2. Objetivos 159
14.3. Desarrollo 160
14.3.1. Índices 160
14.3.2. Índices en PostgreSQL 161
14.4. Ejercicios 168

15.Optimización de consultas **173**
15.1. Meta 173
15.2. Objetivos 173
15.3. Desarrollo 173
15.4. Ejercicios 183

16.Transacciones **187**
16.1. Meta 187
16.2. Objetivos 187
16.3. Desarrollo 187
16.3.1. Transacciones en PostgreSQL 189
16.4. Ejercicios 195

17.Concurrencia **201**
17.1. Meta 201
17.2. Objetivos 201
17.3. Desarrollo 201
17.3.1. Concurrencia dentro de PostgreSQL 202
17.4. Ejercicios 210

18.Recuperación **215**
18.1. Meta 215
18.2. Objetivos 215
18.3. Desarrollo 215
18.3.1. Técnicas de recuperación 216
18.3.2. PostgreSQL y WAL 216
18.4. Ejercicios 218

19. Modificación de recursos utiliados por el SMBD **223**
19.1. Meta . . . 223
19.2. Objetivos . . . 223
19.3. Desarrollo . . . 223
19.3.1. *Escalamiento* y *Aceleración* . . . 224
19.3.2. Reporte de resultados de pruebas . . . 225
19.4. Ejercicios . . . 226

20. Datawarehouse **231**
20.1. Meta . . . 231
20.2. Objetivos . . . 231
20.3. Desarrollo . . . 231
20.3.1. OLAP y OLTP . . . 232
20.4. Microsoft© SQL Server™ 2000 . . . 234
20.4.1. Creación de cubos de múltiples dimensiones . . . 235
20.4.2. Tipos de almacenamiento . . . 238
20.4.3. Visor de cubos de múltiples dimensiones . . . 239
20.5. Ejercicios . . . 244

A. Temarios **247**

B. Lineamientos generales **253**

C. Base de datos para el primer curso **255**
C.1. Introducción . . . 255
C.2. Consideraciones . . . 255
C.3. Modelo Entidad-Relación . . . 256
C.3.1. Entidades . . . 256
C.3.2. Relaciones . . . 257
C.4. Modelo Relacional . . . 258
C.4.1. Dependencias Funcionales . . . 260
C.4.2. Formas Normales . . . 260
C.5. UML . . . 260
C.6. Código . . . 262

D. Base de datos para el segundo curso **269**
D.1. Introducción . . . 269
D.2. Consideraciones . . . 269
D.3. Modelo Entidad-Relación . . . 270
D.3.1. Entidades . . . 270
D.3.2. Relaciones . . . 271
D.4. Modelo Relacional . . . 271
D.4.1. Dependencias Funcionales . . . 272

D.4.2. Formas Normales 272
D.5. Diagrama de clases - UML 273
D.6. Código 273

E. Microsoft© SQL Server™ 2000 **277**
E.1. Introducción 277
E.1.1. Historia 278
E.1.2. Componentes 278
E.2. Obtención e instalación 280
E.3. Servicios de Análisis (*'Analysis Services'*) 286
E.3.1. Instalación 289
E.4. Fuentes de datos 291
E.4.1. PostgresSQL para Windows© 291

F. pgAdminIII **293**
F.1. Obtención e instalación 293
F.2. La interfaz 294
F.2.1. Agregando servidores 295
F.2.2. Herramienta de consultas de pgAdmin III 295
F.2.3. Herramientas administrativas 296

Bibliografía **299**

Índice alfabético **302**

Índice de tablas

2.1. Conversión de elementos ER - UML. 18

3.1. Tipos de datos en Postgresql. 26

5.1. Principales expresiones dentro de la claúsula `WHERE` 48
5.2. Principales funciones escalares. 50
5.3. Operadores de conjuntos de tuplas. 51
5.4. Ejemplos de utilización de la cláusula `LIKE`. 53

10.1. Métodos getXXX recomendados para recuperar datos. 119

14.1. Parámetros en tiempo de ejecución para el optimizador de consultas. . 167

16.1. Tabla de actualizaciones en precios. 189

17.1. Niveles de aislamiento según SQL. 203

Índice de figuras

2.1. Un diagrama Entidad-Relación. 14
2.2. Interfaz de Dia. 15
2.3. Un diagrama de clases UML. 17

3.1. Sintaxis básica de la sentencia `CREATE`. 25
3.2. Sintaxis de la sentencia `DROP`. 28
3.3. Sintaxis básica de la sentencia `ALTER`. 29

4.1. Sintaxis básica de la sentencia `INSERT`. 36
4.2. Sintaxis de la sentencia `DELETE`. 38
4.3. Sintaxis básica de la sentencia `UPDATE`. 39

5.1. Estructura básica de la sentencia `SELECT`. 46
5.2. Sintaxis de la cláusula `ORDER BY`. 49
5.3. Uso de los operadores de conjuntos de tuplas. 51
5.4. Sintaxis de los operadores `LIKE` y `SIMILAR TO`. 52

6.1. Sintaxis de la cláusula `GROUP BY`. 60
6.2. Representación de una ejecución `GROUP BY` con respecto a dos atributos. 62
6.3. Sintaxis de la cláusula `HAVING`. 63
6.4. Tablas ejemplo para el producto cartesiano. 63
6.5. Sintaxis básica de la cláusula `JOIN`. 64
6.6. Tablas de ejemplo para la operación de división. 67
6.7. Versión alternativa de la división en SQL. 68
6.8. Sintaxis de la cláusula `CREATE VIEW`. 69

7.1. Sintaxis básica de las restricciones soportadas por PostgreSQL. 78

7.2. Sintaxis básica de las restricciones de integridad en la creación de una tabla. 81

8.1. Conexión de clientes a través de Internet. 86
8.2. Sintaxis de la cláusula `CREATE USER`. 88
8.3. Sintaxis de la cláusula `CREATE GROUP`. 88
8.4. Sintaxis de la cláusula `ALTER GROUP`. 89
8.5. Sintaxis básica de `GRANT` sobre tablas. 90
8.6. Sintaxis básica de `GRANT` sobre bases de datos. 90
8.7. Sintaxis básica de `REVOKE` sobre tablas. 90
8.8. Sintaxis básica de `REVOKE` sobre bases de datos. 91
8.9. Sintaxis de un registro en 'pg_hba.conf' para clientes locales. 92
8.10. Sintaxis de un registro en 'pg_hba.conf' para clientes remotos. 92

9.1. Relación *R* inicial. 102
9.2. Relación *S*. 102
9.3. Relación *R* en 1FN. 103
9.4. Descomposición de *R* en *R1*, *R2* y *R3* que se encuentran en 2FN. . . . 104
9.5. Descomposición de *R* en *R1*, *R2*, *R3* y *R4* que se encuentran en 3FN. . 105
9.6. Descomposición de *S* en *S1* y *S2* que se encuentran en la 4FN. 106

10.1. Modelo de dos capas. 113
10.2. Modelo de tres capas. 114
10.3. Acceso a una base de datos por medio de un controlador JDBC Tipo 4. 114

11.1. Sintaxis de la función `OPEN`. 130
11.2. Modos básicos de apertura de archivos en Perl. 131
11.3. Ejemplos de apertura de archivos en Perl. 132
11.4. Sintaxis de la función `PRINT`. 133

14.1. Sintaxis básica para la creación de un índice 161
14.2. Sintaxis de la cláusula `DROP` para índices. 162
14.3. Sintaxis de la cláusula `ANALYZE`. 163
14.4. Sintaxis de la cláusula `EXPLAIN`. 164

15.1. Ejecución del agrupamiento después de la operación de reunión. 175
15.2. Ejecución del agrupamiento antes de la operación de reunión. 176
15.3. Sintaxis de la cláusula `CASE`. 182

16.1. La consulta más externa debe esperar a la finalización de la interna. . . 188
16.2. La consulta más externa no termina aunque la interna si. 189
16.3. Sintaxis de la cláusula `SET TRANSACTION`. 193

17.1. Sintaxis de la cláusula `LOCK TABLE`. 207

20.1. Esquema general de los DSS. 232
20.2. Conformación de los sistemas OLAP. 234
20.3. Asistente para la creación de cubos en SQL Server™ 2000. 236
20.4. Editor de cubos en SQL Server™ 2000. 237
20.5. Asistente de almacenamiento de SQL Server™ 2000. 237
20.6. El examinador de cubos de SQL Server™ 2000. 239
20.7. Sentencia SQL - Información resumida respecto a consola. 240
20.8. Sentencia SQL - Información resumida respecto a tipo. 240

C.1. Diagrama Entidad - Relación. 258
C.2. Modelo Relacional. 259
C.3. Diagrama de Clases -UML. 261

D.1. Diagrama Entidad - Relación. 271
D.2. Modelo Relacional. 272
D.3. Diagrama de Clases -UML. 273

E.1. Componentes de SQL Server™ 2000. 281
E.2. Elección de una instalación local de SQL Server™ 2000. 282
E.3. Creación de una nueva instancia de SQL Server™ 2000. 283
E.4. Asignación de nombre a la nueva instancia de SQL Server™ 2000. . . . 283
E.5. Elección del servidor SQL Server™ 2000 y herramientas de cliente. . . . 284
E.6. Elección de nombre de una nueva instancia de SQL Server™ 2000. . . . 284
E.7. Elección de una instalación personalizada. 285
E.8. Selección de todos los componentes de SQL Server™ 2000. 285
E.9. Elección de características en los servicios de SQL Server™ 2000. . . . 286
E.10. Elección del método de autenticación. 287
E.11. Selección del código de caracteres de la base de datos. 288
E.12. Selección de las bibliotecas utilizadas para conectarse al servidor. . . . 288
E.13. El Administrador Corporativo de SQL Server™ 2000. 289
E.14. Selección de componentes del Servidor de Análisis. 290
E.15. El Administrador de Análisis de SQL Server™ 2000. 290
E.16. Exportar/Importar tablas de la base de datos con pgAdmin III. 291

F.1. Ventana principal de pgAdmin III. 294
F.2. Ventana donde se agrega un servidor. 295
F.3. Ventana principal del 'Query Tool'. 296
F.4. Herramienta de mantenimiento. 297
F.5. Herramienta para la creación de copias de respaldo. 298
F.6. Asistente de permisos. 298

Introducción

Los Sistemas Manejadores de Bases de Datos Relacionales (SMBDR) fueron ideados y creados como una solución al problema del manejo de la información, esto entre los años de 1965 y 1970 [1], desde entonces han surgido múltiples sistemas basados en éstos, que abordan el problema desde otra perspectiva e incluso permiten la solución de otros problemas que han surgido recientemente y que con los SMBDR 'convencionales' es más complicado de resolver, por ejemplo la 'Minería de Datos' (*Data Mining*) y el 'Procesamiento de transacciones en línea' (*Online Transaction Processing*).

En la actualidad es difícil pensar en un ámbito de nuestra vida cotidiana que no este involucrado directa o indirectamente con una base de datos y por ello es fundamental que los alumnos egresados de la carrera de Ciencias de la Computación cuenten con bases sólidas en la materia.

El objetivo de la presente tesis el mostrar una continuidad en las distintas prácticas relacionadas con la materia de Sistemas de Bases de Datos así como también con la materia de Sistemas Manejadores de Bases de Datos, pretendiendo cubrir los puntos más importantes que permitan al alumno la interacción y administración de una base de datos al aplicar intensivamente los conocimientos teóricos adquiridos durante las clases, dado que aún cuando se cuenta con gran material de apoyo (amplio material bibliográfico) en estos momentos no existe material plenamente enfocado a la revisión práctica de los conceptos que se presentan dentro del contexto de la Facultad de Ciencias, lo cual dificulta el aprendizaje y retención de los mismos.

La estructura actual que se mantiene en el plan de estudios se ha organizado de manera un poco diferente, de tal modo que se han cubierto los temas que son considerados más importantes y agregado otros que no son considerados por el plan, pero que representan una introducción a las nuevas tecnologías y tendencias actuales. En el apéndice A se hallan los temarios que se tomaron como base para la realización de las prácticas, dentro de los cuales se encuentra indicado el tiempo sugerido de duración de cada tema y la ubicación de cada práctica. Asimismo es necesario establecer que se han

tomando como parámetros principales en la elección de las prácticas, tanto el conocimiento teórico subyacente a cada práctica así como la utilidad misma que presenta para la obtención de experiencia.

Es importante recalcar que aun cuando la carrera de Ciencias de la Computación esta dedicada a la formación de científicos de la materia, por cuestiones que escapan al alcance de la tesis, el número de personas egresadas que se dedican a la investigación científica formal es muy bajo y por el contrario el resto ingresa a un mercado laboral nacional donde es muy importante contar con experiencia en el manejo de las bases de datos, dado que es mínimo el número de sistemas que no se relacionan con una base de datos de una u otra forma.

El contenido de las prácticas esta dirigido a alumnos que cursen el séptimo u octavo semestre de la carrera, considerando que el alumno ya cubrió los requisitos previos del plan de estudios de la carrera. Por lo cual no se estará entrando en detalles[1] durante el desarrollo de cada práctica, asumiendo que los alumnos cuentan con los suficientes conocimientos teóricos obtenidos durante los semestres anteriores.

El total de las prácticas, 20, se divide en dos partes y esta planeado para su aplicación durante un par de cursos de dieciséis semanas cada uno, se consideran 9 prácticas obligatorias y 1 práctica optativa para el primer curso, similarmente se presentan 9 prácticas obligatorias y 1 práctica optativa para el segundo curso.

Las prácticas siguen un esquema general[2] , que presenta al profesor y a los alumnos las metas que se planean alcanzar al finalizar la práctica, así como el objetivo general de la misma. Se continua con una introducción teórica en el tema tratado, un desarrollo general y se finaliza con una sección de preguntas y ejercicios que han de realizarse y entregarse[3] . Además se encuentran intercaladas en el desarrollo de las prácticas, una serie de actividades las cuales complementan y facilitan su realización y la comprensión del tema, mismas que se recomienda también sean entregadas.

[1] Nos referimos a todos aquellos elementos que de una u otra forma el alumno debe de manejar perfectamente en estos semestres de la carrera, por ejemplo manejo de comandos en el sistema operativo Linux.

[2] Plantilla basada en [4] y en [32].

[3] Según los documentos "Lineamientos generales sobre la entrega de prácticas para los cursos de Sistemas de Bases de Datos y Sistemas Manejadores de Bases de Datos", que se presentan en el apéndice B.

Convenciones

Durante el desarrollo del presente trabajo se irán incorporando múltiples conceptos, herramientas y en general elementos nuevos para el alumno. Para este fin se tomarán las siguientes convenciones generales.

La introducción de una nueva herramienta, se presenta mediante texto delimitado por comillas dobles y utilizando la serie de fuentes **bold**, por ejemplo:

"... **"PostgreSQL"** es un sistema manejador de bases de datos ..."

Por otra parte, la presentación de un nuevo concepto y los términos computacionales cuya traducción sea necesaria realizar, se manifiestan con un texto simple utilizando la forma de fuente *italics*. Por ejemplo:

"... para habilitar los enchufes '(*sockets*)' TCP/IP es necesario ..."

Además, la forma de fuente *slanted* es utilizada para denotar nombres de tablas y de atributos que pertenecen a las tablas. Por ejemplo, un nombre de tabla aparece de la siguiente forma:

"... la tabla *cliente* identifica a un cliente dentro de ..."

Por otro lado, el nombre de un atributo se presenta así:

"... aun cuando el atributo '*teléfono*' suele ser muy socorrido, ..."

Y la presentación de una tabla junto con sus atributos aparece de la siguiente manera:

"... las columnas de la tabla *cliente[nombre_cliente,..., ap_mat_cliente]*, permiten ..."

La familia `typewriter` se aplica para denotar los siguientes elementos:

- El prompt del intérprete de comandos, por ejemplo:

  ```
  #bash=>
  ```

- El prompt de la terminal de conexión a la base de datos. Ejemplo:

  ```
  nombre_bd=#
  ```

- Las cláusulas de SQL. Ejemplo:

  ```
  SELECT * FROM tabla;
  ```

- Las palabras reservadas de SQL. Ejemplo:

  `BEGIN, COMMIT,...`

- Mensajes en pantalla mostrados por el sistema. Ejemplo:

  ```
  (1 fila)
  ```

- Elementos aislados del código fuente escrito en algun lenguaje de programación, por ejemplo:

  "...crear una instancia de la clase '`Connection`' de JDBC ..."

Para representar código fuente en un ambiente general se utiliza el tamaño de fuente footnotesize.

De igual forma, se han determinado pequeños ambientes que determinan elementos específicos, y son los siguientes:

- **Actividad X** , denota una actividad complementaria.

- `texto` , delimita código fuente muy particular.

Primera Parte
Sistemas de Bases de Datos

Sistemas Manejadores de Bases de Datos

Semana:	Segunda semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de bases de datos. Manejo de comandos en Linux. Nociones de programación.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Conocer el proceso de instalación del SMBD PostgreSQL.

La primera práctica se realizará en la segunda semana del curso, esto porque es necesario que el profesor presente el curso a los alumnos y estén familiarizados con los conceptos básicos en el ámbito de las bases de datos.

Esta práctica es obligatoria dado que los alumnos trabajarán con el sistema manejador de bases de datos PostgreSQL durante el curso y su conocimiento es fundamental para el desarrollo de las prácticas posteriores.

PRÁCTICA 1

Sistemas Manejadores de Bases de Datos

1.1. Meta

Que el alumno se familiarice con el sistema manejador de bases de datos que utilizará durante el curso.

1.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Instalar el sistema manejador de bases de datos PostgreSQL.
- Crear bases de datos.
- Crear usuarios dentro de las bases de datos.
- Manejar opciones importantes dentro de la creación de bases de datos para un administrador de bases de datos.
- Utilizar el sistema manejador de bases de datos en cuestión.

1.3. Desarrollo

Usualmente el manejo de la información en la actualidad se lleva a cabo mediante un sistema manejador de bases de datos, generalmente relacional, que permite interactuar con la información contenida en una base de datos.

Es importante conocer un panorama general de la instalación de este software en por lo menos un sistema operativo, dado que el papel de administrador de la base de datos es primordial para el desarrollo de futuras aplicaciones.

La práctica se enfoca a dar a modo de introducción los detalles y las consideraciones básicas sobre este proceso en el caso de PostgreSQL.

1.3.1. PostgreSQL

"PostgreSQL" es un sistema manejador de bases de datos objeto-relacional que fue desarrollado por la Universidad de California en Berkley. Es de libre distribución y se encuentra bajo licencia "Berkeley Software Distribution"(BSD). Para el presente trabajo se utilizó la versión 8.0.2 que se ejecuta sobre cualquier distribución del sistema operativo **Linux**.

Actualmente se ha posicionado como una elección contra opciones comerciales ya que contiene características importantes y cumple con gran parte del estándar **SQL2003** [17], además es distribuido bajo licencia BSD la cual permite trabajar sin tener que preocuparse por pagar licencias.

La elección del sistema manejador de bases de datos se hizo basándose principalmente en los siguientes puntos:

- Gratuidad para el desarrollo de aplicaciones en el ámbito académico.
- Disponibilidad de una versión sobre la distribución Fedora Core 3 del sistema operativo Linux, que es el sistema operativo con el que se cuenta en las instalaciones de la Facultad de Ciencias.
- Gran cantidad de documentación disponible.
- Múltiples características presentes para el desarrollo.

1.3.2. Instalación

Hay dos formas principales de obtener e instalar este sistema manejador: la primera y más sencilla es simplemente seleccionando su instalación dentro de la distribución Fedora Core 3 del sistema operativo Linux, y configurárlo posteriormente. Esta primera opción, en el contexto del curso, tiene la desventaja de que al ser un mecanismo automático de instalación, el alumno no está conciente del proceso subyacente que involucra la instalación. Por otro lado, la segunda, la cual se optara por realizar, es obtener el código fuente del sistema y llevar a cabo el proceso de instalación de manera manual y general. Cabe señalar que es posible configurar múltiples características del sistema, pero para las prácticas iniciales no será necesario hacerlo en este momento.

Antes de iniciar es prioritario hacer hincapié que, si no se es el administrador del sistema operativo, entonces es necesario contar con los permisos de administrador, ya

que será requerido crear un usuario diferente en el sistema operativo, esto por razones de seguridad, así como otorgar permisos de escritura para el usuario en cuestión sobre un directorio, que es donde finalmente se almacenará la información.

La obtención del sistema manejador de bases de datos no debe ser problema para los alumnos, dado que se puede descargar el código fuente desde el sitio de PostgreSQL (`www.postgresql.org`) o bien desde cualquiera de los sitios que mantienen una copia de este (*mirrors*). Es importante poner atención en todos los requerimientos y cumplir con cada uno de ellos para evitar el recibir mensajes de error, mismos que son generalmente originados por problemas en las rutas donde se encuentran las bibliotecas compartidas o las herramientas necesarias para la compilación.

Cabe señalar que en la distribución del sistema operativo que se utiliza, la instalación de bibliotecas no debe ser problema si se seleccionan las adecuadas durante la instalación del sistema operativo. Una vez que se cuente con el código fuente descargado se procederá a una instalación típica.

Como algún usuario con los permisos de administrador (no como usuarios '*root*' pues representa problemas de seguridad y siempre es posible crear otro usuario [19]), hay que proceder a descomprimir el archivo que bajado del sitio de PostgreSQL y seguir las instrucciones que se dan en su pagina web[1] .

Actividad 1.1

Revisa tu archivo de configuración para actualizar las variables de entorno y las bibliotecas compartidas. Para más detalles consulta la documentación en:

`http://www.postgresql.org/docs/8.0/static/install-post.html`

1.3.3. postmaster

El proceso principal de PostgreSQL se llama *postmaster*. Este proceso administra la comunicación entre los procesos que se ejecutan en el servidor y aquellos iniciados por peticiones al servidor (*frontend* y *backend*). Para que una aplicación pueda tener acceso a la base de datos, debe conectarse (ya sea localmente o por medio de una conexión en red) al proceso *postmaster*, y éste ejecuta un proceso distinto dentro del servidor (*postgres*) para manejar la conexión. El proceso *postmaster* también se encarga de administrar la comunicación entre los múltiples procesos del servidor.

El proceso *postmaster* requiere en algunos casos de parámetros para su ejecución, por ejemplo el directorio de datos[2] .

[1] `http://www.postgresql.org/docs/8.0/static/installation.html`

[2] Ver lista en `http://www.postgresql.org/docs/8.0/static/app-postmaster.html`.

Adicionalmente, como se mostrará posteriormente, siempre es importante tener un registro de las actividades del SMBD. Por omisión esta información se muestra en la salida estándar, pero se puede conservar esta información, redireccionando la salida estándar a un archivo, esto se logra en Linux mediante el comando de direccionamiento ">". Ejemplo de esto es el siguiente:

```
#bash=> postmaster -i -D /directorio_de_datos > logfile
```

donde "`/directorio_de_datos`" es el lugar que se especifico en la instalación y donde se almacenan físicamente los datos. La opción "`-i`" permite al proceso estar en espera de peticiones TCP/IP y así atenderlas adecuadamente[3] .

Actividad 1.2

Dado que la instalación no deja al proceso *postmaster* como un servicio, debemos configurarlo como tal para que el sistema manejador de bases de datos se ejecute automáticamente cada vez que iniciamos la computadora. Revisa tus archivos de configuración y modifícalos para que inicie como un daemon (servicio). Para detalles más específicos revisa:

`http://www.postgresql.org/docs/8.0/static/postmaster-start.html`

1.3.4. psql

La aplicación para conectarse a PostgreSQL basada en una terminal es *psql*. Permite introducir consultas o sentencias SQL de manera interactiva y apreciar los resultados de éstas. De manera alternativa, la entrada puede provenir de un archivo externo.

Una vez que se tenga el proceso correspondiente al SMBD PostgreSQL, llamado *postmaster*, ejecutándose en el sistema operativo, es necesario llevar a cabo una conexión con el mismo, cabe aclarar que PostgreSQL buscará en forma automática, una base de datos con el nombre del usuario en cuestión. Por lo anterior, se debe tomar en cuenta que, si así se desea, se deberá crear una base de datos con el mismo nombre del usuario a la cual se va a conectar el usuario ya que en Fedora Core 3, PostgreSQL relaciona a esta con el nombre de usuario a menos que se indique lo contrario. Así que en condiciones normales (cuando el nombre de la base de datos coincide con el del usuario) simplemente bastará con ejecutar en el intérprete de comandos la siguiente instrucción:

```
#bash=> psql
```

[3] `http://www.postgresql.org/docs/8.0/static/postmaster-start.html`

además es posible crear grupos y manejar permisos específicos para grupos o usuarios, esto deja tener un mejor control sobre quienes y como interactúan con las bases de datos.

Comandos

Una vez hecha la instalación, estarán a disposición ciertos comandos, algunos permiten crear usuarios y otros las bases de datos. Solamente se utilizan en la práctica a los mas significativos y no se entra en detalle sobre sus diferentes opciones.

Existe a disposición el comando *createdb* cuya funcionalidad es crear bases de datos, y se utiliza desde el intérprete de comandos, de la siguiente forma:

```
#bash=> createdb nombre_base_de_datos
```

la instrucción anterior crea una base de datos dentro del sistema, misma que estará físicamente almacenada en el directorio de datos de la instalación. Se recomienda tener por lo menos otra base de datos donde trabajar adicional a la que genera el sistema [4] .

Para crear un usuario, se cuenta el comando *createuser*, que además de crear el usuario permite especificar si tendrá una contraseña y si este nuevo usuario tiene la posibilidad de crear a más usuarios o más bases de datos. El comando se ejecuta así:

```
#bash=> createuser nombre_usuario
```

las opciones serán mostradas interactivamente en la terminal. Hay que tener en cuenta que un usuario de la base de datos no es el mismo que un usuario dentro del sistema operativo, además de que los usuarios de inicio no tendrán acceso a una base de datos que no tenga el mismo nombre que el usuario a menos que sean especificados los permisos.

Existe además una herramienta gráfica, *pgAdminIII*, que simplifica la administración del sistema manejador y de las bases de datos que se tengan, pero por el momento no será analizada[5] dado que se desea que el estudiante lleve a cabo en este momento las tareas manualmente.

[4] *template1*

[5] Su utilización se presenta en el apéndice F - pgAdminIII.

1.4. Ejercicios

1. Crea una nueva base de datos con el nombre **cursoSBD**.

2. Crea dos usuarios nuevos de la base de datos, uno con el nombre de la base de datos y otro con uno distinto. ¿Qué sucede cuando quieres iniciar la sesión con cada uno?

3. Investiga para que sirven las opciones "`-d`" y "`-U`" del comando psql.

4. Genera un reporte de tu instalación. Indica los detalles y consideraciones más importantes.

Modelado de Bases de Datos

Semana:	Tercera semana.
Tiempo de entrega:	Dos semanas.
Prerequisitos:	Conceptos del modelo Entidad-Relación. Introducción a UML.
Herramientas:	Dia.
Objetivo:	Crear diagramas Entidad-Relación de la base de datos. Crear diagramas de clases de UML de la base de datos.

La segunda práctica debe realizarse durante la tercer semana del curso, habiendo ya el profesor impartido los conceptos relativos al modelo Entidad-Relación.

Es importante que, aun cuando no es un tema propio de las materias de este trabajo, los alumnos desarrollen una metodología para la construcción de aplicaciones utilizando los diagramas del lenguaje UML. Cabe aclarar que este último lenguaje se ha convertido en el lenguaje estándar de modelado para el desarrollo de aplicaciones.

PRÁCTICA 2

Modelado de Bases de Datos

2.1. Meta

Que el alumno aprenda a modelar una base de datos utilizando diferentes tipos de diagramas (Entidad-Relación y diagramas de clases de UML).

2.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Modelar una base de datos basándose en el modelo Entidad-Relación.
- Moldear una base de datos utilizando los diagramas de clases de UML.
- Identificar las diferencias entre los diagramas de clases de UML y los diagramas Entidad-Relación.

2.3. Desarrollo

La creación de aplicaciones de software se encuentra muy relacionada con el diseño y explotación de bases de datos. Dada esta situación, es necesario contar con herramientas capaces de representar gráficamente y en una forma entendible el modelo de las categorías de información involucradas en las aplicaciones.

Para lograr este fin será conveniente el valerse de software que cuente con una interfaz gráfica y sirva para nuestros propósitos de modelado de bases de datos.

2.3.1. Diagramas Entidad-Relación

Los diagramas Entidad-Relación constituyen una herramienta muy conveniente de representar las entidades y sus relaciones. Tienen la característica de ser altamente intuitivos por lo que la representación de los datos llevada a cabo con éstos, puede ser compartida con el usuario en etapas tempranas del análisis y diseño de la aplicación.

Es importante mencionar que este tipo de diagramas no pueden ser utilizados para representar otros elementos necesarios durante el desarrollo de los sistemas, como por ejemplo la interacción de los usuarios con las aplicaciones.

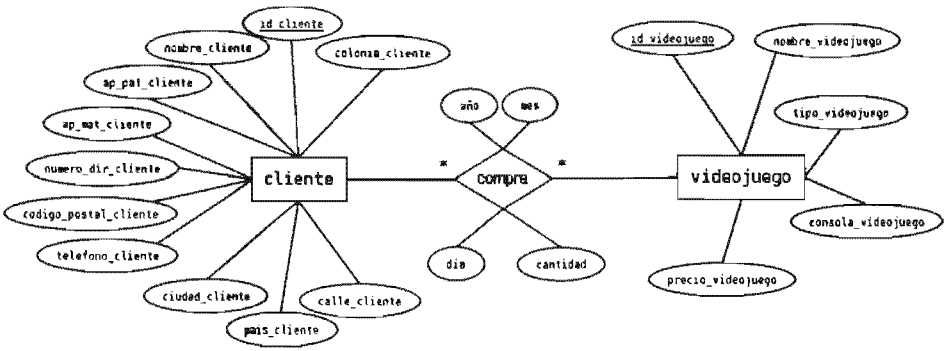

Figura 2.1: Un diagrama Entidad-Relación.

2.3.2. Dia

"Dia" es un software de dibujo basado en vectores que permite la creación de muchos tipos de diagramas, por ejemplo los diagramas de Entidad-Relación, los diagramas de clases de UML entre otros. Fue creado originalmente por Alexander Larsson y actualmente es sustentado por Cyrille Chépélov y Lars Clausen; es distribuido bajo la licencia "General Public License" (GPL) y posee características notables, por ejemplo, permite la adición de conjuntos de diagramas utilizando archivos XML y la exportación a distintos formatos gráficos. Éstas propiedades lo hacen ser la elección para nuestros propósitos.

Para el presente trabajo se asumirá que el software Dia se encuentra instalado previamente en las computadoras donde se llevan a cabo las prácticas. Lo anterior se debe a tres consideraciones importantes: la primera es con respecto a la poca utilidad práctica que representa para las materias el instalar este software, la segunda se basa en no ser necesario apegarnos a una herramienta específica y por último es casi un hecho que el software se encuentre instalado dado que por omisión el paquete completo se presenta en una instalación típica.

Actividad 2.1

Revisa donde se encuentra instalado Dia dentro de tu sistema y actualiza tus variables de entorno. Para localizarlo ejecuta el siguiente comando:

```
#bash=> which dia
```

Una vez hecho esto, simplemente ejecuta lo siguiente para iniciar el programa:

```
#bash=> dia &
```

Dia es muy sencillo de utilizar ya que cuenta con pocos elementos visuales[1] y permite un diseño fácil y rápido. Es posible apreciar la interfaz del programa en la figura 2.2, donde se presenta un área de trabajo en la parte central y el área de herramientas en la parte izquierda, es aquí donde se selecciona el tipo de elementos gráficos que se utilizarán. Siendo los que nos competen, por el momento, los de Entidad-Relación. Hay dos características que son importantes nombrar con respecto a Dia, la primera es la posibilidad de guardar el diagrama creado en un formato compatible con XML, esto permite portar los diagramas a otras aplicaciones para modificarlo. La segunda, es que también permite exportar los diagramas a otros formatos gráficos, a saber "PostScript encapsulado" (eps) y "Gráfico Portable de Red" (png) que son formatos gráficos vectoriales[2] .

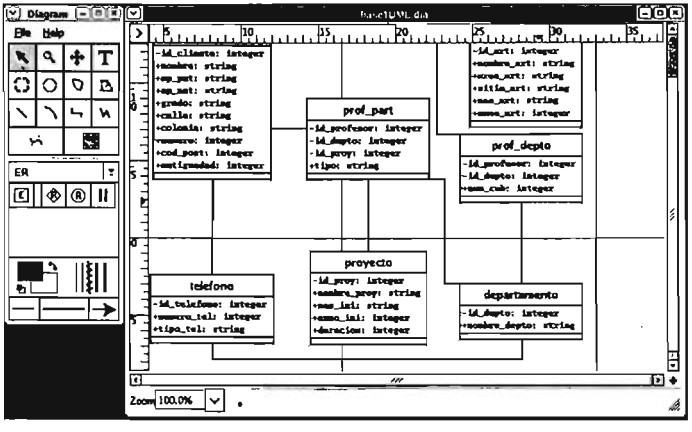

Figura 2.2: Interfaz de Dia.

[1] No significa que por eso posea pocas características.

[2] Estos formatos gráficos no permiten que se deteriore la calidad de la imagen a diferencia de los formatos gráficos de mapas de bits.

Actividad 2.2

Ejecuta Dia y crea un nuevo espacio de trabajo, genera en él tu diagrama correspondiente al diagrama Entidad-Relación de la base de datos del curso.

2.3.3. UML

El Lenguaje Unificado de Modelado (*UML - Unified Modeling Language*) es un lenguaje gráfico que permite visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML brinda una forma de modelar elementos conceptuales como lo son procesos de negocio y funciones de sistema, además de aspectos concretos como lo son las clases en un lenguaje determinado, componentes de software reutilizables y esquemas de base de datos.

Por ello dentro del desarrollo mismo de los sistemas es prioritario formalizar el modelado de las bases de datos dentro de un lenguaje gráfico que no sea desconocido por las diferentes áreas que interactúan en la creación del sistema, es aquí donde entra UML.

UML es un lenguaje estándar para el modelado de sistemas y en particular para el modelado de bases de datos, en nuestro ámbito es preciso considerar los detalles esenciales para el traslado directo de un diagrama Entidad-Relación a un tipo especial de diagrama dentro de UML, el '*diagrama de clases*'.

Diagramas

UML define 12 tipos de diagramas, que se dividen en tres categorías: cuatro de ellos sirven para representar las estructuras estáticas de las aplicaciones, cinco para representar diferentes aspectos del comportamiento dinámico; y tres para representar formas en las cuales se puede organizar y administrar los módulos de las aplicaciones.

- Los diagramas estructurales incluyen:
  1. Diagramas de clases,
  2. Diagramas de objetos,
  3. Diagramas de componentes y
  4. Diagramas de desarrollo.
- Los diagramas de comportamiento son:
  1. Diagramas de casos de uso,
  2. Diagramas de secuencia,

3. Diagramas de actividad,
4. Diagramas de colaboración y
5. Diagramas de estado.

- Los diagramas para la administración de actividad están compuestos por:

1. Paquetes,
2. Subsistemas y
3. Modelos.

El diagrama de clases

El diagrama de clases presenta un mecanismo de implementación neutral para modelar los aspectos de almacenado de datos del sistema. Las clases persistentes, sus atributos, y sus relaciones pueden ser implementadas directamente en una base de datos orientada a objetos o bien en una relacional. Aun así, en el entorno de desarrollo actual, la base de datos relacional es el método más usado para el almacenamiento de datos.

El diagrama de clases de UML se puede usar para modelar algunos aspectos del diseño de bases de datos relacionales, pero no cubre toda la semántica involucrada en el modelado relacional, mayoritariamente la noción de atributos llave que relacionan entre sí unas tablas con otras.

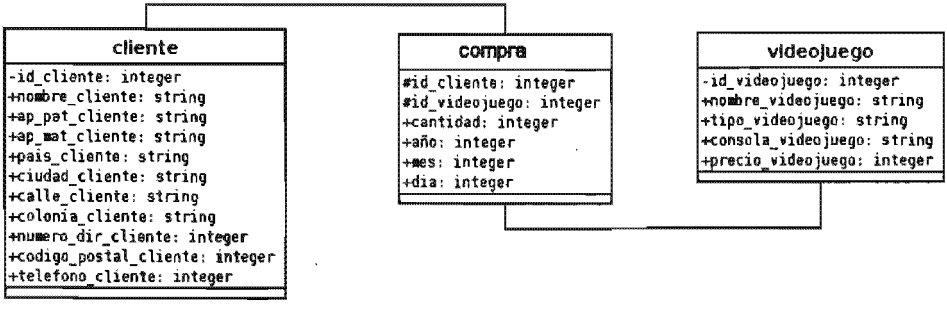

Figura 2.3: Un diagrama de clases UML.

Del modelo Entidad-Relación al Diagrama de Clases

El diagrama de clases se puede usar para modelar el estructura lógica de la base de datos, independientemente de si es orientada a objetos o relacional, con clases representando tablas, y atributos de clase representando columnas.[3] .

[3] Por esto, es recomendable mantener como anexo el diagrama del modelo Entidad-Relación.

Aun cuando no existe un algoritmo o regla general para obtener un diagrama de clases en UML a partir de un diagrama Entidad-Relación, se optará por seguir ciertas recomendaciones [2] y generar el diagrama de clases de UML a partir del diagrama Entidad-Relación generado inicialmente.

Los distintos elementos del diagrama Entidad-Relación, se convierten en elementos del diagrama de clases. Se observan las principales conversiones entre elementos de distintos modelos, en la tabla 2.1. La idea general es determinar dentro del diagrama de clases, como encaja el diagrama Entidad-Relación; que atributos son llaves primarias y cuales son llaves foráneas basándose en las relaciones con otras entidades. Construyendo, implícitamente, un modelo lógico que sea conforme a las reglas de normalización de datos.

Modelo Entidad-Relación	Diagrama de clases (UML)
Entidad	Clase
Relación	Asociación
Relación con atributos	Clase de asociación
Atributos	Atributos
Llaves	Atributos
Multiplicidad	Multiplicidad
Herencia	Herencia
Relación unaria	Agregación

Tabla 2.1: Conversión de elementos ER - UML.

Actividad 2.3

Revisa el diagrama Entidad-Relación realizado en la actividad anterior y utilizando Dia, genera su diagrama de clases de UML.

2.4. Ejercicios

1. Crea un nuevo diagrama dentro de Dia y genera el diagrama Entidad-Relación relativo a tu proyecto.

2. Crea un nuevo diagrama dentro de Dia y genera el diagrama de clases de UML relativo a tu proyecto.

Guarda tus diagramas en formato '.eps', '.dia' y '.png'.

SQL
Lenguaje de Definición de Datos

Semana:	Quinta semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos básicos de SQL.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Crear las tablas correspondientes a la base de datos.

La tercera práctica debe realizarse durante la quinta semana del curso, habiendo ya el profesor impartido los conceptos relativos al modelo Relacional y los ayudantes la introducción a SQL.

La práctica es de suma importancia dado que el lenguaje SQL es el estándar para la interacción con una base de datos y representa el primer encuentro del alumno con elementos más concretos de creación y explotación de una base de datos.

PRÁCTICA 3

SQL
Lenguaje de Definición de Datos

3.1. Meta

Que el alumno aprenda a crear las tablas relativas a un modelo lógico de una base de datos dentro de un SMBD.

3.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Crear tablas dentro del sistema manejador de bases de datos.
- Modificar los atributos de las tablas creadas.
- Conocer los tipos de datos que existen dentro del SMBD.
- Identificar los componentes del modelo relacional e implementarlos.

3.3. Desarrollo

Hasta este momento se cuenta con una representación del modelo lógico de la base de datos, ya sea con un diagrama de clases de UML o con un diagrama Entidad-Relación sin embargo no se ha instrumentado el diseño en un SMBD. Dado que se ha elegido a PostgreSQL, con los comandos básicos del estándar se podrá materializar el modelo que el alumno ha diseñando durante el curso.

Para ello, se utilizará **SQL** que es el lenguaje estándar en cuanto a definición y manipulación de bases de datos.

3.3.1. SQL

"**SQL**" (Structured Query Language) es un lenguaje estándar definido inicialmente por ANSI en 1986[1] que permite una manipulación y administración de las bases de datos, se encuentra constantemente en proceso de actualización y hoy en día el estándar más reciente es SQL2003.

La documentación es amplia[2] , el estándar se encuentra dividido en los siguientes documentos [17]: SQL/Framework, SQL/Foundation, SQL/CLI (Call-Level Interface), SQL/PSM (Persistent Stored Modules), SQL/MED (Management of External Data), SQL/OLB (Object Language Binding), SQL/Schemata, SQL/JRT (Java Routines and Types) y SQL/XML (XML-related specifications).

Para fines prácticos, dividimos a las sentencias de SQL dentro de las tres categorías siguientes:

- DDL (Lenguaje de Definición de Datos) - Estas sentencias definen la estructura de la base de datos, incluyendo registros, columnas, tablas, índices, y consideraciones específicas de la base de datos tal como su ubicación. Los principales comandos incluidos en esta parte son `CREATE` y `DROP`.
- DML (Lenguaje de Manipulación de Datos) - Estas sentencias son utilizadas para obtener y manipular datos. Aquí se abarcan los comandos fundamentales que incluyen `DELETE`, `INSERT`, `SELECT`, y `UPDATE`.
- DCL (Lenguaje de Control de Datos) - En este rubro se ubican las sentencias que permiten controlar la seguridad y permisos de los objetos o partes de la base de datos. Los principales comandos incluidos aquí son `GRANT`, `DENY`, y `REVOKE`.

Dado que se necesita la definición del modelo relacional, se presenta inicialmente el Lenguaje de Definición de Datos.

Actividad 3.1

Inicia una sesión dentro de PostgreSQL conectándote a tu base de datos.

```
#bash=> psql -d nombre_bd
```

[1] De hecho SQL se basa en SEQUEL pero hasta entonces no se había definido un estándar.

[2] Aunque la documentación del estándar no está disponible en forma gratuita, hay gran cantidad de documentos en `http://www.wiscorp.com/SQLStandards.html`

3.3.2. Creación de tablas en la base de datos

Inicialmente la base de datos no contendrá tablas útiles mas que las propias de la aplicación[3] , así que de acuerdo con el modelo lógico es necesario crear las tablas relativas al mismo. Para ello se utilizará la cláusula de SQL: CREATE. Su sintaxis básica en PostgreSQL se presenta en la figura 3.1[4] .

```
CREATE TABLE nombre_de_la_tabla (
   { nombre_columna tipo_de_dato [ DEFAULT expresion_default ]
                                 [ restriccion_columna [...] ]
   [ INHERITS ( tabla_padre [,...])] );

   donde restriccion_columna es:

{NOT NULL | NULL | UNIQUE | PRIMARY KEY | CHECK (expresion)}
```

Figura 3.1: Sintaxis básica de la sentencia CREATE.

Por el momento es importante observar que para cada entidad o clase, dependiendo del diagrama, se generará una tabla y para cada atributo presente se crea una columna dentro del modelo relacional. Con ello en mente, resta por analizar los tipos de datos que se soportan dentro de PostgreSQL. Existen múltiples tipos, pero la tabla 3.1 muestra los más importantes[5] . Cabe aclarar que por el momento sólo interesa la reducción del diagrama lógico a tablas. Los temas de integridad y normalización se tratan en las prácticas 7 y 9 respectivamente.

Actividad 3.2

Investiga que funcionalidad presenta el comando \h.
Ingresa lo siguiente:

```
nombre_bd=# \h CREATE
```

De los tipos de datos que cuenta PostgreSQL, por el momento el que interesa aclarar

[3] Hay algunas tablas de interés para alguien que sea administrador de la base de datos, por ejemplo *pg_class*.

[4] La sintaxis completa se puede consultar en : `http://www.postgresql.org/docs/8.0/interactive/sql-createtable.html`

[5] El listado completo puede consultarse en: `http://www.postgresql.org/docs/8.0/static/datatype.html`

Tipo de Dato	Descripcion
bigint	Entero con signo de 8 bytes
boolean	Valores true/false
varchar(n)	Cadena variable de n caracteres
char(n)	Cadena fija de n caracteres
date	Fechas (año, mes, día)
int	Entero con signo de 4 bytes
numeric[p,s]	Numero exacto con precisión
serial	Entero de 4 bytes de autoincremento
smallint	Entero de 2 bytes

Tabla 3.1: Tipos de datos en Postgresql.

es el `serial`, este tipo de dato es muy eficiente para manejar de inicio las llaves primarias dentro de las tablas ya que los valores que obtiene se incrementan en uno por cada tupla que agreguemos a la tabla en cuestión[6] .

Actividad 3.3

Dentro de tu sesión de PostgreSQL crea las tablas relativas a la base de datos del curso. Para crear la tabla *profesor* introduce lo siguiente:

```
nombre_bd=# CREATE TABLE profesor (
                id_profesor BIGINT PRIMARY KEY,
                nombre_profesor VARCHAR(35),
                ap_pat VARCHAR(35),
                ap_mat VARCHAR(35),
                grado VARCHAR(2),
                calle VARCHAR(35),
                colonia VARCHAR(25),
                numero SMALLINT,
                cod_post INTEGER,
                antiguedad SMALLINT
            );
CREATE TABLE
nombre_bd=#
```

Nota que la sentencia puede ir en una sola línea de texto, pero por comodidad es mejor dividirla en varias. Es importante observar que todas las sentencias terminan con ';'. Termina de agregar las restantes tablas.

[6] Es posible manejar también esto con una secuencia, de hecho es más eficiente, pero por el momento se utilizará el tipo de dato `serial`.

Una vez ejecutada la o las sentencias CREATE, y si no se recibe algún mensaje de error por parte del SMBD, podemos verificar la estructura de la tabla recién creada ejecutando el siguiente comando dentro de nuestra base de datos:

```
nombre_bd=# \d nombre_de_tabla
```

Ejecutando sobre la tabla *profesor* obtendremos una salida semejante a la siguiente:

```
nombre_bd=# \d profesor
            Tabla "public.profesor"
    Columna      |     Tipo      |  Modificadores
-----------------+---------------+------------------
id_profesor      | bigint        | PRIMARY KEY
nombre_profesor  | varchar(35)   |
ap_pat           | varchar(30)   |
ap_mat           | varchar(30)   |
grado            | varchar(2)    |
calle            | varchar(40)   |
colonia          | varchar(20)   |
numero           | smallint      |
cod_post         | smallint      |
Indices:
      id_profesor_pkey PRIMARY KEY, btree <id_profesor>

nombre_bd=#
```

3.3.3. Eliminación de tablas

Posiblemente durante el proceso de creación de las tablas se presenten errores, ya sea por un mal diseño o simplemente por un error en la captura, para cualquiera que sea el caso, SQL provee de la cláusula DROP, que permite la eliminación de una tabla[7] .

Existe la posibilidad de modificar alguno de los atributos que fuimos creando en nuestra base de datos, pero dado que aún estamos en las etapas iniciales del proceso de creación, ciertamente es mejor eliminar la tabla y crearla nuevamente. Para el presente trabajo se utilizará la cláusula DROP, su sintaxis se presenta en la figura 3.2.

[7] De hecho DROP permite eliminar de casi cualquier estructura, incluso la base de datos.

```
DROP TABLE nombre_tabla [,...] [ CASCADE | RESTRICT ]
```

Figura 3.2: Sintaxis de la sentencia DROP.

Actividad 3.4

Dentro de tu sesión de PostgreSQL verifica la estructura de la tabla *profesor* relativa a la base de datos del curso. Posteriormente eliminala.
¿Qué mensaje aparece en la terminal de psql?
¿Existe algún tipo de protección para evitar eliminar una tabla? ¿Consideras que es necesario?
Agrega nuevamente la tabla *profesor*.

3.3.4. Modificación de tablas

En el caso de contar con un información almacenada en una tabla, en la cual posteriormente observamos existen elementos que no representan totalmente la realidad que estamos modelando, es conveniente modificar la estructura de la tabla para obtener una representación más precisa. SQL ofrece la sentencia ALTER para modificar una variedad de objetos presentes en un SMBD, su sintaxis dentro de PostgreSQL se presenta en la figura 3.3.

El proceso de modificación de una tabla es costoso, en términos de los recursos utilizados por el SMBD, y directamente proporcional al tamaño de la tabla, por eso es recomendable llevar a cabo un diseño cuidadoso y utilizar esta sentencia únicamente en casos realmente necesarios.

Actividad 3.5

Dentro de tu sesión de PostgreSQL obtén la estructura de la tabla *profesor* y modifica los siguientes atributos:
1. Asigna al atributo *nombre_profesor* el nombre de *nombre*.
2. Asigna al atributo *grado* un nuevo tamaño de 35.
Ahora verifica la estructura de la tabla *profesor*.

```
ALTER TABLE [ ONLY ] nombre_tabla [ * ] acción [,...];

donde acción es alguna de las siguientes :

        RENAME columna TO nueva_columna
        RENAME TO nuevo_nombre
        ADD columna tipo_dato [ restricción_columna[...] ]
        DROP columna [ RESTRICT | CASCADE ]
        ALTER columna TYPE tipo_dato [ USING expresión ]
        ALTER columna SET DEFAULT expresión
        ALTER columna DROP DEFAULT
        ALTER columna  SET | DROP  NOT NULL
        ADD restricción_tabla
        DROP CONSTRAINT nombre_restricción[ RESTRICT | CASCADE ]
```

Figura 3.3: Sintaxis básica de la sentencia ALTER.

3.3.5. Proceso por lote

En general, es conveniente contar con un archivo que contenga las sentencias que construirán nuestra base de datos, esto permite enfocarse más a cuestiones de diseño y evitar en cierto grado los errores de sintaxis, además, facilita el análisis de la base de datos una vez creada. Asimismo representa una herramienta que permite migrar el esquema de la base de datos a otro SMBD en forma rápida y segura.

Por lo general, los archivos que contengan instrucciones SQL, se guardarán con la extensión '*.sql*'. Cabe aclarar que la extensión que tengan los archivos no es relevante para PostgreSQL dado que este SMBD puede procesar cualquier archivo con sentencias validas de SQL aunque posea cualquier extensión.

Actividad 3.6

Inicia una sesión en editor de textos e ingresa las sentencias CREATE para crear las tablas relativas a la base de datos del curso, *cursoSBD*. Por ejemplo, con emacs:

```
#bash=> emacs creaBD.sql
```

Para iniciar el procesamiento de las instrucciones contenidas en un archivo *.sql*, se tienen varias opciones, y cada una es mejor que otra dependiendo de la situación que se

presente. La primera forma consiste en ejecutar desde PostgreSQL, las sentencias SQL contenidas en el archivo .sql en cuestión con la opción \i Por ejemplo:

```
nombre_db=# \i creaBD.sql
```

de este modo, el sistema presentará en pantalla mensajes informativos correspondientes a la ejecución de las sentencias SQL incluidas en el archivo.

La segunda forma de lograr lo anterior, es ejecutar desde una terminal "shell"[8] el comando psql con la opción "-f". Ejemplo:

```
#bash=> psql -f creaBD.sql
```

Modificando la salida estándar

PostgreSQL permite modificar la salida estándar hacia un archivo donde se guarden todos los resultados y mensajes que en condiciones normales se presentan en pantalla. La ventaja de esto radica en agrupar los resultados en un solo archivo para su posterior procesamiento. Para hacer uso de esta característica dentro de PostgreSQL basta con iniciar la conexión a la base de datos con psql y suministrar por párametro la opción "-o {nombre_archivo}". Donde 'nombre_archivo' representa el archivo donde se almacenan los resultados. El nombre del archivo puede proporcionarse por medio de una ruta relativa al área de trabajo o bien una ruta absoluta.

Por último hay que mencionar que todos los mensajes se almacenan en el archivo en cuestión y esto puede ser un problema al analizar una consulta dada, puesto que no se perciben los resultados en pantalla.

[8] bash en nuestro caso.

3.4. Ejercicios

1. Crea un listado donde muestres que tipo de dato, de los soportados por PostgreSQL, corresponde a cada uno de los atributos de las tablas de tu base de datos. Lleva a cabo la misma actividad pero con la base de datos del curso.

2. Crea las tablas relativas a tu base de datos y a la base de datos del curso, siguiendo tu diagrama de clases y diagrama Entidad-Relación. Para las llaves primarias utiliza un tipo de dato serial y guarda tus sentencias SQL en un archivo con la extensión *.sql*.

3. Verifica tu esquema, analizando cada una de las tablas con el comando \d.

SQL
Lenguaje de Manipulación de Datos

Semana:	Sexta semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos básicos de SQL. Álgebra relacional.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Realizar operaciones de inserción, eliminación y actualización en la base de datos.

La práctica cuatro es importante dado que es el primer encuentro del alumno con las sentencias de manipulación de datos del lenguaje SQL, las cuales son las más comunes y utilizadas dentro de la explotación de datos. Por ello es importante que los alumnos manejen bien los conceptos del modelo Relacional y estén familiarizados con el álgebra relacional.

PRÁCTICA 4

SQL
Lenguaje de Manipulación de Datos

4.1. Meta

Que el alumno aprenda los fundamentos de la manipulación de datos, como la inserción, actualización y eliminación de información de la base de datos.

4.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Insertar tuplas dentro de la base de datos.
- Eliminar tuplas de la base de datos.
- Actualizar valores de las tuplas dentro de la base de datos.

4.3. Desarrollo

Aún cuando ya se creo la base de datos dentro del SMBD, es preciso agregar los datos. Esta tarea es sumamente importante dado que un proceso de carga de los datos de entrada con errores puede ocasionar múltiples fallas como pueden ser errores de integridad o duplicidad de información que dejen en un estado inconsistente a la base de datos.

4.3.1. Inserción de datos

Se presentan dos formas de insertar datos, la primera más genérica que es por medio de SQL y que teóricamente muestra la manera estándar de realizar el proceso. La segunda, utilizando un proceso "por lote". Que se aplica particularmente al SMBD PostgreSQL dentro de Fedora Core 3, pero sirve de base para generar procesos semejantes dentro de otro SMBD e incluso dentro de otro sistema operativo.

Inserción con SQL

Antes de iniciar el proceso de inserción de datos, es recomendable verificar que se cuenta por un lado con los datos y por otro con la tabla ya creada, es importante aclarar que si bien ya se encuentran cargados los datos, la tabla puede ser modificada (alterada) como se presento anteriormente. Para verificar la estructura de la tabla, es preciso recordar la utilización del comando \d. La sentencia INSERT se encarga de agregar los datos a las tablas, es importante tomar en cuenta el orden de los atributos, pues la sentencia sigue el orden estipulado a menos que se indique lo contrario. Su sintaxis básica se presenta en la figura 4.1.

```
INSERT INTO nombre_tabla [ ( nombre_columna [,...] ) ]
             { VALUES ({ expresion | DEFAULT }[,...] ) | consulta }
```

Figura 4.1: Sintaxis básica de la sentencia INSERT.

En la base de datos del curso, *cursoSBD*, una sentencia INSERT típica es la siguiente:

```
nombre_bd=# INSERT INTO profesor VALUES ( 1,'Hugo', 'G', 'G', 'M en C',
'Calle 1', 'Colonia 1', 1, 04420);
```

o bien, utilizando el valor por omisión dentro del atributo *id_profesor*, la sentencia luciría de la siguiente forma:

```
nombre_bd=# INSERT INTO profesor VALUES ( DEFAULT, 'Hugo', 'G', 'G',
'Calle 1', 'Colonia 1', DEFAULT, 04420);
```

Hay que notar dos aspectos, primeramente el orden en que se colocan los datos después de la palabra VALUES es importante ya que existe una correspondencia uno a uno entre los datos insertados y los atributos definidos de la tabla en cuestión. Si se desea modificar el orden de los datos por insertar, los atributos deben aparecer explícitos, en nuestro ejemplo la sentencia que se presenta a continuación es equivalente a la anterior:

```
nombre_bd=# INSERT INTO profesor ( nombre, ap_pat, id_profesor, ap_mat
```

```
grado, calle, colonia, numero, cod_postal) VALUES ('Hugo','G',DEFAULT,
'G', 'M en C', 'Calle 1', 'Colonia 1', 1, 04420);
```

observese que el utilizar un atributo de tipo serial resulta en un manejo más eficiente tanto para el diseñador como para el SMBD, esto porque internamente el manejo de números es más sencillo que el manejo de cadenas alfanuméricas. Además la representación interna de este tipo de datos consiste en números que garantizan la creación de valores únicos dentro del dominio que tienen las llaves primarias.

Inserción por lote

Aún cuando la inserción de datos puede hacerse de manera directa con el SMBD vía una terminal, esta es un tanto ineficiente para el manejo de grandes cantidades de datos ya que estamos expuestos a cometer errores "de dedo". Para evitar este tipo de errores podemos realizar la inserción utilizando un archivo que contenga las sentencias `INSERT`. En última instancia esto es una herramienta que permite dejar al sistema procesar la información contenida en archivos de un usuario.

El principio básico se fundamenta en asegurar que la información por ingresar a la base de datos se encuentra depurada y en cierta forma, es accesible al sistema de manera transparente. Actualmente la mayoría de los sistemas manejadores permiten importar los datos contenidos en otros archivos al sistema, siempre que presenten cierto formato definido por uno mismo. En el caso particular de PostgreSQL, utilizaremos el comando `COPY` de la siguiente forma:

```
nombre_bd=# COPY nombre_tabla FROM archivo WITH DELIMITER 'caracter';
```

el comando permite importar datos de archivos externos, e insertarlos directamente dentro de la tabla en cuestión, hay que remarcar que es importante el orden en que aparecen los datos, ya que a diferencia de la sentencia `INSERT`, aquí no es posible modificar el orden en el cual se insertan los datos.

El caracter '`caracter`' que aparece al final de la sentencia puede ser cualquier símbolo que delimite a los campos dentro del archivo.

Actividad 4.1

Inicia una sesión dentro de PostgreSQL conectándote a la base de datos.

```
#bash=> psql -d nombre_bd
```

Ingresa por lo menos 15 tuplas dentro de la tabla correspondiente a la tabla de *profesor* y una con los siguientes datos:

id_profesor	=	1	grado	=	'M en C'
nombre	=	'Hugo'	calle	=	'Calle 1'
ap_pat	=	'G'	colonia	=	'Colonia 1'
ap_mat	=	'G'	numero	=	20
cod_post	=	04430			

4.3.2. Eliminación de datos

A efecto de eliminar información de nuestra base de datos, desde una *tupla* hasta el total de una tabla, se cuenta con el comando DELETE. Su sintaxis es la siguiente:

```
DELETE FROM [ ONLY ] nombre_tabla [ WHERE condiciones_booleanas ]
```

Figura 4.2: Sintaxis de la sentencia DELETE.

Como podemos observar, la sentencia es relativamente sencilla sin embargo es importante resaltar tres aspectos. El primero y más importante radica en que la condición booleana no es mas que el valor de una expresión que regresa valores del tipo boolean y determina las tuplas que serán eliminadas; segundo, que aun cuando aparece la condición booleana como un parámetro optativo, es necesario especificarlo siempre pues el omitirlo provoca un borrado total de los datos contenidos en la tabla. El tercer aspecto importante es que las condiciónes booleanas pueden generarse a partir de una simple comparación o hasta del resultado de una consulta sobre otra tablas.

Ejemplo de lo segundo es el siguiente:

```
nombre_bd=# DELETE FROM profesor;
```

donde se ha procedido a vaciar la tabla de profesor ya que como no hay condición alguna, se eliminan todos los registros. Ejemplo del tercer aspecto es el siguiente:

```
nombre_bd=# DELETE FROM profesor WHERE id_profesor = 1 ;
```

ahora, solamente se esta eliminando el profesor al cual le corresponde el '*id_profesor*' con valor de 1. Por último se tiene el siguiente ejemplo sobre eliminación de datos:

```
nombre_bd=# DELETE FROM profesor WHERE id_profesor IN (SELECT id_art
FROM articulo);
```

La eliminación de datos es un proceso importante dado que en caso de falla durante el mismo, siempre tendrá consecuencias en la nueva captura de los datos. Aun así, se ha visto que también es un proceso muy sencillo.

Por último, hay que notar que aunque existe cierta semejanza entre los comandos `DELETE` y `DROP`, existe un diferencia importante la cual es que `DROP` elimina por completo el esquema que define a la tabla en cuestión, mientras que `DELETE` solamente elimina los elementos que se encuentran dentro de esta, por ello `DELETE` resulta una operación muy costosa cuando la base de datos ha crecido considerablemente.

Actividad 4.2

Inicia una sesión dentro de PostgreSQL conectándote a tu base de datos. Y elimina dos tuplas de tu tabla *profesor*, además elimina la tupla correspondiente al profesor 'Hugo G G'.

¿Por qué la operación de `DELETE` resulta ser costosa para el sistema una vez que la base de datos ha crecido considerablemente?

4.3.3. Actualización de datos

El hecho de que tengamos la información dentro de la base de datos no es, ni por mucho, el fin del proceso de explotación de la misma. Generalmente las aplicaciones que interactúan con la base de datos se encargan de actualizarla, así por ejemplo, en una base de datos de alumnos su promedio se actualiza semestralmente o en una base de datos de cuentas bancarias, el saldo de cada socio comercial se puede actualizar incluso cada segundo.

En SQL el comando que proporciona esta funcionalidad es `UPDATE`. Su sintaxis básica es la siguiente[1] :

```
UPDATE [ ONLY ] nombre_tabla SET columna =
               { expresion | DEFAULT }[,...]
               [ WHERE condiciones_booleanas ];
```

Figura 4.3: Sintaxis básica de la sentencia `UPDATE`.

[1] La documentación completa se encuentra en: `http://www.postgresql.org/docs/8.0/static/sql-update.html`

La utilización de UPDATE es similar a DELETE y por esto, se deben tener en cuenta las mismas precauciones que se nombraron anteriormente.

Actividad 4.3

Inicia una sesión dentro de PostgreSQL conectándote a la base de datos del curso y actualiza la tupla correspondiente a 'Hugo G G' modificando el valor de la colonia por el nuevo valor de: 'Colonia Nueva'.

Actualiza el atributo '*grado*' de todas las tuplas de la tabla *profesor* con el valor de: 'M. en C.'.

4.4. Ejercicios

1. Inserta 25 tuplas más a cada tabla de tu base de datos, para las tablas que corresponden a relaciones dentro del diagrama Entidad-Relación, inserta 35 tuplas. Realiza lo mismo para las tablas de la base de datos del curso.

2. Crea un archivo por lote para que inserte 36 tuplas más a cada tabla de tu base de datos y otras 35 tuplas a la base de datos del curso.

3. Elimina 10 tuplas de cada una de las tablas de la base de datos.

4. Elimina todas las tuplas de la tabla *articulo* si el atributo '*año*' tiene un valor inferior a 1979.

5. Genera un archivo por lote que actualice los siguientes elementos de la base de datos:

   a) A todas las tuplas de la tabla *articulo* actualizar el atributo '*año*' con el valor de 2005.

   b) A todas las tuplas de la tabla *proyecto* actualizar el atributo '*duracion*' con el valor de 12 si el atributo '*mes_ini*' tiene el valor de 'Enero'.

   c) A todas las tuplas de la tabla *profesor* actualizar el atributo '*grado*' con el valor de 'Dr.' si el atributo '*antiguedad*' tiene un valor superior a 5.

6. Genera un archivo por lotes que elimine las tablas de la base de datos.

7. Existe un método alternativo a la utilización de `COPY` para insertar datos a la base de datos. ¿En qué consiste?[2]

Guarda todas tus sentencias en un archivo con la extensión *.sql*

[2] Pista: Para ello se utiliza el comando `\i algo.sql`.

SQL
Lenguaje de Manipulación de Datos (II)

Semana:	Séptima semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos básicos de SQL. álgebra Relacional.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Realizar consultas sobre la base de datos.

La práctica cinco resulta ser una de la más importantes en este curso, ya que trata la parte de manipulación de datos, en el ámbito de consultas, siendo la herramienta más utilizada.

Es importante que se desarrollen en clase una serie de ejercicios generales que muestren el funcionamiento de las cláusulas estándar de SQL, y en la práctica se generen más ejemplos de consultas, esto para que el funcionamiento de cada uno sea mejor comprendido.

Del mismo modo, los ayudantes deberán haber realizado múltiples ejercicios dentro del álgebra relacional y SQL, para que la práctica se lleve a cabo dentro del plazo previsto.

PRÁCTICA 5

SQL
Lenguaje de Manipulación de Datos (II)

5.1. Meta

Que el alumno aprenda fundamentos de la creación de consultas a una base de datos.

5.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Realizar consultas a la base de datos.

5.3. Desarrollo

Con los datos cargados dentro de la base de datos, queda por delante lo más importante, que son las consultas a la misma. De nada sirve el hecho de tener información almacenada si no es posible explotarla y es aquí donde el Lenguaje de Manipulación de Datos entra en su rol más importante, ya que desde su concepción misma su finalidad ha sido obtener información de los datos almacenados y presentarlos en un formato sencillo para su posterior procesamiento o explotación. El proceso mismo de la consulta inicia desde la formulación y determinación de "¿Qué es lo que se desea obtener?", ya que si desde un inicio no se tiene noción de la información que se solicita, entonces el proceso será más lento y presentará mayor complicación.

A partir de esta noción preliminar es posible obtener conjuntos y subconjuntos de datos que permiten ir construyendo la solución a la consulta mediante pasos intermedios. Por esto, se ha puesto especial atención a la manera en que se irán construyendo consultas, cada vez más complejas, pero que se basan en resultados previos.

Es necesario hacer notar que se sugiere abarcar la mayor parte de las cláusulas mostradas aquí, ya que aunque no forman un porcentaje significativo del estándar de SQL, si permiten abarcar gran parte de las consultas más comunes.

Por otro lado, es importante que los alumnos cuenten con una base de datos no muy grande en cuanto al número de tuplas, puesto que al ser las primeras consultas generadas, estas no serán del todo óptimas. El tema de optimización forma parte del segundo grupo de prácticas.

5.3.1. Consultas

Estructura básica

La estructura básica de la cláusula SELECT de SQL se presenta en la figura 5.1. Se considera básica pues en el estándar es el conjunto mínimo de cláusulas necesarias para generar una sentencia que regrese información de la base de datos. Aún cuando en algunos SMBD ciertas cláusulas son opcionales para la obtención de otro tipo de información, por ejemplo PostgreSQL [16], no la consideramos como estructura básica, ya que no forma parte del estándar.

```
SELECT [DISTINCT] lista_atributos
FROM tablas
[WHERE condiciones_booleanas];
```

Figura 5.1: Estructura básica de la sentencia SELECT.

La primera cláusula corresponde a la palabra reservada SELECT que permite obtener atributos que pertenecen una tabla determinada y corresponde a la operación de proyección del álgebra relacional (π). La cláusula FROM indica las tablas que se consideran y contienen los atributos que se determinan en la lista '`lista_atributos`', su equivalencia en el álgebra relacional es el producto cartesiano o la unión ($\times$ y $\bowtie$ respectivamente). Por último, la cláusula WHERE permite denotar expresiones que retornan valores booleanos y determinan las tuplas que se encontraran dentro del resultado y corresponde a su vez a la operación de selección del álgebra relacional (σ).

Tomando en principio la base de datos del curso, se puede obtener la información del atributo '*nombre*' de la tabla *profesor* de la siguiente forma (una vez conectados a la base de datos):

En este caso, simplemente se está haciendo una proyección sobre el atributo '*nombre*' de la tabla *profesor*. El resultado es un conjunto de tuplas. Obviamente, es posible

```
nombre_bd=# SELECT nombre FROM profesor;
  nombre
-----------
  Hugo
    ⋮
  Ana
 (45 filas)

nombre_bd=#
```

aumentar la lista de atributos por seleccionar simplemente añadiéndo los nombres de estos consecutivamente y separándolos por comas.

Por ejemplo, si deseamos agregar a nuestro resultado, el apellido paterno y el grado máximo de estudios de cada uno de los profesores, entonces la consulta queda de la siguiente forma:

```
nombre_bd=# SELECT nombre, ap_pat, grado FROM profesor;
  nombre  |  ap_pat   |  grado
----------+-----------+---------
  Hugo    |     G     |  M. en C.
    ⋮     |     ⋮     |     ⋮
  Ana     |   Cosio   |  M. en C.
 (45 filas)

nombre_bd=#
```

En general, si se desean obtener todos los atributos de las tablas, en lugar de denotarlos explícitamente, podemos recurrir al símbolo "*", en lugar de la lista. Generalmente deseamos solamente tener en nuestro resultado un conjunto de tuplas que cumplan con alguna restricción, por ejemplo que sean mayores a una constante o se encuentren dentro de un rango. Si es el caso recurrimos a la cláusula `WHERE`. En ella podemos denotar un conjunto de condiciones booleanas, las cuales son expresiones que se evalúan con un tipo boolean y determinan las tuplas de nuestro resultado. Se muestran las principales en la tabla 5.1.

Hay que tener en cuenta que es posible obtener conjuntos de tuplas tal que se repita un valor dentro del mismo conjunto, si es el caso y se desea que los resultados sean únicos dentro del conjunto buscado, hay que utilizar la palabra reservada `DISTINCT`, que elimina las tuplas que tienen valor repetido.

Tipo de expresión	Elemento SQL	Ejemplo
Igualdad	=	`WHERE atributoK = 'Cadena';`
Desigualdad	<>	`WHERE atributoK <> 20;`
Mayor que	>	`WHERE atributoK > 20;`
Menor que	<	`WHERE atributoK < 20;`
Rango	BETWEEN...AND	`WHERE atributoK BETWEEN valor1 AND valor2;`
Pertenencia	IN	`WHERE atributoK IN (1, 2, 3);`
Existencia	EXIST	`WHERE EXISTS (SELECT atributoK FROM tabla2 WHERE atributoL = Y);`
"Y" lógico	AND	`WHERE atributo1 = X AND atributo2 = Y AND...AND atributoN = Z;`
"O" lógico	OR	`WHERE atributo1 = X OR atributo2 = Y OR...OR atributoN = Z;`
No pertenencia	NOT	`WHERE atributoK NOT IN (1, 2, 3);`

Tabla 5.1: Principales expresiones dentro de la claúsula WHERE

Actividad 5.1

Inicia una sesión dentro de PostgreSQL conectándote a tu base de datos y revisa la documentación de la sentencia SELECT identificando cada una de las claúsulas que se han mencionado. Además verifica la existencia de las tablas que contiene tu base de datos.

```
nombre_bd=# \h SELECT
nombre_bd=# \d
```

Ordenamiento

En SQL, también se cuenta con una serie de elementos que permiten obtener de una manera ordenada los resultados de una consulta. La cláusula que logra esto es ORDER BY. Su sintaxis se muestra en la figura 5.2.

```
SELECT lista_atributos
FROM tabla
[WHERE condiciones_booleanas]
ORDER BY columnas [ASC | DESC];
```

Figura 5.2: Sintaxis de la cláusula ORDER BY.

El resultado se ordena con respecto a la o las columnas que se indiquen en cláusula. Además, como se puede apreciar, es valido indicar el orden deseado, ya sea ascendente o descendente. Hay que tomar en cuenta que el ordenamiento, implica un proceso adicional a la consulta[1] , por lo que hay que ser cauteloso con su utilización.

Un ejemplo de esta operación en la base de datos del curso es el siguiente:

```
nombre_bd=# SELECT nombre, ap_pat FROM profesor ORDER BY ap_pat ASC;
  nombre  |  ap_pat
----------+-----------
  Jordi   |  Austrich
    ⋮     |     ⋮
  Julio   |  Zapata
 (45 filas)

nombre_bd=#
```

[1] El proceso adicional solamente se presenta cuando no existe alguna operación de reunión (JOIN).

Funciones escalares básicas

El estándar SQL2003 [17] define múltiples funciones escalares sobre un conjunto de tuplas que tienen su dominio en valores escalares[2] (enteros, enteros sin signo, flotantes, flotantes dobles, enteros "grandes", etc ...), pero existe un subconjunto que le daremos atención especial pues su utilización es altamente requerida para resolver consultas comunes. Este subconjunto de funciones escalares, así como su descripción se muestran en la tabla 5.2.

Nombre	Descripción
SUM	Suma el valor número de los elementos.
COUNT	Cuenta el número de los elementos.
AVG	Promedio del conjunto.
MAX	Valor número mayor del conjunto.
MIN	Valor número menor del conjunto.
COUNT(DISTINCT)	Cuenta el número de elementos diferentes.

Tabla 5.2: Principales funciones escalares.

Así, para obtener el total de profesores que han publicado por lo menos un articulo, procedemos de la siguiente forma: se busca en la tabla *prof_art* cuantos '*id_profesor*' distintos existen:

```
nombre_bd=# SELECT COUNT(DISTINCT id_profesor) FROM prof_art;
  count
----------
      7
 (1 fila)
```

Del resultado obtenido, es importante hacer hincapié en lo siguiente:

1. El resultado es una tabla con un renglón y una columna.
2. El nombre de la columna es el nombre de la función.
3. En la cláusula SELECT, se puede invocar mas de una función, en cuyo caso la tabla resultante tendrá una columna por cada función.
4. El tipo de dato del resultado, es entero.
5. No es posible el anidamiento de funciones, no esta por demás comentar que no son validas expresiones como la siguiente `SUM(MAX(COUNT(atributo))`.

[2] De hecho algunas de estas funciones también abarcan otros dominios como los caracteres o texto.

Actividad 5.2

Dentro de tu sesión en PostgreSQL, experimenta con las cláusulas relativas a las funciones escalares. ¿Todas hacen sentido sobre columnas donde el tipo dato es algún numérico? ¿Y con columnas de tipo 'Date/Time' ?

Dentro de tu base de datos ejecuta la siguiente sentencia:

```
nombre_bd=# SELECT MAX(COUNT(cod_post)) FROM profesor;
```

¿Porqué crees que no es valido el anidamiento de funciones?
En una consulta, para encontrar el máximo de un conteo, ¿Cómo procederías?

Operadores de conjuntos

Los resultados obtenidos en las expresiones estudiadas en esencia son tablas, y como tales pueden usarse como operandos en otras expresiones. Se dice que dos tablas son compatibles entre si, si poseen el mismo grado, orden y dominio para cada uno de sus atributos que contiene. Los principales operadores se muestran en la tabla 5.3 y su sintaxis se muestra en la figura 5.3.

Operador	Descripción
UNION	Unión de todos los elementos
INTERSECT	Elementos que están en ambos conjuntos simultaneamente.
EXCEPT	Elementos que están en un conjunto pero no en otro

Tabla 5.3: Operadores de conjuntos de tuplas.

```
    (sentencia_select)
{UNION | INTERSECT | EXCEPT }
    (sentencia_select);
```

Figura 5.3: Uso de los operadores de conjuntos de tuplas.

Con respecto a la sintaxis de estas operaciones hay que mencionar que las sentencias que aparecen en el rubro de `sentencia_select` pueden ser cualquiera que no contenga

alguno de los operadores ORDER BY, LIMIT o FOR UPDATE [3] .

Operador IS NULL

Se sabe que las bases de datos relacionales utilizan una marca especial para denotar la ausencia de valor en un atributo dado, a saber NULL (η).

Dado que el tratamiento de esta marca es diferente a los demás valores de los diferentes tipos de datos, la prueba de existencia se lleva a cabo con el predicado IS NULL. Un ejemplo de lo anterior es el siguiente:

```
nombre_bd=# SELECT nombre FROM profesor WHERE ap_mat IS NULL;
  nombre
----------
 (0 filas)
```

Equivalencia en cadenas

Aun cuando se cuenta con el operador de igualdad en cadenas, en ocasiones es necesario encontrar tuplas que cumplan con algún patron de texto similar entre ellas, para eso se utilizan los operadores LIKE y SIMILAR TO. LIKE fue la primera aproximación de SQL para el manejo de expresiones regulares en cadenas, su sintaxis se presenta en la figura 5.4.

```
SELECT lista_atributos
FROM tabla
WHERE {condiciones_boolenas |
       cadena [NOT] {LIKE | SIMILAR TO} patrón [ESCAPE caracter]};
```

Figura 5.4: Sintaxis de los operadores LIKE y SIMILAR TO.

En este caso, se quiere encontrar las tuplas tales que en la cadena cadena se encuentre o no el patrón denotado en patrón. Para la definición del patrón, existen dos símbolos especiales, el primero es el guión bajo (_) que sirve para representar la equivalencia con un único caracter, el segundo corresponde al símbolo de porcentaje (%) que concuerda cualquier cadena con cero o más caracteres en ella. Si el patrón no contiene algún símbolo de porcentaje o guión bajo, entonces el operador LIKE actúa como una simple igualdad de cadenas de caracteres.

[3] A excepción de UNION que permite la utilización de ORDER BY y de LIMIT siempre y cuando se encuentren entre paréntesis las sentencias SELECT.

El caracter que continua después de ESCAPE se toma como el caracter de escape para caracteres o símbolos especiales, por omisión el caracter que se usa es el de diagonal invertida (\). Ejemplos de su utilización, se aprecian en la tabla 5.4.

Sentencia.	Descripción.	Ejemplos
`SELECT atributo FROM tabla WHERE atributoK LIKE 'a%';`	Regresa las tuplas tales que el atributoK empiece con la letra 'a' y tenga 0 o más caracteres	'a', 'aT', etc.
`SELECT atributo FROM tabla WHERE atributoK LIKE 'a_%';`	Regresa las tuplas tales que el atributoK empiece con la letra 'a' y tenga 1 o más caracteres	'aR', 'aqRt', etc.
`SELECT atributo FROM tabla WHERE atributoK LIKE '_a_';`	Regresa las tuplas tales que el atributoK empiece con un ca--racter, tenga una 'a' y otro ca--racter.	'Qaw', 'maT', etc.
`SELECT atributo FROM tabla WHERE atributoK LIKE '__%';`	Regresa las tuplas tales que el atributoK contenga por lo menos dos caracteres.	'aa','UOP', etc.

Tabla 5.4: Ejemplos de utilización de la cláusula LIKE.

Dada la poca versatilidad que se presentó con el operador LIKE, para el estándar de SQL99 se introdujo el operador SIMILAR TO [15]. Su sintaxis es similar a LIKE y se muestra en la figura 5.4.

La funcionalidad del operador SIMILAR TO es muy semejante a LIKE[4] , excepto por el hecho de que interpreta los patrones utilizando las definiciones estándar de SQL respectivas a expresiones regulares. Estas últimas son una forma híbrida entre la notación usada por LIKE y la notación común de expresiones regulares.

Al igual que LIKE, SIMILAR TO tiene éxito solamente si el patrón concuerda completamente con la cadena, hecho que no sucede en las expresiones regulares comunes,

[4] La opción de ESCAPE funciona exactamente igual que con LIKE.

donde el patrón puede concordar con cualquier parte de una cadena. Del mismo modo, `SIMILAR TO` utiliza como caracteres comodines los símbolos de guión bajo (_) y de porcentaje (%) para uno o varios caracteres respectivamente. Adicionalmente, `SIMILAR TO` ofrece soporte para los siguientes metacaracteres tomados de las expresiones regulares *POSIX*:

- | denota una elección entre dos o más alternativas.
- * denota repetición del elemento anterior cero o más veces.
- + denota repetición del elemento anterior una o más veces.
- ( ) sirven para agrupar varios elementos como un elementos lógico único.
- [...] especifican clase de caracteres, tal cual en las expresiones regulares POSIX[5]

Actividad 5.3

Dentro de tu sesión en PostgreSQL realiza la siguiente consulta:

```
nombre_bd=# SELECT nombre FROM profesor
            EXCEPT
            SELECT cod_post FROM profesor WHERE cod_post = 04430;
```

¿Qué te indica el sistema?, ahora realiza la siguiente consulta:

```
nombre_bd=# SELECT nombre FROM profesor
            UNION
            SELECT ap_mat FROM profesor ORDER BY ap_mat;
```

¿Qué te indica el sistema? Modifica la claúsula `ORDER BY` para que ordene el resultado por *nombre*. ¿Qué sucede? ¿Por que si lleva a cabo la consulta?

[5] Por ejemplo [A..Z] denota cualquier carácter entre la letra "A" y la letra "Z".

5.4. Ejercicios

Verifica que se encuentre la tabla *profesor* y a continuación realiza las siguientes consultas desde la línea de comandos de PostgreSQL, crea un archivo con extensión *.sql* para guardar posteriormente las consultas que vayas generando.

Obten:

1. Los nombres de los profesores, tal que sean distintos entre ellos.
2. Cuantos códigos postales distintos se tienen registrados.
3. El nombre de los profesores que tienen un código postal mayor a 57100.
4. El nombre de los profesores que tienen un código postal mayor a 57100 pero menor que 98000.
5. El nombre de los profesores cuyo apellido paterno es 'Lopéz' y tienen un código postal menor a 57100.
6. El nombre de los profesores que tienen un código postal igual a 03400, 04420 o 57100.
7. La cantidad de profesores que existe.
8. El nombre, apellido paterno y apellido materno de los profesores, tales que su apellido paterno empiece con la letra 'M', además ordena tu resultado por el nombre de manera ascendente.
9. El nombre, apellido paterno y apellido materno de los profesores, tales que su apellido materno empiece con la letra 'C', además ordena tu resultado por el nombre de manera descendente.
10. El nombre, apellido paterno y apellido materno de los profesores, tales que su apellido paterno empiece con la letra 'M' y su apellido materno empiece con la letra 'C' , sin utilizar el operador lógico `AND`.
11. El nombre de los profesores tales que no tienen un grado de 'M. en C' , utilizando la claúsula `EXCEPT`.
12. El nombre de los profesores tales que no tienen asignado un departamento.
13. La cantidad de nombres de profesores que existe.

SQL
Lenguaje de Manipulación de Datos (III)

Semana:	Novena semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos básicos de SQL. Álgebra Relacional.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Realizar consultas más elaboradas sobre la base de datos.

La práctica seis continua con la creación de consultas. Utilizando para ello un número más amplio de sentencias SQL que permiten encontrar solución a un conjunto mayor de consultas.

Se insiste nuevamente en que los ayudantes lleven a cabo el mayor número posible de ejercicios.

PRÁCTICA 6

SQL
Lenguaje de Manipulación de Datos (III)

6.1. Meta

Que el alumno aprenda más sentencias SQL para responder a una mayor variedad de consultas a las bases de datos.

6.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Crear sentencias SQL para responder a consultas de mayor complejidad.

6.3. Desarrollo

Hasta ahora se presentado un conjunto de consultas relativamente sencillas. La cantidad de consultas que pueden responderse efectivamente con las sentencias SQL anteriores, es mínimo. Para obtener información de la base de datos generalmente se tienen que hacer operaciones más complejas que involucran el manejo de las cláusulas anteriormente presentadas y las que se estudian en esta práctica.

6.3.1. Consultas

Agrupamiento

El agrupamiento, permite reunir las tuplas de una relación en conjuntos que comparten una o varias características tomando en consideración un atributo específico (o varios). Esto se efectúa mediante la sentencia `GROUP BY`. Su sintaxis se aprecia en la figura 6.1.

Es importante considerar que el SMBD al iniciar el proceso de esta cláusula crea grupos con respecto a los distintos valores que se encuentren en el primer atributo obtenido. Posteriormente se crea una nueva división dentro de los grupos con respecto al segundo atributo y así sucesivamente. Se muestra un ejemplo de esto en la figura 6.2.

```
SELECT atributo1, atributo2,...,atributoN
FROM tabla
[WHERE condiciones_booleanas]
GRUOUP BY atributo1, atributo2,...,atributoN;
```

Figura 6.1: Sintaxis de la cláusula `GROUP BY`.

Generalmente las consultas que se realizan sobre propiedades o características presentes en un grupo, pueden ser objeto de uso de las funciones escalares, es decir, primeramente la cláusula `GROUP BY` genera un determinado número de grupos, tantos como distintos valores se encuentren en las tuplas con respecto al atributo en cuestión, y a cada uno de estos grupos formados se puede aplicar la función escalar, por ejemplo `SUM( )` ó `MAX( )`.

A modo de ejemplo, si la tabla *profesores* contiene los siguientes valores:

Tabla:	profesores				
id_profesor	nombre	ap_pat	ap_mat	antigüedad	grado
1	Javier	Garcia	Garcia	5	M. en C.
2	Max	Neummian	Coto	5	Lic.
3	Jordi Inaki	Austrich	Senosiain	6	M. en C.
4	Ana Elena	Escalante	Hernadez	8	Lic.
5	Jorge	Perez	Lopez	6	Fis.
6	Jose	Galaviz	Casas	5	M. en C.
7	Miguel	Lara	Aparicio	5	Lic.

y se ejecuta una consulta para contar el total de profesores por antigüedad, se obtiene lo siguiente:

```
nombre_bd=# SELECT COUNT(id_profesor), antiguedad
            FROM profesores GROUP BY 'antiguedad';
 count(int8) | antiguedad
 ------------+------------
       1     |     8
       2     |     6
       4     |     5
 (3 filas)
nombre_bd=#
```

Es importante mencionar que si se suministra un atributo a la cláusula SELECT que no sea el mismo por el cual se esta agrupando o no se le aplique a este una función escalar, entonces es necesario agrupar con respecto al atributo dentro de la cláusula GROUP BY.

Selección sobre grupos

En diversas circunstancias, surge la necesidad de encontrar características propias de cada grupo creado, para ello se utiliza la cláusula HAVING que funciona de manera similar a la cláusula WHERE pero aplicado a grupos. Su sintaxis se muestra en la figura 6.3.

Actividad 6.1

Dentro de tu sesión en PostgreSQL realiza la siguiente consulta:

```
nombre_bd=# SELECT COUNT(nombre), grado FROM profesor
            GROUP BY grado;
```

¿Es seguro que obtengamos el número de profesores por grado académico?
¿Y con la siguiente consulta?

```
nombre_bd=# SELECT COUNT(id_profesor), grado FROM profesor
            GROUP BY grado;
            HAVING COUNT(*) >10;
```

¿Qué función tiene la sentencia COUNT(*) >10?

Tabla:	profesores				
id_profesor	nombre	ap_pat	ap_mat	antigüedad	grado
1	Javier	García	García	5	M. en C.
2	Max	Neumman	Coto	5	Lic.
3	Jordi Iñaki	Austrich	Senosiain	6	M. en C.
4	Ana Elena	Escalante	Hernández	8	Lic.
5	Jorge	Pérez	López	6	Fis.
6	José	Galaviz	Casas	5	M. en C.
7	Miguel	Lara	Aparicio	5	Lic.

a) `GROUP BY 'antigüedad'`

↓

Tabla:	profesores			*antigüedad*	
id_profesor	nombre	ap_pat	ap_mat	*antigüedad*	grado
4	Ana Elena	Escalante	Hernández	8	Lic.
3	Jordi Iñaki	Austrich	Senosiain	6	M. en C.
5	Jorge	Pérez	López	6	Fis.
1	Javier	García	García	5	M. en C.
2	Max	Neumman	Coto	5	Lic.
6	José	Galaviz	Casas	5	M. en C.
7	Miguel	Lara	Aparicio	5	Lic.

b) `GROUP BY 'antigüedad','grado'`

↓

Tabla:	profesores			*antigüedad, grado*	
id_profesor	nombre	ap_pat	ap_mat	*antigüedad*	*grado*
4	Ana Elena	Escalante	Hernández	8	Lic.
3	Jordi Iñaki	Austrich	Senosiain	6	M. en C.
5	Jorge	Pérez	López	6	Fis.
1	Javier	García	García	5	M. en C.
6	José	Galaviz	Casas	5	M. en C.
2	Max	Neumman	Coto	5	Lic.
7	Miguel	Lara	Aparicio	5	Lic.

Figura 6.2: Representación de una ejecución `GROUP BY` con respecto a dos atributos.

```
SELECT atributo1, atributo2,...,atributoN
FROM tabla
[WHERE condiciones_booleanas]
GRUOUP BY atributo1, atributo2,...,atributoN
HAVING condiciones_sobre_grupos;
```

Figura 6.3: Sintaxis de la cláusula HAVING.

Consultas sobre varias tablas

Debido al proceso de diseño de las bases de datos, la información se almacena en distintas tablas, esto permite eliminar múltiples efectos no deseados, como la redundancia de datos o la inconsistencia de los mismos, por lo tanto es común la necesidad de obtener datos contenidos en más de una tabla durante una consulta dada. Para realizar esta tarea se cuenta con la operación de reunión de tablas (o JOIN) y el producto cartesiano de tablas que son operaciones equivalentes a las definidas dentro del álgebra relacional ($\bowtie$ y $\times$ respectivamente).

El producto cartesiano, es una operación implícita dentro de una consulta, que se lleva a cabo si aparece más de una tabla en la cláusula FROM. También se cuenta con el operador explícito CROSS JOIN que es equivalente.

Tabla T1		
id	peso	nombre
1	10	nombre1
2	15	nombre2
3	16	nombre3

Tabla T2		
id	nombre	peso
2	nombre2	15
4	nombre4	25
5	nombre5	5

Figura 6.4: Tablas ejemplo para el producto cartesiano.

El producto cartesiano se lleva a cabo teniendo como operandos las tablas especificadas en la cláusula FROM[1] . A modo de ejemplo sobre el funcionamiento del producto cartesiano, consideremos a las tablas *T1* y *T2* que se muestran en la figura 6.4.

Para realizar el producto cartesiano entre las dos tablas, se procede con la siguiente consulta[2] :

Si se desea obtener una reunión de tablas (reunión natural ($\bowtie$) o reunión theta

[1] Ya sea tablas explicitas o bien tablas que proceden de la realización de una sub-consulta

[2] El mismo resultado se obtiene al emplear la cláusula `SELECT * FROM T1 CROSS JOIN T2;`

```
nombre_bd=# SELECT * FROM T1, T2;
 id  | peso | nombre  | id  | nombre  | peso
-----+------+---------+-----+---------+------
  1  |  10  | nombre1 |  2  | nombre2 | 15
  1  |  10  | nombre1 |  4  | nombre4 | 25
  1  |  10  | nombre1 |  5  | nombre5 |  5
  2  |  15  | nombre2 |  4  | nombre4 | 25
  2  |  15  | nombre2 |  2  | nombre2 | 15
  2  |  15  | nombre2 |  5  | nombre5 |  5
  3  |  16  | nombre3 |  5  | nombre5 |  5
  3  |  16  | nombre3 |  2  | nombre2 | 15
  3  |  16  | nombre3 |  4  | nombre4 | 25
(9 filas)

nombre_bd=#
```

($\bowtie_\theta$)) hay que utilizar la cláusula `WHERE` para denotar la expresión booleana que deben cumplir las tuplas. Por ejemplo:

```
nombre_bd=# SELECT * FROM T1, T2 WHERE T1.id = T2.id;
 id  | peso | nombre  | id  | nombre  | peso
-----+------+---------+-----+---------+------
  2  |  15  | nombre2 |  2  | nombre2 | 15
(1 fila)

nombre_bd=#
```

La reunión entre tablas, representa el método más utilizado para encontrar información que se encuentre distribuida en varias tablas, actualmente es preferible utilizar la sintaxis que acontinuación se especifica dado que es más clara.

```
SELECT atributos
FROM tabla1 [NATURAL]
            [INNER | [LEFT | RIGHT | FULL] [OUTER] | CROSS]
            JOIN tabla2 [ON condición] ;
```

Figura 6.5: Sintaxis básica de la cláusula `JOIN`.

Obsérvese los diversos tipos de reuniones, de los cuales el `NATURAL JOIN` es el más utilizado por varias razones, la principal es que realiza el apareamiento de las tuplas pertenecientes a las tablas que recibe como operandos tomando en cuenta aquellos

atributos que comparten los mismos nombres en las tablas (siempre y cuando estos atributos posean los mismos dominios). Las tres últimas reuniones, `[LEFT | RIGHT | FULL] [OUTER] JOIN` evitan que las tuplas de una tabla que no contengan elementos que no tengan una contraparte en la otra, sean desechadas. Las tuplas que no se apareen, se agregan al resultado con marcas `NULL`.

Actividad 6.2

Dentro de tu sesión en PostgreSQL realiza lo siguiente:

Crea cuatro tablas con las siguientes sentencias:

```
1. CREATE TABLE tablaX (identificador bigint PRIMARY KEY,
                        nombre varchar(10));
2. CREATE TABLE tablaY (id bigint PRIMARY KEY,
                        identificador varchar(10));
3. CREATE TABLE tablaZ (identificador bigint PRIMARY KEY,
                        nombre varchar(10));
4. CREATE TABLE tablaW (nombre varchar(10),
                        identificador bigint PRIMARY KEY);
```

Ahora ingresa en tu base de datos las siguientes sentencias:

```
1. INSERT INTO tablaX VALUES (1, 'Hugo');
2. INSERT INTO tablaX VALUES (2, 'Paco');
3. INSERT INTO tablaX VALUES (3, 'Luis');
4. INSERT INTO tablaX VALUES (4, 'Tony');
5. INSERT INTO tablaY SELECT * FROM tablaX;
6. INSERT INTO tablaZ SELECT * FROM tablaX;
7. INSERT INTO tablaW SELECT nombre, identificador FROM tablaX;
```

¿Qué sucede con las siguientes consultas?

```
1. SELECT * FROM tablaX NATURAL JOIN tablaY;
2. SELECT * FROM tablaX NATURAL JOIN tablaZ;
3. SELECT * FROM tablaX NATURAL JOIN tablaW;
```

¿Cómo funciona el operador `NATURAL JOIN`?

Anidamiento de consultas

El resultado de una consulta es a su vez una tabla que puede ser utilizada como operando en otra consulta.

Es interesante observar que podemos utilizar el resultado de una función escalar (`MAX( )`, `MIN( )`, etcétera) como operando de una cláusula `WHERE` siempre y cuando se ejecuten correctamente. Considérese el siguiente ejemplo:

```
nombre_bd=# SELECT nombre
            FROM  profesores
            WHERE  profesores.antiguedad =   (SELECT max(antiguedad)
                                              FROM profesores);
  antiguedad
 ---------
    25
 (1 fila)
nombre_bd=#
```

También es posible utilizar el resultado de una consulta dentro de la cláusula `FROM` para especificar que la operación de reunión de tablas se realizará sobre una nueva tabla creada en tiempo de ejecución, es decir, no existe físicamente como tal dentro de nuestra base de datos. Un ejemplo de esto es el siguiente:

```
nombre_bd=# SELECT AP.nombre, AP.ap_pat, AM.ap_mat FROM
                (SELECT nombre, ap_pat FROM profesores) AS AP,
                (SELECT ap_mat FROM profesores) AS AM
            WHERE AP.ap_pat = AM.ap_mat;
  nombre | ap_pat | ap_mat
 --------+--------+--------
   Hugo  | Garcia | Garcia
 (1 fila)
nombre_bd=#
```

Para utilizar una tabla creada a partir de una consulta es necesario utilizar la cláusula `AS`, que es equivalente a un renombramiento dentro del álgebra relacional (ρ). Es importante remarcar que el nuevo nombre asignado a la tabla temporal no debe causar conflictos con la o las tablas que se encuentren en un ambiente superior.

La división

La división en el álgebra relacional es un operador muy útil ya que permite encontrar respuesta a las consultas del tipo "¿Qué entidades están aparejadas con la totalidad de otro conjunto de entidades?". Desafortunadamente no existe un operador de división relacional en SQL[3] , así que esta funcionalidad debe ser expresada en terminos de otros operadores.

Se han desarrollado varias soluciones para abordar este problema por ejemplo [7, Date] y [28, Ramakrishnan], pero nos basaremos en una definición un tanto más intuitiva presentada en [22, Victor M. Matos, Rebecca Grasser].

Considérese las tablas *publica[id_profesor,id_art]* y *articulos[id_art]* mostradas en la figura 6.6. La tabla *publica* representa la lista de los profesores y los artículos en los cuales han participado. El primer atributo '*id_profesor*' es la identificación del profesor y el segundo atributo '*id_art*' representa el número de identificación del artículo. Así, si un profesor ha participado en la publicación de un artículo, en esta tabla aparecerá su identificación junto con el artículo donde ha participado. La tabla *articulos[id_art]* representa el total de los artículos publicados hasta el momento.

La tabla *resultado[id_profesor]* muestra la lista de identificadores de los profesores que han participado en TODOS los artículos publicados (mostrados en la tabla *articulos[id_art]*). La versión alternativa de la división implementada en SQL se presenta en la figura 6.7 y permite encontrar el conjunto mostrado en la tabla *resultado*.

Tabla: publica	
id_profesor	id_art
1	1
1	2
2	3
2	2
2	1
3	2
3	1

Tabla: articulos
id_art
1
2
3

Tabla: resultado
id_profesor
2

Figura 6.6: Tablas de ejemplo para la operación de división.

Esta versión de la división utiliza cláusulas de pertenencia (`IN`), de agrupación (`GROUP BY`), de conteo (`COUNT`) y de condición sobre grupos (`HAVING`). La cláusula `GROUP BY id_profesor` es responsable de crear una tupla de cada subconjunto de la partición efecutada sobre el atributo '*id_profesor*'.

La sentencia `WHERE` selecciona de la tabla *publica*, aquellas tuplas que el valor del atributo '*id_art*' aparezca en la tabla *articulos*, de tal forma que se seleccionan los profesores que han participado en algun(os) articulo(s). Por último, se cuenta el número de

[3] En SQL se cuenta con el operador de división '/' pero sobre valores escalares.

```
SELECT id_profesor
FROM publica
WHERE id_art IN (SELECT id_art FROM articulos)
GROUP BY id_profesor
HAVING COUNT(*) = (SELECT COUNT (*) FROM articulos);
```

Figura 6.7: Versión alternativa de la división en SQL.

elementos de cada partición (de cada profesor) y sólo se seleccionan aquellos profesores cuyo número de participaciones es igual al total de artículos (cláusula `HAVING`).

Es importante comentar que para garantizar un funcionamiento adecuado de esta expresión hay que verificar el que las tuplas contenidas en las tablas sean únicas.

6.3.2. Vistas

Las vistas son un elemento importante de SQL, su principal utilidad es que permiten tener "visiones a la medida". En principio, una vista es simplemente el renombramiento de una consulta arbitraria, con la ventaja de encapsular los detalles de la estructura de las consultas y tablas que en ella participan, instrumentando de esta forma la independica lógica de los datos, además de facilitar con ello la creación de nuevas consultas más complejas.

La vista no es materializada sino que la expresión asociada a ella se ejecuta cada vez que se hace alguna referencia a la vista. Actualmente dentro de PostgreSQL, las vistas son únicamente de consulta pero se puede 'simular' el funcionamiento de una vista actualizable al crear reglas que permitan la inserción, actualización o borrado en la vista manejando las acciones en otras tablas. La utilización de operaciones diferentes a las consultas sobre vistas, dada su complejidad, aún no ha sido resuelta satisfactoriamente ya que éstas operaciones no se pueden realizar directamente sobre la vista sino que deben traducirse a operaciones equivalentes sobre las tablas reales lo que ocasiona problemas complejos de integridad y numerosos efectos secundarios.

Aún así, las ventajas de utilizar vistas son múltiples:

- Se pueden crear vistas que tienen identificadores de los atributos diferentes a los de las tablas originales sin tener que duplicar los datos.

- Se pueden utilizar para la instrumentación de la seguridad de los datos. No todos los usuarios pueden ver o conocer la totalidad del esquema conceptual. Por lo que a cada usuario se le puede proporcionar sus vistas correspondientes. Pueden existir datos ocultos a unos usuarios y conocidos a otros según las limitaciones referentes asociadas a cada usuarios.

```
CREATE VIEW nombre [ ( nombre_columnas [, ...] ) ] AS consulta ;
```

Figura 6.8: Sintaxis de la cláusula CREATE VIEW.

- Permiten limitar las posibles operaciones (consultas) que se puedan realizar sobre los datos.
- Simplifican la interfaz con el usuario.
- Permiten denominar a una operación compleja con un nombre simple y expresivo bajo el cual pueda ser invocada más fácilmente.
- Favorecen la independencia lógica de los datos. Se puede variar la estructura de la base de datos sin necesidad de efectuar cambios profundos en los programas, sino tan sólo en las vistas.
- Facilitan la presentación de los datos y realizan una presentación lógica de estos.

Un ejemplo de la creación de una vista, es el siguiente:

```
nombre_bd=# CREATE VIEW datos_Generales AS
            SELECT nombre, ap_pat, ap_mat
            FROM profesor
            WHERE antiguedad > 15;
CREATE VIEW
```

Actividad 6.3

Dentro de tu sesión en PostgreSQL crea una vista con el nombre de *profesor_t* que incluya los siguientes atributos de la tabla 'profesor':

1. '*nombre*'
2. '*grado*'
3. '*antiguedad*'

Tal que solamente contenga a aquellos profesores con una antigüedad menor de diez años y un grado académico de 'Doctor'

Intenta agregar la siguiente tupla a esta vista recién creada:

```
nombre_bd=# INSERT INTO profesor_t VALUES ('René',5 ,'Doctor');
```

¿Qué te indica el sistema?

6.4. Ejercicios

Verifica que se encuentre la tabla '*profesores*' y a continuación realiza las siguientes consultas desde la línea de comandos de PostgreSQL y crea un archivo con extensión *.sql* para guardar posteriormente las consultas que vayas haciendo.

Recuerda que puedes crear vistas para facilitar las consultas cuando obtienen proporciones relativamente grandes.

Obten:

1. El nombre, departamento y grado máximo de estudios de cada uno de los profesores.
2. Las distintas antigüedades entre todos los profesores.
3. El nombre de los profesores que tienen una antigüedad mayor a 10 años y han participado en proyectos con una duración mayor de 8 meses.
4. El nombre y apellidos de los profesores cuyo apellido paterno es 'Lopéz' y tienen una antigüedad menor de 15 años.
5. El nombre y apellidos de los profesores que tienen una antigüedad igual a 5, 10 o 20 años.
6. El promedio de antigüedad de todos los profesores.
7. El nombre y departamento de los profesores ordenados por departamento de manera descendente.
8. El nombre y apellidos de los profesores que pertenecen al departamento de 'Matemáticas' ordenados según su antigüedad de manera ascendente.
9. El nombre y apellidos de los profesores con la antigüedad máxima de entre todos los profesores.
10. La cantidad de profesores por cada tipo distinto de grado máximo de estudios.
11. El nombre y apellidos de los profesores que tienen una antigüedad menor a la antigüedad media de todos los profesores.
12. El nombre y apellidos de los profesores que pertenecen a TODOS los departamentos.
13. El nombre y apellidos de los departamentos que contienen a TODOS los profesores.
14. El nombre y apellidos de los profesores que tienen más de 2 números telefónicos.

15. El nombre y apellidos de los profesores y su relación con los números telefónicos, es decir, cuantos números telefónicos tiene asociado cada profesor, en caso de no tener asociado alguno, este debe aparecer también.

16. El nombre y apellidos de los profesores que han participado en todos los artículos publicados en el año de 2003.

17. Los nombres y apellidos de los profesores que pertenecen al departamento de 'Física' y han participado en más de dos proyectos.

18. INTERSECT es una cláusula cuya funcionalidad puede ser emulada por otras. Genera una expresión equivalente a la siguiente:

```
(SELECT nombre FROM profesor WHERE ap_pat = García)
INTERSECT
(SELECT nombre FROM profesor WHERE ap_mat = García)
```

19. Considera la siguiente consulta:

```
SELECT MAX(SUM) FROM
       (SELECT SUM(COUNT) FROM
               (SELECT COUNT(numero) FROM profesor) AS tmp1) AS tmp2;
```

Si se ha dicho que el anidamiento de funciones escalares no es válido. ¿Por qué se ejecuta correctamente? Justifica tu respuesta.

Integridad

Semana:	Décimo primera semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos básicos de SQL. Conceptos básicos de integridad.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Implementar los distintos tipos de integridad.

La práctica siete se presenta como una de las más sencillas de realizar, dado que las restricciones de integridad debieron ser vistas en clase por los profesores y ayudantes con anterioridad suficiente y de ser posible, ya se cuente con una idea clara sobre este tema.

PRÁCTICA 7

Integridad

7.1. Meta

Que el alumno aprecie la importancia de los diferentes tipos de integridad presentes en una base de datos y las medidas mínimas para su mantenimiento.

7.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar los tipos de integridad.
- Entender la importancia de cada tipo de integridad.
- Identificar posibles violaciones a las reglas de integridad dentro de una base de datos.

7.3. Desarrollo

Hasta ahora se ha visto la estructura de una base de datos como una abstracción que puede representarse de varias formas y la mejor depende del problema que se quiera resolver. También se ha hecho un primer acercamiento a la implementación de la base de datos pero no se han tomado las consideraciones para crear un buen diseño del modelo de la realidad.

Se considera un buen diseño a aquel que protege, en la medida de lo posible, contra la cometida de errores de inconsistencia y brinda cierto nivel de integridad. Hay que tener

en cuenta el no depender de la verificación de los datos de entrada en las aplicaciones que se comunican con la base de datos, sino utilizar las facilidades provistas por el SMBD que lleven a cabo este tipo de verificación.

7.3.1. La Integridad

La integridad es una propiedad que garantiza que el modelo de la realidad sea lo más parecido a esta y se conserve consistente con ella de manera constante por el tiempo. Se han definido múltiples tipos de integridad [28], pero para nuestro estudio sólo se consideran los siguientes:

- Integridad de entidad.
- Integridad de dominio.
- Integridad de usuario.
- Integridad referencial.

Integridad de entidad

La integridad de entidad se refiere a que cualquier información del sistema se puede identificar por medio de tres elementos:

1. La relación a la que pertenece (tabla).
2. La columna a la que esta relacionado (atributo).
3. Identificación individual dentro de la tabla donde se encuentra.

La tercera regla de integridad se aplica a las llaves primarias de las relaciones base: Ninguno de los atributos que componen la llave primaria puede ser nulo.

Por definición, una llave primaria es un identificador irreducible que se utiliza para identificar de modo único las tuplas. Que es irreducible significa que ningún subconjunto de la llave primaria sirve para identificar las tuplas de modo único. Si se permite que parte de la llave primaria sea nula, se está diciendo que no todos sus atributos son necesarios para distinguir las tuplas, con lo que se contradice la irreducibilidad. Hay que notar que esta regla sólo se aplica a las relaciones base y a las llaves primarias, no a las llaves foráneas, aunque generalmente la aparición de una llave foránea con una marca `NULL` es un error. Para definir una llave primaria dentro de una tabla, se sigue la sintaxis presentada en la figura 3.1, de las opciones presentes, es importante conocer el funcionamiento de `NOT NULL`, la cual indica que el atributo en cuestión no puede contener valores nulos; por su parte `UNIQUE`, que indica que un valor dentro de una columna es único dentro del mismo.

Es importante hacer notar que una restricción `UNIQUE` no se cumple cuando existen dos o más tuplas en la tabla donde los valores de todas las columnas incluidas en la restricción son idénticos. Sin embargo, las marcas `NULL` no son consideradas equivalentes en una comparación, esto significa que aún en la presencia de una restricción `UNIQUE` es posible encontrar un número ilimitado de tuplas que tengan la marca `NULL` en la columna de la restricción.

En PostgreSQL, toda la información se almacena en el catálogo del sistema [28], que es a fin de cuentas una tabla más.

Actividad 7.1

Investiga que información almacena la tabla del sistema *pg_class* y después revisa su contenido.

```
nombre_bd=# SELECT * FROM pg_class;
```

Integridad de dominio

Este tipo de integridad se da sobre el conjunto de valores que puede tomar cada una de las tuplas pertenecientes a una tabla (su dominio). Este dominio puede estar o no definido por el SMBD[1] pero siempre y automáticamente se generan reglas que verifican el cumplimiento de este tipo de integridad. Por ejemplo, si se crea la siguiente tabla *temporal*:

```
nombre_bd=# CREATE TABLE temporal (
              id_temporal bigint);
CREATE TABLE
nombre_bd=#
```

entonces el sistema al recibir una sentencia del tipo:

```
nombre_db=# INSERT INTO temporal VALUES ('Cadena');
ERROR: invalid input syntax for integer: "Cadena"
nombre_bd=#
```

regresa una advertencia de que no se logro la inserción ya que los tipos de datos no coinciden.

[1] Existe la posibilidad de crear nuevos tipos de datos a partir de los definidos en el sistema.

Durante la creación de la tabla se puede tener una expresión en la definición de cada columna para determinar el dominio de los valores que puede tener cada una de las tuplas de la tabla. El sistema se encarga de revisar (por cada una de las tuplas que se inserten) que cumplan con estas restricciones de dominio. La sintaxis para definir alguna regla de integridad de dominio se muestra en la figura 7.1.

```
CREATE TABLE nombre_tabla (
  {nombre_columna tipo_dato
  [DEFAULT expresión | restricción | restriccion_tabla]}
);
```

Figura 7.1: Sintaxis básica de las restricciones soportadas por PostgreSQL.

Dentro de 'expresión' puede suministrarse cualquier expresión[2] válida de SQL que regrese un valor apropiado dentro del dominio original del dato, esto es, si se tiene un `bigint` como tipo de dato en la definición entonces dentro de la expresión se debe tener alguna que regrese un entero. Si se presenta en la definición como tipo de dato un `varchar( )` entonces la expresión debe regresar una cadena, y así sucesivamente. Si se denota una 'restricción', ésta puede definirse con las cláusulas `NULL`, `UNIQUE`, `NOT NULL` entre otras, pero es de particular interés la claúsula `CHECK`, la cual permite definir restricciones sencillas dentro del domino de nuestros atributos. Por ejemplo es posible tener lo siguiente:

```
nombre_db=# CREATE TABLE dias_y_positivos (
       positivos bigint CHECK (positivos > 0)
       dias varchar(10) CHECK (dias IN ('Lunes', 'Domingo'))
       );
CREATE TABLE
nombre_db=#
```

y las inserciones subsecuentes a la creación de la tabla, tendrán que cumplir con estas restricciones de dominio. Es importante mencionar que la cláusula `CHECK` no puede manejar sub-consultas como restricción.

[2] Puede ser una función misma, por ejemplo, en la creación de índices se recurre a la función 'nextval' definida para las secuencias.

Actividad 7.2

Dentro de tu sesión en PostgreSQL, revisa las opciones con las que cuenta PostgreSQL para la integridad de dominio.

```
nombre_db=# \h CREATE TABLE
```

Experimenta con las claúsulas relativas a la creación de secuencias y la utilización del tipo de dato 'serial'.
¿Cuál es más conveniente? ¿Por qué?

Integridad de usuario

La integridad de usuario esta relacionada con la autorización que tiene cada usuario de la base de datos y su interrelación con otros usuarios y con la base de datos misma. Es tema más avanzado de administración y no se profundiza en él sino hasta la siguiente práctica.

Integridad referencial

La integridad referencial consiste en un sistema de reglas que utilizan la mayoría de las bases de datos relacionales para asegurarse que las tuplas de tablas relacionadas son válidas y que no se borren o cambien datos relacionados de forma accidental produciendo errores de consistencia.

Cuando se define una columna como llave foránea, las tuplas de la tabla pueden contener en esa columna la marca `NULL` o bien un valor que existe en la otra tabla. Eso es lo que se denomina integridad referencial y consiste en que los datos que hacen referencia a otros (llaves foráneas) deben ser correctos. La integridad referencial hace que el SMBD se asegure de que no hayan en las llaves foráneas valores que no estén en la tabla referenciada.

La integridad referencial se activa en cuanto es creada una llave foránea y a partir de ese momento se comprueba cada vez que se modifiquen datos que puedan alterarla. Los escenarios en los cuales se pueden producir errores en los datos contenidos en una llave foránea son:

- Cuando se inserta una nueva tupla en la tabla y el valor de la llave foránea no existe en la tabla referenciada.

- Cuando se modifica el valor de la llave primaria de una tupla que tiene referencias en otras tablas.

- Cuando se modifica el valor de la llave foránea, el nuevo valor debe existir en la tabla referenciada.
- Cuando se elimina una tupla de la tabla principal y esa tupla tiene referencias en otras tablas.

Asociada a la integridad referencial están los conceptos de actualizar los registros en cascada y eliminar registros en cascada. El actualizar y/o eliminar registros en cascada, son opciones que se indican cuando es definida la llave foránea y que informan al SMBD qué acciones llevar a cabo en los casos comentados en el punto anterior.

Actualizar registros en cascada: Indica al SMBD que al modificar un valor del atributo llave de la tabla referenciada, automáticamente se actualice el valor de la llave foránea de las tuplas relacionadas en la tabla referenciante. Si no se tiene definida esta opción, no se puede cambiar los valores de la llave primaria de la tabla referenciada.

Eliminar registros en cascada: Indica al SMBD que al eliminar una tupla de la tabla referenciada automáticamente se borran también las tuplas relacionadas en la tabla referenciante. Si no se tiene definida esta opción, no se pueden borrar tuplas de la tabla referenciada si estas tienen tuplas relacionadas en la tabla referenciante.

Por lo tanto, para cada llave foránea de la base de datos habrá que contestar a tres preguntas:

- ¿Tiene sentido que la llave foránea acepte nulos?
- Regla de borrado: ¿Qué ocurre si se intenta borrar la tupla referenciada por la llave foránea?
  - Restringir: no se permite borrar la tupla referenciada.
  - Propagar: se borra la tupla referenciada y se propaga el borrado a las tuplas que la referencian mediante la llave foránea.
  - Anular: se borra la tupla referenciada y las tuplas que la referenciaban obtienen la marca `NULL` en la llave foránea (sólo si acepta nulos).
- Regla de modificación: ¿Qué ocurre si se intenta modificar el valor de la llave primaria de la tupla referenciada por la llave foránea?
  - Restringir: no se permite modificar el valor de la llave primaria de la tupla referenciada.
  - Propagar: se modifica el valor de la llave primaria de la tupla referenciada y se propaga la modificación a las tuplas que la referencian mediante la llave foránea.

- Anular: se modifica la tupla referenciada y las tuplas que la referenciaban obtienen la marca NULL en la llave foránea (sólo si acepta nulos).

Actividad 7.3

Investiga la implementación de la integridad en PostgreSQL. ¿Acepta todas las opciones aquí descritas?

Crea dos tablas, una de ellas con una llave foránea que haga referencia a la otra.
Elimina la tabla referenciada con la instrucción DROP.

¿Se puede llevar a cabo la operación?
¿Qué te indica el sistema?

PostgreSQL implementa la integridad referencial desde la creación de la tabla (aunque también se da la posibilidad de realizarla mediante la cláusula **ALTER TABLE**), su sintaxis se presenta en la figura 7.2. Hay que notar que la cláusula **FOREIGN KEY** no es más que una restricción de tabla, tal como se presenta en la figura7.1.

```
FOREIGN KEY ( columna [,...] ) REFERENCES tabla_referenciada
     [ ( columna_referenciada [,...] ) ] [ MATCH tipo_coincidente ]
     [ ON DELETE accion ] [ ON UPDATE accion ]
```

Figura 7.2: Sintaxis básica de las restricciones de integridad en la creación de una tabla.

7.4. Ejercicios

1. Crea tres tablas temporales dentro de tu base de datos (*T1*, *T2* y *T3*), de tal forma que la tabla *T1* tenga por lo menos 2 atributos con tipo de dato 'bigint' y uno de ellos sea su llave primaria, la tabla *T2* tenga también por lo menos 2 atributos con tipo de dato 'bigint' y uno de ellos sea llave primaria y a su vez llave foránea referenciando a la llave primaria de *T1*. Por último crea *T3* también con 2 atributos con tipo de dato 'bigint' y uno de ellos sea llave foránea referenciando a *T1*, el otro llave foránea referenciando a *T2* y los dos sean llave primaria de *T3*.

   a) Inserta por lo menos 20 tuplas dentro de cada tabla.

   b) Implementa las llaves foráneas con la opción `ON UPDATE CASCADE` y `ON DELETE CASCADE` y procede a eliminar y actualizar algunas tuplas.

   c) Implementa las llaves foráneas con la opción `ON UPDATE CASCADE` y `ON DELETE RESTRICT` y procede a eliminar y actualizar algunas tuplas.

   d) Implementa las llaves foráneas con la opción `ON UPDATE RESTRICT` y `ON DELETE CASCADE` y procede a eliminar y actualizar algunas tuplas.

   e) Implementa las llaves foráneas con la opción `ON UPDATE RESTRICT` y `ON DELETE RESTRICT` y procede a eliminar y actualizar algunas tuplas.

   ¿Qué sucede con los valores en cada uno de los casos? Genera un reporte con tus observaciones y conclusiones.

2. La sintaxis de la sentencia `FOREIGN KEY` acepta en el parámetro 'accion' los valores de `NO ACTION`, `SET NULL` y `SET DEFAULT`. ¿Qué funcionalidad práctica encuentras en cada uno de ellos? Genera un ejemplo donde demuestres lo anterior.

Guarda todos tus resultados y sentencias creadas en un archivo con extensión *.sql*

Seguridad

Semana:	Décimo segunda semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos básicos de SQL. Conceptos de seguridad. Conceptos de redes de computadoras.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Comprender los principios de seguridad de una base de datos.

La practica ocho presenta el primer acercamiento a los temas de seguridad que se refieren en las bases de datos. El tema es amplio, pero se pretende cubrir los aspectos más simples y significativos del mismo.

PRÁCTICA 8

Seguridad

8.1. Meta

Que el alumno comprenda la importancia de la seguridad en una base de datos y le permita definir restricciones básicas de seguridad.

8.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar los niveles de seguridad.
- Comprender la importancia de la seguridad dentro de una base de datos.
- Implementar un esquema simple de seguridad.

8.3. Desarrollo

El principal objetivo de un SMBD es proveer medios para acceder a la información almacenada, pero esta información no siempre debe encontrarse disponible para todos los usuarios de la base de datos, es decir, dependiendo del tipo de usuario de la base de datos será la información a la que deberá tener acceso. Ejemplo de esta situación surge en muchas empresas donde posiblemente un departamento tiene acceso a la información de ventas, pero no necesariamente deba tener acceso al historial médico de

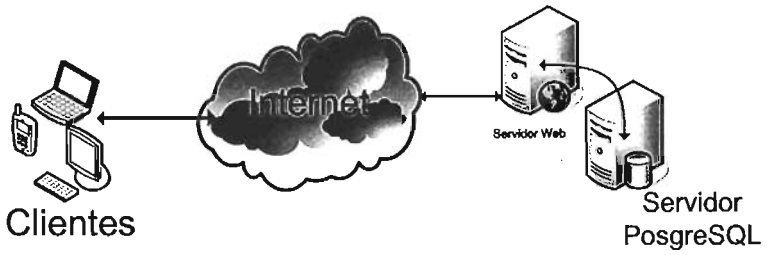

Figura 8.1: Conexión de clientes a través de Internet.

todos los trabajadores de la empresa. Internamente los SMBD deben proveer herramientas para mantener seguros los recursos internos contra accesos internos de usuarios no autorizados y accesos externos por parte de usuarios ajenos al sistema.

En la actualidad es limitado el numero de bases de datos que reciben únicamente consultas de clientes locales, generalmente la información proviene de conexiones de clientes que se encuentran físicamente en otra computadora o dispositivo que permite la conexión por medio de alguna red (por ejemplo de Internet). En estos casos, la información que se transmite es muy delicada en su contenido, ya que puede representar desde números de cuentas bancarias hasta información personal, por ello es recomendable siempre contar con un canal de comunicación seguro. Para obtener información o para almacenarla, se debe crear una conexión a la base de datos desde la aplicación en cuestión, posteriormente enviar una consulta válida, obtener el resultado de la consulta y cerrar la conexión presentando los resultados al cliente. Esto se puede apreciar gráficamente en la figura 8.1.

La primera consideración importante es evitar que cualquier usuario se conecte con privilegios de 'super-usuario'[1] , dado que si se presenta este suceso, dicha sesión podrá destruir la base de datos en cuestión. Esta clase de seguridad generalmente no compete al SMBD sino al sistema operativo anfitrión donde se encuentre instalado.

Otro punto importante se refiere a los privilegios de los usuarios dentro del ambiente en el cual se desarrolla cada sesión de usuario, para ello se tienen a disposición las cláusulas `GRANT` y `REVOKE`.

8.3.1. Seguridad dentro de PostgreSQL

La seguridad de una base de datos, dentro de PostgreSQL, se lleva a cabo en distintos niveles:

- Protección de los archivos.- Todos los archivos almacenados dentro de la base de datos son protegidos ante la lectura de cualquier otra cuenta que no sea la del

[1] '*root*' o administrador, dependiendo del sistema operativo en cuestión.

'super-usuario' de PostgreSQL (generalmente 'postgres')[2] .

- Conexión.- Las conexiones de cualquier cliente al servidor de bases de datos son por omisión, únicamente vía 'enchufes'[3] del sistema operativo, no enchufes TCP/IP. Para modificar este comportamiento hay que suministrar el parámetro "-i" al iniciar el proceso postmaster.

- Restricción de conexiones.- Las conexiones de cada cliente al servidor pueden restringirse por direcciones IP y/o usuarios. Esto se controla por medio del archivo de configuración '*pg_hba.conf*'[4] .

- Permisos a usuarios.- Los administradores pueden otorgar permisos mediante la cláusula `GRANT` y revocar permisos mediante la cláusula `REVOKE` sobre cualquier objeto de la base de datos (por ejemplo tablas, vistas, funciones, etcétera). Inicialmente cada usuario en PostgreSQL se designa con un nombre y opcionalmente una contraseña, los usuarios no tienen privilegios de escritura en objetos que no crearon.

- Modificación de esquemas.- Las cláusulas que destruyen o modifican la estructura de una clase existente, tales como `ALTER`, `DROP TABLE` o `DROP INDEX`, cuentan con funcionalidad solamente para el dueño del esquema. Estás operaciones no se permiten en ningún caso sobre los catálogos del sistema.

- Creación de grupos de usuarios.- Los usuarios pueden agruparse, de modo que los privilegios pueden restringirse a grupos.

- Creación y utilización de vistas.- Esto permite restringir la información a la cual usuarios determinados tienen acceso.

- Herencia de permisos.- Los usuarios poseen permisos sobre los objetos de las bases de datos, pero sólo ellos pueden otorgar o restringir operaciones sobre los objetos que les pertenecen o sobre los que tienen privilegios.

Control de usuarios

Cada conjunto de base de datos, tiene asociado un conjunto de usuarios, estos usuarios son totalmente distintos de los usuarios administrados por parte del sistema operativo. Los usuarios poseen objetos de la base de datos y pueden asignar privilegios en estos objetos a otros usuarios. En PostgreSQL se cuenta con la cláusula `CREATE USER` que permite crear nuevos usuarios dentro de las bases de datos. Su sintaxis se presenta en la figura 8.2.

[2] Adicionalmente sólo el 'super-usuario' (root) del sistema operativo tiene todos los privilegios sobre los archivos de PostgreSQL.

[3] O *sockets*

[4] Este y demás archivos de configuración de PostgreSQL se encuentran en el directorio `$PGHOME/data`

```
CREATE USER nombre_usuario [ [ WITH ] opción [...] ];

donde opción puede ser:
       CREATEDB | NOCREATEDB
      | CREATEUSER | NOCREATEUSER
      | IN GROUP nombre_grupo [,...]
      | [ENCRYPTED | UNENCRYPTED] PASSWORD 'contraseña'
      | VALID UNTIL 'dia'
```

Figura 8.2: Sintaxis de la cláusula CREATE USER.

Actividad 8.1

Inicia sesión dentro del sistema y realiza la siguiente consulta:

```
SELECT username FROM pg_user;
```

¿Qué información se almacena en la tabla del sistema *pg_user*?
Obten todos los elementos de la tabla e investiga que significa cada uno.

Existe la posibilidad de crear grupos de usuarios dentro de PostgreSQL, estos grupos no tienen que ser grupos de usuarios del sistema operativo anfitrión y su funcionalidad primordial es poder administrar los privilegios sobre objetos de la base de datos de una manera más sencilla, ya que los privilegios otorgados a un grupo se propagan a los usuarios que pertenecen al grupo a menos de que se indique lo contrario. La cláusula para la creación de un grupo se presenta en la figura 8.3. Para agregar a un usuario o eliminarlo del grupo, se cuenta con la cláusula ALTER GROUP. Su sintaxis se presenta en la figura 8.4.

```
CREATE GROUP nombre_grupo;
```

Figura 8.3: Sintaxis de la cláusula CREATE GROUP.

```
ALTER GROUP nombre_grupo [ADD | DROP] USER nombre_usuario1,...;
```

Figura 8.4: Sintaxis de la cláusula ALTER GROUP.

Actividad 8.2

Inicia sesión dentro del sistema y realiza la siguiente consulta:

```
SELECT groname FROM pg_group;
```

¿Qué información se almacena en la tabla del sistema *pg_group*?
Obten todos los elementos de la tabla e investiga que significa cada uno.
¿Cómo se relaciona esta tabla con *pg_user*?

Cuando un objeto se crea en la base de datos, se asignan permisos de lectura y de modificación a su creador. El dueño del objeto es entonces, quien haya ejecutado la sentencia de creación del mismo. Para cambiar el dueño del objeto en cuestión se utiliza la sentencia ALTER. Por omisión solamente el dueño y el 'super-usuario' pueden modificar al objeto. Para permitir que otro usuario lo utilice, los privilegios se deben otorgar (o revocar posteriormente).

Existen distintos tipos de privilegios y estos se otorgan o revocan sobre las distintas cláusulas. Los privilegios existentes son: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE y ALL PRIVILEGES. Para otorgar un privilegio se utiliza la sentencia GRANT y para retirar un privilegio se cuenta con la sentencia REVOKE.

Los privilegios especiales para el dueño (DROP, GRANT, REVOKE, etcétera) son siempre implícitos para el mismo y no pueden ser otorgados o retirados, pero el dueño de la tabla puede escoger el retirar algunos privilegios ordinarios, por ejemplo dejar a una tabla en modo de sólo lectura de datos.

GRANT

La cláusula GRANT, determina los privilegios de un objeto para uno o más usuarios o grupos. Su sintaxis se presenta en las figuras 8.5 y 8.6. Los privilegios se agregan a los que se encuentren previamente definidos, si hay alguno. La palabra PUBLIC indica que los privilegios se otorgan a todos los usuarios, incluso a los que se generen posteriormente, además, PUBLIC define a un grupo de usuarios al cual pertenecen todos los usuarios.

Si se utiliza la opción "WITH GRANT OPTION", entonces quien recibe los privilegios descritos, puede otorgar esos permisos.

```
GRANT {{SELECT | INSERT | UPDATE | DELETE | RULE
            | REFERENCES | TRIGGER }[...] | ALL [ PRIVILEGES ] }
ON [ TABLE ] tablename [,...]
TO {nombre_usuario | GROUP nombre_grupo | PUBLIC }
                          [,...] [ WITH GRANT OPTION ]
```

Figura 8.5: Sintaxis básica de GRANT sobre tablas.

```
GRANT {{CREATE | TEMPORARY | TEMP }[,...] | ALL [ PRIVILEGES ] }
ON DATABASE dbname [,...]
TO {nombre_usuario | GROUP nombre_grupo | PUBLIC }
                          [,...] [ WITH GRANT OPTION ]
```

Figura 8.6: Sintaxis básica de GRANT sobre bases de datos.

Actividad 8.3

Inicia sesión dentro del sistema y realiza lo siguiente:

```
nombre_bd#=> \z nombre_tabla;
```

¿Qué información se presenta ?

REVOKE

```
REVOKE [ GRANT OPTION FOR ]
{{SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES | TRIGGER }
[,...] | ALL [ PRIVILEGES ] }
 ON [ TABLE ] tablename [,...]
FROM {nombre_usuario | GROUP nombre_grupo | PUBLIC }[,...]
[ CASCADE | RESTRICT ]
```

Figura 8.7: Sintaxis básica de REVOKE sobre tablas.

La cláusula REVOKE elimina los privilegios obtenidos a uno o más usuarios o grupos. La sintaxis básica de REVOKE se presenta en las figuras 8.7 y 8.8, para tablas y bases de datos respectivamente. De nueva cuenta la palabra PUBLIC hace referencia implícita

```
REVOKE [ GRANT OPTION FOR ]
{{CREATE | TEMPORARY | TEMP }[,...] | ALL [ PRIVILEGES ] }
ON DATABASE dbname [,...]
FROM {nombre_usuario | GROUP nombre_grupo | PUBLIC }[,...]
[ CASCADE | RESTRICT ]
```

Figura 8.8: Sintaxis básica de REVOKE sobre bases de datos.

al grupo de todos los usuarios. Es importante mencionar que un usuario tiene por privilegios la suma de todos los privilegios obtenidos ya sea por parte de un grupo en el cual este presente o por privilegios obtenidos vía PUBLIC. Así, por ejemplo, revocar el privilegio SELECT de PUBLIC no significa necesariamente que todos los usuarios han perdido el privilegio de SELECT en ese objeto, ya que pudieron haber obtenido el privilegio directamente (al crearlo) o bien por otro grupo.

Asimismo si se ocupa la opción "GRANT OPTION FOR" solamente se pierde la propiedad de revocar el privilegio, no el privilegio propiamente. En caso contrario, ambos se pierden.

Autenticación de usuarios

La autenticación es el proceso mediante el cual el servidor y postmaster verifican que en la petición de acceso a los datos por parte de un usuario sea realmente él quien la solicita. Todos los usuarios que realizan una petición a PostgreSQL son verificados contra el contenido de la clase 'pg_user' para asegurarse que están realmente autorizados para realizar la petición. Sin embargo, la verificación del usuario actual se lleva a cabo de distintas formas:

- Desde el intérprete de comandos: un proceso ejecutándose en segundo plano ('*backend*') iniciado desde el intérprete de comandos del usuario toma la identificación real del usuario ('*user-id*') antes de realizar un '*SETUID*'[5] al identificador de usuario del usuario de PostgreSQL. El identificador de usuario real es utilizado para la verificación del control de acceso.

- Desde la red: el acceso al puerto TCP de postmaster se encuentra disponible para cualquier usuario. El administrador de la base de datos debe configurar el archivo 'pg_hba.conf' para determinar que sistema de autenticación será utilizado

[5] Un total de 12 bits que expresan las operaciones del archivo que son permitidas en función del proceso que acceda a este archivo. Los bits **SETUID** y **SETGID** en archivos ejecutables se emplean para permitir que un programa se ejecute bajo los privilegios de un usuario distinto al que lanza la ejecución del programa.

de acuerdo al servidor anfitrión[6] que recibe la conexión y a que base de datos se conecta.

Control de acceso basado en anfitriones

El control de acceso basado en anfitriones es el nombre para el control de acceso que implementa PostgreSQL en el manejo de los clientes y sus permisos para acceder a las bases de datos. Cada base de datos contiene un archivo llamado 'pg_hba.conf' que controla quien puede conectarse a la base de datos. El cliente que accede a una base de datos debe concordar con alguna entrada dentro del archivo 'pg_hba.conf', de otro modo todos los intentos de conexión de ese cliente serán rechazados con el mensaje de error: `"User authentication failed"`.

El archivo pg_hba.conf se compone de registros, uno por línea. Cada registro se encuentra compuesto por campos numéricos que se separan por espacios o tabuladores. Se tienen dos formatos generales, aquellos que abarcan a las conexiones locales y aquellos que cubren las conexiones remotas. La sintaxis de cada formato se presentan en la figura 8.9 y en la figura 8.10, respectivamente.

```
local nombre_bd usuario método_autenticación [opciones]
```

Figura 8.9: Sintaxis de un registro en 'pg_hba.conf' para clientes locales.

Donde 'nombre_bd' denota la base de datos a la cual se aplica la regla. Los valores posibles son: '*sameuser*', '*samegroup*' y '*all*' ('*all*' determina que se aplica a todas las bases de datos). El campo 'usuario' indica el usuario al cual aplica la regla y puede tomar los valores: '*all*', el nombre de un usuario, un grupo de usuarios (teniendo como prefijo el símbolo "+" o bien una lista de estos separando los valores por comas. El campo marcado como 'método_autenticación' determina el método que el usuario debe utilizar para autenticarse cuando se conecte a la base de datos, puede tomar los valores: '*trust*', '*reject*', '*md5*', '*crypt*', '*password*','*krb4*', '*krb5*' o '*ident*'.

```
{host | hostssl | hostnossl}
nombre_bd usuario dirección método_autenticación
```

Figura 8.10: Sintaxis de un registro en 'pg_hba.conf' para clientes remotos.

En el caso de conexiones remotas, el primer parámetro indica el tipo de servidor anfitrión que se tendrá: '*host*' utiliza enchufes TCP/IP ya sean sencillos o cifrados por

[6] '*Host server*'.

SSL, '*hostssl*' indica enchufes TCP/IP cifrados por SSL y '*hostnossl*' indica enchufes TCP/IP sencillos. El campo 'dirección' es una dirección que indica el conjunto de máquinas a las cuales afecta la regla, está compuesta de una dirección IP y una mascara CIDR[7] .

Alternativamente se puede incluir una dirección IP y la máscara de red en columnas separadas para denotar el conjuntos de anfitriones, estas se escriben en notación decimal puntual.

Métodos de autenticación

Los siguientes métodos de autenticación se encuentran implementados para conexiones locales:

- 'trust'.- La conexión se permite incondicionalmente.
- 'reject'.- La conexión se niega incondicionalmente.
- 'crypt'.- Pide al cliente por la contraseña del usuario. Esta se envía cifrada utilizando crypt[8] y se compara contra la contraseña que se mantiene en la tabla '*pg_shadow*'. Si las contraseñas concuerdan, la conexión se establece.
- 'password'.- Pide al cliente por la contraseña del usuario. Esta se envía en texto sencillo y el procedimiento es similar al descrito anteriormente.

Para conexiones remotas se tienen los siguiente métodos:

- 'ident'.- El servidor 'ident'[9] en el cliente se utiliza para autenticar al usuario. Se puede indicar un nombre de mapas después de la palabra 'ident' para permitir que los clientes asocien sus nombres a nombres de usuarios dentro de PostgreSQL. Estos mapas de nombres se mantienen en el archivo 'pg_ident.conf'.
- 'krb5' y 'krb4'.- Implementan la autenticación por medio del sistema 'Keberos' en sus versiones 4 y 5.
- 'md5'.- Los mensajes enviados y datos por parte del cliente y hacia el mismo son validados utilizando la función MD5. La suma de comprobación de la función MD5 verifica la integridad de los datos sometiéndolos a un algoritmo hash una vez que se reciben. El valor resultante se compara con el valor obtenido por el algoritmo hash que se envía junto con los datos. Si ambos coinciden, significa que los datos no han sufrido alteraciones o manipulaciones y puede confiarse en su integridad.

[7] '*Classless Inter-Domain Routing*' es un entero entre 0 y 32 para IPv4 y entre 0 y 128 para IPv6, que determina el número de bits significativos en la mascara.

[8] Es la función de cifrado de contraseñas basada en DES.

[9] La ident es una parte de la mascara de usuario típica de IRC que esta compuesta por un sobrenombre, una ident y un host o IP de la siguiente forma: sobrenombre!ident@host.

No todos los métodos se encuentran presentes por omisión dentro de una instalación típica, por ejemplo para habilitar SSL, es necesario compilar PostgreSQL utilizando como parámetro la opción de "–with-openssl[=DIR]" al comando 'configure' (además de contar con las utilidades necesarias de SSL).

Es importante recordar que los métodos de cifrado simplemente se encargan se cifrar la información que se envía y recibe por parte del cliente y del servidor, por ello si un atacante a la base de datos obtiene acceso a la misma, es la información almacenada en el servidor la que corre peligro, los métodos anteriores no protegen los datos persistentes en la base de datos.

Actividad 8.4

Inicia sesión dentro del sistema como el usuario postgres, revisa los archivos. 'pg_hba.conf' y 'pg_ident.conf'.

Identifica cada elemento dentro de las reglas que aparecen por omisión.

Actividad 8.5

Inicia sesión dentro del sistema como el usuario postgres.
Crea dos nuevos usuarios, uno con las opciones de 'PASSWORD' y UNENCRIPTED'.
Crea otro con las opciones de 'PASSWORD' y 'ENCRIPTED'.

Verifica el contenido de la tabla *pg_shadow*.
¿Qué información encuentras en la tabla?

8.4. Ejercicios

1. Crea tres usuarios ('HUGO','PACO' y 'LUIS') y dos tablas (*tabla_A* y *tabla_B*). Siendo el usuario 'HUGO' otorga los privilegios de 'SELECT' sobre la tabla '*tabla_A*' (con la opción de 'GRANT') a 'PACO'. Después haz que 'PACO' otorgue los mismos privilegios a 'LUIS'. Responde:

   a) ¿'HUGO' puede quitarle los privilegios a 'LUIS'?

   b) Haz que 'HUGO' otorgue los privilegios de 'SELECT' sobre la tabla 'tabla_A' a 'LUIS'. Elimina el privilegio 'SELECT' al grupo PUBLIC sobre la tabla '*tabla_A*'. ¿Puede 'LUIS' continuar realizando sentencias 'SELECT' sobre la tabla '*tabla_A*'?

2. Investiga que reglas implementan las siguientes lineas si se encuentran dentro del archivo 'pg_hba.conf'

   a) local trust

   b) host all 127.0.0.1 255.255.255.255 trust

   c) host all 192.168.0.10 255.255.255.0 reject

   d) host all 192.168.0.3 255.255.255.0 password

   e) host all 192.168.0.0 255.255.255.0 crypt

   f) host all all 127.0.0.1/32 md5

3. Crear dos nuevos usuarios ('claseP' y 'claseSP'), además de una vista llamada *antiguos* que contenga la información de los tres profesores con mayor antigüiedad mayor. Ahora modifica los permisos de manera que el usuario 'claseP' pueda leer la vista antiguos y que el usuario 'claseSP' no. Ingresa con el usuario 'claseP' y realiza una consulta SELECT. Ingresa con el usuario 'claseSP' y realiza una consulta SELECT. ¿Qué información presenta el sistema en cada caso?

Normalización

Semana:	Décimo tercera semana.
Tiempo de entrega:	Una semana.
Prerequisitos:	Conceptos de dependencias funcionales. Conceptos de formas normales.
Herramientas:	PostgreSQL v.8.0.2.
Objetivo:	Aplicar de forma práctica los principios del proceso de normalización de una base de datos.

La práctica correspondiente al tema de normalización es la última de las practicas que se consideran de carácter obligatorio. Aborda el problema, manejando ejemplos concretos donde se perciben todos los problemas que implica una base de datos no normalizada. Por ello, es de suma importancia que el alumno domine los conceptos relativos a dependencias funcionales, dependencias multivaloradas y formas normales.

Además se considera indispensable que los alumnos hayan visto en clase, múltiples ejercicios sobre estos temas para terminar la práctica en los tiempos establecidos.

PRÁCTICA 9

Normalización

9.1. Meta

Que el alumno aprenda a crear bases de datos que cumplan con los principios de la teoría de la normalización de bases de datos y comprenda la importancia que tiene esto.

9.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Detectar la forma normal en que se encuentran las tablas de una base de datos.
- Aplicar los principios de normalización en una base de datos.
- Generar bases de datos con determinados niveles de normalización.

9.3. Desarrollo

El modelo relacional y las formas normales fueron inicialmente definidos por el Dr. E. F. Codd [8], y posteriormente estos conceptos han sido extendidos por otros autores. El proceso de normalización se basa en el concepto de las formas normales y comprende la descomposición de una relación dada en otras relaciones que presenten una mejor estructura, dicha descomposición debe ser reversible para que no se pierdan datos en el proceso. Por lo tanto solamente interesan las descomposiciones sin pérdida

de información. Este concepto está íntimamente ligado con el concepto de dependencias funcionales. La normalización tiene dos metas: eliminar los datos redundantes (por ejemplo, almacenar los mismos datos en tablas distintas) y asegurar que las dependencias entre los datos tengan sentido (solamente almacenar datos relacionados entre si en una tabla). Es importante conseguir éstas metas dado que reducen la cantidad de espacio ocupado por la base de datos y asegura que los datos son almacenados de una manera lógica.

La normalización es un proceso de ayuda en el diseño de una base de datos, pero no es la panacea; es recomendable tener presente el proceso de normalización, así como con sus principios, pero el diseño no debe basarse solamente en estos elementos.

Durante la práctica su presenta una variante de la base de datos del curso, con ella se analiza cada forma normal alcanzada y las consideraciones que se tomaron para obtener esta forma normal.

9.3.1. Un ejemplo práctico

Considerando una variante de la base de datos del curso, se tiene la necesidad de almacenar la información referente al registro de los profesores, sus artículos que han publicado, los cursos que imparten, el material utilizado en esos cursos y el departamento académico de cada uno de los profesores. Para ello se considera que es necesario almacenar únicamente la siguiente información:

- Datos generales de los profesores, como lo son: nombre, apellido paterno, apellido materno y dirección.
- Datos de los departamentos académicos de los profesores, como son: nombre del departamento académico del profesor y el cubículo asignado.
- Datos generales de los artículos publicados por el profesor, que son: nombre del artículo, área de estudio (en la cual se clasifica al artículo) y nombre de la publicación donde aparece.
- Datos de los libros utilizados por los profesores durante sus cursos, tales como: el nombre del curso que imparte y el libro que utiliza el profesor.

Los supuestos iniciales sobre esta información que deseamos resaltar son los siguientes:

1. No existen ciudades con el mismo nombre en diferente país.
2. Un profesor puede publicar muchos artículos.
3. Un articulo puede estar relacionado con varios autores (co-autoría).
4. Un profesor solamente puede tener un domicilio registrado.

5. En un curso se pueden utilizar libros distintos.

6. Un curso puede ser impartido por más de un profesor.

Para representar esta información se crean las relaciones siguientes:

- *R[nom, ap, am, cp, cd, pa, art]* : Contiene los datos referentes a un profesor y sus artículos publicados, está compuesta de los atributos:
  - '*nom*' : nombre del profesor.
  - '*ap*' : apellido paterno del profesor.
  - '*am*' : apellido materno del profesor.
  - '*cp*' : código postal de la dirección del profesor.
  - '*cd*' : ciudad de nacimiento del profesor.
  - '*pa*' : país de nacimiento del profesor.
  - '*art*' : datos del artículo del profesor.
- *S[curso, nom, ap, am, libro]* : Contiene los datos referentes a un profesor y los libros que utilizá en los cursos que imparte el profesor. Está compuesta por los siguientes atributos:
  - '*curso*' : curso que imparte el profesor.
  - '*nom*' : nombre del profesor.
  - '*ap*' : apellido paterno del profesor.
  - '*am*' : apellido materno del profesor.
  - '*libro*' : nombre del libro que utiliza el profesor en el curso.

Un ejemplo de la información contenida en R se presenta en la figura 9.1 y es posible observar que no se encuentra en primera forma normal.

La primera forma normal indica que no deben existir conjuntos dentro de los valores de los atributos, ya que estos deben ser atómicos. La propuesta inicial es crear un esquema donde se 'aplane' el contenido de cada uno de los conjuntos. La nueva estructura de la relación R queda como se muestra en la figura 9.3, en ella se presenta a cada artículo (publicado por el profesor en cuestión), de manera 'fragmentada'. Así, tenemos que la nueva relación R toma la siguiente constitución:

- *R[nom, ap, am, cp, cd, pa, , nomA, pubA, areaA, cub, depto]* : Contiene los datos referentes a un profesor, sus artículos publicados y las propiedades de estos, está compuesta de los atributos:
  - '*nom*' : nombre del profesor.

nom	ap	am	cp	cd	pa	art				
Xio	Li	Koo	021	DF	Mex	public1	Science	Mat	AM	Mat
						public2	Science	Mat	AM	Mat
						public3	ACM pub	Fis	AM	Mat
Yung	Tai	Chu	044	Gda	Mex	public4	ACM pub	Mat	BM	Mat
						public3	ACM pub	Fis	BM	Mat
						public6	Science	Fis	BM	Mat
Zap	Wan	Tyl	047	Mty	Mex	public5	IEEE pub	Quim	CF	Fis
						public7	Natural	Biol	CF	Fis

Figura 9.1: Relación R inicial.

- '*ap*' : apellido paterno del profesor.
- '*am*' : apellido materno del profesor.
- '*cp*' : código postal de la dirección del profesor.
- '*cd*' : ciudad de nacimiento del profesor.
- '*pa*' : país de nacimiento del profesor.
- '*nomA*' : nombre del artículo.
- '*pubA*' : nombre de la publicación donde aparece el artículo.
- '*areaA*' : área de estudio donde se enmarca el artículo.
- '*cub*' : cubículo donde se desarrollo el artículo del profesor.
- '*depto*' : departamento al que pertenece el cubículo.

curso	nom	ap	am	libro
SBD	Xio	Li	Koo	Introduction to Database Systems
SBD	Xio	Li	Koo	Database Design Principles
SBD	Yung	Tai	Chu	Introduction to Database Systems
SBD	Yung	Tai	Chu	Database Design Principles
SMBD	Xio	Li	Koo	Introduction to Database Systems
SMBD	Xio	Li	Koo	Complete Database Book
SMBD	Xio	Li	Koo	SQL Bible

Figura 9.2: Relación S.

Es preciso mencionar que la estructura de la relación S no se ha modificado hasta el momento, ya que únicamente se ha realizado el análisis sobre la relación R. Un ejemplar de ésta última se aprecia en la figura 9.2.

nom	ap	am	cp	cd	pa	nomA	pubA	areaA	cub	depto
Xio	Li	Koo	021	DF	Mex	public1	Science	Mat	AM	Mat
Xio	Li	Koo	021	DF	Mex	public2	Science	Mat	AM	Mat
Xio	Li	Koo	021	DF	Mex	public3	ACM pub	Fis	AM	Mat
Yung	Tai	Chu	044	Gda	Mex	public4	ACM pub	Mat	BM	Mat
Zap	Wan	Tyl	047	Mty	Mex	public5	IEEE pub	Quim	CF	Fis
Zap	Wan	Tyl	047	Mty	Mex	public7	Natual	Biol	CF	Fis
Yung	Tai	Chu	044	Gda	Mex	public3	ACM pub	Fis	BM	Mat
Yung	Tai	Chu	044	Gda	Mex	public6	Science	Fis	BM	Mat

Figura 9.3: Relación R en 1FN.

De la relación presentada en la figura 9.3, el posible observar que se presentan múltiples problemas.

1. Si un profesor se va, entonces se deben eliminar todos los artículos donde él aparece como autor. Esto se conoce como anomalía de borrado.

2. Si un nombre de un artículo es asignado a otro profesor entonces se debe eliminar la referencia anterior. En el ejemplo si se asigna el articulo '*public5*' y '*public7*' a otro profesor, entonces se debe eliminar al profesor '*Zap*'. Esta es la anomalía de actualización.

3. Si se tiene un nuevo artículo el cual aún no cuenta con profesor asignado, simplemente no se puede agregar a la base. Esta es la anomalía de inserción.

Actividad 9.1

El esquema relacional R se encuentra en la primera forma normal. Agrega este esquema como la tabla R dentro de tu base de datos y verifica si se presentan las anomalías anteriormente descritas.

Con esto en mente, es posible analizar el esquema relacional R y determinar que no se encuentra en segunda forma normal ya que existen dependencias parciales de los atributos con respecto a las llaves candidatas; esto trae como consecuencia una serie de duplicación innecesaria de información. Esta afirmación surge luego de listar las dependencias funcionales presentes en este esquema relacional R y analizar a cada uno de los atributos constituyentes.

Las dependencias funcionales encontradas luego del análisis preliminar son las siguientes[1] :

[1] Por brevedad se omiten las dependencias funcionales triviales.

1. $nom \longrightarrow ap, am, cp, calle, cd, pa$:
2. $nA \longrightarrow pubA, areaA$:
3. $cd \longrightarrow pa$:
4. $nom, nomA \longrightarrow pubA, areaA$:
5. $nomA \longrightarrow pubA, areaA$:
6. $nom \longrightarrow cub, depto$:

Así que para obtener la segunda forma normal, se procede a una descomposición de R. En este caso, es conveniente crear una nueva relación para cada una de estas dependencias funcionales y conservando en otra relación el resto de los atributos. El resultado de este procedimiento se presenta en la figura 9.4.

nom	ap	am	cp	cd	pa
Xio	Li	Koo	021	DF	Mex
Yung	Tai	Chu	044	Gda	Mex
Zap	Wan	Tyl	047	Mty	Mex

R1

nom	nomA	pubA	areaA
Xio	public1	Science	Mat
Xio	public2	Science	Mat
Xio	public3	ACM pub	Fis
Yung	public4	ACM pub	Mat
Yung	public3	ACM pub	Fis
Yung	public6	Science	Fis
Zap	public5	IEEE pub	Quim
Zap	public7	Natural	Biol

R2

nom	cub	depto
Xio	AM	Mat
Yung	BM	Mat
Zap	CF	Fis

R3

Figura 9.4: Descomposición de R en *R1*, *R2* y *R3* que se encuentran en 2FN.

En la segunda forma normal, se reduce la cantidad de redundancia existente en las relaciones, pero no se elimina. En el ejemplo, la relación *R1* presenta la dependencia funcional $cd \longrightarrow pa$. Esto quiere decir que existe una dependencia funcional transitiva ($nom \longrightarrow cd$ y $cd \longrightarrow pa \Longrightarrow$ nom $\longrightarrow$ pa) .

Para solventar esta redundancia presente, es conveniente descomponer el esquema nuevamente. En el caso específico de las dependencias transitivas, es propio crear una

nueva relación donde se integren lo atributos que participan en la dependencia en cuestión. Por ello, la nueva descomposición crea la relación *R4*. El resultado de esto, se presenta en la figura 9.5 (solamente se muestran las relaciones afectadas por esta nueva descomposición, a saber *R1* y *R4*).

nom	ap	am	cp	cd
Xio	Li	Koo	021	DF
Yung	Tai	Chu	044	Gda
Zap	Wan	Tyl	047	Mty

*R*1

cd	pa
DF	Mex
Gda	Mex
Mty	Mex

R4

Figura 9.5: Descomposición de *R* en *R1*, *R2*, *R3* y *R4* que se encuentran en 3FN.

El esquema relacional obtenido se encuentra en tercera forma normal, de hecho se encuentra en forma normal Boyce-Codd.

Actividad 9.2

Se dijo que el último esquema relacional se encuentra en la forma normal Boyce-Codd. ¿Es cierto?
¿Qué consideración es fundamental para determinarlo?

Dentro de tu base de datos, codifica las cuatro tablas anteriores e inserta 10 tuplas en cada una de ellas. ¿Se detecta alguna inconsistencia o duplicidad de información?

Continuando con el proceso de normalización, para obtener la cuarta forma normal, se considera ahora la relación *S* de la figura 9.2, donde se cumple el conjunto de dependencias multivaluadas $D = \{$ *curso* $\twoheadrightarrow$ *nom, ap, am* y *curso* $\twoheadrightarrow$ *libro*$\}$.

El análisis de la dependencia multivaluda *curso* $\twoheadrightarrow$ *profesor*, determina que aún cuando el atributo *curso* no tiene una correspondencia ÚNICA con *profesor* (no se cumple la dependencia funcional *curso* $\longrightarrow$ *profesor*), cada tupla en *curso* si posee un conjunto 'bien definido' de profesores correspondientes en *profesor*. Por 'bien definido' se quiere dar a entender que, precisamente, para un curso dado *c* y un libro *l*, el conjunto

de profesores p que concuerda en par (c, l) en la relación S depende solamente del valor c (no hay diferencia en cual valor particular de l sea elegido).

El análisis para la dependencia multivaluada *curso* $\twoheadrightarrow$ *libro* es similar. Para obtener un esquema relacional donde todas las relaciones cumplan con la cuarta forma normal, generalmente basta con crear una nueva relación (por cada dependencia multivaluada donde el determinante no sea superllave) donde el atributo que no es superllave para la relación original, sea una superllave para una de las nueva relaciones que se generan a partir de la descomposición de la relación original.

El resultado de aplicar esto sobre el esquema S se presenta en la figura 9.6

curso	nom	ap	am
SBD	Xio	Li	Koo
SBD	Yung	Tai	Chu
SMBD	Xio	Li	Koo

S1

curso	libro
SBD	Introduction to Database Systems
SBD	Database Design Principles
SMBD	Introduction to Database Systems
SMBD	Complete Database Book
SMBD	SQL Bible

S2

Figura 9.6: Descomposición de S en $S1$ y $S2$ que se encuentran en la 4FN.

9.4. Ejercicios

1. Añade nuevas relaciones al esquema relacional de la práctica, de manera que se presente una dependencia multivaluada y lleva el esquema hasta la forma normal Boyce-Codd.

2. ¿Qué sucede si en el esquema relacional de la práctica se considera que un profesor puede compartir su cubículo? ¿Se preserva la forma normal Boyce-Codd?

3. Considera la siguiente relación:

nombre	materia
Hugo	Análisis Cuantitativo de las Masas presentes en un cuerpo estable bajo presión alta - un estudio sistemático.
Paco	Seminario Administrativo Avanzado - "La calidad en el desarrollo de métodos de investigación eficientes"
René	Seguridad en computo distribuido utilizando un modelo de calidad para el desarrollo de infraestructura altamente sustentable
Luis	Temas Selectos de Biología Marina - "Métricas de calidad en la investigación del Delfín Azul en las aguas naturales de México"
Hugo	Temas Selectos de Biología Marina - "Métricas de calidad en la investigación del Delfín Azul en las aguas naturales de México"
Hugo	Seguridad en computo distribuido utilizando un modelo de calidad para el desarrollo de infraestructura altamente sustentable
Paco	Seguridad en computo distribuido utilizando un modelo de calidad para el desarrollo de infraestructura altamente sustentable
Luis	Análisis Cuantitativo de las Masas presentes en un cuerpo estable bajo presión alta - un estudio sistemático.

   Crea las tablas relativas, inserta las tuplas presentadas y lista las dependencias funcionales, justificando tu respuesta. ¿Es conveniente realizar alguna descomposición? En caso de ser así, genera un nuevo esquema relacional donde presentes las relaciones resultantes y crea las nuevas tablas.

4. ¿Qué es una descomposición sin perdida de información? Genera un esquema relacional y divídelo en tres relaciones donde presentes una descomposición con perdida. Genera las tablas relativas a cada una de las relaciones resultantes e ingresa 20 tuplas a cada una de las tablas.

5. Del esquema anterior, muestra una descomposición sin perdida.

Guarda tus resultados en archivos con extensión '.sql'.

Conexión a Bases de Datos

Semana:	Décimo cuarta semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Programación con Java.
Herramientas:	PostgreSQL v8.0.2 JDK v5.0 (v1.5.0) Controlador JDBC PostgreSQL (v3)
Objetivo:	Crear una aplicación para conectarse a una base de datos.

La práctica diez es la práctica optativa. Se considera como tal, ya que las tecnologías de la información, se encuentran en constante cambio y no es deseable el apegarse a una tecnología sino adaptar los conocimientos teóricos a la herramienta más actual o que mejor se adecue a nuestras necesidades inmediatas.

Por otro lado, tampoco es consideración el hecho de desarrollar una interfaz que cumpla con todos los principios ergonómicos, sino simplemente generar la interfaz mínima que permite la interacción con la base de datos.

PRÁCTICA 10

Conexión a Bases de Datos

10.1. Meta

Que el alumno desarrolle en forma práctica los conocimientos obtenidos al aplicarlos con una tecnología que le permita desarrollar aplicaciones que se conecten a una base de datos.

10.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar los elementos de una conexión a una base de datos.
- Crear aplicaciones que se conecten a una base de datos.
- Interactuar con una base de datos de manera externa.

10.3. Desarrollo

La base de datos no es un elemento aislado dentro de una organización, sino que representa un medio para compartir la información necesaria entre todos los componentes de la misma. Por ello es importante conocer medios que permitan esta actividad.

Se han desarrollado distintas tecnologías para obtener estos resultados, algunas sobresalen por su facilidad de uso, otras por sus características y unas mas por ser simplemente soluciones a la medida. Destacan entre ellas: ODBC y JDBC.

10.3.1. ODBC

"ODBC" (Open DataBase Connectivity), es una interfaz basada en el lenguaje de programación C para las aplicaciones que manejan SQL, provee un medio consistente de comunicación con las bases de datos y de acceso con los metadatos de la base. Cada compañía provee controladores específicos o 'puentes'[1] para su SMBD. Consecuentemente, gracias a ODBC y SQL, es posible establecer una conexión a una base de datos y manipularla en una forma estándar.

Aún cuando SQL permite la manipulación de bases de datos de manera sencilla, éste no fue diseñado para ser un lenguaje de aplicación general; sino que fue especificado en un principio con la idea de interactuar con bases de datos. Independientemente de los últimos cambios al estándar actual, SQL2003, es común utilizar otro lenguaje de programación donde se encuentren incrustadas sentencias SQL que se envían al SMBD y posteriormente se procesen los resultados.

El principal inconveniente, se refiere a la imposibilidad de escribir programas que se ejecuten en distintas plataformas, aún cuando el tópico de estandarización en la conectividad a bases de datos se ha tratado de resolver. Por ejemplo, si se escribe un programa cliente en C++, posiblemente sea necesario volver a codificarlo si se transporta a otra plataforma.

10.3.2. JDBC

"JDBC" (Java DataBase Connectivity), es un API de Java™ que permite al programador ejecutar instrucciones en SQL, enviarlas al servidor de bases de datos y procesar los resultados obtenidos. Para que una aplicación pueda realizar operaciones en una base de datos, ha de tener una conexión con ella, que se establece a través de un controlador (*driver*), que convierte el lenguaje de alto nivel a sentencias de SQL y son recibidas por la base de datos. Es decir, las tres acciones principales que realizará JDBC son: establecer la conexión a una base de datos, ya sea remota o no; enviar sentencias SQL a esa base de datos y, por último, procesar los resultados obtenidos.

JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa escrito en Java™ acceder a sistemas de bases de datos de forma homogénea. La aplicación de Java™ debe tener acceso a un controlador JDBC adecuado y este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

La necesidad de JDBC, a pesar de la existencia de ODBC, viene dada porque ODBC es un interfaz escrita en lenguaje C, que al no ser un lenguaje portable, haría que las aplicaciones también pierdan portabilidad. Además, ODBC tiene el inconveniente de que se ha de instalar manualmente en cada máquina; al contrario que los drivers JDBC, que se encuentran incluidos en las plataformas Java™ (J2SE y J2EE).

[1] *bridges*

Toda la conectividad de bases de datos de Java™ se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. La especificación JDBC requiere que cualquier driver JDBC sea compatible con al menos, el estándar SQL92.

Acceso con JDBC a bases de datos

El API JDBC soporta dos modelos distintos de acceso a bases de datos, los modelos de dos y tres capas.

Modelo de dos capas

Este modelo se basa en que la conexión entre la aplicación Java™ (o el '*applet*' que se ejecuta en un navegador) se conecta directamente a la base de datos. Esto significa que el controlador JDBC específico para conectarse con la base de datos, debe residir en el sistema local. La base de datos puede estar en cualquier otra máquina y se accede a ella por medio de una conexión a la red. Esta es la configuración de típica Cliente/Servidor: el programa cliente envía instrucciones SQL a la base de datos, ésta las procesa y envía los resultados de vuelta a la aplicación.

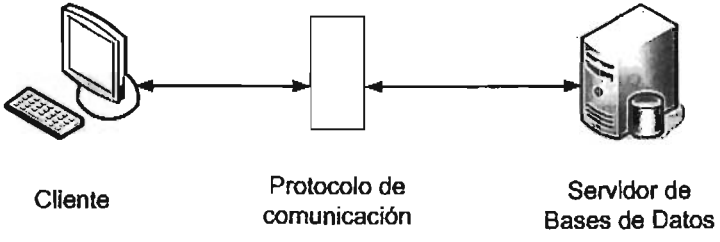

Figura 10.1: Modelo de dos capas.

Modelo de tres capas

En este modelo de acceso a las bases de datos, las instrucciones son enviadas a una capa intermedia entre Cliente y el Servidor, la cual se encarga de enviar las sentencias SQL a la base de datos y obtener el resultado desde la base de datos. En este caso el usuario no tiene contacto directo, ni a través de la red, con la máquina donde reside la base de datos.

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los controladores JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de controlador.

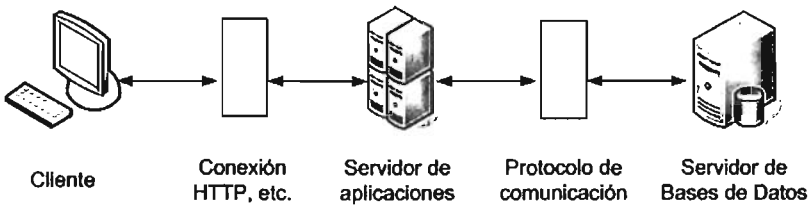

Figura 10.2: Modelo de tres capas.

Tipos de controladores

Un controlador JDBC es una clase que implementa la interfaz 'JDBC Driver', acepta peticiones SQL por parte del programa y manipula esto para crear peticiones a una base de datos dada. Existen 4 tipos de controladores, su clasificación se basa en la forma en que opera cada uno de ellos.[2]

Protocolo nativo completamente en Java

Este tipo de controlador convierte las llamadas JDBC en llamadas del protocolo de red utilizado directamente por el SMBD. Esto permite llamadas directas desde la máquina cliente al servidor del SMBD y conjunta una solución práctica para accesos en 'Intranets'. El controlador es independiente de la plataforma y una vez compilado se puede utilizar en cualquier sistema. PostgreSQL proporciona un controlador de esta clase.

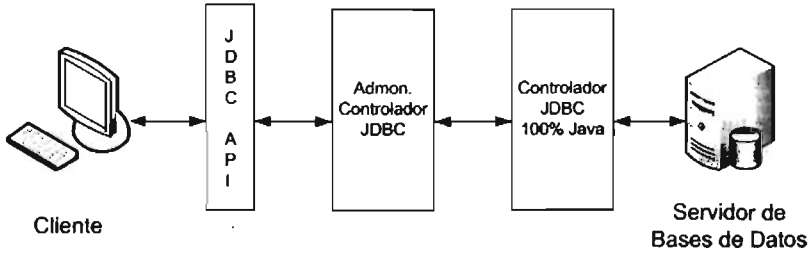

Figura 10.3: Acceso a una base de datos por medio de un controlador JDBC Tipo 4.

[2] Para más información sobre los distintos tipos de controladores JDBC, se puede visitar la página `http://java.sun.com/products/jdbc/driverdesc.html`

Actividad 10.1

Revisa donde se encuentra instalado java dentro de tu sistema (actualiza tus variables de entorno) con el siguiente comando:

```
#bash=> which java
```

Investiga si se encuentran instalados los controladores JDBC de PostgreSQL:

```
#bash=> find -name postgresql*.jar /
```

De no ser así, descarga el ultimo controlador JDBC de la siguiente página: `http://jdbc.postgresql.org/` y procede a instalarlo.

Asegúrate de actualizar tus variables de entorno si es necesario (pon atención especial en `CLASSPATH`).

10.3.3. Proceso general de conexión

El proceso de conexión a una base de datos desde JDBC se compone, de manera general, de las siguientes etapas:

- Establecer la conexión a la base de datos.
- Crear y enviar las sentencias SQL al SMBD.
- Recibir la información y procesar estos resultados.

Estableciendo la conexión a la base de datos

La primera tarea que debe ejecutarse, es importar el controlador JDBC, esto se logra con la siguiente instrucción en java:

```
import java.sql.*;
```

posteriormente hay que cargar el controlador, esto se lleva a cabo con la instrucción siguiente:

```
Class.forName("org.postgresql.Driver");
```

existe otro método [13], pero ciertamente este es el más sencillo.

Dentro de JDBC, una base de datos es representada por un *URL* (Uniform Resource Locator) y dentro de PostgreSQL, este toma alguna de las siguientes estructuras:

- jdbc:postgresql:nombreBaseDeDatos
- jdbc:postgresql://servidorBaseDeDatos/nombreBaseDeDatos
- jdbc:postgresql://servidorBaseDeDatos:puerto/nombreBaseDeDatos

Para obtener la conexión a la base de datos, es necesario crear una instancia de la clase 'Connection' de JDBC. y posteriormente asignar la conexión al ejecutar el método 'DriverManager.getConnection()'. Ejemplo general de su uso es el siguiente:

```
Connection db = DriverManager.getConnection(url, usuario, contraseña);
```

hay que recordar, que usuario y contraseña, hacen referencia al usuario de la base de datos y su contraseña que tenga asignada, además son cadenas de texto que pueden provenir de múltiples objetos de Java™ ya sea mediante la lectura a partir de un archivo, de la entrada estándar por parte del usuario u otras.

En caso de tener éxito la ejecución del método, se obtiene una conexión que puede utilizarse para interactuar con el SMBD enviando las sentencias SQL apropiadas.

Creación y envío de sentencias SQL al SMBD

Una vez establecida la conexión a la base de datos por parte de la aplicación, es posible enviar sentencias SQL al SMBD. Para este propósito, se cuenta con tres clases para trabajar con las sentencias SQL.

- `Statement`.- Trabaja con una sentencia SQL sin parámetros.
- `PreparedStatement`.- Utilizado con sentencias SQL que se ejecutan con frecuencia, por tanto son pre-compiladas y pueden tomar parámetros.
- `CallableStatement`.- Trabaja con procedimientos almacenados.

Es preciso aclarar que en el presente trabajo solamente se estudia a la clase 'Statement'. Un objeto 'Statement' es el que envía nuestras sentencias SQL al controlador de la base de datos. Para ello, sólo se crea una instancia del objeto y se ejecuta, pasando como parámetro la sentencia SQL que desea enviar. Para una sentencia SELECT, el método a ejecutar es 'executeQuery'. Para sentencias que crean o modifican tablas, el método a utilizar es 'executeUpdate'[3] .

Es necesario que exista un objeto Connection que represente una conexión activa para el objeto Statement. En el siguiente ejemplo, se utiliza un objeto Connection llamado 'db'.

[3] Para mayor información sobre los métodos que implementa Statement, consultar `http://java.sun.com/j2se/1.5.0/docs/api/`

```
Statement sentencia = db.createStatement();
```

en ese momento se crea una instancia del objeto Statement '(sentencia)', pero no tiene ninguna sentencia SQL para enviar al controlador de la base de datos. Para lograr esto, se necesita ejecutar un método 'executeQuery' ó 'executeUpdate', dependiendo de la finalidad. Por ejemplo, en el siguiente fragmento de código, se suministra executeUpdate con la sentencia SQL del ejemplo anterior.

```
sentencia.executeUpdate("CREATE TABLE prueba "+
                        "(id_prueba INTEGER, nombre VARCHAR(30)"+
                        "cantidad INTEGER, total INTEGER);");
```

es importante mencionar que el método 'executeUpdate' recibe como parámetro una cadena, por lo tanto la asignación de la sentencia SQL puede llevarse a cabo en otro momento del programa y al ejecutar 'executeUpdate', sólo basta con enviar como parámetro la cadena en cuestión. En este caso interesa únicamente el método 'executeUpdate' puesto que permite enviar sentencias SQL las cuales no regresan resultados por parte de la base de datos, son sentencias DDL.

Por otro lado, el método más utilizado para ejecutar sentencias DML de SQL es 'executeQuery'. Este método se utiliza para ejecutar sentencias SELECT, que comprenden la mayoría de las sentencias SQL.

Procesar resultados

El método 'executeQuery' se utiliza para enviar las sentencias DML de SQL, dado que en una ejecución correcta entrega los resultados de la consulta en un objeto llamado 'ResultSet', por eso es necesario declarar un objeto de la clase y crear una instancia del mismo, la cual contendrá los resultados. Un ejemplo es el siguiente:

```
ResultSet resultados = sentencia.executeQuery (
                "SELECT nombre, grado, cod_post FROM profesor");
```

el objeto 'resultados' contiene las tuplas de resultado de la consulta. Para acceder a ellas se utiliza el método 'next()', que devuelve 'true' si hay datos y 'false' en otro caso. Este método utiliza un 'cursor' para desplazarse sobre el conjunto de tuplas del resultado, inicialmente se posiciona justo antes de la primera tupla de un objeto 'ResultSet'. Esto implica que primero se debe ejecutar el método next para mover el cursor a la primera tupla y convertirla en la tupla actual. Sucesivas invocaciones del método 'next' moverán el cursor de tupla en tupla secuencialmente.

Actividad 10.2

Inicia una sesión dentro de tu base de datos y verifica con el comando '\d' las tablas que tienes actualmente.
Crea un archivo llamado 'conecta.java' que contenga el siguiente código.

```
import java.sql.*;
public class conecta {
  public static void main(String args[]){
    String url = "jdbc:postgresql";
    String database = "nombre_bd";
    String username = "usuario";
    String password = "contrasena";
    Connection conexion = null;
    Statement sentencia = null;
    try{
       Class.forName("org.postgresql.Driver");
       conexion = DriverManager.getConnection(url +":"+ database, username, password);
       System.out.println("Conexion exitosa");
       sentencia = conexion.createStatement();
       sentencia.executeUpdate("CREATE TABLE prueba ( id_prueba INTEGER,"+
                    "nombre VARCHAR(30), cantidad INTEGER, total INTEGER)");
    }
    catch (SQLException e){
       System.out.println("Error - "+ e );
    }
    finally {
       if (connection!= null){
          try{ connection.close();}
          catch(SQLException e) {
          System.out.println("Error - "+ e );
          }
       }
    }
  }
}
```

Guarda y compila el archivo. Ejecuta el programa creado y verifica que se haya creado a tabla 'prueba'.

A partir de la versión 2.0 de JDBC, se puede manipular la posición del cursor en

cualquier dirección (también se manejan posiciones absolutas y relativas). Para extraer cada atributo de la tupla actual se debe usar un método '`getXXX`' de acuerdo al tipo de dato que es el atributo. Se presenta una lista con los tipos básicos de SQL y sus respectivos métodos get en la tabla 10.1.

Tipo SQL	Tipo Java™	getXXX
BIT	boolean	getBoolean
SMALLINT	short	getShort
INTEGER	int	getInt
BIGINT	long	getLong
REAL	float	getFloat
FLOAT	double	getDouble
DOUBLE	double	getDouble
NUMERIC	java.math.BigDecimal	getBigDecimal
CHAR	java.lang.String	getString
VARCHAR	java.lang.String	getString
DATE	java.sql.Date	getDate
TIME	java.sql.Time	getTime
BINARY	byte[ ]	getUnicodeStream

Tabla 10.1: Métodos getXXX recomendados para recuperar datos.

Para mantener un acceso consistente sobre los datos que estén contenidos en un objeto '`ResultSet`', cualquier estructura de iteración puede ser utilizada. Así, una vez resuelta la conexión a la base y asignado el '`ResultSet`', es posible realizar lo siguiente:

```
String nombre = "";
String grado = "";
int codigo_postal = 0;

while (resultados.next()) {
       nombre = resultados.getString("nombre");
       grado = resultados.getString("grado");
       codigo_postal = resultados.getInt("cod_post");
       System.out.println("Profesor : "+ nombre + "Grado máximo : "+
              grado + "Código postal : "+ codigo_postal);
}
```

es importante observar dos detalles: primero que para cada tipo de dato obtenido, es necesario crear una instancia del objeto correspondiente, según la tabla 10.1; y que para obtener los datos correspondientes a la primera tupla del resultado es necesario hacer una invocación al método '`next()`' del objeto '`ResultSet`' en cuestión ('`resultados`' en el ejemplo).

También es posible obtener los datos pasando como parámetro la posición que ocupan los datos dentro de cada tupla del 'ResultSet'. Un ejemplo que demuestra este tipo de referencias, se presenta a continuación:

```
while (resultados.next()) {
       String nombre = resultados.getString(1);
       String grado = resultados.getString(2);
       int codigo_postal = resultados.getInt(3);
       System.out.println("Profesor : "+ nombre + "Grado máximo : "+
              grado + "Código postal : "+ codigo_postal);
}
```

10.4. Ejercicios

1. Crea un programa en Java™ utilizando JDBC, que permita la conexión a la base de datos del curso y la interacción con la misma ofreciendo las siguientes funcionalidades:

   - Creación de nuevas tablas.
   - Inserción de tuplas.
   - Eliminación de tuplas.
   - Eliminación de tablas.

Segunda Parte
Sistemas Manejadores de Bases de Datos

Acceso a memoria

Semana:	Segunda semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de SQL. Programación en lenguaje Perl. Manejo de archivos (E/L).
Herramientas:	Intérprete de Perl.
Objetivo:	Observar la diferencia de velocidades presente entre los distintos tipos de memoria.

La práctica once es la primera relativa al segundo curso, Sistemas Manejadores de Bases de Datos, y se lleva a cabo durante la segunda semana del curso, dado que es necesario contar con una pequeña introducción al curso por parte del profesor y de los ayudantes.

Dentro de esta práctica, se muestra la importancia que tiene el factor de lecturas y escrituras a disco dentro de cualquier sistema de cómputo.

PRÁCTICA 11

Acceso a Memoria

11.1. Meta

Que el alumno observe el impacto en el rendimiento de un sistema de cómputo, debido al factor de escrituras y lecturas en un dispositivo físico como el disco duro.

11.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar la importancia de los dispositivos físicos.
- Apreciar las distintas velocidades a las que opera cada uno de los niveles de la jerarquía de memoria.
- Determinar el impacto que tienen las operaciones de escritura o de lectura (E/L) de un disco duro en el rendimiento de un sistema.

11.3. Desarrollo

En principio, la creación y el desarrollo de programas y sistemas de software requieren de memoria, un elemento fundamental para su funcionamiento, puesto que ahí se manipulan todos los datos que intervienen en las operaciones[1] . El escenario ideal sería

[1] De hecho, también requieren tener en memoria al programa, esto por ser computadoras con *arquitectura de Von Neumman.*

contar con gran cantidad de memoria y de gran velocidad o desempeño, pero esto obviamente no es posible en todos los casos. Es por eso, que desde los inicios de la computación como tal se ha definido una jerarquía de memoria ([29] y [3]), en la cual se clasifica a esta, tomando en cuenta su velocidad y costo.

11.3.1. La jerarquía de memoria

Es posible definir múltiples niveles en esta jerarquía de memoria, pero se presenta una clasificación básica que consta de 5 niveles, a saber en orden decreciente respecto a la velocidad con que operan:

1. Los registros del procesador.
2. El cache L1.
3. El cache L2.
4. La memoria RAM.
5. El disco duro.

Cada nivel es un subconjunto del anterior[2] , pero al ser más lento, es sustancialmente más barato y por ello existe la posibilidad de tener mayor capacidad. Es importante remarcar que la velocidad con la cual se desempeña cada uno de los dispositivos es sustancialmente diferente, por ejemplo, el orden de magnitud en el cual la memoria RAM ejecuta sus transferencias de datos se ubica dentro de los nanosegundos, mientras que el disco duro tiene una tasa de transferencia sobre la métrica de los milisegundos. Esto indica que por cada transferencia que realiza el disco duro, la memoria RAM ha realizado posiblemente un millón de operaciones.

En el área de bases de datos, el papel que toman los discos duros es fundamental, ya que aún cuando el mayor número de respaldos de información se hace sobre cintas magnéticas, el disco duro es el dispositivo por excelencia donde se mantienen las base de datos operacionales y de análisis.

Por otro lado, el SMBD como cualquier programa en ejecución dentro del sistema operativo, consume la memoria disponible del mismo; ya sea por el proceso mismo del servidor (postmaster en nuestro caso), por los procesos que se encargan de las tareas administrativas y que también se ejecutan en segundo plano o bien por los procesos que atienden a las consultas que recibe. Eventualmente la memoria RAM disponible disminuye y el sistema operativo se encarga de hacer uso de la memoria virtual, esto es escribir al disco duro los datos con los cuales se esta trabajando y en último término, retrasar la ejecución de las consultas por la transferencia de datos que se presenta entre el disco y la memoria RAM. Por esto, las tareas que abarcan lecturas y escrituras a disco u otros dispositivos de almacenamiento masivo, deben llevarse a cabo con la menor frecuencia posible para aumentar el desempeño general del sistema.

[2] Generalmente sucede esto pero hay casos donde no, como se menciona en [5].

11.3.2. Perl

"Perl" (Practical Extraction and Report Languaje) es un lenguaje de programación creado a principio de la década de los años noventa por Larry Wall, que cuenta con distribuciones gratuitas[3] .

Perl es un lenguaje creado inicialmente para la manipulación de cadenas de caracteres, archivos y procesos. Esta manipulación se ve simplificada por el importante número de operadores a disposición del usuario. El lenguaje Perl se percibe habitualmente como un lenguaje intermedio entre los *shell scripts* y la programación en C, ya que los programas escritos en este lenguaje de programación son una sucesión de instrucciones y son similares a los shell scripts porque no existe un procedimiento principal como la subrutina *main* en C. Sin embargo, es similar al lenguaje C en su sintaxis y en el número funciones que permiten la manipulación de cadenas de caracteres y archivos.

El lenguaje Perl no es precompilado, pero aún así es más rápido que la mayoría de lenguajes interpretados, en especial que el Bourne Shell. Esto se debe a que los programas escritos en Perl son analizados, interpretados y compilados por el intérprete *perl* antes de su ejecución.

Estas características hacen que en general el mantenimiento y la depuración de un programa en Perl sea una tarea mucho más sencilla que el mismo programa escrito en C, y en particular permitirá, dado lo mínimo de las instrucciones necesarias, apreciar la diferencia radical que existe entre el manejo de la información en la memoria principal y el manejo de la misma información cuando se presentan las escrituras y lecturas a disco duro.

Actividad 11.1

Verifica que Perl se encuentre instalado en tu sistema:

```
#bash=>which perl
/usr/bin/perl
#bash=>
```

En caso contrario, descarga e instala el archivo en formato '.rpm' de la siguiente dirección: `http://www.activestate.com/Products/Download/Download.plex?id=ActivePerl`

Hay que aclarar que no se pretende dar un tutorial sobre Perl, únicamente se mostrarán los elementos que son útiles para la realización de la práctica. Además la optimización de código tampoco es un tema de la presente práctica.

[3] Documentación de Perl en: `http://www.perl.com/pub/q/documentation`

Archivos

Uno de los usos más probados y comunes de Perl es el procesamiento de archivos de texto para generar otros archivos de texto[4] que implica cierto proceso sobre los datos leídos originalmente. Además, Perl no conoce otra entrada, más que la proveniente de archivos (asociando la entrada, salida y salida de errores con archivos de ambiente). El ciclo del uso normal de un archivo en Perl, es el siguiente:

1. Apertura.- Mediante la función 'open' inicializa una variable de archivo.
2. Uso.- Lectura, generalmente secuencial por líneas.
3. Cerrado .- Mediante la función '`close`'.

Perl posee distintas implementaciones para manipular el contenido de archivos binarios y archivos de texto en forma aleatoria, pero el uso de archivos de texto secuenciales es lo mas común. Hay tres tipos de archivos, los cuales se manejan de modo muy similar:

- Archivos comunes.
- Programas de los que se obtiene la entrada o la salida.
- La entrada, salida o salida estándar de errores.

Apertura

La entrada, salida y salida de errores estándar, están abiertas por omisión, pero en cualquier otro sentido se utilizan igual que cualquier archivo, debe tenerse cuidado al planificar los entubamientos de información para evitar que los programas esperen por siempre una entrada que no le ha de llegar.

Los archivos convencionales se abren con la instrucción 'open', en la figura 11.1 se muestra su sintaxis básica.

```
open VARCHIVO, EXPRESION
```

Figura 11.1: Sintaxis de la función `OPEN`.

Donde `VARCHIVO` es el nombre de la variable mediante la que se hace referencia al archivo, no tiene indicador de clase y por convención se maneja en mayúsculas; y `EXPRESION` es el nombre y modo en que habrá de abrirse el archivo.

Los archivos, se dan con los nombres nativos del sistema operativo anfitrión y los modos se indican al inicio del nombre con las cadenas mostradas en la figura 11.2.

[4] En la presente práctica, se necesita crear archivos con extensión '.sql' para cargar los datos posteriormente en PostgreSQL.

Modo	Descripción
<archivo	Abre 'archivo' para lectura.
>archivo	Abre 'archivo' para escritura, borrándolo si existe.
>>archivo	Abre 'archivo' para escritura, agregando la nueva información al final
+<archivo	Abre 'archivo' para lectura y escritura.
+>archivo	Abre 'archivo' para lectura y escritura borrando 'archivo' al abrirlo.
| programa	Ejecuta 'programa' y reasigna su entrada estándar a lo que se escriba en el.
programa |	Ejecuta 'programa' y reasigna su salida estándar para lectura de nuestro programa.

Figura 11.2: Modos básicos de apertura de archivos en Perl.

Actividad 11.2

En Perl se tienen tres tipos de datos básicos y dos especiales. A saber: escalares, arreglos asociativos, hash, de archivo y Glob (obsoleto en Perl 5).

Investiga en que consiste cada tipo.

Uso y Cerrado

Básicamente, se puede usar un archivo de dos formas, por renglones, (terminados por el carácter de vuelta de carro) o por carácter (byte). El operador que permite leer de un archivo es `'<>'` teniendo la variable de archivo dentro de el, así por ejemplo:

```
$reng=<STDIN>;
```

lee un renglón de la entrada estándar y lo coloca en la variable '`$reng`', incluida la vuelta de carro que termina al renglón. Si el renglón termina con carácter de fin de archivo, este no se incluye en el renglón, y lecturas sucesivas de `STDIN` esperaran indefinidamente por una nueva entrada[5] .

Por otro lado, también es valido utilizar lo siguiente:

```
@contenido=<STDIN>;
```

aquí se esta evaluando al operador `'<>'` en un contexto de arreglo, y su comportamiento varía, como arreglo `'<>'` regresa un arreglo de cadenas, donde cada cadena es

[5] `STDIN` es la variable de archivo que identifica la entrada estándar, así como `STDOUT` es la variable que identifica la salida estándar y `STDERR` es la que identifica la salida estándar de errores.

un renglón del archivo, de modo que la expresión anterior leerá todo lo que se introduzca en la entrada estándar y lo coloca en el arreglo @contenido. Esta es una forma más segura de usar la entrada estándar, pues ahora cada elemento del arreglo equivale a una lectura sucesiva de STDIN en contexto escalar.

Si se sustituye STDIN por cualquier otra variable de archivo (o valor de variable de referencia a una variable de archivo) entonces se estará cargando el archivo ya sea renglón por renglón o todo completo a variables de memoria.

La función open regresa al evaluarse el valor de la variable de archivo o nulo si no se puede abrir el archivo en el modo deseado, de modo que si se utiliza como condición lógica resultara verdadera en caso de lograr abrir el archivo. Ejemplos de esto se ilustran en la figura 11.3.

`open(DATOS,"</tmp/datos.sql");`	Apertura de lectura del archivo /tmp/datos.sql asociado a la variable de archivo DATOS.
`open(SAL,">salida.txt");`	Creación del archivo "salida.txt" en el directorio desde donde se invoque el programa, sólo de escritura y asociado a la variable SAL.

Figura 11.3: Ejemplos de apertura de archivos en Perl.

El cerrado de los archivos en una operación recomendable ya que garantiza que todos los buffers han sido vaciados (o regresa falso en caso de que haya algún error) y se lleva a cabo con la función 'close'.

Lectura de datos

La lectura de la información contenida en los archivos, se realiza con la operación '<>'. El operador '<>' realiza la lectura de texto de el archivo, cuya variable se coloca entre < y >. Ahí es evaluado en un contexto de escalar, regresa un renglón, incluyendo el carácter de fin de línea. Si se evalúa en contexto de lista, regresa un arreglo de todos los renglones del archivo incluyendo sus terminadores de línea. Por ejemplo:

```
$reng=<ENTRADA>;
```

carga una línea del archivo asociado a ENTRADA (que debe ser de lectura) y

```
@contenido=<ENTRADA>;
```

toma todas las líneas del archivo y las coloca como elementos consecutivos del arreglo `@contenido`.

Debe mencionarse, que cuando se evalúa el operador '<>' para un archivo, el apuntador de posición en el archivo avanza, de modo que en aplicaciones sucesivas (de contexto escalar) se obtienen renglones sucesivos. Y si se aplica en contexto de arreglo, una evaluación sucesiva no regresara más renglones. Además, el operador '<>' tiene la propiedad de tomar valor por defecto cuando no se especifica que archivo debe usar, esta característica tiene dos modalidades:

1. El ultimo archivo abierto por 'open'.
2. El siguiente parámetro como nombre de archivo a abrir y procesar como de lectura.

Escritura

Una vez abierto un archivo para escritura, la forma mas común de escribir en él, es mediante la función `print`, de hecho, `print` esta diseñada para escribir a archivos, pero el archivo que utiliza por defecto para escribir es el asociado a la salida estándar `STDOUT`, la sintaxis para especificar que archivo debe utilizarse se puede apreciar en la siguiente figura.

```
print ARCHIVO LISTA
```

Figura 11.4: Sintaxis de la función `PRINT`.

Donde `ARCHIVO` es la variable de tipo archivo donde se dirigirá la salida y `LISTA` es una lista con los valores a imprimir, (si solamente es un escalar se interpreta como una lista de un solo elemento).

Si se desea emplear las variables de archivo guardadas en arreglos, hashes o como resultado de expresiones se les deberá poner en un bloque de código (entre { })[6] .

Actividad 11.3

Investiga que utilidad poseen las siguientes funciones de Perl.
1. 'seek'.
2. 'chop'.
3. 'rand'.

[6] Para mas información consultar la función '*print*' en la referencia de Perl.

11.4. Ejercicios

1. Crea un programa en Perl que genere un archivo llamado 'insertaClientes.sql' con las sentencias de inserción[7] de los datos de la tabla 'clientes' de la base de datos del curso tal que:

   a) El programa abra los tres archivos de texto proporcionados ('nombres.txt', 'apellidos.txt' y 'paises.txt'), busque aleatoriamente en cada uno de ellos un renglón, tome el valor, lo imprima en una línea del archivo 'insertaclientes.sql' y cierre nuevamente cada uno de los archivos. Repita el proceso para generar los clientes necesarios.

   b) El programa abra los tres archivos de texto proporcionados ('nombres.txt', 'apellidos.txt' y 'paises.txt'), busque aleatoriamente en cada uno de ellos un renglón, tome el valor, lo imprima en una línea del archivo 'insertaClientes.sql', deje abiertos los archivos, repita el proceso para generar los clientes necesarios y por último cierre los archivos.

   c) El programa abra los tres archivos de texto proporcionados ('nombres.txt', 'apellidos.txt' y 'paises.txt'), asigne cada uno a un arreglo y busque aleatoriamente en cada uno de ellos un elemento, tome el valor, lo imprima en una línea del archivo 'insertaClientes.sql', repita el proceso para generar los clientes necesarios y por último cierre los archivos.

2. Genera un reporte de tus experimentos al llevar a cabo la ejecución de cada programa para crear: 10, 1000, 100000, 1000000 y 7000000 de clientes.
   Incluye una gráfica comparativa y las conclusiones con respecto al rendimiento de cada programa en función del tiempo necesario para generar los archivos[8] .

[7] `INSERT INTO 'tabla' VALUES (lista_de_valores);`

[8] No olvides incluir en el reporte la configuración del equipo en donde ejecutaste las pruebas, tanto de hardware como de software.

Niveles RAID

Semana:	Tercera semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos de SQL. Conceptos de niveles RAID. Conceptos de administración de sistemas. Conceptos sobre particiones de discos.
Herramientas:	Sistema Operativo Linux - Distribución Fedora Core 3. PostgreSQL v.8.0.2
Objetivo:	Observar la implementación de un arreglo de discos con niveles RAID 0, 1 y 5 vía software y cuantificar los beneficios que aporta esto.

La práctica doce introduce al alumno de manera sencilla y directa con los niveles RAID y todos los beneficios que se presentan en ellos. Es recomendable que se cuente con el hardware adecuado disponible para la realización de la práctica y los permisos necesarios para la modificación de la configuración del software y hardware. Además de experiencia en el área administrativa dentro del sistema operativo Linux.

PRÁCTICA 12

Niveles RAID

12.1. Meta

Que el alumno comprenda la importancia que tiene la implementación de los niveles RAID en un SMBD así como todos los beneficios que aporta esto.

12.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Implementar distintos niveles RAID dentro de un sistema.
- Apreciar las mejoras en cuanto a rendimiento y robustez obtenidas al implementar los distintos niveles RAID.
- Elegir correctamente el nivel de RAID adecuado para un sistema dependiendo de las necesidades y metas.

12.3. Desarrollo

Los discos duros, como todos los dispositivos en una computadora, han ido disminuyendo en precio y aumentando en capacidad, por eso, ahora es económicamente viable el reunir gran número de discos duros en un sistema.

El tener muchos discos mejora el desempeño en la escritura y lectura de datos, si estos están conectados en paralelo. Más aun, el tenerlos así puede aumentar la fiabilidad

del almacenamiento de datos, dado que información redundante puede almacenarse en múltiples discos. Esto es, la falla de un disco, no supone la falla total del sistema.

12.3.1. RAID

Una variedad de organizaciones en discos duros, conocida como RAID ('*Redundant Array of Inexpensive Disks*' - Arreglo Redundante de Discos No costoso[1]) ha sido propuesta para manejar este problema de fiabilidad y desempeño. Hay que hacer hincapié de que la operatividad de los discos en RAID se da en función de mejorar el desempeño general del sistema y su robustez y no por cuestiones económicas de almacenamiento.

Los primeros niveles RAID fueron dados en 1987 [26], donde se definieron inicialmente 5 niveles RAID, aunque posteriormente se agregaron 2 más y de este conjunto se han generado combinaciones que abarcan características de dos o más niveles. Los niveles RAID ofrecen grandes diferencias entre rendimiento e integridad de los datos, dependiendo de las especificaciones de cada nivel. No hay un nivel RAID perfecto para todos los usuarios, ya que cada uno de ellos cumple distintos propósitos. A continuación se presentan los niveles RAID que se implementarán en la práctica.

RAID 0: Data Striping Without Parity (DSA)

RAID 0:
Datos en bandas de discos sin paridad y sin corrección de errores.

También conocido como "separación ó fraccionamiento"[2] . Los datos se desglosan en pequeños segmentos y se distribuyen entre varias unidades. Este nivel de arreglo no ofrece tolerancia al fallo. Al no existir redundancia, RAID 0 no ofrece ninguna protección de los datos. El fallo de cualquier disco del arreglo tendría como resultado la pérdida de los datos y sería necesario restaurarlos desde una copia de seguridad. Por lo tanto, RAID 0 no se ajusta realmente al acrónimo RAID.

Consiste en una serie de unidades de disco conectadas en paralelo que permiten una transferencia simultánea de datos a todos ellos, con lo que se obtiene una gran velocidad en las operaciones de lectura y escritura. La velocidad de transferencia de datos aumenta en relación al número de discos que forman el conjunto. Esto representa una gran ventaja en operaciones secuenciales con archivos de gran tamaño. Por lo tanto, este arreglo es aconsejable en aplicaciones de tratamiento de imágenes, audio, vídeo o CAD/CAM, es decir, es una buena solución para cualquier aplicación que necesite un almacenamiento a gran velocidad pero que no requiera tolerancia a fallos.

[1] 'Inexpensivo' sería una traducción literal, pero esa palabra no existe en nuestro idioma. Además el termino 'Inexpensive' ha sido remplazado en múltiples ocasiones por 'Independent', pero el primer término es el correcto ya que con ese fue ideado inicialmente por investigadores de la Universidad de California en Berkeley, en 1987 [26].

[2] '*Striping*'.

Se necesita un mínimo de dos unidades de disco para implementar una solución RAID 0.

RAID 1: Mirrored Disk Array (MDA)

RAID 1:
Conjunto de discos en espejo.

Se basa en la utilización de discos adicionales sobre los que se realiza una copia en todo momento de los datos que se están almacenando. RAID 1 ofrece una excelente disponibilidad de los datos mediante la redundancia total de los mismos.

De esta manera se asegura la integridad de los datos y la tolerancia al fallo, pues en caso de avería, se sigue trabajando con los discos no dañados sin detener el sistema. Los datos se pueden leer desde la unidad duplicada sin que se produzcan interrupciones.

RAID 1 es una alternativa costosa para los grandes sistemas, ya que las unidades se deben añadir en pares para aumentar la capacidad de almacenamiento y se desaprovecha la mitad de la capacidad total del conjunto de discos.

Sin embargo, RAID 1 es una buena solución para las aplicaciones que requieren redundancia cuando hay sólo dos unidades disponibles. Los servidores de archivos pequeños son un buen ejemplo.

Se necesita un mínimo de dos unidades para implementar una solución RAID 1.

RAID 5: Independent Disk Array (IDA)

RAID 5:
Sistema de discos con acceso independiente con integración de códigos de error mediante una paridad.

Este arreglo ofrece tolerancia al fallo, pero además, optimiza la capacidad del sistema permitiendo una utilización de hasta el 80 % de la capacidad del conjunto de discos. Esto lo consigue mediante el cálculo de paridad de la información y su almacenamiento alternativo por bloques en todos los discos del conjunto. De esta manera, si cualquiera de las unidades de disco falla, se puede recuperar la información en tiempo real, mediante una operación lógica de 'O exclusivo', sin que el servidor deje de funcionar.

RAID 5 guarda la paridad de los datos dentro de los discos y no hace falta un disco dedicado para guardar dichas paridades. La paridad se genera haciendo un **XOR** de los datos, creando la zona de paridad PAR, la paridad nunca se guarda en los discos que contienen los datos que han generado dicha paridad, ya que en el caso que uno de ellos se estropease, bastaría con aplicar la operación **XOR** en los datos restantes para que el dato almacenado en el disco descompuesto, se restablezca. RAID 5 no asigna un disco específico a esta tarea sino que asigna un bloque alternativo de cada disco para esta escritura. Al distribuir la función de comprobación entre todos los discos, se disminuye el cuello de botella y con una cantidad suficiente de discos puede llegar a eliminarse completamente, proporcionando una velocidad equivalente a un RAID 0. RAID 5 es

el nivel de RAID más eficaz y el de uso preferente para las aplicaciones de servidor básicas.

Se necesita un mínimo de tres unidades para implementar una solución RAID 5, aunque su resultado óptimo de capacidad se obtiene con siete o más unidades. RAID 5 es la solución más económica por megabyte, que ofrece la mejor relación de precio, rendimiento y disponibilidad para la mayoría de los servidores.

12.3.2. Hardware RAID

Actualmente la implementación de un sistema RAID puede llevarse a cabo mediante la utilización de hardware dedicado o bien simular su funcionamiento mediante el software. En el primer caso, el sistema basado en el hardware gestiona el subsistema independientemente de la máquina y presenta a la misma un único disco por conjunto de discos RAID.

Un ejemplo del hardware RAID es el que se conecta a un controlador SCSI y presenta el conjunto de discos RAID en una sola unidad de disco. Un sistema externo RAID se encarga de mover la "inteligencia" RAID a un controlador que se encuentra en un subsistema de discos externo. Todo el subsistema está conectado a la computadora con un controlador SCSI normal y para esta es como si se tratara de una sola unidad de disco.

Los controladores RAID también tienen la forma de tarjetas que actúan como un controlador SCSI del sistema operativo pero se encargan de todas las comunicaciones del disco actual. En estos casos, se conectan las unidades de disco al controlador RAID como si se tratara de un controlador SCSI pero se tienen que añadirlas a la configuración del controlador RAID; de todas maneras el sistema operativo nunca nota la diferencia.

12.3.3. Software RAID

El software RAID implementa los diversos niveles de RAID en el código del kernel (dispositivo de bloque) del sistema operativo Linux[3] . Ofrece la solución más barata ya que las tarjetas de controladores de disco y los chasis "hot-swap" son bastante costosos[4] y no se requieren. El software RAID funciona con discos ATA, SATA (que son más baratos) y SCSI.

El controlador MD ('*Multiple Devices*') en el kernel de Linux es un ejemplo de la solución RAID que es completamente independiente del hardware. El rendimiento del conjunto de discos del software RAID depende del rendimiento y de la carga que presenta el procesador en un momento dado. El software RAID, cuenta con las siguientes características principales:

- Proceso de reconstrucción de subprocesos.

[3] En nuestro caso particular en la distribución Fedora Core 3, ya que el kernel ofrece soporte desde su versión 2.4.X.

[4] Un chasis de "hot-swap" permite quitar un disco duro sin tener que apagar la computadora.

- Configuración basada en el kernel.
- Portabilidad de los conjuntos de discos entre máquinas Linux sin reconstrucción.
- Reconstrucción de los conjuntos de discos con el uso de los recursos que no se usan del sistema.
- Soporte para las unidades de disco en las que se pueden hacer cambios "en caliente" (hot-swappable).
- Detección automática de procesador con el objetivo de obtener beneficios de las mejoras del mismo.

El RAID por software, dada su naturaleza, tiende a ser una solución más flexible que el RAID por hardware. el lado negativo de esto, es que en general se requiere más ciclos del procesador, aunque con los procesadores 'rápidos' de hoy en día, el rendimiento del software RAID aumenta considerablemente con respecto al hardware RAID.

Actividad 12.1

Dado que en Fedora Core 3, el software RAID es mas sencillo configurarlo durante la instalación. Será preciso contar una máquina para este propósito

Inicia el proceso de instalación de Fedora Core 3.
Durante la **Configuración de la partición del disco**, selecciona

Partición manual con Disk Druid.

En **Disk Druid**, elije **Nuevo** para crear una nueva partición.

Crea las particiones "*/boot*" , "/" (raíz) con el **Tipo de sistema de archivos** ext3, además del "*swap*".

Crea otra partición y selecciona **software RAID** desde el menú **Tipo de sistema de archivos** (observa que no se puede elegir un punto de montaje esto se hace una vez que se ha creado el dispositivo RAID).

Para **Unidades admisibles**, selecciona el(los) disco(s) donde se quiera configurar el arreglo RAID. Si se cuenta con varios discos, todos los discos podrán ser seleccionados desde aquí y deberán anularse de la selección los discos que **no** tengan un arreglo RAID.

Introduce el tamaño de la partición.

Selecciona **Tamaño fijo** para hacer la partición de un tamaño especificado,

Haz click en **OK** para volver a la pantalla principal.

Repite estos pasos para crear tantas particiones como necesite la configuración RAID.

Ten en cuenta que no todas las particiones no tienen porqué ser RAID.
Por ejemplo, se puede configurar tan sólo la partición */home* como un dispositivo RAID por software.

Actividad 12.1 - continuación

Una vez que se hayan creado todas las particiones como particiones **software RAID**, hay que hacer lo siguiente:

Selecciona el botón **RAID** en la pantalla principal de particionamiento **Disk Druid**.

A continuación, aparecerá un dialogo nuevo donde se puede crear un dispositivo RAID.

Introduce un punto de montaje llamado */psql*.

Selecciona por tipo de sistema de archivos para la partición recién creada: 'ext3'.

Selecciona **md0** como el nombre del dispositivo RAID.

Elige RAID 0.

Las particiones RAID que se acaban de crear aparecerán en la lista **Miembros RAID**.

Selecciona cuales particiones de éstas deben ser usadas para crear el dispositivo RAID.

Después de hacer click en **OK**, el dispositivo RAID aparecerá en la lista **Descripción de la unidad**.

Llegados a este punto, podemos continuar con el proceso típico de instalación. pero **NO** selecciones la instalación de PostgreSQL.

Después de tener tu sistema ya instalado, descarga la última versión disponible de PostgreSQL e instálala. Teniendo el cuidado de hacerlo sobre la partición creada en la actividad anterior, así como también de ubicar el directorio de datos en dicha partición.

12.4. Ejercicios

1. Realiza la instalación de otros dos sistemas, uno con nivel RAID 1 y otro con nivel RAID 5.

2. Carga una nueva base de datos en todos los sistemas utilizando sentencias 'INSERT INTO'.

   a) Para 10000 tuplas en la tabla.

   b) Para 1000000 tuplas en cada tabla.

3. Ejecuta las siguientes consultas:

   a) Obten el nombre de todos los cliente cuyo nombre empieza con las letras 'Ma'.

   b) Obten el nombre de todos los clientes que hay realizado una compra en el año 2004.

   c) Obten el nombre y apellido de todos los clientes que han comprado algún videojuego de la consola 'PS2'.

   d) Obten para cada consola, el nombre de la consola y el total de videojuegos disponibles para ella.

   e) Obten el nombre de la consola con menor número de videojuegos.

Genera un reporte de tus experimentos al llevar a cabo las inserciones en las tablas y la ejecución de cada consulta. Incluye una gráfica comparativa y las conclusiones con respecto al rendimiento de cada sentencia en función del tiempo[5] necesario que se tardo en ejecutar para cada nivel RAID. [6] ¿Qué nivel de RAID fue el mejor en términos generales? ¿Por qué consideras esto?

EXTRA

Genera una base de datos que contenga imágenes dentro de alguna o alguna de sus tablas. Repite los experimentos y analiza los nuevos resultados. ¿Existe alguna relación con el tamaño de los datos almacenados que afecta el rendimiento de algún nivel RAID?

[5] Para tomar los tiempos de una consulta, PostgreSQL cuenta con el comando '\timing'.

[6] No olvides incluir en el reporte la configuración del equipo en donde ejecutaste las pruebas, tanto de hardware como de software.

Sistemas de archivos

Semana:	Quinta semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos de SQL. Conceptos de Sistemas de Archivos. Manejo de archivos (E/S). Conocimientos de Sistemas Operativos.
Herramientas:	Sistema Operativo Linux - Distribución Fedora Core 3. PostgreSQL v8.0.2
Objetivo:	Aprender la creación y utilización de los sistemas de archivos.

La práctica número trece posee una importancia determinante, ya que se encamina a la comprensión y manejo del sistema de archivos del sistema operativo. Siendo en nuestro caso, el que se encarga de la administración del espacio en disco utilizado por el SMBD. Es importante que los alumnos ya cuenten con bases sólidas en relación a los sistemas operativos.

PRÁCTICA 13

Sistemas de archivos

13.1. Meta

Que el alumno comprenda la importancia que tiene el escoger un sistema de archivos adecuado para las distintas bases de datos operacionales que pueda utilizar.

13.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar la importancia de los sistemas de archivos.
- Preparar un sistema operativo para soportar diferentes sistemas de archivos.
- Montar diversos sistemas de archivos.
- Escoger un sistema de archivos adecuado a las necesidades de las bases de datos operacionales que tenga.

13.3. Desarrollo

Una necesidad evidente de casi cualquier aplicación de computadora, como es una base de datos, es la del almacenamiento y posterior recuperación de la información. Tal información puede tener un tamaño relativamente pequeño o un tamaño desmesurado.

Además, tal información se debe poder almacenar por períodos de tiempo muy diversos: desde unos pocos segundos hasta unos meses, o incluso para siempre[1] . En algunas ocasiones, también se necesita que una misma información sea compartida por varios usuarios de la base de datos simultáneamente.

Para este propósito se podría pensar en el espacio de direcciones de la propia aplicación, pero tal solución acarrearía una serie de problemas, principalmente los siguientes:

- El espacio de direcciones de un programa posee un tamaño limitado
- Cuando el programa termina, la información almacenada en su espacio de direcciones se pierde.

Estas limitaciones se contraponen a las tres condiciones básicas que debe cumplir todo sistema que se ocupe del almacenamiento de información a largo plazo:

1. La cantidad de información que debe ser posible almacenar ha de ser potencialmente grande.
2. La información debe sobrevivir a la conclusión de cualquier proceso que la genere y/o manipule.
3. Debe ser posible que varios procesos puedan acceder de forma concurrente a la información almacenada.

La solución a los problemas enunciados anteriormente es común y reside en el empleo de otros medios distintos del espacio de direcciones: los medios de almacenamiento masivo.

El uso de estos medios de almacenamiento introduce un nuevo problema. El manejo de estas memorias secundarias es muy diferente del manejo que se hace de la memoria principal. Y no sólo eso: entre diferentes medios de almacenamiento masivo, el acceso y manipulación de los datos en ellos almacenados puede ser muy distinto. A nivel interno, no es lo mismo acceder a la información almacenada en un disco duro que a la almacenada en una cinta o en un disco compacto óptico (*CD-ROM*).

El principal medio de almacenamiento que se emplea en el caso de las bases de datos operacionales, es el disco duro, por ello durante el desarrollo de los SMBD, se ha presentado una mejora constante en la forma en que se almacena la información en los discos duros, cada una de estas técnicas presenta características que permiten llevar a cabo un mejor aprovechamiento de todos los recursos del sistema y con esto hacer eficiente la tarea de interactuar con el SMBD. La forma lógica, y en ocasiones física, en la cual se procede a mantener la información de la base de datos es conocida como el sistema de organización de archivos.

Es importante mencionar que en algunos casos, los SMBD poseen la característica de implementar su sistema de archivos independiente del sistema de archivos en el

[1] Aunque 'para siempre' es un termino muy relativo.

cual se encuentra el sistema operativo anfitrión. Estos sistemas, generan particiones en disco duro conocidas como particiones "RAW" o crudas[2] y se administran de manera independiente.

Para la práctica, al utilizar PostgreSQL, dependemos de los sistemas de archivos soportados por el sistema operativo Linux.

Las principales características que se analizan en estos sistemas corresponden a las operaciones básicas que se llevan acabo, como puede ser el manejo de gran cantidad de datos, acceso "rápido"[3] para la obtención de información, actualizaciones convenientes y economía en el espacio por almacenar, aunque recientemente este último rubro ha dejado de ser un factor ha considerar debido al abaratamiento de los discos duros. Otras características que se toman en cuenta son la capacidad de representar estructuras del mundo real, la protección de privacidad y el mantenimiento de la integridad. Como en todos los aspectos de cómputo, estos principios tienden a contraponerse y por eso al presentarse los diferentes sistemas de organización de archivos, no es posible hacer la generalización y dar a uno como la solución a todos los problemas. Se deben considerar los beneficios de cada uno de ellos, y en base a ello decidir cual es la mejor opción para el problema que se presenta.

El número de sistemas de organización de archivos que existen numeroso y diverso, por ello solamente se optara por analizar un pequeño subconjunto de estos y que se encuentren soportados dentro del sistema operativo Linux.

Actividad 13.1

Verifica las particiones activas en la computadora y los tipos de cada una.

```
#bash=> more /etc/fstab
```

Investiga que significa cada una de las columnas que componen al archivo.

[2] Un ejemplo de esto es Oracle *10g*©.

[3] Relativamente hablando, puesto que existe una diferencia de más de 5 ordenes de magnitud, con respecto al acceso a datos en memoria.

Actividad 13.2

Verifica las particiones del sistema operativo.

```
#bash=> fdisk -l
```

Ahora identifica el tamaño de cada una de ellas, utiliza:

```
#bash=> fdisk -s particion
```

donde 'partición' es alguna de las listadas anteriormente. Puedes verificar la documentación de `fdisk` en `www.rt.com/man/fdisk.8.html`.

Actividad 13.3

Crea un diagrama de la disposición del disco duro e identifica el espacio necesario para crear 5 particiones independientes del sistema operativo.
Ahora, con el comando *fdisk* crea estas particiones:

```
#bash=> fdisk {dispositivo}
```

donde 'dispositivo' es el dispositivo de almacenamiento primario, por ejemplo, el identificado con `/dev/hda`, que generalmente es el primer disco IDE

Verifica que TODAS estas nuevas particiones sean del tipo 'Ext3' y reinicia el sistema.

13.3.1. Sistemas de archivo sin Journaling

Ext2FS

Durante mucho tiempo, el sistema de archivos estándar en Linux fue el Ext2. Éste fue diseñado por Wayne Davidson con la colaboración de Stephen Tweedie y Theodore Tsó. Es una mejora al sistema anterior, Ext, diseñado por Rémy Card. El Ext2 está basado en '*i-nodos*' (asignación indexada). Cada i-nodo mantiene la *meta-información*[4] del archivo y los apuntadores a los bloques con los datos "reales".

En el sistema de archivos ext2, el i-nodo es el bloque de construcción básico; cada archivo y directorio del sistema de archivos es descrito por un y sólo un i-nodo. Los

[4] Información que describe las características propias del archivo.

i-nodos ext2 para cada grupo de bloque se almacenan juntos en la tabla de i-nodos con un mapa de bits que permite al sistema rastrear la pista de i-nodos reservados y libres.

La tabla de i-nodos se descompone en varias partes: cada parte está contenida en un grupo de bloques. Esto permite utilizar estrategias de asignación particulares: cuando un bloque debe asignarse, el núcleo intenta asignarlo en el mismo grupo que su i-nodo, a fin de minimizar el desplazamiento de las cabezas de E/L en la lectura del archivo.

Para mejorar el rendimiento de las operaciones de E/S, los datos del disco son temporalmente almacenados en la memoria RAM a través del '*page-cache*' y '*buffer-cache*'[5] . Los problemas surgen si hay un corto de suministro eléctrico antes que los datos modificados en la memoria ('*dirty buffers*') sean grabados nuevamente al disco, se generará una inconsistencia en el estado global del sistema de archivos. Por ejemplo, un nuevo archivo que todavía no fue "creado" en el disco u otros que hayan sido borrados pero sus i-nodos y bloques de datos todavía permanecen como "activos" en el disco.

El fsck ('*file system check*') fue la herramienta que resolvía dichas inconsistencias, pero el fsck tiene que analizar la partición completa y verificar las interdependencias entre inodos, bloques de datos y contenidos de directorios. Con la ampliación en la capacidad de los discos, la recuperación de la consistencia del sistema de archivo se ha convertido en una tarea que requiere mucho tiempo, por lo que crea problemas serios de disponibilidad de los servidores afectados. Esta es la razón principal de que los sistemas de archivos hayan importado de las bases de datos las técnicas de transacciones y recuperación, y así hayan aparecido los sistemas de archivos con "*journaling*".

Actividad 13.4

Para activar el soporte dentro del sistema operativo para los nuevos sistemas de archivos, hay que compilar el kernel para generar uno nuevo que tenga activadas estas opciones.

Para ello, descarga de `www.kernel.org` el código fuente del kernel más reciente, de preferencia uno de versión superior o igual a la '2.6.11'.
Sigue las instrucciones para su compilación y no olvides activar en el menú de configuración la opción de soporte para sistemas de archivos Ext2FS, Ext3FS, XFS, JFS y ResiserFS, todos compilados como módulos.

Reinicia el sistema con tu nuevo kernel.

[5] Son estructuras de datos que se encuentran en el modulo de administración de memoria del kernel de linux[21].

13.3.2. Sistemas de archivo con Journaling

Un sistema con journaling es un sistema de archivos tolerante a fallos en el cual la integridad de los datos está asegurada porque las modificaciones de la meta-información de los archivos son primero grabadas en un registro cronológico (log o journal) antes que los bloques originales sean modificados. En el caso de un fallo del sistema, un sistema con journaling "integral" asegura que la consistencia del sistema de archivos es recuperada. El método más común consiste en grabar previamente cualquier modificación de la meta-información en un área especial del disco, el sistema realmente grabará los datos una vez que la actualización de los registros haya sido completada. A la hora de recuperar la consistencia luego de un fallo, el módulo de recuperación analizará el registro y sólo repetirá las operaciones incompletas en aquellos archivos inconsistentes, es decir que la operación registrada no se haya llevado a cabo finalmente. Los primeros sistemas de archivos con journaling fueron creados a mediados de los ochenta e incluyen a Veritas (VxFS), Tolerant y JFS de IBM©. La demanda de sistemas de archivos que soporten terabytes de datos, miles de archivos por directorios y compatibilidad con arquitecturas de 64 bits ha hecho que en los últimos años haya crecido el interés de la disponibilidad de sistemas con journaling en Linux.

El kernel actual (2.6.11) posee soporte para los siguientes cuatro sistemas de archivos con esta característica:

- Ext3 desarrollado por S. Tweedie[6] (`http://e2fsprogs.sourceforge.net/`),
- XFS de Silicon Graphics© (`http//oss.sgi.com/projects/xfs/`),
- JFS de IBM© (`http://jfs.sourceforge.net/`) y
- ReiserFS de Namesys© (`http://www.namesys.com`).

Mientras que ReiserFS es un sistema totalmente nuevo escrito desde cero, XFS, JFS y Ext3 son derivados de sistemas ya existentes. XFS está basado, y comparte parcialmente el mismo código, en el sistema desarrollado por Silicon Graphics© para sus estaciones de trabajo y servidores. JFS fue desarrollado por IBM© para su sistema OS/2 Warp, que a su vez es derivado del sistema JFS de AIX. Ext3 es una extensión a Ext2 y agrega dos módulos independientes: un módulo de transacciones y un módulo de registro.

Durante la práctica, se realizara una comparación en rendimiento de cada uno de estos sistemas de archivos, al utilizar el SMBD PostgreSQL.

Árboles B

La técnica básica para mejorar el rendimiento comparado a sistemas de archivos tradicionales de Unix es evitar el uso de listas enlazadas o mapas de bits ya que sufren de

[6] co-desarrollador del Ext2

problemas inherentes de escalabilidad. La mayoría de los nuevos sistemas usan árboles B o variaciones de ellos (árboles B^+). Debido a la estructura de los árboles B, estos son más robustos en rendimiento pero al mismo tiempo los algoritmos de gestión y balanceo son más complejos.

Actividad 13.5

Basándote en tu diagrama de la disposición del disco duro. Identifica el espacio reservado para crear la partición del sistema de archivos XFS.
Descarga el paquete RPM de herramientas de XFS, xfsprogs, e instálalas :

```
#bash=> rpm -Uhv xfsprogs-X-X-X.rpm
```

Desmonta la partición ext3 donde colocarás el sistema de archivos XFS.

```
#bash-> umount /dev/hdaX
```

Ahora crea la partición con este sistema de archivos:

```
#bash=> ./mkfs.xfs /dev/hdaX
```

y crea el directorio que corresponderá a esta partición:

```
#bash=> mkdir /XFS
```

Solamente resta 'montarlo':

```
#bash=> mount -t xfs /dev/hdaX /XFS
```

Actividad 13.6

Basándote en tu diagrama de la disposición del disco duro. Identifica el espacio reservado para crear la partición del sistema de archivos JFS.
Descarga el paquete RPM de herramientas de JFS, jfsutils, e instálalas :

```
#bash=> rpm -Uhv jfsutils-X-X-X.rpm
```

Desmonta la partición ext3 donde colocarás el sistema de archivos JFS.

```
#bash-> umount /dev/hdaY
```

Ahora crea la particion con este sistema de archivos:

```
#bash=> ./mkfs.jfs /dev/sdaY
```

Crea el directorio que corresponderá a este partición:

```
#bash=> mkdir /JFS
```

Por último, hay que 'montarlo':

```
#bash=> mount -t jfs /dev/sdaY /JFS
```

13.4. Ejercicios

1. Realiza las instalaciones de las particiones exclusivas con los sistemas de archivos:
   a) JFS.
   b) XFS.
   c) ReiserFS.
   d) Ext3FS.
2. Edita el archivo '`/etc/fstab`' para que las particiones de los sistemas de archivos se monten al iniciar el sistema.
3. Crea 5 directorios de datos para la base de datos, uno dentro de cada partición de los sistemas de archivos recién creados. Genera un reporte sobre las condiciones de cada una de las instalaciones de los sistemas de archivo.
4. Carga una nueva base de datos en todos los sistemas de archivos utilizando sentencias `INSERT INTO`.
   a) Para 10000 tuplas en la tabla.
   b) Para 1000000 tuplas en cada tabla.
5. Ejecuta las siguientes consultas:
   a) Obten el nombre de todos los cliente cuyo apellido empieza con las letras 'Ga'.
   b) Obten el nombre de todos los clientes que hay realizado una compra en el mes 1 del año 2005 .
   c) Obten el nombre y apellido de todos los clientes que han comprado algún videojuego de la consola 'PS2'.
   d) Obten para cada consola, el nombre de la consola y el total de videojuegos disponibles para ella. Ordena el resultado de forma descendente con respecto al total de videojuegos disponibles.
   e) Obten el nombre de la consola con menor número de videojuegos entre el segundo mes de 1999 y el décimo mes del 2005.

   Genera un reporte de tus experimentos al llevar a cabo las inserciones en las tablas y la ejecución de cada consulta. Incluye una gráfica comparativa y las conclusiones con respecto al rendimiento de cada sentencia en función del tiempo necesario que se tardo en ejecutar para cada sistema de archivo[7] . ¿Qué sistema de archivos fue el mejor en términos generales? ¿Por qué consideras esto?

[7] No olvides incluir en el reporte la configuración del equipo, tanto de hardware como de software, en donde ejecutaste las pruebas.

Manejo de Índices

Semana:	Séptima semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos de SQL. Conceptos de Índices.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Aprender la creación y uso de los índices y manejar opciones de configuración de PostgreSQL.

La práctica número catorce se enfoca al análisis de las consultas que recibe un SMBD utilizando para ello los índices. Es importante que los alumnos ya cuenten con bases solidas en relación concepto de índices, además de comprender los distintos algoritmos utilizados durante la operación de reunión de tablas.

PRÁCTICA 14

Manejo de índices

14.1. Meta

Que el alumno comprenda la importancia que tiene la creación y el manejo adecuado de los índices, así como las opciones de configuración que presenta PostgreSQL.

14.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar los distintos tipos de índices.
- Identificar cuando crear índices.
- Elegir los tipos de índices a utilizar según las consultas más frecuentes.
- Identificar los algoritmos utilizados durante la operación de reunión de tablas.
- Forzar al planificador de consultas a elegir distintos algoritmos.
- Elegir el algoritmo de reunión más adecuado según las distintas consultas que se presenten.

14.3. Desarrollo

14.3.1. Índices

Un índice para un archivo en el sistema, funciona de manera muy semejante a un catalogo para libros dentro de una biblioteca. Si se busca un libro según el nombre de su autor entonces se realiza una consulta en el catalogo de autores (o en su defecto, dentro de la sección "búsqueda por autores" del programa en cuestión) y es ubicado rápidamente el lugar donde se encuentra el libro. Por símil, los índices son las herramientas que se utilizan para esto dentro de las bases de datos: Localizar registros rápidamente.

Los índices se clasifican tomando distintos parámetros, por ejemplo, existe una clasificación basada en el tipo de valores que contienen:

- Índices ordenados.- Son índices que se basan en valores previamente ordenados.
- Índices hash.- Estos índices se basan en valores que son distribuidos uniformemente dentro de un conjunto de entradas. Estas entradas son determinadas por una función conocida como 'función hash'.

Cada estructura de un índice es asociada con una llave en particular, así al 'pedir' una llave, se obtiene la ubicación del registro al que corresponde. Un archivo pude contener múltiples índices, los cuales pueden basarse sobre cualquier atributo pero si se crea sobre el atributo por el cual se encuentra ordenado físicamente entonces el índice se conoce como *índice primario*. Por otro lado, los índices en los cuales las llaves de búsqueda difieren del orden secuencial del archivo son llamados *índices secundarios*. Cabe señalar que un índice primario en una base de datos no siempre es sobre el atributo que funciona como llave primaria.

Dentro de los índices ordenados se genera una subdivisión basada en como se agrupan los valores relativos a las llaves de búsqueda. Existen:

- Índices densos.- En este caso una entrada de registro aparece por cada llave de búsqueda. El registro del índice contiene el valor de búsqueda y un apuntador a la primera aparición del valor de búsqueda en el archivo principal.
- Índices dispersos.- Un registro de índice es creado sólo para algunos valores, y cada uno contiene la misma información que uno denso. Para encontrar un registro se localiza una entrada en los índices con el mayor valor de búsqueda que sea menor o igual al valor de búsqueda suministrado, posteriormente se recorren los apuntadores hasta encontrar el valor deseado.

Generalmente es más rápido localizar un registro si se genera un índice denso en lugar de uno disperso, pero un disperso involucra menos espacio y sobretodo menos mantenimiento.

Índices de múltiples niveles

Existe la posibilidad de que aún al cargar un índice en la memoria, éste índice crezca demasiado y sea necesario hacer múltiples lecturas a disco, lo cual implicaría mayor sobrecarga al sistema. En estos casos es preferible realizar un nuevo índice que contenga como entradas los valores del índice anterior, en otras palabras, se crea un índice de índices.

14.3.2. Índices en PostgreSQL

PostgreSQL implementa cuatro tipos diferentes de índices, cada uno pensado idealmente para cubrir ciertos tipo de consultas. Los cuatro tipos de índices que implementa PostgreSQL son:

- Árboles B.
- Árboles R.
- Hash.
- GiST.

PostgreSQL utiliza los árboles B como el tipo de índice por omisión, esto dado que permiten manejar consultas que involucran igualdades y rangos dentro de dominios que permiten cierto orden.

Los índices tipo árboles R, son ideales para consultas sobre datos espaciales y los tipo hash para consultas que involucran únicamente igualdades. Por su parte, los índices GiST no son propiamente índices, sino que son estructuras en las cuales se implementan diversas estrategias de índices[1] . La sintaxis para la creación de estos índices se presenta en la figura 14.1 Es preciso considerar que con la creación de un índice no basta para

```
CREATE INDEX nombre ON tabla [ USING tipo_índice ]
    ( {columnas | ( expresión )})
```

Figura 14.1: Sintaxis básica para la creación de un índice

que se consiga un desempeño óptimo dado que en un inicio el planificador de consultas

[1] Información sobre los tipos de datos que manejan los índices GiST, se encuentra en `http://www.postgresql.org/docs/8.0/static.html`

no cuenta con los valores estadísticos más actuales del catalogo del sistema. Un ejemplo de creación de índice es el siguiente:

```
nombre_db=# CREATE INDEX indice_Uno ON cliente USING HASH (atributo);
  (CREATE INDEX)
```

Así como existe la instrucción para crear un índice, también existe la instrucción DROP respectiva para eliminarlo. Su sintaxis se presenta en la figura 14.2.

```
DROP nombre_indice;
```

Figura 14.2: Sintaxis de la cláusula DROP para índices.

Actividad 14.1

Cuando se crea una tabla en PostgreSQL que incluye una llave primaria se crea implícitamente un índice sobre esa columna.
¿Por qué crees que sucede eso?

Revisa la estructura de cada tabla para obtener la información de su estructura.
Crea un índice de tipo árbol B sobre la columna '*nombre_cliente*' de la relación *cliente*.
Crea un índice de tipo árbol B sobre la columna '*codigo_postal*' de la relación *cliente*.

Revisa la estructura de las tablas para obtener información sobre los índices recién creados.

ANALYZE

Como se menciono, los índices no obtienen las últimas estadísticas almacenadas en los catálogos del sistema, para ello siempre es conveniente que una vez creado el índice se realice la tarea de mantenimiento conocida en PostgreSQL como ANALYZE.

ANALYZE genera estadísticas sobre los contenidos de las tablas de la base de datos y almacena los resultados en el catalogo del sistema. Éstas estadísticas son utilizadas

por el planificador de consultas para la determinación de planes de ejecución eficientes correspondientes a las consultas que recibe el SMBD. Su sintaxis se presenta en la figura 14.3.

```
ANALYZE [ VERBOSE ] [ tabla [ (columna [, ...] ) ] ]
```

Figura 14.3: Sintaxis de la cláusula `ANALYZE`.

Por eso, es importante que una vez creado un índice, se ejecute esta instrucción sobre el mismo. La opción `VERBOSE` presentara una descripción más detallada de las operaciones que lleva a cabo `ANALYZE` durante su ejecución.

De igual forma, es conveniente realizar la operación de `ANALYZE` una vez que se elimina un índice, ya que evita al planificador de consultas utilizar información del catalogo no actualizada.

EXPLAIN

PostgreSQL utiliza los índices para reducir los tiempos de ejecución de las consultas que recibe. Pero internamente PostgreSQL implementa distintos métodos para obtener los resultados. Por ejemplo, cuando recibe una consulta donde se involucra una opción de igualdad, en caso de que no exista un índice, utilizará una búsqueda secuencial para regresar las tuplas que cumplen con la condición. O bien, si existe el índice en cuestión, el sistema lo utiliza. El planificador de consultas de PostgreSQL, permite manipular múltiples opciones en su configuración para moldear su comportamiento dependiendo de las consultas que reciba nuestra base de datos.

Actividad 14.2

Elimina los índices creados anteriormente y realiza a operación de `ANALYZE` sobre las tablas donde originalmente se encontraban los índices.
Realiza las siguientes consultas, tomando el tiempo de ejecución de cada una de ellas:

1. Obten el nombre de los videojuegos tales que su precio sea igual a 490.90.

2. Obten el nombre de los cliente y su ciudad, tales que su código postal sea igual a 57100.

Ahora crea dos índices tipo árbol B, uno sobre el atributo '*código postal*' de la tabla *cliente* y otro sobre el atributo '*precio*' de la tabla *videojuego*.
Nuevamente realiza las consultas tomando el tiempo de ejecución.
¿Se redujo el tiempo?

Realiza las operaciones de `ANALYZE` respectivas sobre cada uno de los índices y vuelve a ejecutar las consultas.
¿Qué mejora se presento?

¿Por qué sucedió esto?

Para conocer el trabajo interno que realiza PostgreSQL con cada consulta recibida, se cuenta con la cláusula `EXPLAIN`, que presenta el plan de ejecución que el sistema utiliza para la consulta en cuestión. Su sintaxis se presenta en la figura 14.4. Al utilizar

```
EXPLAIN consulta;
```

Figura 14.4: Sintaxis de la cláusula `EXPLAIN`.

`EXPLAIN` sobre una consulta, se observa el plan de ejecución. El cual se encuentra compuesto por diferentes elementos, por ejemplo, número de tuplas leídas de disco y el tamaño promedio de cada tupla pero sobretodo y siendo de particular atención para nuestro interés, el algoritmo utilizado en la consulta. Un ejemplo de la utilización de `EXPLAIN` es el siguiente:

```
nombre_db=# EXPLAIN SELECT nombre_cliente
            FROM cliente NATURAL JOIN compra
            WHERE compra.cantidad = 1000;
                                    QUERY PLAN
-------------------------------------------------------------------------
 Nested Loop (cost=0.00..29032.26 rows=7331 width=10)
    ->Seq Scan on compra (cost=0.00..3054.56 rows=7331 width=8)
       Filter: (cantidad = 1000)
    ->Index Scan using cliente_pkey on cliente (cost=0.00..3.53 rows=1
 width=18)
       Index Cond: (cliente.id_cliente = "outer".id_cliente)
 (5 filas)
```

Obviamente para distintos tipos de consultas se generan distintos tipos planes de ejecución, por ejemplo para las búsquedas sobre una sola tabla, se observan las búsquedas secuenciales o búsquedas que utilizan un índice si es el caso.

Actividad 14.3

Repite las consultas de la actividad 14.2 utilizando y sin utilizar los índices

Observa los distintos algoritmos que utiliza PostgreSQL en cada consulta.
¿Qué sucede cuando aplicas `EXPLAIN` sobre un `ANALYZE` previo?

Opciones de configuración.

Cada uno de estos algoritmos utilizados para la obtención del resultado, puede o no ser benéfico, por ejemplo, en una tabla donde rara vez se ejecuta una consulta pero se realizan bastantes operaciones de inserción o actualización, puede ser mejor no crear más índices que los propios de las llaves primarias, dado que casi siempre se debería realizar una actualización del catalogo antes de la consulta (esto para utilizar los valores más actuales del mismo).

Para modificar el funcionamiento de PostgreSQL, no es necesario reiniciar el servidor porque los algoritmos que se implementan en el sistema pueden ser activados y desactivados durante la ejecución del mismo. Los principales parámetros que son posibles de modificar de este modo se presentan en la tabla 14.1[2] .

[2] Para mayor información sobre el total de los parámetros que se pueden modificar en tiempo de ejecución ver `http://www.postgresql.org/docs/8.0/static/runtime-config.html`

Para modificar el parámetro en cuestión, hay que ejecutar el comando **SET**. Por ejemplo:

```
nombre_db=# SET enable_seqscan TO FALSE;
 (SET)
```

Lo que se logra con las modificaciones a estos parámetros es forzar al optimizador de consultas, a utilizar o no, los distintos algoritmos implementados internamente. Hay que remarcar que el modificar estos parámetros permanentemente no es una buena idea, puesto que los cambios son aplicados de manera global a todas las consultas recibidas por el SMBD. Así que puede presentarse el caso en el cual una modificación sea beneficiosa para una consulta, pero posiblemente esto será en detrimento de múltiples consultas[3] .

Actividad 14.4

Ejecuta la siguiente consulta, tomando su plan y tiempo de ejecución:

```
nombre_db=# SELECT nombre_cliente
            FROM cliente NATURAL JOIN compra
            WHERE compra.cantidad = 1000;
```

Observa que PostgreSQL utiliza el algoritmo de 'nested loop' para ejecutar la reunión (`JOIN`) interna. Deshabilita este tipo de reunión:

```
nombre_db=# SET enable_nestloop TO FALSE;
SET
```

Repite la consulta. ¿Qué sucede? ¿Sirve el cambio obtenido? ¿Por qué?

[3] Nuevamente es recomendable ejecutar nuevamente **ANALYZE** luego de realizar algún cambio en estos parámetros.

Parámetro.	Tipo.	Define.
enable_hashjoin	(boolean)	Habilita o no el uso de 'hash-joins' por parte del planificador de consultas.
enable_indexscan	(boolean)	Habilita o no el uso de búsquedas secuenciales en índices por parte del planificador de consultas.
enable_mergejoin	(boolean)	Habilita o no el uso de 'merge-joins' por parte del planificador de consultas.
enable_nestloop	(boolean)	Habilita o no el uso de 'nested-joins' por parte del planificador de consultas. No es posible inhabilitar los 'nested-joins' totalmente pero al modificar este parámetro, el planificador preferirá utilizar otros métodos para los joins si están disponibles.
enable_seqscan	(boolean)	Habilita o no el uso de búsquedas secuenciales por parte del planificador de consultas. No es posible deshabilitar totalmente la opción, pero el planificador preferira utilizar otros métodos disponibles.
enable_sort	(boolean)	Habilita o no el uso de ordenamientos explícitos por parte del planificador de consultas. No es posible deshabilitar totalmente este parámetro, pero forza al planificador a no utilizarlo si exite otro método disponible.

Tabla 14.1: Parámetros en tiempo de ejecución para el optimizador de consultas.

14.4. Ejercicios

Dentro de la base de datos, elimina todos los índices que no sean sobre alguna llave primaria, contesta la preguntas y realiza lo siguiente[4] .

1. Ejecuta las consultas que a continuación se listan, tomando en cuenta las consideraciones para cada una de ellas:

   a) Obten el número de clientes que tienen por nombre 'René'.
   - Realiza la consulta sin agregar índice alguno.
   - Realiza la consulta con un índice tipo árbol B sobre '*nombre_cliente*'.
   - Realiza la consulta con un índice tipo hash sobre '*nombre_cliente*'.

   b) Obten el nombre de los videojuegos tales que su precio sea estrictamente mayor que 490.90
   - Elimina los índices creados anteriormente y realiza la consulta.
   - Realiza la consulta con un índice tipo árbol B sobre '*precio_videojuego*'.
   - Realiza la consulta con un índice tipo hash sobre '*precio_videojuego*'.

   c) Obten el nombre de los clientes que han comprado videojuegos cuyo precio es igual a 550.50 y viven en 'Yibuti'.
   - Elimina los índices creados anteriormente y realiza la consulta.
   - Realiza la consulta con índices tipo árbol B sobre '*precio_videojuego*' y sobre '*ciudad_cliente*'.
   - Realiza la consulta con índices tipo hash sobre '*precio_videojuego*' y sobre '*ciudad_cliente*'.

   d) Obten el nombre y apellidos de los clientes que han comprado videojuegos cuya consola es la 'PC' y son del tipo 'RPG'.
   - Elimina los índices creados anteriormente y realiza la consulta.
   - Realiza la consulta con índices tipo árbol B sobre '*consola_videojuego*' y sobre '*tipo_videojuego*'.
   - Realiza la consulta con un índices de múltiples columnas y de tipo árbol B sobre '*consola_videojuego*' y sobre '*tipo_videojuego*' simultáneamente.
   - Realiza la consulta con índices tipo hash sobre '*consola_videojuego*' y sobre '*tipo_videojuego*'.

   Genera un reporte de tus experimentos e incluye conclusiones con respecto al rendimiento de cada consulta en relación al tiempo que tomaron en ejecutarse cada una de ellas al utilizar índices y al no utilizarlos[5] .

[4] No olvides ejecutar `ANALYZE` después de la creación y eliminación de un índice.

[5] No olvides incluir en el reporte la configuración del equipo, tanto de hardware como de software, en donde ejecutaste las pruebas.

2. Ejecuta las siguientes consultas y modifica los parámetros en tiempo de ejecución que se piden[6] .

   a) `SELECT id_cliente, nombre_videojuego FROM compra, videojuego WHERE videojuego.id_videojuego = 1963657;`.

      - Realiza la consulta sin modificar parámetro alguno.
      - Realiza la consulta modificando 'enable_nestloop' a OFF.
      - Realiza la consulta modificando 'enable_hashjoin' a OFF y 'enable_nestloop' a ON.

   b) `SELECT id_cliente, nombre_videojuego FROM compra NATURAL JOIN videojuego WHERE videojuego.id_videojuego = 1963657;`

      - Realiza la consulta sin modificar parámetro alguno.
      - Realiza la consulta modificando 'enable_nestloop' a OFF.
      - Realiza la consulta modificando 'enable_hashjoin' a OFF y 'enable_nestloop' a ON.

   A partir de las consultas anteriores: ¿Qué puedes concluir sobre la equivalencia de operaciones dentro de SQL?

   Genera un reporte de tus experimentos e incluye conclusiones con respecto al rendimiento de cada consulta en relación al tiempo que tomaron en ejecutarse cada una de ellas al utilizar determinado algoritmo para la operación `JOIN`.

3. El parámetro 'enable_sort', ¿en que condiciones será bueno desactivarlo? ¿Por qué?

4. Considera la siguiente consulta

```
SELECT * FROM videojuegos, compra
WHERE videojuego.id_videojuego = compra.id_videojuego
AND cantidad < 25;
```

   a) ¿Qué tipo de algoritmo utiliza el planificador de consultas? Escribe el plan y su costo estimado.

   b) Inhabilita este último 'join' y recalcula el costo para el mejor plan. Describe el mejor plan producido por el optimizador en este caso y explica las diferencias.

   c) Inhabilita ahora el algoritmo utilizado en el punto anterior. ¿Qué 'join' utiliza ahora el optimizador? Describe el mejor plan obtenido.

[6] Recuerda que puedes ver el plan de ejecución al aplicar `EXPLAIN` sobre `ANALYZE` de la consulta.

5. Considera la siguiente consulta

```
SELECT nombre_cliente, ap_mat_cliente
FROM cliente CROSS JOIN compra CROSS JOIN videojuegos
WHERE nombre_cliente LIKE '%e'
AND videojuegos.consola_videojuegos = 'PC';
```

   Utiliza **EXPLAIN** junto con **ANALYZE** para determinar el plan de ejecución y su tiempo estimado, modifica el orden de las tablas y genera los demás planes de ejecución.

6. PostgreSQL implementa los índices sobre múltiples columnas
   ¿Qué tipo de consultas se benefician con esto?

Optimización de consultas

Semana:	Novena semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Consultas anidadas en SQL. Manejo de índices y vistas. Conceptos de equivalencias entre operadores.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Aprender el diseño y la creación de consultas eficientes.

La práctica número quince se enfoca a la optimización de las consultas que recibe un SMBD, utilizando para ello todos los recursos disponibles, tanto herramientas propias del sistema como el diseño lógico de las consultas.

Es importante que los alumnos ya cuenten con bases sólidas en creación de consultas complejas, manejo de índices, vistas y equivalencias entre operaciones.

PRÁCTICA 15

Optimización de consultas

15.1. Meta

Que el alumno comprenda la importancia que tiene el proceso de diseño de consultas y el impacto que tiene esto en el desempeño del sistema.

15.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar las equivalencias entre las distintas cláusulas de SQL.
- Identificar cuando utilizar estás equivalencias.
- Desarrollar un proceso concreto para la creación de consultas eficientes.

15.3. Desarrollo

En general es posible obtener el resultado de una consulta de muy diversas formas, esto se debe en parte a que existen operaciones equivalentes dentro de SQL y en parte a que el proceso de diseño de una consulta obedece en gran parte a la intuición del programador. La desventaja de lo anterior, es que posiblemente no se genere una consulta que tenga el mejor desempeño posible aún cuando el planificador de consultas del SMBD haga la optimización interna. Por ello es importante conocer ciertos elementos,

por ejemplo técnicas y equivalencias entre operaciones, que permitan crear consultas que se ejecuten de la manera más eficiente[1] .

El catalogo del sistema

Aún cuando parece redundante, es importante ejecutar tareas de mantenimiento de manera constante, sobre todo en las tablas donde se mantienen índices que están siendo utilizados continuamente. En PostgreSQL, es recomendable ejecutar la cláusula `ANALYZE` sobre los índices para conservar actualizados los valores estadísticos que utiliza el SMBD al generar los planes de ejecución.

Eliminación de atributos

La primera y más simple de las mejoras que hay que tomar en cuenta es la eliminación de atributos. Considerando la siguiente consulta: "Obtener los nombres de los clientes que han realizado una compra", la solución más simple es la siguiente.

```
SELECT nombre_cliente
FROM cliente, compra
WHERE cliente.id_cliente = compra.id_cliente;
```

La sentencia SQL anterior es correcta, mas no es la más eficiente, sobretodo si toda esta información se encuentra en una base de datos distribuida. Internamente el sistema acarrea todos los atributos que pertenecen a las dos tablas, y con ello realiza la operación correspondiente al producto cartesiano. La evidencia se hace mayor cuando se consideran atributos que son de tamaño muy extenso, por ejemplo imágenes o archivos binarios. Una sentencia equivalente que obtiene únicamente los atributos de interés tiene la siguiente estructura:

```
SELECT nombre_cliente FROM
(SELECT id_cliente, nombre_cliente FROM cliente) AS clienteOp
NATURAL JOIN
(SELECT id_cliente FROM compra) AS compraOp ;
```

El agrupamiento

El agrupamiento es una operación muy importante ya que permite encontrar respuesta a un amplio conjunto de consultas distintas. Sin embargo es importante utilizarlo adecuadamente.

[1] Hay que recordar que en un sistema de cómputo el rendimiento siempre se encuentra determinado por el medio más lento que participa el proceso, así que, cuando se realice una consulta externa el tiempo de respuesta estará determinado por la velocidad de transmisión de la red.

Considerando la consulta "Obtener el identificador de cada cliente así como el total de sus compras"[2] , generalmente se opta por realizar una operación de agrupamiento sobre el resultado obtenido de la reunión de tablas, esto es, realizar la reunión de las tablas '*cliente*' y '*compra*' y posteriormente aplicar la operación de agrupamiento a la tabla resultante, la sentencia en SQL queda de la siguiente forma:

```
SELECT id_cliente, COUNT(id_cliente) AS cantidad
FROM cliente NATURAL JOIN compra
GROUP BY id_cliente
```

El resultado es completamente correcto, pero nuevamente estamos acarreando información redundante dentro de nuestra consulta. Un ejemplo gráfico de la ejecución de esta sentencia se muestra en la figura 15.1.

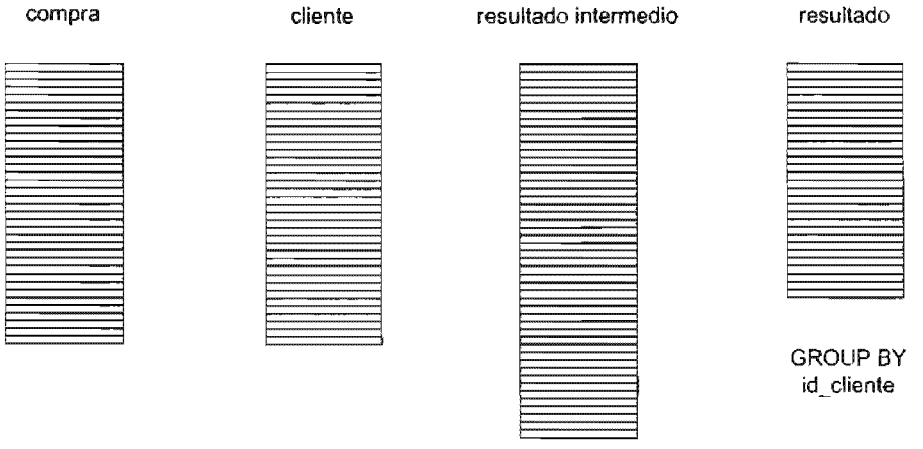

Figura 15.1: Ejecución del agrupamiento después de la operación de reunión.

La optimización de la consulta se sigue de las siguientes observaciones. Primeramente hay que observar que al consultar la tabla de '*compra*' se obtienen los identificadores de los clientes que han hecho una compra, por lo tanto es en esta relación donde es posible agrupar el resultado para que la relación que participe en la reunión con la tabla '*cliente*' sea de menor tamaño. De esta forma, ocupara menos memoria y se evitan lecturas innecesarias a disco. La idea general se muestra en la figura 15.2.

Dado esto, se puede imaginar que siempre es conveniente llevar a cabo una agrupación antes de una reunión. Pero no es así, se ha visto (en [25]) que el tamaño de los grupos formados por la sentencia `GROUP BY` influye en el rendimiento de la consulta.

[2] La consulta mas general sería obtener el nombre u otros datos, pero eso requiere de una operación más y no esta vinculada con la operación de agrupamiento.

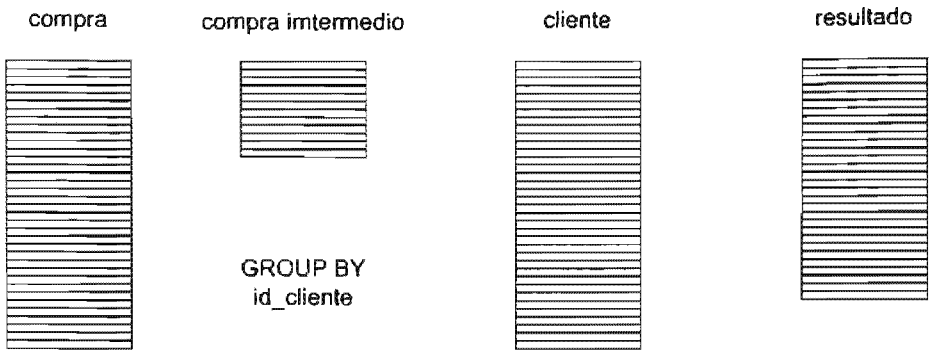

Figura 15.2: Ejecución del agrupamiento antes de la operación de reunión.

Lo cual se explica de la siguiente forma: Si el número de grupos formados durante el agrupamiento es semejante al número de elementos de la tabla original, entonces no se obtiene ganancia al reducir el tamaño de la misma sino que por el contrario, se introduce una sobrecarga de trabajo al realizar el agrupamiento y la consulta se ejecuta similarmente a la primera forma (figura 15.1).

Actividad 15.1

Ejecuta las sentencias SQL anteriores y obten su plan de ejecución utilizando la sentencia `EXPLAIN`.
Examina los distintos algoritmos utilizados por el planificador de consultas de PostgreSQL.
Inhabilita los algoritmos utilizados con la sentencia `SET` y ejecuta nuevamente las sentencias SQL.
Compara el rendimiento de las consultas.
¿Existe alguna mejora considerable?

Índices compuestos

Los índices compuestos son simplemente aquellos que están basados en múltiples atributos. PostgreSQL únicamente usa un índice por tabla cuando está procesando una consulta. Esto significa que si se tienen varias columnas que frecuentemente aparecen juntas en una cláusula `WHERE`, se presenta la oportunidad de acelerar estas consultas al crear un índice compuesto. Pero es preciso tener cuidado, ya que se puede interpretar que el contar con un índice compuesto permite acelerar arbitrariamente las consultas

que se ejecutan sobre los atributos donde se encuentra el índice. Esto es parcialmente cierto cuando se observa que si una tabla tiene un índice formado por múltiples columnas, cualquier prefijo puede ser usado por el optimizador de consultas de PostgreSQL para encontrar las tuplas. Por ejemplo, si se crea un índice compuesto sobre las columnas de la tabla *cliente[nombre_cliente, ap_pat_cliente, ap_mat_cliente]*, entonces se tienen capacidades de búsqueda con índices sobre *'nombre_cliente'*, *'nombre_cliente, ap_pat_cliente'* y *'nombre_cliente, ap_pat_cliente, ap_mat_cliente'*.

```
CREATE INDEX indice_compuesto1 ON cliente (
             nombre_cliente, ap_pat_cliente, ap_mat_cliente);
```

PostgreSQL no puede usar un índice parcial si las columnas no forman un prefijo del índice. Suponiendo que se cuenta con sentencias SELECT como estas:

1. `SELECT * FROM cliente WHERE nombre_cliente = 'Rene';`

2. `SELECT * FROM cliente WHERE nombre_cliente = 'Rene'`
   `AND ap_pat_cliente = 'Villeda';`

3. `SELECT * FROM cliente WHERE ap_pat_cliente = 'Villeda';`

Debido a la forma en que PostgreSQL construye los índices compuestos, éste puede usar el índice `'indice_compuesto1'` sólo para resolver consultas basadas en el nombre, o en el nombre y los apellidos, sin embargo, no usará el índice en una consulta que haga referencia únicamente a la columnas de los apellidos. Por lo tanto, sólo las dos primeras consultas hacen uso del índice `'indice_compuesto1'`.

La idea es que los índices compuestos pueden usarse frecuentemente para acelerar algunas consultas complejas, pero es necesario entender sus limitaciones y es recomendable ejecutar algún tipo de prueba en vez de asumir que estos índices siempre ayudan a mejorar el rendimiento de una consulta.

Actividad 15.2

Ejecuta las sentencias SQL anteriores y obten el plan de ejecución de las consultas utilizando la sentencia `EXPLAIN`.
Verifica para que consultas se utilizan búsquedas exhaustivas y para cuales se se utilizan los índices.

Reuniones explícitas

La operación de reunión de tablas, es posiblemente la más utilizada dada la forma en que se encuentran estructuradas las bases de datos, por ello es de particular importancia optimizar al máximo a las consultas en este rubro.

La operación de reunión (`JOIN`) se encarga de relacionar los valores que se encuentran en una tabla con los valores de otra tabla, la operación es binaria por lo cual siempre trabaja teniendo como entradas a 2 tablas aún cuando se tenga una expresión que involucre más tablas (cuando se utiliza una sub-consulta está se ejecuta antes para servir como fuente de datos a la operación de reunión inmediata).

Es posible controlar al planificador de consultas de modo que permita la utilización de reuniones explícitas. La importancia de este hecho radica en que el espacio de soluciones, los planes de ejecución que se generán para una consulta dada, es bastante amplio (de hecho son $n! * |arboles|$ [31]). Por ello, conviene, utilizar heurísticas en la obtención de un plan de ejecución eficiente.

Por ejemplo, en una consulta como la siguiente:

```
SELECT * FROM a, b, c WHERE a.id = b.id AND b.ref = c.id;
```

el planificador es libre de reunir las tablas dadas en cualquier orden, por ejemplo puede generar un plan de ejecución tal que reuna las tablas A y B utilizando la condición `WHERE a.id = b.id`, y luego reunir C a esta tabla resultante utilizando la otra condición `WHERE`. O bien, puede reunir B con C y luego ese resultado con A. O puede unir A con C y luego reunir el resultado con B, pero sería ineficiente dado que el producto cartesiano de A y C se tendría que materializar, dado que no hay condición aplicable a la cláusula `WHERE` para permitir la optimización de la consulta.

Es importante observar que el planificador tendría que generar todos los planes de ejecución posibles, verificar sus costos (que indudablemente son variables) y escoger el mejor. Cuando las consultas requieren de dos o tres tablas, no hay muchas opciones, pero cuando el número se incrementa, el proceso de encontrar el plan de ejecución (no de obtener el resultado) puede tomar bastante tiempo. El planificador de PostgreSQL cambiará de una búsqueda exhaustiva a una búsqueda genética probabilística [14]. La búsqueda génetica toma menos tiempo, pero no necesariamente encuentra al mejor plan de ejecución.

Para forzar al planificador a utilizar un orden estricto de reuniones (cuando se tienen más de dos tablas) se debe modificar el parámetro '*join_collapse_limit*' y asignarle el valor de 1 utilizando la cláusula `SET`.

Actividad 15.3

Cambia el parámetro '*join_collapse_limit*' a 1.
Ejecuta la siguiente consulta y obten el plan de ejecución de la misma:

```
SELECT * FROM cliente
NATURAL JOIN (compra NATURAL JOIN videojuego)
WHERE codigo_postal_cliente = 44230
AND consola_videojuego = 'PlaySation 2';
```

Ahora, ejecuta la siguiente consulta y obten el plan de ejecución.

```
SELECT * FROM (cliente NATURAL JOIN compra)
NATURAL JOIN videojuego
WHERE codigo_postal_cliente = 44230
AND consola_videojuego = 'PlaySation 2';
```

Compara el rendimiento de las consultas.
¿Existe alguna mejora considerable?

Operadores de conjuntos

Las operaciones de `NOT IN`, `IN`, `EXISTS` y `NOT EXISTS` se utilizan a menudo para responder a consultas donde interesa saber si existen registros que cumplen o no con alguna propiedad específica. Esto permite generar consultas que son sencillas de entender pero no siempre son lo más eficientes.

Por ejemplo, considerando la consulta "Obtener los nombres de los clientes que no han realizado una compra", una de sus soluciones es la siguiente:

```
SELECT nombre_cliente
FROM cliente
WHERE nombre_cliente NOT IN
      (SELECT nombre_cliente
       FROM cliente NATURAL JOIN compra);
```

Pero observando el plan de ejecución generado es posible apreciar que la solución es muy ineficiente dado que trabaja materializando el resultado temporal donde busca la no existencia de cada elemento externo. Ciertamente es posible reducir el trabajo al aplicar un agrupamiento dentro de la cláusula `SELECT` más interna, pero aún así el trabajo de la materialización contempla la mayor sobrecarga de tiempo por la escritura a disco.

Actividad 15.4

Obten el plan de ejecución de la consulta anterior y describelo de manera detallada.
¿Por qué utiliza la materialización?

Escribe una sentencia equivalente utilizando la cláusula `EXISTS`.
Compara el rendimiento de las consultas.
¿Existe alguna mejora considerable?

Por otro lado, las operaciones dadas por las cláusulas `UNION`, `INTERSECTION` y `EXCEPT`, tampoco ofrecen el mejor rendimiento en ciertas consultas. Esto se tiene por la forma en la cual construyen sus resultados. Básicamente reciben dos conjuntos, sobre los cuales operan y su eficiencia esta determinada por el tiempo que toman en generarse estos conjuntos.

Considerando la siguiente consulta: "Obtener los nombres de los clientes que viven en 'Tanzania' o en 'Canada' o en 'Inglaterra' y no tienen por código postal alguno de los números 674420 o 57100".

La consulta se puede dividir en dos consultas y trabajar con sus resultados, por ejemplo:

```
SELECT nombre_cliente FROM cliente
EXCEPT (
     SELECT nombre_cliente FROM cliente
     WHERE codigo_postal_cliente <> 674420 OR
     codigo_postal_cliente <> 57100)
EXCEPT (
     SELECT nombre_cliente FROM cliente
     WHERE pais_cliente = 'Tanzania' OR pais_cliente = 'Canada'
     pais_cliente = 'Inglaterra' 57100);
```

La idea principal es tener todos los nombres de los cliente e ir eliminado aquellos que cumplen o no con las condiciones. Esto, por supuesto es ineficiente dado que por cada condición se esta analizando la tabla '*cliente*' y no se aprovecha el recorrido inicial sobre la tabla.

En estos casos siempre es mejor utilizar las cláusulas `IN` y `NOT IN` operando sobre las condiciones que se desea que cumplan o no los registros del resultado. Una sentencia SQL más eficiente para responder a la anterior consulta sería la siguiente:

```
SELECT DISTINCT nombre_cliente FROM cliente
WHERE codigo_postal_cliente NOT IN (674420, 57100)
AND pais_cliente IN ('Tanzania','Canada','Inglaterra');
```

Actividad 15.5

Obtene el plan de ejecución de las consultas anteriores y describelos de manera detallada.

Escribe, si es posible, una sentencia equivalente utilizando la cláusula EXISTS.

Actualizaciones

Las actualizaciones dentro de SQL suelen ser operaciones costosas ya que en condiciones normales, estas operaciones consisten en búsquedas secuenciales sobre la tabla donde se hace la modificación pertinente. El problema que se presenta en este tipo de operaciones se vuelve más notorio cuando se llevan a cabo distintas actualizaciones sobre una misma tabla y atributo.

Por ejemplo si se desea realizar en la base de datos del curso una actualización en los precios de los videojuegos, de tal forma que si el precio del videojuego es menor que 250.50 se requiere de un aumento en el precio de 15 % y en caso contrario se necesita una disminución en el precio de 5 %. Es posible generar las siguientes sentencias:

```
UPDATE videojuego SET precio_videojuego = precio_videojuego
     * 1.15 WHERE precio_videojuego < 250.50;

UPDATE videojuego SET precio_videojuego = precio_videojuego
     * 0.95 WHERE precio_videojuego >= 250.50;
```

Las sentencias están bien estructuradas, pero se presentan dos problemas, el primero y más importante se refiere al hecho de que la segunda actualización tomará en cuentan valores posiblemente modificados durante la primera operación y los precios no se actualizan de la forma deseada. El segundo problema está relacionado con el rendimiento: para cada una de las actualizaciones se debe recorrer la tabla '*videojuego*', lo cual es costoso porque involucra lecturas a disco. Para solventar este problema, se cuenta con la cláusula CASE que posee una utilidad muy semejante a la estructura de control 'IF' presente en la mayor parte de los lenguajes de programación orientados a objetos. Su sintaxis se presenta en la figura 15.3.

Con esta cláusula, es posible obtener una sentencia SQL más eficiente. Para el ejemplo anterior, se tiene lo siguiente:

```
UPDATE videojuego SET precio_videojuego = CASE
WHEN precio_videojuego < 250.50 THEN precio_videojuego * 1.15
ELSE precio_videojuego * 0.95 END;
```

La idea es sencilla, se realiza la actualización de la misma forma, haciendo una lectura secuencial sobre la tabla '*videojuego*', pero a diferencia de las anteriores sentencias en esta, el recorrido de la tabla solamente se lleva a cabo en una ocasión. Cuando se lee cada registro, se comprueba el valor que se tiene en el atributo del precio y dependiendo de esto, se realiza una u otra actualización.

```
CASE WHEN condición THEN resultado
     [WHEN ...]
     [ELSE resultado]
END
```

Figura 15.3: Sintaxis de la cláusula CASE.

15.4. Ejercicios

1. Ejecuta las siguientes consultas:

   a) `SELECT attname, n_distinct, most_common_vals FROM pg_stats WHERE tablename = 'cliente';`

   b) `SELECT relname, relkind, reltuples, relpages FROM pg_class WHERE relname LIKE 'cli%';`

   ¿Qué información proporcionan?
   ¿De que manera la puede utilizar el planificador de consultas?
   ¿Y el optimizador de consultas?

2. Cuando la consulta involucra reuniones externas, el planificador tiene menor libertad para elegir planes de ejecución. Explica porque sucede esto. Y genera dos consultas donde demuestres lo explicado anteriormente. Incluye planes de ejecución obtenidos.

3. Según lo visto en la tercera actividad de la práctica podemos tener control e indicar al planificador el orden de ejecución de las reuniones (`JOIN`). ¿De qué nos sirve esto?

4. Crea una sentencia en SQL equivalente a la siguiente consulta pero que se ejecute de manera más eficiente[3] :

   ```
   SELECT * FROM cliente NATURAL JOIN compra NATURAL JOIN videojuego
   WHERE consola_videojuego = 'Atari 5200';
   ```

   Genera el plan de ejecución, comparalo con el de la consulta presentada y explica porque se dieron las mejoras[4] .

5. Considera la siguiente consulta:

   a) `SELECT nombre_videojuego FROM videojuego WHERE precio_videojuego >= ALL (SELECT precio_videojuego FROM videojuego);`

   ¿Qué información se obtiene? ¿Qué puedes decir con respecto a su desempeño? Analiza los planes de ejecución y genera una consulta que obtenga la misma información pero con un mejor desempeño.

[3] No generes más índices

[4] No olvides incluir en el reporte la configuración del equipo, tanto de hardware como de software, en donde ejecutaste las pruebas.

Explica el plan de ejecución de tu nueva consulta y comparalo con la presentada aquí.

6. Considera la consulta de la actividad número cuatro de la presente práctica. Obten una consulta equivalente a las que utilizan los operadores `NOT IN` y `NOT EXISTS`. Muestra y explica a detalle su plan de ejecución.
¿Puedes mejorar el rendimiento aún más? ¿Cómo?

7. Considere la consulta de la actividad número cinco de esta práctica. Obten una consulta equivalente pero más eficiente[5] . Muestra su plan de ejecución y explica ¿cómo mejoraste el rendimiento?

8. Considera la siguiente consulta

```
SELECT nombre_cliente, ap_mat_cliente
FROM cliente CROSS JOIN compra CROSS JOIN videojuegos
WHERE nombre_cliente LIKE '%e'
AND videojuegos.consola_videojuegos = 'PC';
```

Utiliza `EXPLAIN` junto con `ANALYZE` para determinar el plan de ejecución y su tiempo estimado, modifica el orden de las tablas y los distintos algoritmos para la reunión y genera el mejor plan de ejecución.

9. Crea las siguientes vistas:

   a) `videoDiv (id_videojuego)`

   b) `compraDiv (id_cliente, id_videojuego)`

Para encontrar el '*id_cliente*' del cliente que ha comprado TODOS los videojuegos tenemos la siguiente sentencia SQL que nos representa una división relacional:

```
SELECT id_cliente
FROM compraDiv
WHERE id_videojuego IN (SELECT id_videojuego FROM videoDiv)
GROUP BY id_cliente
HAVING COUNT(*) = (SELECT COUNT (*) FROM videoDiv);
```

Encuentra una sentencia equivalente que tenga un mejor rendimiento, muestra el plan de ejecución y explica porque es mejor al compararlo con el plan de ejecución de esta consulta.

[5] Puedes utilizar índices

Transacciones

Semana:	Décima semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de transacciones. Programación en Java.
Herramientas:	PostgreSQL v8.0.2 JDK v5.0 (1.5.0) Controlador JDBC PostgreSQL
Objetivo:	Manejar transacciones y resolver los posibles problemas que se presenta con esto.

La práctica número dieciséis se enfoca a la utilización de transacciones dentro del SMBD. Se plantea un análisis e implementación dentro de PostgreSQL.

Los ayudantes deben haber presentado diversos ejercicios de seriabilidad, para mejor aprovechamiento de la práctica.

PRÁCTICA 16

Transacciones

16.1. Meta

Que el alumno comprenda la importancia que tiene el concepto de transacción dentro de un SMBD y las consecuencias de no aplicarlo.

16.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar las situaciones adecuadas para manejar transacciones explícitas.
- Implementar transacciones explícitas utilizando puntos de verificación y puntos de salvado.

16.3. Desarrollo

El concepto de transacción es fundamental en todos los SMBD, permite agrupar en una forma lógica a un conjunto de operaciones y manejarlas como una sola. Los estados intermedios que se presentan no son detectables ni para el propio usuario que ejecutó la transacción, ni para el resto de las transacciones que se estén ejecutando concurrentemente en el sistema, ya sea iniciadas por el mismo usuario o por otro. Si alguna operación perteneciente a la transacción no puede realizarse completamente, ninguna de las operaciones que forman parte de la transacción se realiza y la base de datos se regresa al estado previo al inicio de la transacción.

Propiedades ACID

Es deseable que las transacciones posean un conjunto de propiedades, las cuales les confieren sus características y permiten llevar a cabo su funcionalidad. Este conjunto de propiedades se conoce como '*propiedades ACID*' y son las siguientes:

- Atomicity (Atomicidad) - Todas las operaciones que componen a la transacción se ejecutan como una única instrucción, de modo que, si se ejecuta una se ejecutan todas, si no se ejecuta alguna entonces no se ejecuta ninguna.

- Consistency (Consistencia) - La base de datos debe encontrarse en un estado consistente antes y después de la ejecución de la transacción.

- Isolation (Aislamiento) - Cada transacción actúa en la base de datos como si fuese la única transacción que se está ejecutando y su presencia en el sistema debe ser desapercibida por las demás transacciones.

- Durability (Durabilidad) - Al concluir la ejecución correcta de una transacción, ésta deja a la base de datos con los cambios de manera permanente esto es, se garantiza que todas la actualizaciones son almacenadas en un registro de manera permanente, generalmente en disco duro, antes de que la transacción sea registrada completamente.

Hasta el momento se ha trabajado implícitamente con la idea de transacciones: cuando se realiza una consulta que involucra otra consulta anidada, la consulta exterior para completar su ejecución necesita del resultado obtenido por la consulta más interna de modo que o bien terminan las dos consultas o no termina la consulta exterior. Ejemplo de esto se presenta en la figura 16.1.

```
SELECT nombre_cliente
FROM   cliente
WHERE  codigo_postal_cliente =
       (SELECT MAX(codigo_postal_cliente)
        FROM cliente);
```

Figura 16.1: La consulta más externa debe esperar a la finalización de la interna.

Hay que observar que no es una verdadera transacción, dado que en una transacción las operaciones dependen totalmente una de otra mientras que en las consultas anidadas las operaciones externas dependen de las internas, no así las internas que eventualmente pueden terminar mientras que las externas no. Ejemplo de esto se muestra en la figura 16.2.

```
SELECT nombre_cliente
FROM cliente
WHERE codigo_postal_cliente =
     (9090990 /
     (SELECT MAX(codigo_postal_cliente)
      FROM cliente
      WHERE codigo_postal_cliente = 0)) ;
```

Figura 16.2: La consulta más externa no termina aunque la interna si.

16.3.1. Transacciones en PostgreSQL

La utilidad práctica de las transacciones se aprecia cuando es necesario garantizar la ejecución de un conjunto de operaciones sobre la base de datos, pero se requiere que este conjunto de operaciones se realice en su totalidad. Las situaciones en donde se presenta esto son múltiples. A continuación se muestra un ejemplo: en la base de datos del curso se tiene que dar una modificación en los precios a los cuales se vende cada videojuego, esto dependiendo de la plataforma, ya que los nuevos precios tienen una fluctuación entre 15 % de reducción y 30 % de aumento (ver tabla 16.1). La solución creada en primera instancia consiste en ejecutar una actualización para cada una de las plataformas.

Plataforma	Variación en precio
Atari 2600	-5 %
Dreamcast	-2 %
Game Boy Advance	-10 %
GameCube	-15 %
PlayStation Two	+25 %
PlayStation Portable	+30 %
Sega 32X	+10 %
Xbox	+28 %

Tabla 16.1: Tabla de actualizaciones en precios.

El problema surge cuando se considera que cada actualización dentro de la base de datos tarda aproximadamente m segundos. Entonces según nuestra tabla de modificaciones, se tienen que hacer 8 operaciones donde cada una tarda m segundos en

promedio. En un sistema ideal, es posible ejecutar sin problema alguno cada una de las consultas por separado, es decir, se realizan las siguientes actualizaciones:

```
UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.10 * precio_videojuego)
WHERE consola_videojuego = 'Game Boy Advance';

UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.5 * precio_videojuego)
WHERE consola_videojuego = 'Atari 2600';

                    ⋮

UPDATE videojuegos SET precio_videojuego =
precio_videojuego + (.28 * precio_videojuego)
WHERE consola_videojuego = 'Xbox';
```

En un ambiente real es factible que al menos una operación falle, de modo que si durante la ejecución de alguna actualización se presenta un error en el servidor, no es posible garantizar que la siguiente actualización se logre ejecutar, más aún, tampoco se puede regresar la base de datos a un estado previo a la operación actual. Para solventar estos problemas se recurre a las transacciones.

En PostgreSQL, una transacción se delimita con las cláusulas `BEGIN TRANSACTION` y `COMMIT`, entre las cuales se colocan todas las operaciones que se desea formen parte de la transacción. Para el conjunto anterior de actualizaciones, la transacción queda así:

```
BEGIN TRANSACTION;
UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.10 * precio_videojuego)
WHERE consola_videojuego = 'Game Boy Advance' ;

UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.5 * precio_videojuego)
WHERE consola_videojuego = 'Atari 2600';
                    ⋮
COMMIT;
```

Por otro lado, si se desea no llevar a cabo las actualizaciones o si ocurre un error, existe la posibilidad de utilizar la cláusula `ROLLBACK` en lugar de `COMMIT` y todas las actualizaciones serán canceladas.

Internamente PostgreSQL trata cada sentencia SQL como si se ejecutará dentro de una transacción. Y un conjunto de sentencias rodeadas por `BEGIN TRANSACTION` y `COMMIT` se conoce como un '*bloque de transacciones*'.

Es importante mencionar que cuando se ejecutan dos o más transacciones concurrentemente, éstas no pueden notar los cambios incompletos efectuados por otras transacciones. Por ello, las transacciones deben ser unidades lógicas indivisibles no solamente en términos de su efecto permanente en la base de datos, sino también en términos de su visibilidad mientras se ejecutan. Los cambios efectuados por una transacción no son detectados hasta que se termina de ejecutar, con lo cual todas sus modificaciones realizadas sobre la base de datos, llegan a ser registradas simultáneamente.

Puntos de guardado

Es posible controlar las sentencias que componen a las transacciones de un modo más fino, al utilizar los puntos de guardado o '*savepoints*'. Éstos, permiten selectivamente descartar secciones de una transacción, mientras el resto se lleva a cabo. Su definición se logra en PostgreSQL con la cláusula `SAVEPOINT`. `SAVEPOINT` denota un conjunto de operaciones de las cuales se garantiza su ejecución, independientemente del resto de las operaciones que componen a la transacción. Este conjunto comprende a todas las operaciones denotadas a partir del inicio de la transacción (`BEGIN TRANSACTION`) o de un `SAVEPOINT` anterior. Para regresar a un punto de guardado se utiliza la cláusula `ROLLBACK TO`. Con esto, los cambios posteriores al punto de guardado se descartan, mientras que los cambios anteriores se conservan.

En general es preciso delimitar dentro de una transacción que operaciones son criticas y cuales no, a modo de colocar este tipo de operaciones en bloques delimitados por cláusulas `SAVEPOINT`. En nuestro ejemplo, puede ser valido completar ciertas actualizaciones y otras simplemente no. Un punto de guardado se conserva dentro del sistema para futuras referencias, esto trae consigo una carga considerable de recursos por parte del SMBD, por ello es conveniente liberar estos puntos de guardado explícitamente utilizando la cláusula `RELEASE SAVEPOINT`. En caso de utilizar otro punto de guardado, el sistema automáticamente liberará todos los puntos de guardado que se hayan definido anteriormente. Todo esto dentro del bloque de transacciones actual, de modo que estos puntos de guardado no son visibles para otras sesiones dentro de la base de datos. Cuando el bloque de transacciones ejecuta la cláusula `COMMIT`, las modificaciones se vuelven visibles para el resto de las sesiones, si por el contrario ejecuta la cláusula `ROLLBACK` las modificaciones no se presentan.

Regresando al ejemplo, si tiene la situación en donde se realiza una operación errónea[1] dentro de una transacción, es posible evitar esos cambios utilizando `SAVEPOINTS` y retornando a ellos cuando se presente el error. Por ejemplo:

[1] Recordando que las situaciones son más a modo de ejemplo que realistas.

```
BEGIN TRANSACTION;
UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.10 * precio_videojuego)
WHERE consola_videojuego = 'Game Boy Advance' ;
SAVEPOINT punto_uno;
UPDATE videojuegos SET precio_videojuego =
precio_videojuego + (.25 * precio_videojuego)
WHERE consola_videojuego = 'Game Boy Advance' ;
 -- Error, volvimos a aumentar el valor de estos juegos!
ROLLBACK TO punto_uno;
UPDATE videojuegos SET precio_videojuego =
precio_videojuego - (.2 * precio_videojuego)
WHERE consola_videojuego = 'Dreamcast' ;
                          ⋮
UPDATE videojuegos SET precio_videojuego =
precio_videojuego + (.28 * precio_videojuego)
WHERE consola_videojuego = 'Xbox' ;
COMMIT;
```

Actividad 16.1

En una secuencia de operaciones ejecutada directamente sobre el SMBD, la utilidad de ROLLBACK no es tan evidente.
Pero considerando una conexión al SMBD a través de una red de computadoras, tiene mayor importancia.
En Java, específicamente dentro de JDBC se maneja la opción de **Autocommit** con el valor de TRUE. ¿Por qué consideras que se define así?.
¿Qué problemas presenta ?.

Aislamiento de transacciones

El aislamiento de transacciones es un elemento crítico en cualquier SMBD. El nivel de aislamiento de una transacción se encuentra determinado por las condiciones de aislamiento conocidas como 'lecturas sucias' (*dirty reads*), 'lecturas repetidas' (*repeatable reads*) y 'lecturas fantasma' (*phantom reads*). Estas condiciones describen que sucede cuando dos o más transacciones operan sobre los mismos datos.

Niveles de aislamiento

El nivel de aislamiento de una transacción determina que datos puede tomar en cuenta la transacción cuando otras transacciones se encuentran ejecutándose concurrentemente en el sistema. En PostgreSQL se cuenta con la cláusula SET TRANSACTION

para controlar el nivel de aislamiento de una transacción.

SET TRANSACTION permite determinar las características de la siguiente transacción únicamente. Si se desea cambiar el comportamiento de una serie de transacciones, se debe utilizar la cláusula SET CHARACTERISTICS.

Las propiedades de las transacciones que se pueden modificar son dos: el nivel de aislamiento y el modo de acceso (de lectura/escritura y sólo de lectura). Su sintaxis se presenta en la figura 16.3.

```
SET TRANSACTION modo_transaccion [,...]
SET SESSION CHARACTERISTICS AS TRANSACTION modo_transaccion [,...]

Donde modo_transaccion es uno de los siguientes:

ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ |
                  READ COMMITTED | READ UNCOMMITTED }
                  READ WRITE | READ ONLY
```

Figura 16.3: Sintaxis de la cláusula SET TRANSACTION.

Los niveles de aislamiento definidos en PostgreSQL son los siguientes:

- READ COMMITTED - Las operaciones de la transacción actual sólo puede considerar:

  1. Tuplas que no son utilizadas por otra transacción.
  2. Tuplas tales que las transacciones que operan con ellas, las dejen luego de alcanzar un estado de completa ('*commited*') antes que la transacción actual inicie.

  Este es el comportamiento que se da por omisión.

- SERIALIZABLE - Las operaciones de la transacción actual solamente pueden considerar:

  - Tuplas que no son utilizadas por otra transacción.
  - Tuplas tales que las transacciones que operan con ellas, las dejen luego de alcanzar un estado de completa antes de que la primera operación de consulta o actualización de la transacción actual se ejecute.

El estándar SQL define dos niveles adicionales, READ UNCOMMITTED y REPEATABLE READ. Pero en PostgreSQL READ UNCOMMITTED es tratado como READ COMMITTED mientras que REPEATABLE READ se maneja como SERIALIZABLE, esto se implementa así por-

que es la única forma directa de corresponder los niveles de aislamiento estándar dentro de la arquitectura MVCC [18][2] .

El nivel de aislamiento de una transacción no puede modificarse luego de que se ejecute la primera consulta de datos o bien la primera operación que modifique dato alguno (`SELECT`, `INSERT`, `DELETE`, `UPDATE`, `FETCH` ó `COPY`).

El modo de acceso de la transacción determina si la transacción es de lectura/escritura o de sólo lectura. Cuando una transacción es únicamente de lectura, las siguientes sentencias SQL no se permiten: `INSERT`, `UPDATE`, `DELETE`, `COPY TO`, `CREATE`, `ALTER`, `DROP`, `COMMENT`, `GRANT`, `REVOKE`, `TRUNCATE`, `EXPLAIN ANALYZE` y `EXECUTE` (si el comando por ejecutar es alguno de los anteriores).

Hay que notar que si es utilizado un `SET TRANSACTION` sin un `START TRANSACTION` o `BEGIN`, puede parecer que no tienen efecto las cláusulas dado que solamente es una transacción e inmediatamente entra en ejecución al sistema, pero no es así.

[2] Las características de esta arquitectura se analizan en la siguiente práctica.

16.4. Ejercicios

1. Considera las siguientes transacciones:

   T0:
   ```
   BEGIN TRANSACTION ISOLATION LEVEL ______ ;
   SELECT SUM(cantidad)
   FROM compra
   WHERE id_cliente = 920320
   COMMIT;
   ```

   T1:
   ```
   BEGIN TRANSACTION ISOLATION LEVEL ______ ;
   INSERT INTO compra VALUES (920320, 34920, 20, 2005, 1, 15);
   INSERT INTO compra VALUES (920320, 34920, 40, 2005, 1, 16);
   INSERT INTO compra VALUES (920320, 34920, 50, 2005, 1, 17);
   COMMIT;
   ```
   ¿Qué nivel de aislamiento deben tener las transacciones para que se presente una lectura fantasma?

2. Presenta dos transacciones dentro del contexto de la base de datos del curso que hagan presente el fenómeno de 'lectura repetida'. Explica por que sucede.

3. Presenta dos transacciones dentro de la base de datos del curso que hagan presente el fenómeno de 'lectura sucia'. Explica por que sucede.

4. ¿En qué condiciones es valido tener lecturas no repetidas? ¿cuando es bueno tener lecturas repetidas? Ejemplifica ambos casos utilizando tres transacciones por cada ejemplo. (las transacciones deben tener sentido dentro de la base de datos del curso).

   En caso de no ser posible crear el conjunto de transacciones pedido, indicar porque no es posible obtenerlo.

5. Considera las siguiente transacción:

   T0:
   ```
   BEGIN TRANSACTION;
   UPDATE cliente SET ciudad_cliente = 'Distrito Federal'
   WHERE pais_cliente = 'Mexico';
   SAVEPOINT punto_uno;
   UPDATE cliente SET ciudad_cliente = 'Paris'
   WHERE pais_cliente = 'Francia';
   UPDATE cliente SET ciudad_cliente = 'Hamburgo'
   ```

```
WHERE pais_cliente = 'Alemania';
SAVEPOINT punto_dos;
UPDATE cliente SET ciudad_cliente = 'Roma'
WHERE pais_cliente = 'Italia';
UPDATE cliente SET ciudad_cliente = 'Ottawa'
WHERE pais_cliente = 'Canada';
RELEASE SAVEPOINT punto_uno;
ROLLBACK TO punto_dos;
COMMIT;
```

¿Qué actualizaciones se llevaron a cabo?
¿Qué error presenta el sistema? Justifica tu respuesta

6. Considera las siguiente transacción:

T0:

```
BEGIN TRANSACTION;
UPDATE cliente SET ciudad_cliente = 'Distrito Federal'
WHERE pais_cliente = 'Mexico';
SAVEPOINT punto_uno;
UPDATE cliente SET ciudad_cliente = 'Paris'
WHERE pais_cliente = 'Francia';
UPDATE cliente SET ciudad_cliente = 'Hamburgo'
WHERE pais_cliente = 'Alemania';
UPDATE cliente SET ciudad_cliente = 'Roma'
WHERE pais_cliente = 'Italia';
UPDATE cliente SET ciudad_cliente = 'Ottawa'
WHERE pais_cliente = 'Canada';
RELEASE SAVEPOINT punto_uno;
SAVEPOINT punto_dos;
ROLLBACK TO punto_dos;
ROLLBACK;
```

¿Qué actualizaciones se llevaron a cabo?
¿Qué error presenta el sistema? Justifica tu respuesta.
Si se sustituye la última instrucción por `COMMIT`, ¿qué sucede?

7. Considera el siguiente código de un programa escrito en Java:

```
import java.sql.*;
public class transacciones {
  public static void main(String args[]) throws ClassNotFoundException, SQLException {
    String url = "jdbc:postgresql";
    String database = "rvilleda";
    String username = "rvilleda";
    String password = "rvilleda2005";
    Connection connection = null;
    Statement sentence = null;
    Statement update = null;
    ResultSet results = null;
    String consulta01 = "SELECT COUNT(id_cliente)
            FROM cliente WHERE ciudad_cliente = 'Ecatepec';";
    String consulta02 = "SELECT COUNT(id_cliente)
            FROM cliente WHERE ciudad_cliente = 'Berlin';";
    String actualiza01 = "UPDATE cliente SET ciudad_cliente = 'Ecatepec'
            WHERE pais_cliente = 'Mexico';";
    String actualiza02 = "UPDATE cliente SET ciudad_cliente = 'Berlin'
            WHERE pais_cliente = Álemania';";
      try {
      Class.forName("org.postgresql.Driver");
      connection = DriverManager.getConnection(url +":"+ database, username, password);
      System.out.println("Conexion exitosa!");
      connection.setAutoCommit(false);
      System.out.println ("Autocommit DESHABILITADO");
      System.out.println("Ejecutando consulta 1 ");
      sentence = connection.createStatement();
      results = sentence.executeQuery(consulta01);
      results.next();
      System.out.println("Tuplas totales en consulta 01 -: "+ results.getInt(1));
      System.out.println("Ejecutando actualizacion 1");
      update = connection.createStatement();
      update.executeUpdate(actualiza01);
      System.out.println("Ejecutando consulta 2 ");
      sentence = connection.createStatement();
      results = sentence.executeQuery(consulta02);
      results.next();
      System.out.println("Tuplas totales en consulta 02 -: "+ results.getInt(1));
      System.out.println("Ejecutando actualizacion 2");
      update = connection.createStatement();
      update.executeUpdate(actualiza02);
```

```
            connection.commit();
        } catch(SQLException ex) {
            System.out.println("SQLException: " + ex.getMessage());
            if (connection != null)
            try {
                System.out.println("La transaccion ha sido abortada ");
                connection.rollback();
            } catch(SQLException e) {
                System.out.print("SQLException: ");
                System.out.println(e.getMessage());
            }
        }
        finally {
            if (connection != null)
            try {connection.close();} catch(SQLException e) { e.printStackTrace();}
        }
        }
    }
```

Descarga el controlador JDBC de PostgreSQL[3] y actualiza tu variable de entorno `CLASSPATH`. Modifica el programa anterior y ejecutalo sobre la base de datos del curso. Desde otra terminal elimina el proceso correspondiente al programa, justo cuando se inicie la segunda actualización. Responde si se llevaron a cabo las transacciones.

8. Elimina la línea 'connection.setAutoCommit(false);' y repite la prueba del punto anterior. ¿qué sucede con las actualizaciones? ¿cuáles se llevan a cabo?

9. Crea un programa en Java llamado 'transaccion.java' que permita la ejecución de 3 operaciones de actualización, e interactivamente permita llevar a cabo la operación de 'commit' y 'rollback' para cada una de ellas o de manera general.

[3] Se encuentra en `http://jdbc.postgresql.org/`.

Concurrencia

Semana:	Décimo primera semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de SQL. Conceptos de transacciones. Conceptos de concurrencia.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Aprender la utilización de los niveles de aislamiento y bloqueos en un SMBD.

La práctica número diecisiete trata el tema de concurrencia. Es importante que los alumnos manejen sólidamente los conceptos de transacciones y ejecución concurrente, además es recomendable que el profesor y los ayudantes muestren diversos ejemplos en clase para ayudar la compresión de la práctica.

PRÁCTICA 17

Concurrencia

17.1. Meta

Que el alumno comprenda la importancia que tiene el manejo adecuado de los niveles de aislamiento y sus implicaciones en el rendimiento del SMBD.

17.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar los distintos niveles de aislamiento de transacciones.
- Identificar cuando utilizar los distintos niveles de aislamiento.
- Comprender la importancia de los bloqueos.
- Identificar cuando utilizar los distintos tipos de bloqueos.

17.3. Desarrollo

La capacidad de manejar consultas concurrentemente es fundamental en el desempeño y operación de un SMBD, la idea de contener información y hacerla disponible para múltiples usuarios es uno de los principales propósitos de los sistemas actuales.

La concurrencia permite trabajar simultáneamente sobre la misma información sin tener que preocuparse por detalles internos o externos, el sistema se encarga de asignar los recursos disponibles a cada transacción dependiendo del tipo de usuario que la haya

iniciado, sus permisos asignados, tipo de transacción que se desea ejecutar y el orden en el cual arriban al sistema.

Los SMBD logran esto al implementar esquemas de control de concurrencia, los cuales varían dependiendo del tipo de base de datos que se opere.

17.3.1. Concurrencia dentro de PostgreSQL

A diferencia de otros SMBD que utilizan únicamente los bloqueos para el control de concurrencia, PostgreSQL preserva la consistencia de los datos utilizando un modelo conocido como Control de Concurrencia de Múltiples Versiones (*'Multiversion Concurrency Control, MVCC'*). El modelo en cuestión trata a cada transacción brindándole una 'imagen instantánea' de los datos (una versión de la base de datos) tal cual se encontraban previo al inicio de la transacción. Esto protege a las transacciones de leer datos inconsistentes que pudiesen ser causados por otras transacciones concurrentes que actualicen en esos instantes a los mismos datos, de modo que se asegura el aislamiento de cada sesión en la base de datos.

La ventaja de utilizar este modelo es que las transacciones que solamente involucran operaciones de lectura, no tienen conflicto con los bloqueos adquiridos por transacciones que escriben datos, de modo que una lectura nunca bloquea escrituras y viceversa. Los bloqueos a nivel de tabla y de tupla también están disponibles dentro de PostgreSQL para aplicaciones que no pueden adaptarse tan fácilmente al comportamiento de MVCC.

Aislamiento de transacciones en PostgreSQL

El estándar de SQL define cuatro niveles de aislamiento para las transacciones en términos de los tres fenómenos que se deben evitar entre transacciones concurrentes, a saber:

- Lectura sucia (*dirty read*).
- Lectura no repetible (*nonrepeatable read*).
- Lectura fantasma (*phantom read*).

Los cuatro niveles de aislamiento y sus correspondientes comportamientos ante los fenómenos descritos anteriormente se presentan en la tabla 17.1.

En PostgreSQL, se puede solicitar explícitamente el manejo de los cuatro niveles de aislamiento, pero internamente sólo existen dos, 'Lectura Comprometida' y 'Serializable'. La razón de ello es que sólo para estos niveles de aislamiento existe un equivalente dentro del control de concurrencia de múltiples versiones.

Para controlar el nivel de aislamiento de una transacción se tiene a disposición la cláusula `SET TRANSACTION`.

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma
Lectura no comprometida	Posible	Posible	Posible
Lectura comprometida	No posible	Posible	Posible
Lectura repetible	No posible	No posible	Posible
Serializable	No posible	No posible	No posible

Tabla 17.1: Niveles de aislamiento según SQL.

Nivel de aislamiento - lectura comprometida

Este nivel de aislamiento de transacciones es el que se presenta por omisión en PostgreSQL. Cuando una transacción se ejecuta en este nivel de aislamiento, cualquier consulta SELECT sobre los datos que maneja la transacción obtendrá datos comprometidos ('*commited*') antes de que la consulta haya iniciado, no obtiene datos no comprometidos o cambios comprometidos durante la ejecución de la consulta realizados por otras transacciones concurrentes.

Las actualizaciones se comportan de forma semejante en este nivel, si una transacción $T1$ intenta realizar una actualización sobre datos que están siendo modificados por otra transacción $T2$, la segunda transacción en tiempo $T1$, se espera a que la primera termine su ejecución, si $T2$ no modificó la tupla actual por un 'rollback' entonces $T1$ procede a modificar la tupla tomando en cuenta su valor original (dado que $T2$ no lo modificó en absoluto). Por otro lado si $T2$ modificó la tupla actual y el cambio fue comprometido, entonces $T1$ verifica si cumple con la condición o no. Si es el caso entonces lleva a cabo su actualización. Pero si $T2$ modificó la tupla actual de modo que se borra o no cumple con las condiciones de $T1$, esta última simplemente la ignora y procede con las siguiente tupla.

Actividad 17.1

¿Cuál es el problema que presenta el tipo de aislamiento presentado?

Se mencionó que PostgreSQL sólo implementa internamente 2 tipos de aislamiento. ¿Por qué sucede esto?
¿Realmente se resuelven todos los fenómenos no deseados en transacciones concurrentes?

Nivel de aislamiento - serializable

Este nivel de aislamiento provee el control más estricto de todos, emula la ejecución serial en la cual una transacción se ejecuta totalmente tras de otra en lugar de hacerlo

concurrente. Sin embargo pueden ocurrir fallas durante el proceso que retarden las respuestas por parte del servidor.

Cuando una consulta SELECT se ejecuta en este nivel, ésta sólo ve datos comprometidos antes de que la transacción inicie; nunca refleja datos que no han sido comprometidos o cambios debidos a transacciones ejecutadas concurrentemente (aunque si describe los efectos de actualizaciones previas durante la propia transacción aún cuando no se han comprometido).

Las actualizaciones se comportan de forma semejante en este nivel, si una transacción $T1$ intenta realizar una actualización sobre datos que están siendo modificados por otra transacción $T2$, la segunda transacción en tiempo $T1$, se espera a que la primera termine su ejecución, si $T2$ no modificó la tupla actual por un 'rollback' entonces $T1$ procede a modificar la tupla tomando en cuenta su valor original (dado que $T2$ no lo modificó en absoluto). Por otro lado si $T2$ modificó la tupla actual y el cambio se comprometio entonces $T1$ se deshace y regresa a su estado anterior, el sistema informa que no se pudo llevar a cabo la serialización con el mensaje:

```
ERROR: could not serialize access due to concurrent update
```

esto porque $T1$ no puede alterar tuplas modificadas por otras transacciones, en este caso $T2$, después de que la transacción serializable dió inicio.

El modo de aislamiento serializable provee una garantía de que cada transacción recibe una vista consistente de la base de datos, sin embargo, la aplicación que se conecta a la base de datos debe estar preparada para reintentar transacciones cuando se llevan a cabo actualizaciones concurrentes. Dado que el costo de volver a ejecutar una transacción compleja puede ser alto, este nivel de aislamiento se recomienda sólo cuando las transacciones de actualización contienen una lógica compleja tal que si se utiliza el nivel de Lectura Comprometida, pueden generarse respuestas erróneas. Este modo es necesario cuando una transacción ejecuta sucesivas cláusulas que deben trabajar con vistas idénticas de la base de datos.

Actividad 17.2

¿Cuál es el problema que presenta el nivel de aislamiento presentado?

Considera las siguiente tabla *tmp*:

```
tipo | valor
-------------
   1 |  10
   1 |  20
   2 |  100
   2 |  200
```

Ejecuta la siguientes transacciones:

```
T0 = BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO tmp SELECT 2, SUM(value) FROM tmp WHERE tipo = 1;
COMMIT;
```

y

```
T1 = BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
INSERT INTO tmp SELECT 1, SUM(value) FROM tmp WHERE tipo = 2;
COMMIT;
```

¿Qué problemas se presentan si se ejecutan de manera concurrente?

¿Cuál ejecución es correcta? ¿$T0 \longrightarrow T1$ ó $T1 \longrightarrow T0$?

Para garantizar una real serialización, es necesario para el SMBD forzar el *bloqueo de predicados*, lo que significa que una transacción no puede insertar o modificar datos que cumplan con las condiciones `WHERE` de una consulta en otra transacción concurrente. Prácticamente el sistema previene lecturas fantasma al restringir lo que se escribe, mientras que el sistema MVCC previene estas lecturas al restringir lo que se lee.

Aún así se cuenta con la posibilidad de manejar bloqueos explícitamente para prevenir que transacciones concurrentes trabajen sobre los mismos datos de manera simultanea.

Bloqueos explícitos

PostgreSQL provee distintos modos de bloqueos para controlar el acceso concurrente en tablas. Estos modos pueden ser utilizados por las aplicaciones de modo que controlen

los bloqueos cuando el MVCC no presente un comportamiento deseable. Por otro lado, la mayor parte de las cláusulas en PostgreSQL automáticamente adquieren bloqueos de modos apropiados para asegurar que las tablas a las que hacen referencia no sean modificadas o referenciadas en formas incompatibles mientras el comando se ejecuta.

Actividad 17.3

La vista '*pg_locks*' del sistema muestra la lista de los bloqueos actuales que se mantienen en el sistema. Realiza lo siguiente:

```
nombre_db#=> SELECT * FROM pg_locks;
```

e investiga que significa cada columna.

Bloqueos a nivel de tablas

Para obtener los resultados correctos (serializables) en una ejecución concurrente de transacciones, en ciertos casos se debe recurrir al bloqueo de tablas. Este tipo de bloqueo obtiene los privilegios en la tabla en cuestión y le permite a la transacción trabajar con la tabla sin permitir que otra transacción concurrente la modifique de alguna forma.

Este tipo de bloqueo se proporciona con la cláusula `LOCK TABLE`, su sintaxis se presenta en la figura 17.1. `LOCK TABLE` indica a la transacción que espere si es necesario a que cualquier bloqueo que haga conflicto con el deseado sea liberado antes de poner su bloqueo, si se opta por la opción `NOWAIT` entonces no existe espera alguna y la transacción intenta adquirir los bloqueos, en caso de no ser posible, la transacción aborta y se emite un mensaje de error. En ambos casos, si el bloqueo se adquirió, entonces se mantiene hasta la finalización de la actual transacción[1] .

Cuando se otorgan bloqueos automáticamente para las cláusulas que hacen referencias a tablas, PostgreSQL siempre utiliza el modo de bloqueo menos restrictivo[2]

Del mismo modo es posible adquirir cualquiera de estos bloqueos de manera explícita con el comando `LOCK`, recordando que todos estos modos de bloqueo son a nivel de tabla, aún cuando en su nombre aparezca `ROW`. De hecho la única diferencia real entre un modo de bloqueo y otro, es el conjunto de bloqueos con los cuales tienen conflicto cada uno. Dos transacciones no pueden tener bloqueos los cuales tengan conflictos entre si, sobre

[1] No existe la cláusula `UNLOCK TABLE`, para liberar explícitamente los bloqueos.

[2] Para mayor información sobre los distintos niveles de bloqueo, consultar la página siguiente: `http://www.postgresql.org/docs/8.0/interactive/explicit-locking.html`

```
LOCK [ TABLE ] nombre_tabla [,...] [ IN modo_bloqueo MODE ] [ NOWAIT ]

Donde modo_bloqueo es uno de los siguientes:

    ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE
    | SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE
```

Figura 17.1: Sintaxis de la cláusula LOCK TABLE.

la misma tabla al mismo tiempo.

Bloqueos a nivel de tupla

Un bloqueo a nivel de tupla se adquiere automáticamente cuando la tupla es modificada, el bloqueo se mantiene hasta que la transacción termina comprometida o se deshace con un 'rollback'. Los bloqueos a nivel de tupla no afectan las consultas; solamente bloquean a posibles transacciones de escribir en la misma tupla.

Deadlocks

La posibilidad de brindar el uso de bloqueos explícitos acarrea un aumento en la aparición de '*deadlocks*'[3] . Por ejemplo si una transacción $T1$ adquiere un bloqueo exclusivo en una tabla 'A' y luego intenta adquirir otro sobre la tabla B, mientras la transacción $T2$ previamente cuenta con un bloqueo exclusivo en la tabla B y luego pide un bloqueo exclusivo en la tabla A, entonces, las dos transacciones quedarán en estado de espera. PostgreSQL detecta estos conflictos y elimina el deadlock al abortar una transacción.

Hay que notar que los deadlocks también pueden ocurrir por bloqueos a nivel de tupla (y por lo tanto ocurrir aún cuando el bloqueo explícito no se utilice). Lo mejor que se puede hacer para evitar los deadlocks, es verificar que las aplicaciones que utilicen múltiples objetos de la base de datos, lo hagan en orden consistente, así mismo también es buena práctica hacer que las transacciones adquieran los bloqueos más restrictivos.

[3] Abrazo mortal

Actividad 17.4

El 'abrazo mortal', puede ocasionarse aún en escenarios muy simples. Para mostrarlo, manten abiertas dos sesiones en la base de datos del curso y ejecuta las siguientes transacciones:

En la terminal 1:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE videojuego
SET precio_videojuego = (precio_videojuego * .15)
WHERE id_videojuego = 385892;
SELECT COUNT(id_cliente) FROM cliente;
UPDATE videojuego
SET precio_videojuego = (precio_videojuego * .15)
WHERE id_videojuego = 255892;
COMMIT;
```

y en la terminal 2:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE videojuego
SET precio_videojuego = precio_videojuego * .15)
WHERE id_videojuego = 255892;
SELECT COUNT(id_cliente) FROM cliente;
UPDATE videojuego
SET precio_videojuego = precio_videojuego * .15)
WHERE id_videojuego = 385892;
COMMIT;
```

Analiza la salida al finalizar las dos transacciones en sus respectivas terminales.
¿Qué problemas se presentan si se ejecutan de manera simultanea?

Existe un parámetro asociado a la configuración del servidor (que puede ser configurado en tiempo de ejecución) llamado '*deadlock_timeout*'. Éste se puede ajustar mediante el archivo de configuración '*postgresql.conf*' y representa la cantidad de tiempo en milisegundos que debe esperar el sistema al encontrarse un bloqueo antes de verificar si existe una condición de deadlock. La verificación es lenta, así que el servidor no la lleva a cabo

cada que se encuentra otorgando un bloqueo. Incrementar el valor reduce la cantidad de tiempo utilizado en la verificación de la existencia de un deadlock pero decrementa el reporte de deadlocks reales.

17.4. Ejercicios

1. Dado el nivel de aislamiento de Lectura Comprometida, En una terminal ejecuta la siguiente transacción:

   $T0$:
   ```
   SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
   ciudad_cliente
   FROM cliente WHERE pais_cliente = 'Mexico' LIMIT 25;
   ```

   Analiza los resultados.
   Ejecuta en otra terminal ejecuta la transacción:

   $T1$:
   ```
   BEGIN;
   UPDATE cliente SET ciudad_cliente = 'Merida'
   WHERE pais_cliente = 'Mexico';
   ```

   En otra terminal ejecuta la transacción:

   $T2$
   ```
   BEGIN;
   UPDATE cliente SET ciudad_cliente = 'Zacatecas'
   WHERE pais_cliente = 'Mexico';
   ```

   En la terminal de la transacción $T0$ ejecuta nuevamente la consulta y analiza los resultados. Han cambiado los datos?

   Ninguna de las dos transacciones ha terminado, realiza en $T1$ un `COMMIT` y en $T2$ un `ROLLBACK`
   Realiza la consulta de $T0$ y responde ¿Qué ha sucedido? ¿Qué beneficios tiene este nivel de aislamiento? y ¿qué efectos negativos permite?

2. Muestra 4 ejemplos de conjuntos de transacciones sobre la base de datos del curso, de como logra PostgreSQL solventar los fenómenos no deseados que se pueden presentar en transacciones concurrentes.

3. Muestra dos transacciones que generen un deadlock utilizando bloqueos a nivel de tabla, explica porque se genera y vuelve a definirlas para que no se presente el deadlock.

4. Se tienen las siguientes dos transacciones en un momento dado:

T0:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
UPDATE cliente SET ciudad_cliente = 'Nezahualcoyotl'
WHERE nombre_cliente = 'Rene';
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
SAVEPOINT punto_uno;
UPDATE cliente SET ciudad_cliente = 'Ecatepec'
WHERE nombre_cliente = 'Rene'
AND ciudad_cliente = 'Nezahualcoyotl';
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
AND ciudad_cliente = 'Ecatepec';
ROLLBACK punto_uno;
COMMIT;
```

T1:

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
UPDATE cliente SET ciudad_cliente = 'Nezahualcoyotl'
WHERE nombre_cliente = 'Rene';
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
SAVEPOINT punto_uno;
UPDATE cliente SET ciudad_cliente = 'Ecatepec'
WHERE nombre_cliente = 'Rene'
AND ciudad_cliente = 'Nezahualcoyotl';
SELECT id_cliente, nombre_cliente, ap_pat_cliente, pais_cliente,
ciudad_cliente
```

```
FROM cliente WHERE pais_cliente = 'Mexico'
AND nombre_cliente = 'Rene';
AND ciudad_cliente = 'Ecatepec';
ROLLBACK punto_uno;
COMMIT;
```

Si se ejecutan concurrentemente. ¿Qué sucede? Explica a detalle.
Si las dos tuviesen un nivel de aislamiento `SERIALIZABLE`, sucede lo mismo?

5. El bloqueo a nivel de tabla es el único que garantiza una ejecución REALMENTE concurrente y serializable. Sin embargo, ¿qué problema tienen las aplicaciones que adquieren bloqueos a nivel de tabla cuando se conectan? Ejemplifica.

6. Modifica el parámetro 'deadlock_timeout' dentro del archivo 'postgresql.conf' con un valor de 35000 y reinicia el servidor. Ejecuta nuevamente las transacciones del ejercicio 4. ¿Se pueden ejecutar las transacciones? Explica porque si o porque no. En caso afirmativo, ¿qué beneficios trae esto consigo?

Recuperación

Semana:	Décimo segunda semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de SQL. Conceptos de recuperación.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Observar los procesos de recuperación y determinar en los distintos tipo de memoria.

La práctica dieciocho corresponde al tema de recuperación. Dentro de esta práctica, se muestra la importancia que tiene el proceso de recuperación dentro de un SMBD y la forma en la cual se debe optimizar los mecanismos de recuperación para minimizar el impacto de éstos en el rendimiento general del sistema.

PRÁCTICA 18

Recuperación

18.1. Meta

Que el alumno comprenda la importancia que tiene el proceso de recuperación dentro de un SMBD y como afecta al rendimiento del mismo.

18.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar la importancia del proceso de recuperación.
- Identificar el proceso de recuperación implementado en PostgreSQL.
- Configurar PostgreSQL para que se desempeñe mejor dependiendo de sus necesidades.

18.3. Desarrollo

En un sistema, se pueden dar fallas que pongan en riesgo la integridad y la existencia misma de la base y por lo tanto de los datos. Alguno de ellos son:

- Fallas en el hardware (fallas en disco duro, procesador, etcétera).
- Error lógicos (errores en la transacción, división entre cero, etcétera).
- Errores de memoria del sistema operativo anfitrión.

- Excepciones sin programar (los datos necesarios para realizar la transacción no se encuentran).
- Errores del control de concurrencia (generación de 'deadlocks').
- Falla en la transmisión de datos sobre una base de datos distribuida.
- Catástrofes (incendios, robo, etc).

En cualquier caso, el SMBD debe ser capaz de recuperar un estado consistente y conocido de la base.

18.3.1. Técnicas de recuperación

Para soslayar las fallas que se presentan a nivel lógico y de dispositivos, se han elaborado distintas técnicas, que en menor o mayor grado afectan el rendimiento del sistema de manera general. La idea básica detrás de cada una de las técnicas mencionadas es generar siempre una copia de respaldo para garantizar que no se pierde la información que generan las transacciones en un momento dado.

Las principales técnicas son:

- Recuperación basada en Paginación Sombra ('Shadow Paging').
- Recuperación basada en bitácoras ('logs').
- Recuperación basada en actualización diferida.
- Recuperación basada en actualización inmediata.
- Escritura adelantada de la bitácora ('Write-Ahead Logging - WAL').

18.3.2. PostgreSQL y WAL

En PostgreSQL la escritura adelantada de la bitácora se encuentra automáticamente habilitada, así que no es necesaria la intervención del administrador, excepto para asegurar que existe espacio suficiente para almacenar las bitácoras. Estás bitácoras se almacenan como segmentos de archivos normalmente cada uno es de 16 MB, y cada segmento esta dividido en páginas de 8KB cada una. Los buffers y las estructuras de control se encuentran en áreas de memoria compartida que son protegidas por bloqueos. La demanda de esta memoria compartida depende del número de buffers, la configuración por omisión maneja 8 buffers de 8KB cada uno (para el total de 64 KB).

Los puntos de verificación son registros dentro de la bitácora que garantizan que los datos modificados por la secuencia de transacciones anteriores a ellos han sido actualizados en los archivos de datos. En cuanto se da un punto de verificación, todas

las páginas sucias[1] se escriben a disco y un registro especial se escribe en el archivo de la misma. Como resultado, en el evento de una falla, el procedimiento de recuperación determina a partir de que registro de la bitácora (registro 'REDO') debe empezar con las operaciones 'Rehacer', dado que cualquier cambio realizado a los archivos de datos antes que ese registro REDO ya se encuentra en el disco. Una vez escrito el punto de verificación, cualquier segmento de la bitácora escrito en forma anterior al registro 'REDO' no es necesario y se puede eliminar.

El SMBD se encarga de realizar los puntos de verificación según los siguientes 2 parámetros[2] :

1. '*checkpoint_segments*'.- Determina cada cuantos segmentos creados de la bitácora se necesitan tener para realizar un punto de verificación.

2. '*checkpoint_timeout*'.- Determina el tiempo máximo que el sistema deja pasar antes de realizar un nuevo punto de verificación.

El sistema ejecuta el proceso de creación de un punto de verificación según lo primero que ocurra. Los valores por omisión que tienen estos parámetros son de 3 segmentos y de 300 segundos respectivamente. Opcionalmente existe la sentencia SQL de `CHECKPOINT` para forzar un punto de verificación.

[1] Páginas que tienen un bit marcando que no se han escrito a disco.
[2] Se encuentran el archivo 'pg_conf' dentro del directorio `$PGDATA`

18.4. Ejercicios

1. Considera las siguientes transacciones:
   T0:
   ```
   UPDATE cliente SET ciudad_cliente = 'Distrito Federal'
   WHERE pais_cliente = 'Mexico' AND nombre_cliente LIKE 'Re%';
   ```

   T1:
   ```
   BEGIN TRANSACTION;
   INSERT INTO cliente SELECT nextval('seq_id_cliente'), nombre_cliente,
   ap_pat_cliente, ap_mat_cliente, pais_cliente, calle_cliente,
   colonia_cliente, numero_dir_cliente, codigo_postal_cliente,
   telefono_cliente, ciudad_cliente FROM cliente LIMIT 25000;
   COMMIT;
   ```

   Ejecuta ambas transacciones simultáneamente y toma el tiempo que toma en completarse.
   ¿Se marca alguna advertencia durante la ejecución de esto?

2. Modifica los parámetros para la creación de un punto de verificación. Decrementa 'checkpoint_segments' a 1 y 'checkpoint_timeout' a 5. Vuelve a ejecutar las transacciones anteriores[3] .
   Ejecuta ambas transacciones simultáneamente y toma el tiempo que toma en completarse[4] .
   ¿Se marca alguna advertencia durante la ejecución de esto?

3. Regresa los valores originales de los parámetros modificados en el ejercicio anterior. Obten los siguientes valores:

   a) Total de clientes que viven en 'México' en la ciudad 'Distrito Federal';

   b) Total de clientes que viven en 'Estados Unidos' en la ciudad 'New York';

   c) Total de clientes que viven en 'Colombia' en la ciudad de 'Bogota';

   Ahora ejecuta las siguientes transacciones:
   T0:
   ```
   BEGIN TRANSACTION;
   UPDATE cliente SET ciudad_cliente = 'Distrito Federal'
   ```

[3] Únicamente modifica el valor de la nueva cadena de *T0*, intenta con `SET ciudad_cliente = 'Guadalajara'`.

[4] No olvides actualizar el valor de la secuencia para no obtener errores de duplicación en la llave primaria.

```
WHERE pais_cliente = 'Mexico';
UPDATE cliente SET ciudad_cliente = 'New York'
WHERE pais_cliente = 'Estados Unidos';
UPDATE cliente SET ciudad_cliente = 'Bogota'
WHERE pais_cliente = 'Colombia';
COMMIT;
```

$T1$:

```
BEGIN TRANSACTION;
INSERT INTO cliente SELECT nextval('seq_id_cliente'),
nombre_cliente, ap_pat_cliente, ap_mat_cliente, 'Colombia', calle_
cliente, colonia_cliente, numero_dir_cliente, codigo_postal_cliente,
telefono_cliente, ciudad_cliente FROM cliente LIMIT 25000;
COMMIT;
```

Deja pasar 25 segundos y como usuario 'root' termina el proceso[5] que corresponde a la transacción $T0$ e inmediatamente después el correspondiente a la transacción $T1$.
Ejecuta nuevamente los conteos de los clientes. ¿Se llevaron a cabo las actualizaciones?
Ejecuta nuevamente las transacciones y toma sus tiempos de ejecución.

4. Modifica los parámetros tal cual lo hiciste en el ejercicio 2 y lleva a cabo las actividades del ejercicio 3. ¿Qué puedes comentar sobre el impacto que tienen estos parámentros en el rendimiento del sistema? ¿Qué puedes decir respecto a que valores deben tener cada uno de ellos?

[5] 'ps aux | grep usuario' para ver el PID del proceso en cuestión.

Modificación de recursos utiliados por el SMBD

Semana:	Décimo tercera semana.
Tiempo de entrega:	Una semana.
Prerrequisitos:	Conceptos de SQL. Conceptos de escalamiento.
Herramientas:	PostgreSQL v8.0.2
Objetivo:	Observar las ventajas obtenidas al aumentar el procesamiento de computo donde se encuentra. un SMBD.

La práctica número 19 reafirmará la idea intuitiva que se tiene sobre el escalamiento de los recursos con los cuales cuenta un sistema computacional y como afecta en el rendimiento general de éste. Es importante contar con una manera práctica de solventar los problemas inherentes a la modificación del hardware disponible para la práctica.

PRÁCTICA 19

Modificación de recursos utiliados por el SMBD

19.1. Meta

Que el alumno aprecie de manera cuantitativa las mejoras obtenidas en un SMBD al aumentar los recursos de hardware donde se encuentra el SMBD.

19.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar las mejoras obtenidas por el aumento de recursos en el equipo de cómputo.
- Implementar las mejores opciones de escalamiento dependiendo de las distintas situaciones.

19.3. Desarrollo

En general, como en la mayor parte de los sistemas computacionales, se espera que al aumentar los recursos disponibles en un sistema, su desempeño general mejore significativamente. La intuición natural indica que en los casos en que se dupliquen los recursos la mejora substancial sera del doble, si se triplican los recursos la mejora substancial será del triple y así sucesivamente. Pero esto no es cierto ya que influyen muchos factores en ello (por ejemplo la arquitectura del sistema de cómputo, entre muchos otros aspectos). Existen principalmente dos medidas de desempeño en una base de datos:

1. Rendimiento ('*throughput*') .- Métrica que toma el número de tareas que se pueden completar en un tiempo dado.

2. Tiempo de respuesta ('*response time*') .- Métrica que toma el tiempo que tarda el realizarse una tarea dada.

Un sistema que puede procesar gran cantidad de pequeñas tareas puede mejorar el rendimiento al procesar muchas tareas en paralelo, mientras que un sistema que procese tareas grandes puede mejorar el tiempo de respuesta al realizar porciones de estas tareas en paralelo.

Actividad 19.1

En general el aumento de recursos se hace sobre cada componente del sistema, pero es de particular atención el aumento en el número de procesadores (UCP) ya que permite manejar '*paralelismo*' dentro del sistema operativo y en particular dentro del SMBD.

Investiga si PostgreSQL soporta el manejo de paralelismo y como depende del sistema operativo para ello.

19.3.1. *Escalamiento y Aceleración*

Dos aspectos importantes surgen durante el análisis en el aumento de recursos en un sistema de cómputo. Estos son:

- Escalamiento ('*Scaleup*').- Se refiere a ejecutar más tareas en el mismo tiempo al aumentar los recursos, se relaciona con el rendimiento.

- Aceleración ('*Speedup*').- Se refiere a ejecutar una tarea dada en menos tiempo aumentado los recursos, esta relacionado con el tiempo de respuesta.

Actividad 19.2

Investiga las capacidades del sistema donde se encuentra ejecutándose actualmente PostgreSQL.
Realiza un registro detallado de las características encontradas en el sistema e investiga las posibles modificaciones a las que es susceptible el sistema.

19.3.2. Reporte de resultados de pruebas

El objetivo de un reporte de resultados de pruebas, es generar las condiciones en las cuales se han llevado a cabo los experimentos y poder reproducirlos a fin de obtener los mismos resultados, sino se cuenta con esta información entonces se encuentra con la presencia de información poco confiable.

La información contenida en reporte de pruebas tendrá que abarcar los dos rubros, tanto el software como el hardware (puesto que este último es el que se aumenta) y verificar la comparativa en cuanto a desempeño. Dentro de los datos más importantes están:

- Hardware:
  1. Procesadores (Tipo, versión, caches, cantidad y velocidad).
  2. Discos duros (Interfaz, capacidad, cantidad y velocidad).
  3. Memoria RAM (Tipo, cantidad y velocidad).
  4. Conjunto de chips de la tarjeta madre (Tipo, versión y velocidad).
- Software:
  1. Sistema operativo (Nombre y versión).
  2. Sistema Manejador de Bases de Datos (Nombre y versión)
  3. Configuración general (Tamaño de memoria virtual asignado, disposición del almacenamiento, nivel RAID implementado, sistemas de archivos, etcétera).

Actividad 19.3

Ejecuta la siguiente consulta en un equipo que cuente con sólo 64 MB en memoria RAM:

"Obten los nombres y apellidos de los clientes que viven en un país cuya inicial es 'M', tienen un código postal entre '03400' y '57100' y han comprado todos los videojuegos de la plataforma 'PSP' del tipo 'RPG'". El resultado debe encontrarse ordenado por el apellido paterno.

Mide el tiempo que tarda en ejecutarse la consulta, sin modificar parámetros.

Aumenta la memoria a 128 MB y repite la consulta.

Vuelve a tomar el tiempo de ejecución
¿Cúal ha sido la aceleración obtenida?
¿Qué factores influyen en el resultado?

19.4. Ejercicios

Genera un reporte de las siguientes pruebas, incluye en él todo lo mencionado en la sección 19.3.2. Calcula la aceleración obtenida con cada mejora (suponiendo que el aumento de recursos es el doble para cada recurso modificado) y si no se presenta una aceleración linear explica las causas que lo evitan.

Para esto ejecuta las siguientes consultas en la base de datos *cursoSMBD*, sobre un equipo con las siguientes características:

Procesador	Velocidad	Memoria RAM	Disco Duro
1	X	64 MB	ATA
1	X	128 MB	ATA
1	X	256 MB	ATA
1	X	512 MB	ATA
1	2X	64 MB	ATA
1	2X	128 MB	ATA
1	2X	256 MB	ATA
1	2X	512 MB	ATA
2	Y	Z	SCSI

1. El nombre y apellidos de todos los clientes cuyos números telefónicos terminan en numero par.
2. El nombre y plataforma de los videojuegos más vendidos en el año de 2004.
3. La cantidad de nombres distintos existentes entre los videojuegos.
4. Los nombres y plataformas de los videojuegos que no han sido comprados.

Ahora considera las siguientes transacciones y elabora el reporte equivalente pero tomando en cuenta el escalamiento obtenido en cada caso:

1. $T1$:
   - Actualiza todas las ciudades de los clientes que viven en Bélgica y han comprado un videojuego para la consola PS2 durante el año de 1999.
   - Actualiza todos los códigos postales de los clientes que han realizado alguna compra en el mes de Enero (mes 1), aumentando el valor un 10 %.
   - Elimina todos los clientes cuyo '*id_cliente*' sea múltiplo de 5000.
2. $T2$:
   - Elimina las ventas realizadas tales que el '*id_cliente*' sea el mismo que el '*id_videojuego*', para clientes que viven en Rusia.

- Deshaga las modificaciones anteriores.
- Elimina las ventas realizadas tales que el '*id_cliente*' sea el mismo que el '*id_videojuego*', para clientes que viven en China.
- Aumente el precio del videojuegos más vendido en un 25 %.

3. *T*3:

- Elimine todos los clientes cuyo apellido paterno es igual al apellido materno.
- Elimine todos los videojuegos que se han vendido menos de 200 veces durante 2001.
- Elimine todos los videojuegos que se han vendido menos de 200 veces durante 2002.
- Deshaga las anteriores operaciones.
- Disminuya en un 25 % el precio del videjuego menos vendido durante 2001.
- Disminuya en un 25 % el precio del videjuego menos vendido durante 2002.

En esta práctica es importante minimizar el efecto producido por el resto de los procesos presentes en el sistema operativo, si te es posible entra como super-usuario y elimina los procesos no necesarios durante la práctica (por ejemplo intenta llevar a cabo las pruebas en un ambiente que no gráfico). Por último considera NO modificar los parámetros de configuración que se tienen por omisión en PostgreSQL[1] .

[1] Por ejemplo '*enable_mergejoin*', '*checkpoint_segments*' etcétera.

Datawarehouse[2]

Semana:	Décimo cuarta semana.
Tiempo de entrega:	Dos semanas.
Prerrequisitos:	Conceptos de SQL. Conceptos de Datawarehouse. Conceptos de OLTP y OLAP.
Herramientas:	Microsoft Windows© 2000 Server ó Microsoft Windows© XP Professional Microsoft SQL Server™ 2000 Evaluation Edition (versión de prueba 120 días)
Objetivo:	Comprender los principios del Datawarehouse y su implementación en un SMBD comercial.

La práctica número veinte es la práctica optativa relativa al segundo curso. Introduce al alumno en el tema de '*datawarehouse*' y las consideraciones relativas al mismo y permitiéndole apreciar la importancia de esto en el desarrollo de aplicaciones empresariales.

Es importante contar con bases solidas en el tema para no dilatar su desarrollo y terminar en los tiempos estipulados. Así como el contar con los recursos de equipo suficientes para obtener los resultados esperados.

[2] El término correcto es 'almacén de datos', pero dada la aceptación del término en inglés, se utiliza este último durante el presente trabajo.

PRÁCTICA 20

Datawarehouse

20.1. Meta

Que el alumno aprecie la importancia que tienen el datawarehouse en el desarrollo de sistemas empresariales, así como su implementación en un SMBD comercial.

20.2. Objetivos

Al finalizar esta práctica el alumno será capaz de:

- Identificar las ventajas que tiene un datawarehouse.
- Conocer las principales características de un datawarehouse.
- Identificar los problemas que resuelven los datawarehouse.
- Implementar un datawarehouse relativo a la base de datos del curso.

20.3. Desarrollo

Los sistemas para el apoyo en la toma de decisiones ('*Decision Support Systems*' DDS), son sistemas que brindan facilidades para el análisis de información en los negocios. La idea básica de estos sistemas es recolectar datos operacionales de la organización (o empresa) en cuestión y reducirlos en estructuras que faciliten el análisis del comportamiento de los estados operacionales de la organización y con ello le permitan modificar esos estados de una manera más eficiente.

Las bases de datos para el apoyo en la toma de decisiones poseen muchas características, pero la más importante consiste en que la base de datos es únicamente de lectura, aunque las actualizaciones pueden presentarse en cantidad limitada. Otras características importantes de mencionar son:

- Las columnas de las tablas tienden a ser utilizadas combinándose entre si.
- La integridad, en general, no es un problema que se aborde en ellas, ya que se asumen datos 'limpios', es decir datos que han pasado por un proceso de depuración.
- Existen componentes temporales (columnas que hacen referencia a fechas).
- Las bases de datos tienden a ser grandes, especialmente cuando las transacciones del negocio se acumulan durante el tiempo.
- Las bases de datos tienden a contener una cantidad considerable de índices .
- Las bases de datos contienen múltiples controles para la redundancia que maneja.

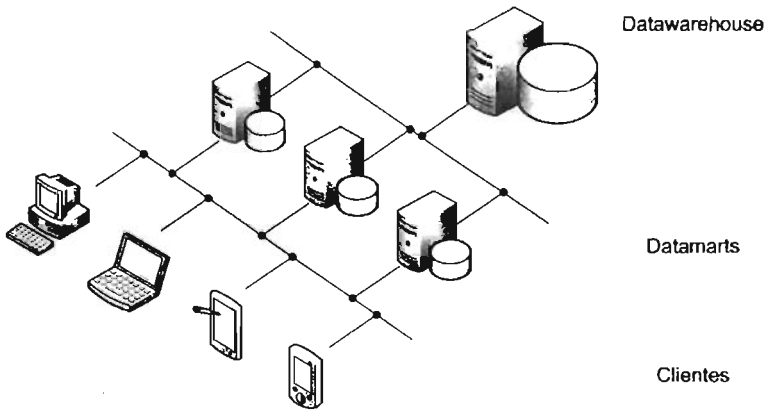

Figura 20.1: Esquema general de los DSS.

20.3.1. OLAP y OLTP

El software de administración de base de datos relacional a nivel corporativo, se diseñó originalmente para almacenar de manera centralizada los datos generados por las transacciones diarias de grandes empresas u organizaciones gubernamentales. Durante décadas, estas bases de datos han crecido para ser sistemas altamente eficaces para grabar los datos que se requieren para realizar las tareas diarias de la empresa. Ya que

el sistema se basa en equipos y registra las transacciones relativas a la organización de la empresa, estos sistemas se conocen como sistemas de procesamiento de transacciones en línea (OLTP, '*Online Transaction Processing*').

Sistemas OLTP Los datos en los sistemas OLTP están organizados básicamente para admitir transacciones. Las transacciones individuales se completan rápidamente y se tiene acceso a cantidades de datos relativamente pequeñas. Los sistemas OLTP están diseñados y ajustados para procesar cientos o miles de transacciones que se introducen al mismo tiempo.

Aunque los sistemas OLTP sobresalen en el proceso de registro de los datos necesarios para admitir operaciones diarias, los datos OLTP no están organizados de una manera que proporcione fácilmente la información requerida por los administradores para planear el trabajo de sus organizaciones. Los administradores necesitan información resumida a partir de la cual puedan analizar tendencias que afecten a su organización o equipo.

Sistemas OLAP Los sistemas diseñados para controlar las consultas requeridas para descubrir tendencias y factores críticos se denominan sistemas de procesamiento analítico en línea (OLAP). Las consultas OLAP normalmente requieren grandes cantidades de datos y este tipo de consultas en los sistemas OLTP tiene múltiples efectos, principalmente el que la consulta tarde mucho tiempo en calcular las respectivas agregaciones necesarias[1] puesto que la información no está preparada para un tratamiento cronológico y la consulta genera una carga de trabajo muy intensa que ralentiza el registro de transacciones de los usuarios habituales del sistema.

Aunado a este hecho, es factible que los diversos sistemas OLTP se encuentren en equipos con software y hardware diferentes y en muchos casos, los datos que se utilizan para identificar elementos en un sistema son distintos entre cada equipo. Los administradores que ejecutan consultas OLAP normalmente necesitan poder hacer referencia a datos de varios de estos sistemas OLTP.

Los datos OLAP están organizados en cubos de múltiples dimensiones, así la estructura de éstos ofrece un mejor rendimiento para las consultas OLAP que los datos organizados en tablas relacionales. La unidad básica de un cubo de múltiples dimensiones se llama medida y son las unidades de datos que se van a analizar. Las medidas están organizadas en dimensiones y éstas dimensiones forman los ejes lógicos del cubo virtual.

[1] Operaciones sobre conjuntos.

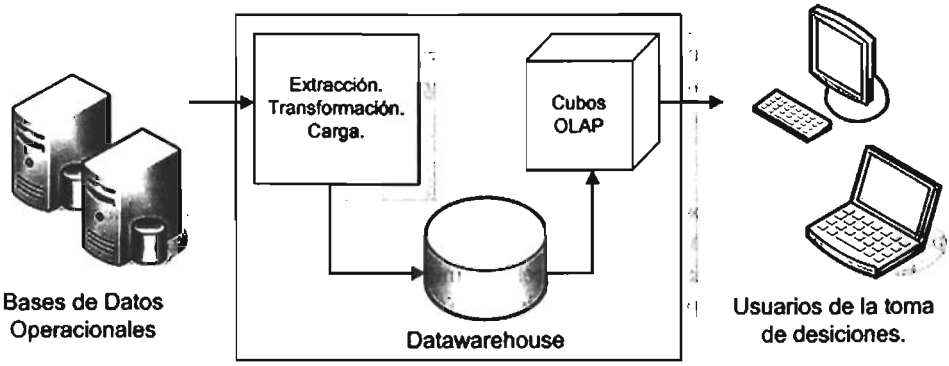

Figura 20.2: Conformación de los sistemas OLAP.

Actividad 20.1

Genera un esquema para crear el datawarehouse relativo a la base de datos del curso y describe las relaciones existentes entre los componentes del mismo.

Para ello considera las consultas que puede recibir el datawarehouse.

20.4. Microsoft© SQL Server™ 2000

Desafortunadamente, PostgreSQL no cuenta con una implementación que maneje los cubos de múltiples dimensiones ni herramientas o esquemas que permitan definir un datawarehouse a partir de fuentes heterogéneas. Por esto se ha decidido utilizar una opción comercial dentro de los SMBD.

SQL Server™ 2000 es el SMBD perteneciente a Microsoft©. Es comercial y presenta características que lo hacen idóneo para el ámbito académico en general y en particular para el presente trabajo.

Las razones de elección de este SMBD con respecto a otras soluciones en el mercado son las siguientes:

- Implementación de características avanzadas (OLAP, cubos de múltiples dimensiones, minería de datos, aceptación de distintas fuentes de datos, etc).
- Disponibilidad de una versión de evaluación de 120 días.

- Suficiente documentación[2] .
- Alta aceptación en el ámbito del desarrollo de software.
- Menor cantidad de recursos utilizados.
- Amplia variedad de herramientas en la creación y automatización de procesos administrativos.

Los detalles sobre su instalación se presentan en el ápendice E.

20.4.1. Creación de cubos de múltiples dimensiones

Un cubo es un conjunto de datos que se construyen de un subconjunto de datos contenidos en un DW, se encuentran organizados y resumidos generalmente en una estructura de múltiples dimensiones definida por un conjunto de dimensiones y de medidas, además proporciona un mecanismo para consultar datos de una manera rápida y uniforme. Los datos se encuentran resumidos y precalculados con anterioridad en lo que se conoce como agregaciones[3] y permiten tiempos de respuesta uniformes y generalmente menores a las consultas realizadas sobre la información contenida en los cubos. De hecho las agregaciones son creadas para un cubo antes de que el usuario acceda a ellos.

Cada cubo posee un esquema, que es el conjunto de tablas (o reuniones de éstas) provenientes del DW de donde, además, obtiene los datos que lo conforman. Generalmente la tabla central del esquema es una tabla de hechos (la fuente de las medidas del cubo). Las otras tablas son las tablas de dimensión, que son las fuentes de las dimensiones del cubo. Un cubo es definido por las medidas y las dimensiones que contiene y cada dimensión del cubo puede contener una jerarquía de niveles para especificar la granularidad categórica disponible para realizar consultas. Esto mismo define que cantidad de consultas pueden tener respuesta dado el diseño original que se ha definido.

Los niveles de la dimensión representan una herramienta poderosa para los usuarios finales ya que les permiten realizar consultas generales y posteriormente refinar sus consultas para revelar detalles que no aparecen en primera instancia. Este tipo de exploración, conocido como '*drill-down*', es común.

La creación de un cubo involucra tres pasos básicos:

- **Definición**. La definición de un cubo se basa en los requerimientos de análisis de los usuarios finales, seleccionar una tabla de hechos e identificar las medidas dentro de la tabla de hechos. Posteriormente seleccionar o crear la dimensiones, cada una compuesta de una o más columnas de otra tabla. En SQL Server™ se cuenta con la herramienta 'Asistente para la cubos' que permite definirlos rápidamente. Además

[2] Disponible en la siguiente dirección electrónica: `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnanchor/html/sqlserver2000.asp`.

[3] Tabla o estructura que contiene datos calculados con anterioridad para un cubo determinado.

se incluye un editor de cubos que permite definir y modificar la estructura de los cubos creados con anterioridad (figura 20.3 y figura 20.4).

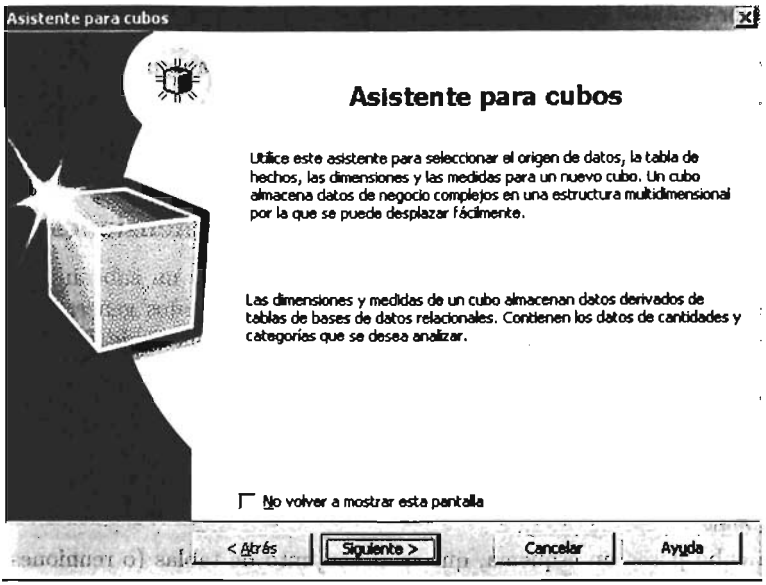

Figura 20.3: Asistente para la creación de cubos en SQL Server™ 2000.

- **Diseño de agregaciones**. Después de definir un cubo, es posible definir las agregaciones utilizando el 'Asistente para almacenamiento y agregado de datos' (figura 20.5). El diseñar las agregaciones consiste en especificar la estrategia que se utiliza para resumir la información.

- **Procesamiento**. Posterior al diseño de las agregaciones, se debe calcular cada una de ellas para crearlas propiamente.

Si luego de procesar un cubo, este se modifica o su fuente de datos cambia, es usualmente necesario el procesar el cubo nuevamente. Se presentan diferentes opciones de procesamiento cada una adecuada a distintas circunstancias[4] .

[4] Para mayor información sobre estas opciones de procesamiento para cubos, hay que consultar la información contenida en SQL Server™(**"Inicio → Microsoft SQL Server → Libros en pantalla → Procesar cubos"**).

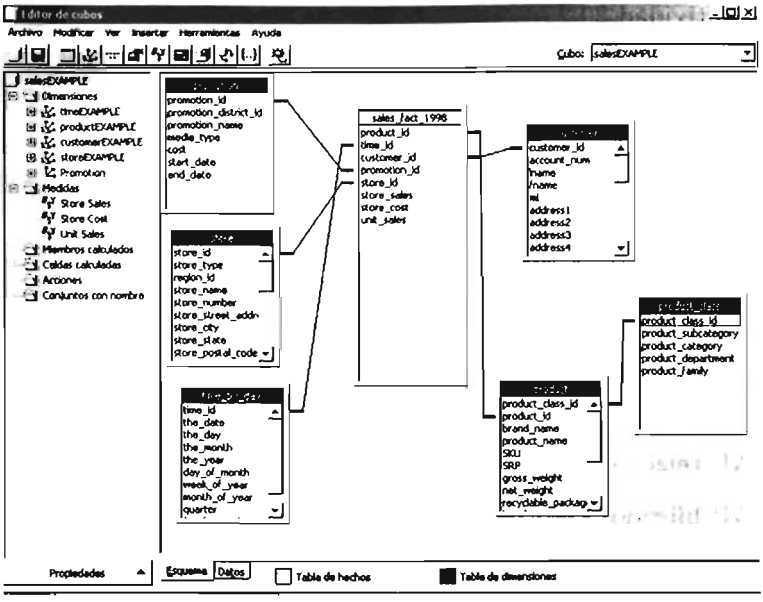

Figura 20.4: Editor de cubos en SQL Server™ 2000.

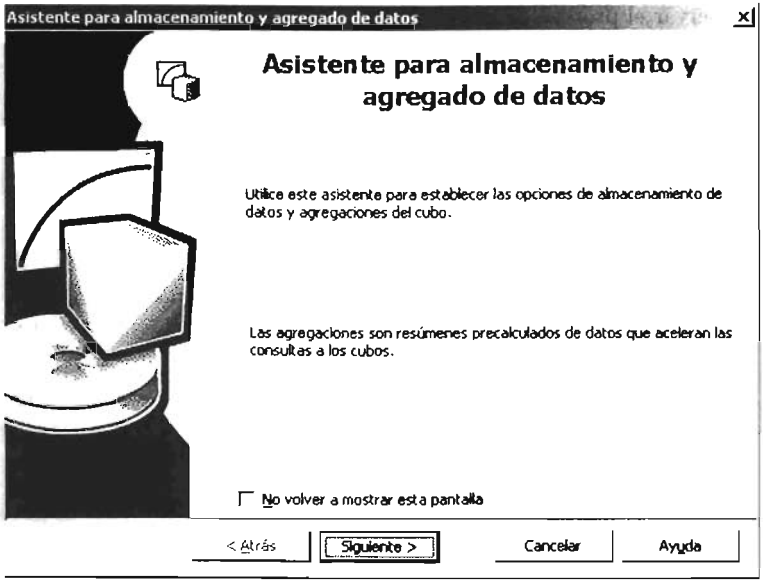

Figura 20.5: Asistente de almacenamiento de SQL Server™ 2000.

Actividad 20.2

Tomando en principio el esquema de la base de datos del curso, genera un esquema de estrella y un esquema de copo de nieve.
¿Cuál es más apropiado para la base de datos del curso? ¿Por qué ?

20.4.2. Tipos de almacenamiento

Existen distintos tipos de almacenamiento para los cubos de múltiples dimensiones, estos afectan el rendimiento, los requerimientos de almacenamiento, las relaciones entre los mismos cubos que se generan así como la ubicación de su almacenamiento. Los tipos de almacenamiento que se soportan en SQL Server™ son :

- **OLAP de múltiples dimensiones (MOLAP).**
- **OLAP relacionales (ROLAP).**
- **OLAP híbridos (HOLAP).**

Actividad 20.3

Para observar el proceso de creación completo dentro de este SMBD, realiza el tutorial incluido en él.

Para visualizar el resultado de los cubos creados no olvides utilizar el explorador de datos de cubos de múltiples dimensiones.

Durante la utilización del 'Asistente para almacenamiento y agregado de datos' es donde se elige el modo más apropiado para cada partición o conjunto de cubos. Adicionalmente es posible utilizar el 'Asistente de optimización basada en costo' para elegir el modo de almacenamiento y optimizar las agregaciones referentes a los cubos.

Actividad 20.4

Considera los tres modos de almacenamiento. ¿Cuál es mejor?
Presenta tres escenarios donde cada modo de almacenamiento se desempeñe mejor.

Para cada uno de los esquemas realizados en la actividad 1, decide que modo de almacenamiento es más conveniente y cuales son las razones de ello.

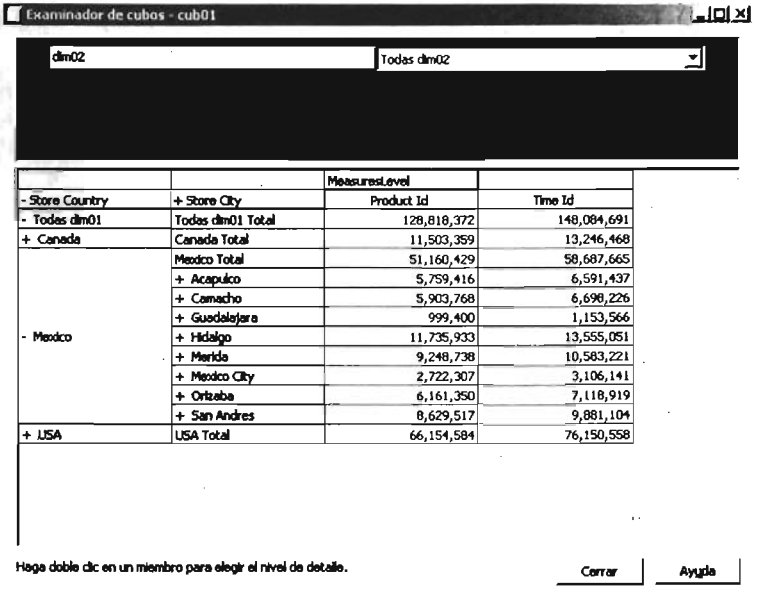

Figura 20.6: El examinador de cubos de SQL Server™ 2000.

20.4.3. Visor de cubos de múltiples dimensiones

Una vez diseñado y procesado cada cubo, es posible observar el resultado de este procesamiento al utilizar el 'Examinador de cubos' (figura 20.6).

El examinador permite observar de manera resumida, el resultado de las agregaciones calculadas y con ello mantener una vista general sobre la información almacenada dentro de las bases de datos operacionales. Sin embargo no es la única herramienta con la cual se cuenta para explotar esta información. Existe un lenguaje que permite la interacción con éstas estructuras y se conoce como MDX.

MDX

El lenguaje de expresiones de múltiples dimensiones (MDX) se utiliza para manipular información estructurada en múltiples dimensiones en los Servicios de Análisis de Microsoft© SQL Server™ 2000. MDX está definido en las extensiones OLAP de OLE DB. De manera similar a SQL en muchos aspectos, MDX proporciona una sintaxis para la recuperación y manipulación de datos de múltiples dimensiones, más no es una extensión del lenguaje SQL; de hecho, SQL puede ofrecer parte de la funcionalidad que ofrece MDX, aunque no con tanta eficacia ni de manera tan intuitiva.

Al igual que con una consulta SQL, cada consulta MDX requiere una solicitud de datos (la cláusula `SELECT`), un punto de inicio (la cláusula `FROM`) y un filtro (la

cláusula WHERE). Éstas y otras palabras clave proporcionan las herramientas utilizadas para extraer partes específicas de datos de un cubo para su análisis. MDX proporciona también un sólido conjunto de funciones para la manipulación de datos recuperados, así como la capacidad de ampliar MDX con funciones definidas por el usuario.

MDX, al igual que SQL, proporciona sintaxis del lenguaje de definición de datos (DDL) para administrar estructuras de datos. Hay comandos MDX para crear y eliminar cubos, dimensiones, medidas y sus objetos subordinados.

CUBE y ROLL UP de SQL

Aún cuando los servicios OLAP de SQL Server™ 2000 brindan estos beneficios, es posible obtener este tipo de información al utilizar únicamente sentencias SQL (estándar SQL92 y posteriores [9]) con las operaciones de 'partir' y 'rebanar' [5] . SQL Server™ 2000 incluye las extensiones CUBE y ROLL UP de SQL que permiten generar agregaciones de agrupaciones en una sola consulta. Como se ha mostrado en el presente trabajo, es posible resumir los datos al utilizar la cláusula GROUP BY con alguna de las funciones de agregación que se han descrito anteriormente (SUM, AVG, MIN, MAX o COUNT), pero en ciertas ocasiones es necesario presentar la información en resumenes que muestren toda la información posible aún cuando no se cumplan ciertas condiciones.

Por ejemplo considerando la consulta "¿Cuáles son las ventas por año de cada plataforma y de cada tipo de videojuego ?" es posible obtener resultados parciales al realizar consultas que agrupen los resultados por año, posteriormente por plataforma y por último calcular la cantidad en cuestión (ver figura 20.7), además de otra que agrupe los resultados por año, posteriormente por tipo y por último que considere la cantidad de cada uno (ver figura 20.8).

```
SELECT año, consola, count(id_videojuego) AS cantidad
FROM compra NATURAL JOIN videojuego
GROUP BY consola, año;
```

Figura 20.7: Sentencia SQL - Información resumida respecto a consola.

```
SELECT año, tipo, count(id_videojuego) AS cantidad
FROM compra NATURAL JOIN videojuego
GROUP BY tipo, año;
```

Figura 20.8: Sentencia SQL - Información resumida respecto a tipo.

Pero no siempre es una solución adecuada, dado que se pueden presentar casos en los cuales no se muestre toda la información, por ejemplo que no se haya vendido alguna unidad de un videojuego particular en un año tendra como efecto el que no aparezca

[5] Conocidas como '*dice*' y '*slice*' respectivamente.

esa información en el resumen final. Para solucionar esto, se cuenta con la cláusula `CUBE`, que permite crear nuevas dimensiones en las consultas. Cuando se utiliza esta cláusula es como si estuviésemos haciendo a la vez todos los `GROUP BY` posibles y además mostrándolos en un único resultado. La sintaxis de la cláusula `SELECT` en SQL Server es amplia[6] pero la utilización de `CUBE` se realiza al agregar la cláusula en una consulta que realice agrupaciones sobre los datos.

Considérese la siguiente vista: *resumenVentas[año, id_videojuego, tipo, consola, precio]*, que almacena la información relevante de las ventas a través de los años. Si se desea conocer el total de ventas por consola y tipo que se han tenido durante los años, es posible obtener parte de la información con la siguiente consulta[7] :

```
SELECT año, consola, tipo, COUNT(id_videojuego) Cantidad
FROM dbo.resumenVentas
GROUP BY tipo, consola, año;
```

La información presentada no es la más completa para presentar un informe detallado, ya que se muestran datos que aún necesitan ser procesados para obtener valores totales (por ejemplo el total de ventas por tipo de videojuego en todos los años). Es aquí donde se procede a agregar la cláusula `WITH CUBE`.

```
SELECT año, consola, tipo, COUNT(id_videojuego) Cantidad
FROM dbo.resumenVentas
GROUP BY tipo, año,consola
WITH CUBE
ORDER BY año DESC, consola DESC, tipo DESC;
```

Con ella, se indica al SMBD, que calcule todas las agregaciones posibles dentro de los datos almacenados.

Actividad 20.5

Abre una sesión en el "Analizador de consultas SQL" dentro de la base de datos del curso y ejecuta las sentencias SQL anteriores. Para ello, primero crea la vista 'resumenVentas'.

Al utilizar la sentencia `WITH CUBE` aparecen varias marcas `NULL`
¿A qué se refiere cada una de ellas?

[6] Ver `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_sa-ses_9sfo.asp`

[7] Es necesario recordar que la sintaxis siguiente se refiere a Transact-SQL, el dialecto SQL propio de SQL Server™ 2000.

Se observa que aparecen varias marcas NULL por el medio de la tabla. Ahora cada fila es una de las posibles combinaciones de tipo con consola, y las filas que contienen una marca NULL se tienen que leer pensando que donde está NULL debería poner 'todas'. Es decir la tupla:

NULL	PSP	Deportes	22

significa que la cantidad total de videojuegos de 'Deportes' para la plataforma 'PSP' en todos los años ha sido de 22. Y la tupla:

2003	NGC	NULL	9

indica que exiten 9 videojuegos vendidos para la plataforma 'NGC' durante el 2003, por último la tupla

NULL	NULL	NULL	116

indica el total de ventas de todas las plataformas de todos los tipos en todos los años (que es 116). Como se aprecia, la marca NULL representa un super agregado en la columna en la que está colocado. Este tipo de NULL no se debe confundir con un NULL normal. Ya que un NULL normal indica que se desconoce el valor, mientras que este NULL indica una agrupación. Pero, ¿cómo se detecta cual de los NULL es un super agregado y cual es un NULL de 'verdad'? En SQL Server™ 2000 existe una función llamada GROUPING que indica cuando una marca NULL es de verdad y cuando es producto de una agregación. Esta función devuelve el valor de 1 si el nombre de la columna suministrada como parámetro se usa como resumen y un 0 si no es así.

Actividad 20.6

Abre una sesión en el "Analizador de consultas SQL" dentro de la base de datos del curso y ejecuta la siguiente sentencia SQL

```
SELECT año, consola, tipo,
'NULL?'= GROUPING(tipo),
COUNT(id_videojuego) Cantidad
FROM dbo.resumenVentas GROUP BY año, consola, tipo
WITH CUBE;
```

¿A qué se refiere cada columna del resultado?
¿Puedes distinguir los distintos tipos de marcas NULL?

ROLL UP Mientras que `CUBE` genera un conjunto de resultados que muestra agregados para todas las combinaciones de valores de las columnas seleccionadas, `ROLLUP` genera un conjunto de resultados que muestra agregados para una jerarquía de valores de las columnas seleccionadas. Es decir, con `CUBE` aparecen los resultados totalizados por consola, por tipo, y por totales absolutos, mientras que con `ROLL UP` sólo aparecerían los totales agrupados por lo que se indique en la sentencia SQL proporcionada. Se obtiene menos información que con `CUBE` pero de manera más clara. Además muchas veces con esto será suficiente

20.5. Ejercicios

1. A partir de los diseños creados en la actividad 1 de esta práctica, genera los cubos relativos a cada uno de ellos. Recuerda definir tus fuentes de datos para cada uno de ellos.
   ¿Cuál es más apropiado para nuestra base de datos?
   ¿Por qué consideras esto?

2. De tus esquemas anteriores, modifica el tipo de almacenamiento utilizado y genera los cubos relativos para cada tipo de almacenamiento (6 cubos en total).

3. Genera sentencias SQL que utilicen las cláusulas `CUBE` y `ROLL UP` para responder a la pregunta "¿Cuáles son las ventas por año de cada plataforma y de cada tipo de videojuego ?" mostrando el total de posibles combinaciones, de modo que no aparezcan marcas `NULL` de agregaciones para los atributos '*tipo*' y '*consola*' (de la vista *resumenVentas*) y en su lugar aparezca la leyenda 'Todos' y 'Todas' respectivamente (investiga la utilidad de las cláusulas `CASE` y `ISNULL`).

Conclusiones

Dentro de las ciencias de la computación, las bases de datos representan un área de conocimiento bastante amplio y por ello es vital para los egresados de la carrera de Licenciatura en Ciencias de la Computación el poseer bases sólidas en la materia, mismas que le permitan desarrollarse posteriormente en distintos ambientes laborales, ya sean de investigación o no.

El presente compendio de prácticas de laboratorio abarca los temas más importantes dentro del área de las bases de datos y plantea ejercicios prácticos, directos y sencillos que permiten al alumno reforzar sus conocimientos teóricos adquiridos durante las clases de las respectivas materias. La utilidad pedagógica de las prácticas queda demostrada con los conocimientos adquiridos por aquellas generaciones de estudiantes a las cuales se les ha impartido el contenido expuesto en el presente trabajo, sin embargo faltaría llevar a cabo un estudio detallado sobre estás generaciones y posteriores, para ponderar la utilidad real de éstas prácticas. Esto último representa más un trabajo de investigación pedagógico que uno dentro del ámbito de las ciencias de la computación y se aparta del objetivo principal del trabajo expuesto.

Asimismo, el conjunto de prácticas representa un primer intento concreto por conjuntar un desarrollo integral de los alumnos dentro de éstas materias, mas sin embargo no representa un trabajo aislado de otras áreas del conocimiento, dado que los elementos aprendidos por los alumnos son aplicables dentro de diversos contextos, por ejemplo sistemas operativos, lenguajes de programación, compiladores, ingeniería de software, inteligencia artificial y análisis de algoritmos entre otros. Por otro lado, hay que comentar que es indispensable para los alumnos el contar con las instalaciones adecuadas así como garantizar el acceso a ellas a fin de tener a su disposición los elementos que les permitan realizar las prácticas en su totalidad.

Sin embargo, como todo elemento del área de las ciencias de la computación, las bases de datos se encuentran en constante cambio y por ello siempre será necesario actualizar el presente contenido temático, estableciendo nuevos objetivos, metas y ejer-

ciclos que se adapten a los conocimientos y tecnologías emergentes.

Por último, es preciso señalar que si bien el plan de estudios actual muestra solamente a la primera materia como obligatoria, la experiencia adquirida durante la aplicación de estas prácticas y cursos, demuestra que es necesario contar con los dos cursos para obtener una formación medianamente completa dado lo extenso de los temas y el tiempo con el que se cuenta para abordarlos.

Todo ello aunado con un tercer curso optativo, donde se exponga ampliamente la utilización de las bases de datos dentro del proceso de desarrollo de aplicaciones empresariales, sus ventajas y beneficios, y se presenten trabajos de investigación recientes en el área, sería idóneo puesto que permitiría al alumno reforzar estos conocimientos así como adentrarse considerablemente en las tecnologías recientes.

Solamente así se garantizará que los egresados tengan la experiencia necesaria para enfrentar áreas laborales que constituyen un marco de referencia dentro de la situación social, económica y tecnológica del país.

APÉNDICE A

Temarios

Sistemas de Bases de Datos

1. Conceptos básicos de las Bases de Datos.

   a) Bases de datos y sistemas manejadores de bases de datos.

   b) Arquitectura de tres niveles (físico, conceptual y externo).

   c) Modelos.

   Práctica 1 - SMBD
   Segunda semana.

2. Modelo Lógicos basados en Objetos.

   a) Modelo Entidad Relación.

   b) Diagramas UML (Diagrama de Clases).

   Práctica 2 - Modelado de Bases de Datos
   Tercera semana.

3. Modelo Relacional.

   a) Estructura.

   b) Álgebra relacional.

   c) Cálculo relacional.

   d) Sistemas relacionales.

e) Reglas de Codd.

4. SQL.

   a) Estructura.

   b) Operadores.

   Práctica 3 - SQL Lenguaje de Definición de Datos
   Quinta semana.
   Práctica 4 - SQL Lenguaje de Manipulación de Datos
   Sexta semana.
   Práctica 5 - SQL Lenguaje de Manipulación de Datos (II)
   Séptima semana.
   Práctica 6 - SQL Lenguaje de Manipulación de Datos (III)
   Novena semana.

5. Integridad.

   a) Integridad de Entidad.

   b) Integridad de Dominio.

   c) Integridad Referencial.

   d) Integridad del Usuario.

   Práctica 7 - Integridad
   Décimo primera semana.
   Práctica 8 - Seguridad
   Décimo segunda semana.

6. Diseño de Bases de Datos.

   a) Normalización.

   b) Formas normales.

   Práctica 9 - Normalización
   Décimo tercera semana.

7. Bases de Datos Orientadas a Objetos y Objeto-Relacionales.

   Práctica 10 - Conexión a Bases de Datos
   Décimo cuarta semana.

8. Revisión general de sistemas manejadores de bases de datos.

Sistemas Manejadores de Bases de Datos

1. El hardware en una base de datos.
   a) La jerarquía de memoria.
   b) Los discos duros.
   c) Los niveles RAID.
   d) Representación de los datos.
   e) Elementos de datos y campos.
   f) Registros.
   g) Representación de direcciones de bloques y registros.
   h) Registros y datos de tamaño variable.

   Práctica 1 - Acceso a memoria
   Segunda semana.
   Práctica 2 - Niveles RAID
   Tercera semana.

2. Los sistemas de archivos.
   a) El sistema de archivos secuencial indexado.
   b) El sistema de archivos indexado.
   c) El sistema de archivos de acceso directo (hash).
   d) Índices de múltiples dimensiones y de mapas de bits.

   Práctica 3 - Sistemas de archivos
   Quinta semana.

3. Procesamiento de Consultas.
   a) Etapas.
      1) Introducción a los operadores en los planes de consultas.
      2) Análisis sintáctico (parseo).
      3) Leyes algebraicas para la improvisación de los planes de ejecución.
      4) Árboles de parseo.
      5) Estimación de costos.
      6) Selección basada en el costo.
      7) El orden de las reuniones (joins).

8) Completando el plan de ejecución.

b) Estructura.

c) La selección.

d) La reunión (join).

1) Algoritmos de una pasada.
2) Reuniones con ciclos aninados.
3) Algoritmos de dos pasadas basados en ordenamiento.
4) Algoritmos de dos pasadas basados en estructuras Hash.
5) Algoritmos basados en índices.
6) Algoritmos que utilizan más de dos pasadas.

e) Evaluación de expresiones (Materialización y pipeline).

f) Transformación y evaluación de expresiones.

g) Optimizaciones.

Práctica 4 - Manejo de índices
Séptima semana.
Práctica 5 - Optimización de consultas
Novena semana.

4. Transacciones.

a) Concepto (ACID).

b) Implementación de ACID.

c) Seriabilidad.

Práctica 6 - Transacciones
Décima semana.

5. Concurrencia.

a) Seriabilidad y sus problemas.

b) Bloqueos y sus tipos.

c) Deadlock: Definición, manejo y prevención.

d) Calendarización.

e) Métodos de control de concurrencia (marcas de tiempo y validación).

Práctica 7 - Concurrencia
Décimo primera semana.

6. Sistemas de recuperación.

   a) Tipos de fallas.

   b) Seriabilidad.

   c) Recuperación.

   d) Tipos de almacenamiento.

   e) Recuperación basada en bitácoras.

   f) Protección contra fallas en los medios.

   g) Paginación sombra.

   h) Recuperación en transacciones concurrentes.

Práctica 8 - Recuperación
Décimo segunda semana.

7. Arquitecturas de los sistemas de Bases de Datos.

   a) Clasificación.

   b) Sistemas paralelos.

   c) Aceleración y escalamiento (speedup, scaleup).

   d) Sistemas distribuidos.

   e) Tipos de redes.

Práctica 9 - Modificación de recursos utilizados por el SMBD
Décimo tercera semana.

8. Bases de Datos Paralelas y Distribuidas.

   a) Introducción.

   b) Paralelismo interconsulta e intraconsulta.

   c) Paralelismo intraoperación e interoperación.

   d) Almacenamiento distribuido.

   e) Replicación y fragmentación.

   f) Procesamiento de consultas distribuidas.

g) Modelo de transacciones distribuidas.

h) Protocolos de Verificación (commit protocols).

i) El coordinador y su selección.

j) Control de concurrencia.

k) Manejo de deadlock.

9. Datawarehouse.

a) Conceptos de DataWarehouse.

b) Ventajas y aplicaciones.

c) Data-Marts.

d) Cubos de múltiples dimensiones.

e) Operaciones.

Práctica 10 - Datawarehouse
Décimo cuarta semana.

APÉNDICE B

Lineamientos generales

Lineamientos generales sobre la entrega de prácticas para los cursos de "Sistemas de Bases de Datos" y "Sistemas Manejadores de Bases de Datos".

- Las prácticas serán INDIVIDUALES, a menos de que se indique lo contrario.
- Las prácticas que se dejen durante el curso, deberán ser entregadas en la fecha que se indique, no importando que sea o no a la hora de clase, pero si debe ser el día estipulado, esto a menos de que se indique lo contrario.
- Existirá una penalización de un punto sobre la calificación por cada día de retraso que se tenga en la entrega de la práctica en cuestión.
- La penalización del punto anterior no aplicará para días festivos, obligatorios de asueto o algún día que no haya labores por causas de fuerza mayor.
- Las prácticas que sean muy semejantes o iguales, contarán con la penalización de anulación para las partes que se encuentren involucradas.
- La entrega de las prácticas se hará enviando por correo electrónico los archivos necesarios para poder revisarlas. (Esto debe hacerse a las direcciones de correo previamente establecidas).
- La fecha de entrega que se toma en cuenta, será la que se registre en el servidor de correo del ayudante o profesor.
- Las especificaciones de las tareas serán resueltas en el documento pertinente[1] .

[1] Se especifica esto, dado que los cursos requieren de tareas complementarias.

- Cualquier peculiaridad no resuelta en el presente documento, será tratada y resuelta específicamente.
- Dependiendo de la práctica, los archivos relativos a la misma deben entregarse dentro de un archivo adjunto, teniendo por nombre el login de la persona que lo envía, en el formato de compresión TAR y ZIP, con la extensión ".tgz" para que ocupe menos espacio y con la estructura presentada a continuación.

```
login
¦---- practica0n
      ¦---- README     En formato PDF, PS o TXT
      ¦---- MAKEFILE   En caso de ser necesario
      L---- Archivos   Resto de archivos necesarios
                       específicos por práctica.
```

Donde $0n$ es el número de tarea a entregar respectivamente.

APÉNDICE C

Base de datos para el primer curso

A continuación se presenta un detalle amplio de la base de datos utilizada para el desarrollo de las prácticas relativas al curso "Sistemas de Bases de Datos" y abarca todas sus especificaciones.

C.1. Introducción

El primer curso, "Sistemas de Bases de Datos", se encuentra enfocado a mostrar características básicas sobre el modelado y manipulación de las bases de datos. La base de datos que se presenta tiene como meta particular abarcar la mayor parte de estos conceptos y mostrarlos detalladamente al alumno.

La complejidad lógica del modelo es mayor con respecto a la base de datos del segundo curso y se recomienda, aunque no es requerimiento necesario, que los alumnos desarrollen una base de datos paralelamente a esta con la cual puedan trabajar y sentirse más involucrados durante su desarrollo

El tamaño de la base de datos no es extenso dado que los temas avanzados no son considerados para el primer curso y lo más importante dentro del primer curso es adquirir nociones prácticas dentro del tema.

C.2. Consideraciones

La base de datos pretende almacenar la información relativa al personal académico de una entidad educativa. En la misma, se tiene registro de las características más relevantes únicamente.

Las principales consideraciones respecto a la base de datos son las siguientes:

- El alcance cubre al área de académicos.
- Del área académica se tiene un registro de los profesores que participen en algún proyecto de investigación o algún articulo publicado.
- La información general de los académicos se contempla.
- Los académicos pueden pertenecer a más de un departamento dentro de la institución educativa.
- Los académicos NO pueden tener más de un grado académico. Pero si más de un número telefónico asociado.
- Los numeros telefónicos sólo contemplan numeros locales o teléfonos móviles y no se manejan extensiones.
- La duración de los proyectos se mide en meses.

C.3. Modelo Entidad-Relación

El modelo entidad-relación obtenido durante la etapa de diseño es el mostrado en la figura C.1. Sus entidades, y relaciones se presentan a continuación:

C.3.1. Entidades

Las entidades que se obtuvieron luego del proceso de abstracción, son las siguientes:

- **Profesor**.- Representa a un profesor y consta de los atributos:
  - id_profesor - Es el identificador del profesor.
  - nombre - Es el nombre del profesor.
  - ap_pat - El apellido paterno del profesor.
  - ap_mat - El apellido materno del profesor.
  - grado - El grado académico que posee el profesor.
  - calle - Nombre de la calla donde vive el profesor.
  - colonia - Colonia donde reside el profesor.
  - número - Número de inmueble donde se ubica el profesor.
  - cod_post - Código postal del profesor.
  - antiguedad - La antigüedad que tiene el profesor en la institución.
- **Teléfono**.- Represente a un numero telefónico. Esta compuesto de:

- id_telefono - Es el identificador del teléfono.
- numero_tel - El número teléfonico.
- tipo_tel - Tipo de teléfono.

- **Artículos**.- Representa a una articulo publicado, consta de los atributos:
  - id_art - Identificador de artículos.
  - nombre_art - El nombre del articulo.
  - area_art - Área a la cual pertenece el articulo.
  - sitio_art - Publicación donde apareció el articulo.
  - mes_art - Mes cuando apareció el articulo.
  - anno_art - Año cuando apareció el articulo.
- **Departamento**.- Representa a un departamento de la institución, cuenta con los siguientes atributos:
  - id_depto - Identificador de departamentos.
  - nombre_depto - Es el nombre del departamento.
- **Proyecto**.- Representa un proyecto de investigación realizado en la institución, tiene los atributos:
  - id_proy - Identificador de proyectos.
  - nombre_proy - Es el nombre del proyecto.
  - mes_ini - Mes de inicio del proyecto.
  - anno_ini - Año de inicio del proyecto.
  - duracion - Duración del proyecto.

C.3.2. Relaciones

Las relaciones que se presentan en el modelo son las siguientes:

- **tel_prof**.- Representa los números telefónicos que tiene un profesor.
- **tel_depto**.- Representa los números telefónicos que tiene un departamento.
- **prof_art**.- Representa los artículos que ha publicado un profesor.
- **prof_depto**.- Representa los departamentos a los cuales pertenece un profesor. Tiene como atributos:
  - num_cub - Es el número de cubículo que tiene el profesor en el departamento.

- **prof_part.**- Representa la participación de un profesor en un proyecto determinado para cierto departamento. Tiene el siguiente atributo.

  - tipo_part - Es el tipo de participación que tiene el profesor en el proyecto.

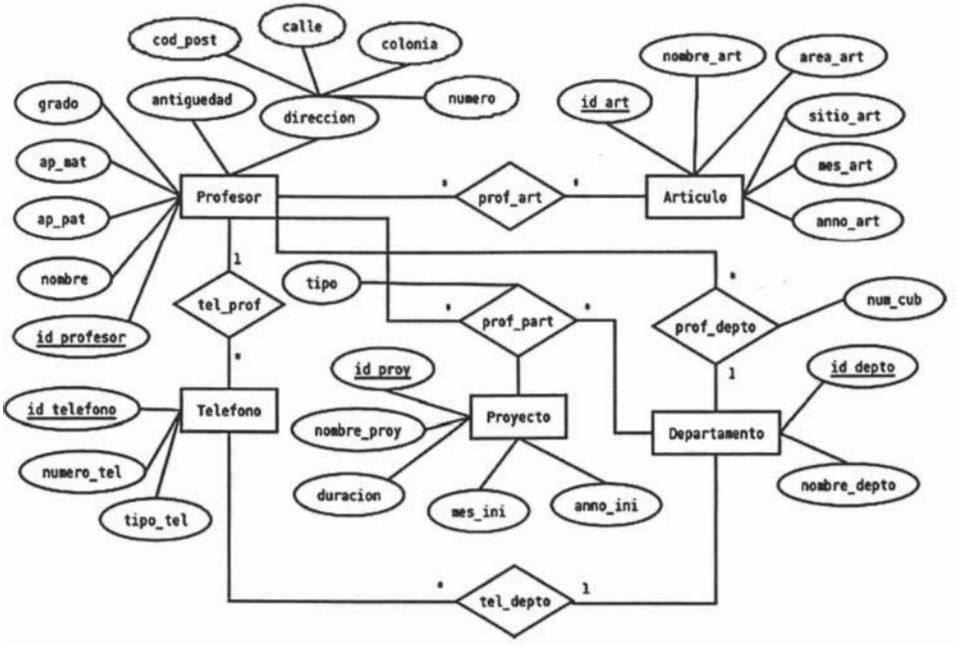

Figura C.1: Diagrama Entidad - Relación.

C.4. Modelo Relacional

Durante el procedimiento de generación de las tablas relativas al modelo relacional se tomaron en cuenta las recomendaciones generales mostradas en [2].

Para la generación de las tablas correspondientes a las relaciones en el modelo Entidad-Relación se tomaron las llaves primarias de las relaciones y se tratarón como llaves foráneas, luego simplemente se les agrego algún atributo si era el caso.

Las tablas que se generan durante el traslado a modelo relacional se presentan en la figura C.2.

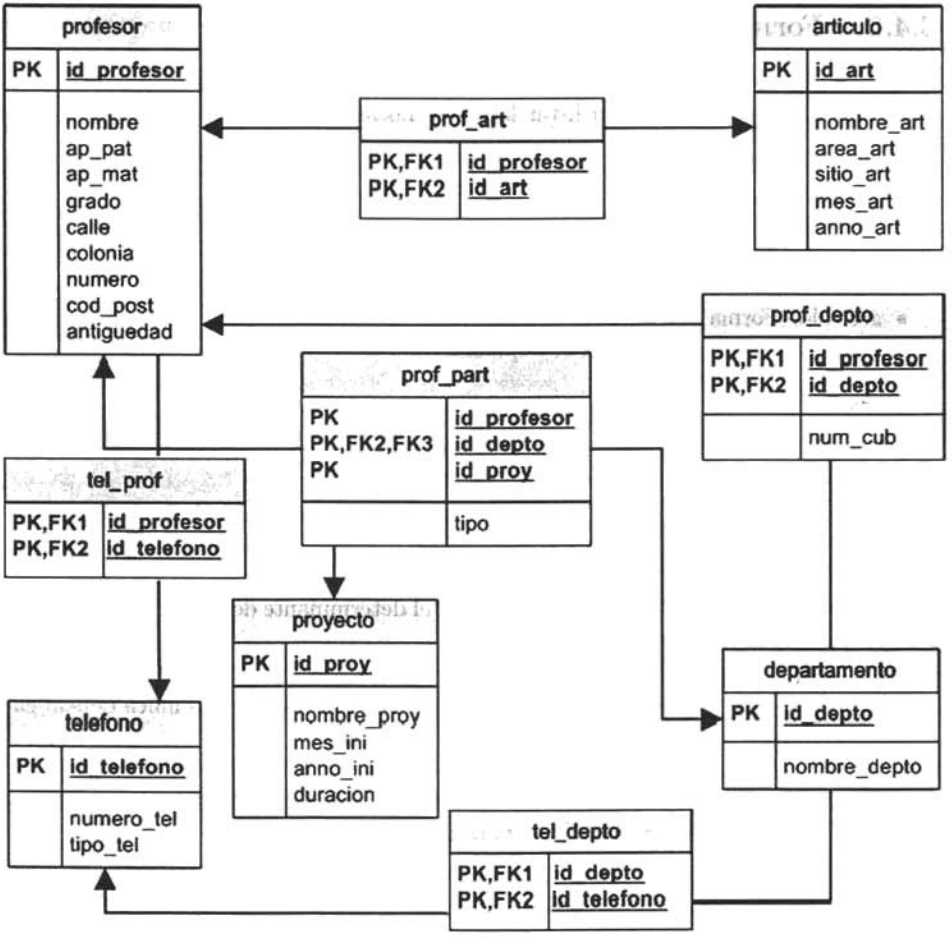

Figura C.2: Modelo Relacional.

C.4.1. Dependencias Funcionales

Las dependencias funcionales que se encuentran en este modelo no son más que las encontradas en el conjunto de dependencias funcionales triviales.

C.4.2. Formas Normales

Una vez que se detectaron las dependencias funcionales presentes en todas las relaciones del modelo, se procedió a listar la forma normal alcanzada dentro de cada una de ellas.

- profesor.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- articulo.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- prof_depto.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- prof_part.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- telefono.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- proyecto.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- departamento.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.

C.5. UML

Del mismo modo, para la generación del diagrama de clases relativo a la base de datos, se procedió tomando en cuenta las recomendaciones de [2]. Los métodos de cada clase se omiten en el diagrama puesto que su utilidad no esta dentro del alcance del proyecto y su utilidad es nula para nuestros fines.

El diagrama de clases correspondiente a la base de datos se presenta en la siguiente figura.

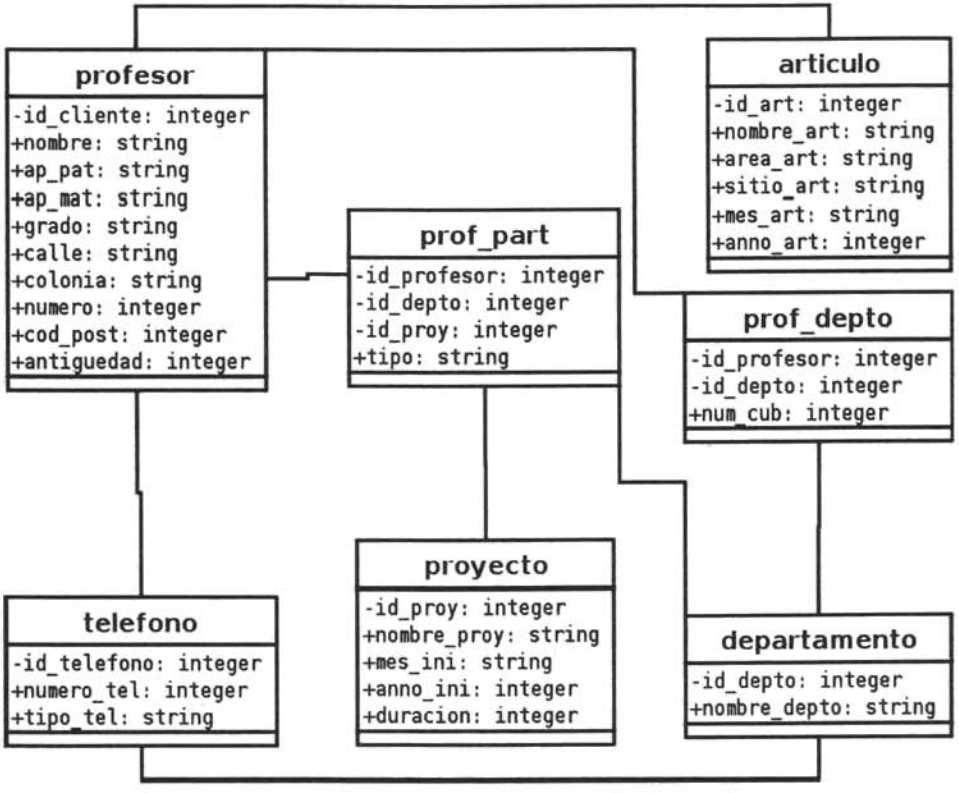

Figura C.3: Diagrama de Clases -UML.

C.6. Código

A continuación se lista el código en SQL utilizado para generar la base de datos del curso de "Sistemas de Bases de Datos" y que para el contenido de las prácticas se hará mención a esta base de datos por el nombre de **"cursoSBD"**.

```
--
-- Eliminación de tablas y secuencias
--
DROP TABLE prof_part;
DROP TABLE prof_depto;
DROP TABLE prof_art;
DROP TABLE tel_prof;
DROP TABLE tel_depto;
DROP TABLE proyecto;
DROP TABLE telefono;
DROP TABLE departamento;
DROP TABLE articulo;
DROP TABLE profesor;
DROP SEQUENCE seq_id_proyecto;
DROP SEQUENCE seq_id_telefono;
DROP SEQUENCE seq_id_departamento;
DROP SEQUENCE seq_id_articulo;
DROP SEQUENCE seq_id_profesor;
--
-- Secuencia SEQ_ID_PROFESOR
-- Secuencia que controla la creación de identificadores
-- de la tabla 'PROFESOR'
--
CREATE SEQUENCE seq_id_profesor INCREMENT 1 START 1;
--
-- Tabla        : 'profesor' Profesor de la Institución
-- id_profesor  : Identificador del profesor
-- nombre       : Nombre del profesor
-- ap_pat       : Apellido paterno del profesor
-- ap_mat       : Apellido materno del profesor
-- grado        : Grado académico máximo
-- calle        : Calle del domicilio
-- colonia      : Colonia del domicilio
-- numero       : Numero del domicilio
-- cod_post     : Código postal del domicilio
-- antiguedad   : Antigüedad del profesor en la Institución
```

```
--
CREATE TABLE profesor (
id_profesor BIGINT DEFAULT (nextval('seq_id_profesor')) PRIMARY KEY,
nombre VARCHAR(35) NOT NULL,
ap_pat VARCHAR(35) NOT NULL,
ap_mat VARCHAR(35) NOT NULL,
grado VARCHAR(35) NOT NULL,
calle VARCHAR(35) NOT NULL,
colonia VARCHAR(25) NOT NULL,
numero SMALLINT NOT NULL DEFAULT (1)
CHECK (numero > 0 ),
cod_post INTEGER NOT NULL
CHECK(cod_post >= 1 AND cod_post <= 99999),
antiguedad SMALLINT NOT NULL DEFAULT (1)
CHECK(antiguedad > 0)
);
--
-- Secuencia SEQ_ID_ARTICULO
-- Secuencia que controla la creación de identificadores
-- de la tabla 'ARTICULO'
--
CREATE SEQUENCE seq_id_articulo INCREMENT 1 START 1;
--
-- Tabla      : 'articulo' Un articulo publicado por un profesor
-- id_art     : Identificador de un articulo
-- nombre_art   : Nombre del articulo
-- area_art    : Area de estudio a la que pertenece el articulo
-- sitio_art   : Nombre donde aparece el articulo (revista, publicacion, etc.)
-- mes_art     : Mes de publicación del articulo
-- anno_art    : Año de publicación del articulo
--
CREATE TABLE articulo (
id_art BIGINT DEFAULT (nextval('seq_id_articulo')) PRIMARY KEY,
nombre_art VARCHAR(35) NOT NULL,
area_art VARCHAR(30) NOT NULL,
sitio_art VARCHAR(35) NOT NULL,
mes_art VARCHAR(15) NOT NULL
CHECK (mes_art IN ('Enero','Febrero','Marzo','Abril','Mayo','Junio','Julio',
'Agosto','Septiembre','Octubre','Noviembre','Diciembre')),
anno_art SMALLINT NOT NULL
CHECK (anno_art > 1900 and anno_art < 2006)
);
```

```
--
-- Secuencia SEQ_ID_DEPARTAMENTO
-- Secuencia que controla la creación de identificadores
-- de la tabla 'DEPARTAMENTO'
--
CREATE SEQUENCE seq_id_departamento INCREMENT 1 START 1;
--
-- Tabla          : 'departamento' Un departamento de la Institución
-- id_depto       : Identificador del departamento
-- nombre_depto   : Nombre del departamento
--
CREATE TABLE departamento (
id_depto BIGINT DEFAULT (nextval('seq_id_departamento')) PRIMARY KEY,
nombre_depto VARCHAR(35) NOT NULL
);
--
-- Secuencia SEQ_ID_TELEFONO
-- Secuencia que controla la creación de identificadores
-- de la tabla 'TELEFONO'
--
CREATE SEQUENCE seq_id_telefono INCREMENT 1 START 1;
--
-- Tabla          : 'telefono' Un numero telefónico
-- id_telefono    : Identificador del telefono
-- numero_tel     : Numero telefónico
-- tipo_tel       : Tipo de numero telefónico
--
CREATE TABLE telefono (
id_telefono BIGINT DEFAULT (nextval('seq_id_telefono')) PRIMARY KEY,
numero_tel SMALLINT NOT NULL
CHECK (numero_tel > 0 and numero_te < 445599999999),
tipo_tel VARCHAR(20) NOT NULL
CHECK (tipo_tel IN ('Celular','Movil','Trabajo','Casa'))
);
--
-- Secuencia SEQ_ID_PROYECTO
-- Secuencia que controla la creación de identificadores
-- de la tabla 'PROYECTO'
--
CREATE SEQUENCE seq_id_proyecto INCREMENT 1 START 1;
--
-- Tabla          : 'proyecto' Un proyecto donde participa un profesor
```

```
-- id_proy        : Identificador del proyecto
-- nombre_proy    : Nombre del proyecto
-- mes_ini        : Mes de inicio del proyecto
-- anno_ini       : Año de inicio del proyecto
-- duracion       : Tiempo que toma el proyecto (en meses)
--
CREATE TABLE proyecto (
id_proy BIGINT DEFAULT (nextval('seq_id_proyecto')) PRIMARY KEY,
nombre_proy VARCHAR(35) NOT NULL,
mes_ini VARCHAR(15) NOT NULL
CHECK (mes_ini IN ('Enero','Febrero','Marzo','Abril','Mayo','Junio','Julio',
'Agosto','Septiembre','Octubre','Noviembre','Diciembre')),
anno_ini SMALLINT NOT NULL
CHECK (anno_ini > 1900),
duracion SMALLINT NOT NULL
CHECK (duracion > 0)
);
--
-- Tabla          : 'prof_art' Detalles de los articulos de cada profesor
-- id_profesor    : Identificador del profesor
-- id_art         : Identificador del articulo
--
CREATE TABLE prof_art(
id_profesor BIGINT NOT NULL,
id_art BIGINT NOT NULL,
CONSTRAINT pk_prof_art PRIMARY KEY (id_profesor,id_art),
CONSTRAINT fk_prof_ FOREIGN KEY (id_profesor)
REFERENCES PROFESOR (id_profesor)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_art FOREIGN KEY (id_art)
REFERENCES ARTICULO (id_art)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
--
-- Tabla          : 'tel_prof' Detalles de los teléfonos de cada profesor
-- id_telefono    : Identificador del teléfono
-- id_profesor    : Identificador del profesor
--
CREATE TABLE tel_prof(
id_telefono BIGINT NOT NULL,
id_profesor BIGINT NOT NULL,
CONSTRAINT pk_tel_prof PRIMARY KEY (id_telefono, id_profesor),
```

```
CONSTRAINT fk_tel_prof FOREIGN KEY (id_telefono)
REFERENCES TELEFONO (id_telefono)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_prof_tel FOREIGN KEY (id_profesor)
REFERENCES PROFESOR (id_profesor)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
--
-- Tabla          : 'tel_depto' Detalles de los teléfonos de cada departamento
-- id_telefono    : Identificador del teléfono
-- id_depto       : Identificador del departamento
--
CREATE TABLE tel_depto(
id_telefono BIGINT NOT NULL,
id_depto BIGINT NOT NULL,
CONSTRAINT pk_tel_depto PRIMARY KEY (id_telefono, id_depto),
CONSTRAINT fk_tel_depto FOREIGN KEY (id_telefono)
REFERENCES TELEFONO (id_telefono)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_depto_tel FOREIGN KEY (id_depto)
REFERENCES DEPARTAMENTO (id_depto)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
--
-- Tabla          : 'prof_depto' Detalles de departamento al que pertenece un profesor
-- id_profesor    : Identificador del profesor
-- id_depto       : Identificador del departamento
-- num_cub        : Numero del cubiculo del profesor dentro del departamento
--
CREATE TABLE prof_depto (
id_profesor BIGINT NOT NULL,
id_depto BIGINT NOT NULL,
num_cub SMALLINT NOT NULL,
CONSTRAINT pk_prof_depto PRIMARY KEY (id_profesor,id_depto),
CONSTRAINT fk_prof FOREIGN KEY (id_profesor)
REFERENCES PROFESOR (id_profesor)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_depto FOREIGN KEY (id_depto)
REFERENCES DEPARTAMENTO (id_depto)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
--
```

```
-- Tabla          : 'prof_part' Detalles del profesor que participa en un proyecto
-- de un departamento
-- id_profesor    : Identificador del profesor
-- id_depto       : Identificador del departamento
-- id_proy        : Identificador del proyecto
-- tipo           : Tipo de participacion
CREATE TABLE prof_part (
id_profesor BIGINT NOT NULL,
id_depto BIGINT NOT NULL,
id_proy BIGINT NOT NULL,
tipo VARCHAR(20) NOT NULL
CHECK (tipo IN ('Titular','Director','Colaborador','Coordinador')),
CONSTRAINT pk_prof_part PRIMARY KEY (id_profesor,id_depto,id_proy),
CONSTRAINT fk_prof FOREIGN KEY (id_profesor)
REFERENCES PROFESOR (id_profesor)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_depto FOREIGN KEY (id_depto)
REFERENCES DEPARTAMENTO (id_depto)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
CONSTRAINT fk_proy FOREIGN KEY (id_proy)
REFERENCES PROYECTO (id_proy)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
```

APÉNDICE D

Base de datos para el segundo curso

A continuación se presenta un detalle amplio de la base de datos utilizada para el desarrollo de las prácticas relativas al curso "Sistemas Manejadores de Bases de Datos" y abarca todas sus especificaciones.

D.1. Introducción

El segundo curso, "Sistemas Manejadores de Bases de Datos", se encuentra enfocado a mostrar características avanzadas del diseño e implementación de herramientas y técnicas a nivel administrativo dentro de la base de datos. De tal forma que a diferencia de la base de datos planeada para el primer curso, en esta será necesario contar con una base de datos relativamente grande pero bastante simple que refleja una realidad muy en particular.

D.2. Consideraciones

La base de datos, pretende reflejar de manera muy sencilla, a la información que almacena una tienda de videojuegos que tiene servicio a nivel mundial.

Las principales consideraciones respecto a la base de datos son las siguientes:

- La información que es importante en relación a un cliente, simplemente son los datos más básicos, por ejemplo, nombre, apellido paterno, apellido materno, dirección, y teléfono; de este último, suponemos que solamente tiene uno.
- Los clientes solamente puede llevar a cabo una compra de un producto a la vez, es decir, no existen "ordenes de compra" que agrupen más de un videojuego.

- Los clientes únicamente pueden llevar a cabo una compra, de cualquier número de elementos, de algún videojuego.
- Aún cuando el atributo de teléfono suele ser muy socorrido, en nuestro contexto, presuponemos que nuestro cliente sólo tiene un teléfono de contacto[1] .
- Es importante poseer un catalogo de los videojuegos que se tienen en existencia.
- Toda información adicional es irrelevante para nuestros propósitos[2] .

D.3. Modelo Entidad-Relación

El modelo Entidad-Relación obtenido durante la etapa de diseño, tomando en cuenta las consideraciones respectivas, es el mostrado en la figura D.1. Sus entidades, y relaciones se enlistan a continuación:

D.3.1. Entidades

Las entidades que se obtuvieron luego del proceso de abstracción, son las siguientes:

- **Cliente**.- Representa a un cliente y consta de los atributos:
  - id_cliente - El identificador único de un cliente.
  - nombre_cliente - Es el nombre de nuestro cliente.
  - ap_pat_cliente - El apellido paterno del cliente.
  - ap_mat_cliente - El apellido materno del cliente.
  - país - El país de residencia del cliente.
  - ciudad_cliente - La ciudad de residencia del cliente.
  - calle_cliente - Nombre de la calla donde vive nuestro cliente.
  - colonia_cliente - Colonia donde reside el cliente.
  - número_dir_cliente - Número de inmueble donde se ubica el cliente.
  - código_postal_cliente - Código postal del cliente.
  - teléfono_cliente - Teléfono del cliente.
- **Videojuego**.- Representa a un videojuego y consta de los atributos:
  - id_videojuego - El identificador único de un videojuego.
  - nombre - Es el nombre de nuestro videojuego.

[1] Consideración muy fuera de la realidad.
[2] Por ejemplo no se considera la información relevante a los proveedores y distribuidores.

- tipo - Es el tipo o genero que posee el videojuego.
- consola - Es la consola sobre la cual se ejecuta el videojuego en particular.
- precio - Refleja nuestro precio de venta del videojuego.

D.3.2. Relaciones

Las relaciones que se presentan en el modelo son las siguientes:

- **Compra**.- Representa a una compra realizada y consta de los atributos:
  - id_cliente - Es el cliente que ha realizado la compra.
  - id_videojuego - Es el videojuego comprado.
  - cantidad - Es el total de videojuegos que compra una cliente.
  - mes - Es el mes de la fecha de compra.
  - día - Es el día de la fecha de compra.
  - año - Es el año de la fecha de compra.

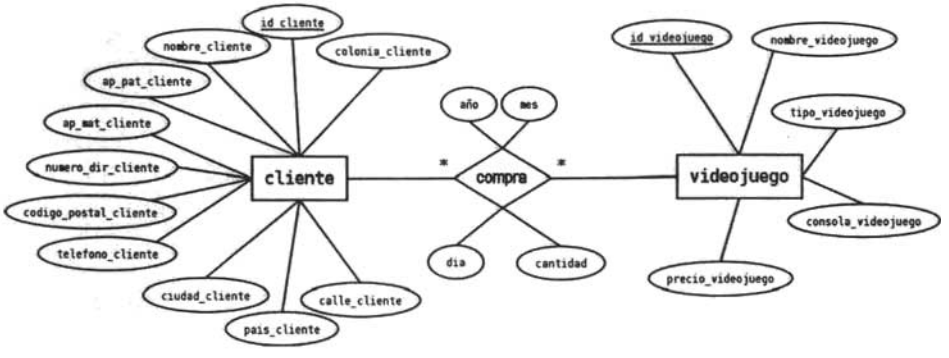

Figura D.1: Diagrama Entidad - Relación.

D.4. Modelo Relacional

Durante el procedimiento de generación de las tablas relativas al modelo relacional se tomaron en cuenta las recomendaciones generales mostradas en [2].

Hay que mencionar que para la tabla 'compra', la llave primaria se conformo al utilizar las dos llaves foráneas, una que hace referencia a un cliente y otra que hace referencia a un videojuego, esto como se menciono anteriormente muestra una realidad

que es difícil que se presente, dado que un cliente solamente puede comprar en una ocasión cierto videojuego.

Las tablas que se generan durante el traslado al modelo relacional se presentan en la figura D.2.

Figura D.2: Modelo Relacional.

D.4.1. Dependencias Funcionales

Las dependencias funcionales que se encuentran en éste modelo no son más que las encontradas en el conjunto de dependencias funcionales triviales.

Esto se da principalmente por cuestiones de simplicidad, ya que por ejemplo en una base de datos operacional, es casi un hecho de que se presente la dependencia funcional $\{ciudad \longrightarrow país\}$ o bien si un teléfono incluye larga distancia, entonces también se presenta la dependencia funcional $\{teléfono \longrightarrow ciudad\}$.

D.4.2. Formas Normales

Una vez que se detectaron las dependencias funcionales presentes en todas las relaciones del modelo, se procedió a listar la forma normal alcanzada dentro de cada una de ellas.

- cliente.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- compra.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.
- videojuego.- Forma normal FNBC, dado que el determinante de la única dependencia funcional es superllave para el esquema.

D.5. Diagrama de clases - UML

Del mismo modo, para la generación del diagrama de clases relativo a la base de datos, se procedió tomando en cuenta las recomendaciones de [2]. Los métodos de cada clase se omiten en el diagrama puesto que su utilidad no esta dentro del alcance del proyecto y su utilidad es nula para nuestros fines.

El diagrama de clases correspondiente a la base de datos se presenta en la siguiente figura.

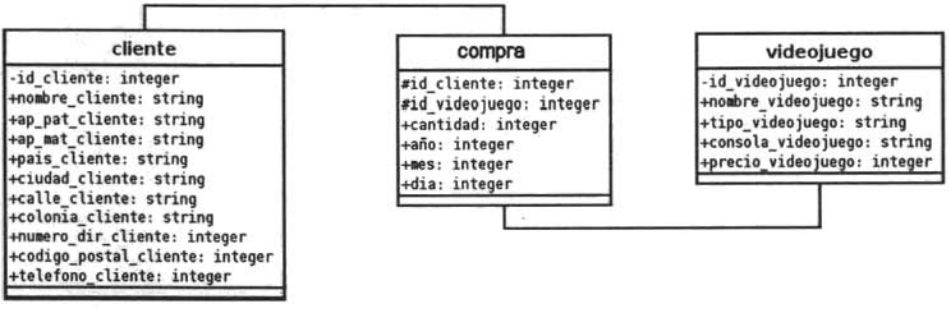

Figura D.3: Diagrama de Clases -UML.

D.6. Código

A continuación se lista el código en SQL utilizado para generar la base de datos del curso de "Sistemas Manejadores de Bases de Datos" y que para el contenido de las prácticas se hará mención a esta base de datos por el nombre de **"cursoSMBD"**.

```
--
-- Eliminación de tablas y secuencias
--
DROP TABLE compra;
DROP TABLE cliente;
DROP TABLE videojuego;
DROP SEQUENCE seq_id_cliente;
DROP SEQUENCE seq_id_videojuego;
--
-- Secuencia SEQ_ID_CLIENTE
-- Secuencia que controla la creación de identificadores
-- de la tabla 'CLIENTE'
--
CREATE SEQUENCE seq_id_cliente INCREMENT 1 START 1;
```

```
-- Tabla                 : 'cliente' Cliente de nuestra tienda
-- id_cliente            : Identificador del cliente
-- nombre_cliente        : Nombre del cliente
-- ap_pat_cliente        : Apellido paterno del cliente
-- ap_mat_cliente        : Apellido materno del cliente
-- país_cliente          : País de domicilio
-- calle_cliente         : Calle de domicilio
-- colonia_cliente       : Colonia del domicilio
-- código_postal_cliente : Código postal
-- teléfono_cliente      : Teléfono del cliente
-- ciudad_cliente        : Ciudad del cliente
--
CREATE TABLE cliente (
id_cliente BIGINT DEFAULT (nextval('seq_id_cliente')) PRIMARY KEY,
nombre_cliente VARCHAR(35) NOT NULL,
ap_pat_cliente VARCHAR(35) NOT NULL,
ap_mat_cliente VARCHAR(35) NOT NULL,
pais_cliente CHAR(35) NOT NULL,
calle_cliente CHAR(35) NOT NULL,
colonia_cliente CHAR(35) NOT NULL,
numero_dir_cliente SMALLINT NOT NULL DEFAULT (1)
CHECK (numero_dir_cliente > 0),
codigo_postal_cliente INTEGER NOT NULL
CHECK(codigo_postal_cliente >= 0 AND codigo_postal_cliente <= 999999),
telefono_cliente INTEGER NULL
CHECK(telefono_cliente > 0 AND telefono_cliente < 9999999999),
ciudad_cliente VARCHAR(35) NOT NULL
);
--
-- Secuencia SEQ_ID_VIDEOJUEGO
-- Secuencia que controla la creación de identificadores
-- de la tabla 'VIDEOJUEGO'
--
CREATE SEQUENCE seq_id_videojuego INCREMENT 1 START 1;
--
-- Tabla              : 'videojuego' Producto de nuestra tienda
-- id_videojuego      : Identificador del videojuego
-- nombre_videojuego  : Nombre del videojuego
-- tipo_videojuego    : Tipo del videojuego
-- consola_videojuego : Consola del videojuego
-- precio_videojuego  : Precio del videojuego
--
```

```
CREATE TABLE videojuego (
id_videojuego BIGINT DEFAULT (nextval('seq_id_videojuego')) PRIMARY KEY,
nombre_videojuego VARCHAR(40) NOT NULL,
tipo_videojuego CHAR(35) NOT NULL,
consola_videojuego CHAR(35) NOT NULL,
precio_videojuego NUMERIC (10, 2) NOT NULL
CHECK(precio_videojuego > 0)
);
--
-- Tabla          : 'compra' Relación de compra.
-- id_cliente     : Identificador del cliente (id_cliente)
-- id_videojuego  : Identificador del videojuego (id_videojuego)
-- cantidad       : Cantidad de videojuegos comprados
-- año            : Año de la fecha de la compra
-- mes            : Mes de la fecha de la compra
-- dia            : Dia de la fecha de la compra
--
CREATE TABLE compra (
id_cliente BIGINT NOT NULL,
id_videojuego BIGINT NOT NULL,
anno SMALLINT NOT NULL DEFAULT (2005)
CHECK (anno > 1990 AND anno < 2006),
mes SMALLINT NOT NULL DEFAULT (1)
CHECK (mes >0 AND mes <13),
dia SMALLINT NOT NULL DEFAULT (1)
CHECK (dia > 0 AND dia < 32),
cantidad BIGINT NOT NULL DEFAULT (1)
CHECK (cantidad > 0),
CONSTRAINT pk_compra PRIMARY KEY (id_cliente,id_videojuego),
CONSTRAINT fk_cliente FOREIGN KEY (id_cliente)
REFERENCES CLIENTE (id_cliente)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT fk_videojuego FOREIGN KEY (id_videojuego)
REFERENCES VIDEOJUEGO (id_videojuego)
MATCH FULL ON DELETE CASCADE ON UPDATE CASCADE
);
```

APÉNDICE E

Microsoft© SQL Server™ 2000

E.1. Introducción

El desarrollo de las compañías y su crecimiento general, han hecho que el control sobre la información sea un factor decisivo en la dirección, administración y toma de decisiones de las mismas. Esto creo una necesidad que fue cubierta desde sus inicios por los SMBD.

Microsoft© SQL Server™ 2000 es una familia de productos que cumple las características y requerimientos para sistemas de procesamiento de grandes bases de datos y sitios comerciales Web, al mismo tiempo provee de un fácil y sencillo acceso a estos servicios de almacenamiento de datos a individuos o negocios pequeños. Es un SMBDR, integrado fuertemente con el núcleo del sistema operativo Microsoft© Windows 2000©, que presenta características para el desarrollo de aplicaciones y un desempeño eficiente.

Con soluciones basadas en servidores SQL, los usuarios emplean sistemas clientes con los cuales pueden obtener información del servidor y manipularla localmente. Este tipo de implementación optimiza el procesamiento de la información, permitiendo a cada componente trabajar con la información independientemente y hacerlo en la manera que mejor se apegue a sus necesidades. El servidor se enfoca en los procesos de la base de datos, mientras que los clientes en la presentación de la información.

Actualmente la última versión de este conjunto de aplicaciones es Microsoft© SQL Server™ 2000, cuenta con múltiples características enfocadas a las empresas con desarrollo en Internet así como un conjunto de extensiones a las características mostradas en versiones anteriores. Las principales características son:

- Base de datos relacional.
    - Compatibilidad con XML.

- Servidores de bases de datos federados.

- Administración Gráfica.
  - Analizador de consultas.
  - Componentes gráficos para administrar el servidor.
- Replicación.
  - Duplicación por transacciones.
  - Duplicación por instantáneas.
  - Duplicación por mezcla.
- Servicios de transformación de datos.
- Servicios de Análisis.
  - Creación y manejo de fuentes OLAP.
  - Mineria de datos.
- Consultas en inglés.

E.1.1. Historia

A finales de la década de los años 80 e inicios de la década de los años 90, el sistema *Sybase* era uno de los más populares e innovadores SMBDR [27]. Estos sistemas se podían encontrar en ambientes UNIX y NetWare, pero el costo para adquirir uno de estos sistemas estaba fuera del alcance de muchos. Entonces, Sybase© y Microsoft© empezaron a cambiar esto. Microsoft© daría las licencias y vendería el sistema SMBDR de Sybase, SQL Server, bajo el nombre de Microsoft© para la plataforma OS/2. Microsoft© SQL Server™ para OS/2 se volvió un sistema de trabajo de grupos aceptable, pero el producto estaba limitado por OS/2 en su escalabilidad y desempeño.

La relación entre Microsoft y Sybase termino en 1992 cuando Microsoft abandono OS/2 en favor de su nuevo sistema operativo, Windows NT Server©, y decidió reescribir el núcleo de SQL Server para Windows NT©. El producto resultante fue Microsoft© SQL Server™ 4.2, que se volvió rápidamente una de las aplicaciones más populares de Windows NT©.

E.1.2. Componentes

SQL Server™ 2000 brinda los siguientes servicios fundamentales a aplicaciones en un ambiente Windows™

- Motor de base de datos relacional.
  Los datos se almacenan conceptualmente en tablas, que están constituidas de columnas y así se forman renglones o tuplas. Las aplicaciones pueden enviar sentencias de SQL al motor de la base de datos, el cual regresa los resultados en una forma de conjuntos de tablas. El dialecto de SQL que soporta SQL Server™ es el conocido como Transact-SQL. También pueden enviar sentencias SQL, búsquedas de XPath y peticiones que hacen que el motor de la base de datos regrese sus resultados en forma de un documento XML.

  El motor de la base de datos también se administra dinámicamente, adquiriendo más recursos cuando más usuarios se conectan a la base de datos y liberando los mismos cuando el usuario termina su sesión. La integración de este motor con Windows 2000© y Windows NT©, en clusters, permiten definir servidores virtuales que se mantienen funcionando aún cuando alguno de los servidores físicos en el nodo, falla. Además, la autenticación de acceso al SMBD puede ser integrada con la autenticación del sistema operativo.

  La característica de búsqueda distribuida del motor de la base de datos, permite que se acceda a los datos de cualquier fuente que lo permita por medio de OLE DB. El motor también es capaz de almacenar registros detallados de todas las transacciones generadas por los sistemas de procesamiento de transacciones en línea (OLTP) y soporta los requerimientos de proceso para las tablas o cubos de múltiples dimensiones en los procesamientos analíticos en línea (OLAP).

- Servicios de Análisis.
  Proveen de herramientas para analizar los datos guardados en los datawarehouses. Ciertos procesos analíticos, pueden tomar mucho tiempo si se usan todos los registros detallados en el sistema OLTP, para evitar eso, periódicamente los datos del sistema OLTP se guardan en los datawarehouses. Los Servicios de Análisis presentan estos datos, provenientes de los hechos y dimensiones de las tablas, como cubos de múltiples dimensiones que pueden ser analizados para obtener información que será importante en la planeación del trabajo futuro.

- Duplicación.
  Permite mantener varias copias de datos en diferentes equipos con el fin de mejorar el rendimiento global del sistema mientras se garantiza al mismo tiempo que las diferentes copias de datos se mantienen sincronizadas. La duplicación de SQL Server admite también la duplicación de datawarehouses y puede duplicar datos en o desde cualquier origen de datos que admita acceso OLE DB.

- Servicios de transformación de datos (DTS).

  Los Servicios de Transformación de Datos admiten la extracción de datos de un origen de datos, realizando en ocasiones complicadas transformaciones de los

datos y su almacenamiento. El componente simplifica el proceso de extracción de datos de varios sistemas OLTP y la creación de los datos en un almacén o puesto de datos OLAP. La utilización de los DTS no se limita a la creación de datawarehouses, además se pueden utilizar en cualquier momento para recuperar datos de un origen de datos, realizar transformaciones complejas de los datos y almacenarlos en otro origen de datos. Los DTS pueden trabajar con cualquier origen de datos al que se pueda tener acceso utilizando OLE BD.

- English Query.
  Permite crear aplicaciones que se pueden personalizar a sí mismas para ajustarse a preguntas del usuario. Un administrador de English Query define al motor de English Query todas las relaciones lógicas entre las tablas y columnas de una base de datos o los cubos de un sistema OLAP. Una aplicación puede presentar entonces al usuario un cuadro en el que puede escribir una cadena de caracteres con una pregunta (en inglés) acerca de los datos de la base de datos o del almacén de datos. La aplicación pasa la cadena al motor de English Query, que analiza la cadena frente a las relaciones definidas entre las tablas o cubos. A continuación, English Query devuelve a la aplicación una instrucción SQL o una consulta MDX que devolverá la respuesta a la pregunta del usuario.

Un esquema donde se presentan los componentes de SQL Server™ 2000 se ilustra en la figura E.1.

E.2. Obtención e instalación

Para la realización de las prácticas se utilizó la versión de evaluación de 120 días de SQL Server™ disponible en la siguiente dirección electrónica:
`http://www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyID=d20ba6e1-f44c-4781-a6bb-f60e02dc1335`

Además hay que considerar la descarga del paquete de servicio 4 ('*Service Pack 4*') de la siguiente dirección electrónica: `http://www.microsoft.com/downloads/details.aspx?FamilyId=8E2DFC8D-C20E-4446-99A9-B7F0213F8BC5&displaylang=es`

Una vez descargados los archivos, se procede a instalar el SMBD. La instalación es un proceso muy directo sobre el sistema operativo, ya que los archivos descargados se encuentran en un formato que permite la extracción autómatica de su contenido. Es importante mencionar que debido a las políticas de seguridad del sistema operativo anfitrión será necesario contar con los permisos pertinentes para habilitar el SMBD como un servicio del sistema operativo[1] . Por omisión los archivos archivos de instalación se almacenan en el directorio "C:\SQLEVAL\", y el archivo ejecutable que inicia propiamente la instalación es "setup.exe". Al ejecutar este archivo, se presentan va-

[1] Semejante a un demonio dentro de Linux.

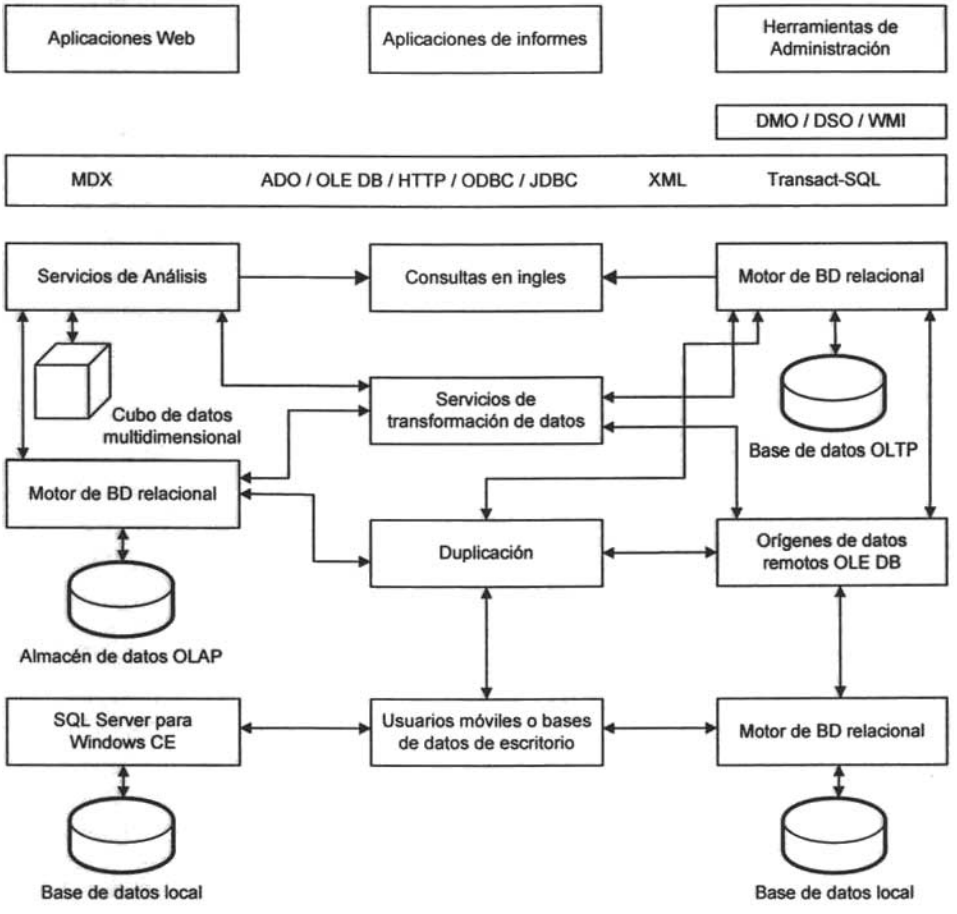

Figura E.1: Componentes de SQL Server™ 2000.

rios mensajes en pantalla que permiten determinar las características de la instalación así como el progreso de la misma.

Primeramente se debe escoger una instalación local, ya que el SMBD se instalará en la máquina donde se tienen los archivos, posteriormente se debe elegir la opción para crear una nueva instancia de SQL Server, esto se presenta en la figura E.3. A continuación se eligen los datos del usuario (nombre y compañia) (figura E.4), la siguiente pantalla del proceso de instalación permite elegir entre las herramientas a instalar, aquí hay que elegir instalar el servidor y las herramientas de cliente (figura E.5).

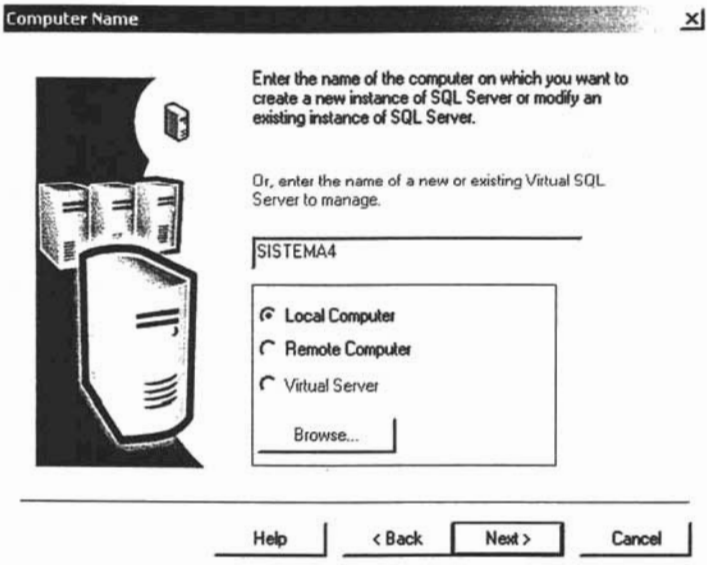

Figura E.2: Elección de una instalación local de SQL Server™ 2000.

El proceso continua con la elección de un nombre para la instancia de SQL Server™ 2000 que se esta instalando, esto porque es posible contener varias instancias independientes de SQL Server dentro de la misma computadora o servidor, existe también la posibilidad de permitir al sistema nombrar cada instancia, pero es recomendable elegir el nombre de la instancia para facilitar su identificación (figura E.6). En la siguiente pantalla es posible elegir entre una instalación típica, una instalación mínima y una personalizada Hay que seleccionar una instalación personalizada (figura E.7) y posteriormente todos los componentes (figura E.8). En la pantalla de configuración de servicios, es recomendable administrar todos los servicios con la misma cuenta de usuario (figura E.9).

Las siguientes pantallas presentan información respecto al modo de autenticación que utilizara SQL Server™ 2000, es recomendable elegir la autenticación por medio

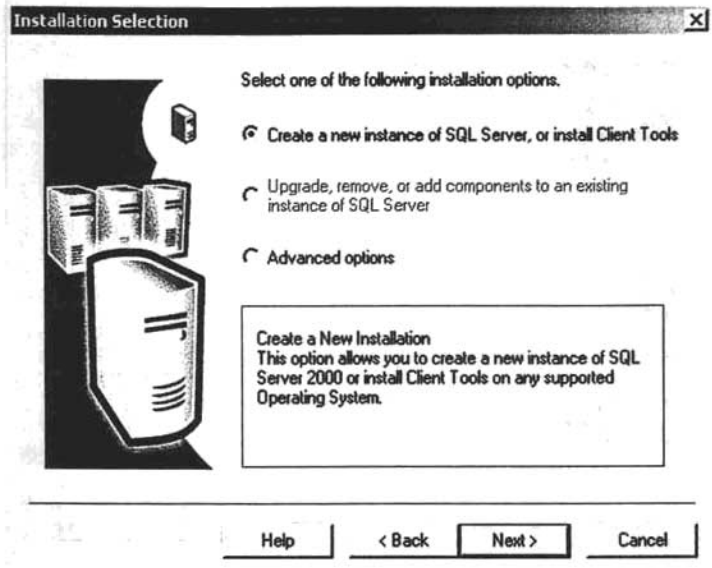

Figura E.3: Creación de una nueva instancia de SQL Server™ 2000.

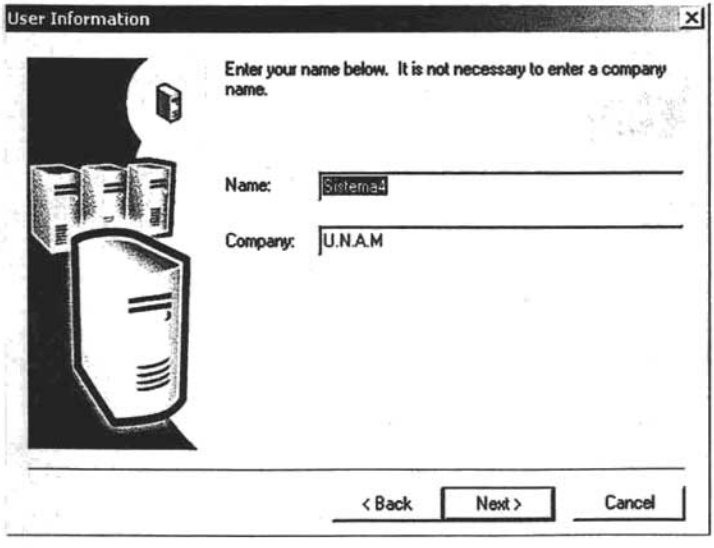

Figura E.4: Asignación de nombre a la nueva instancia de SQL Server™ 2000.

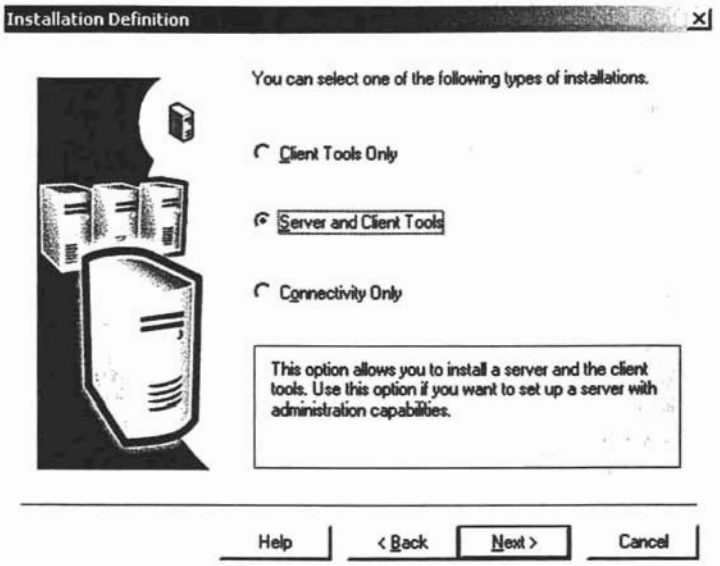

Figura E.5: Elección del servidor SQL Server™ 2000 y herramientas de cliente.

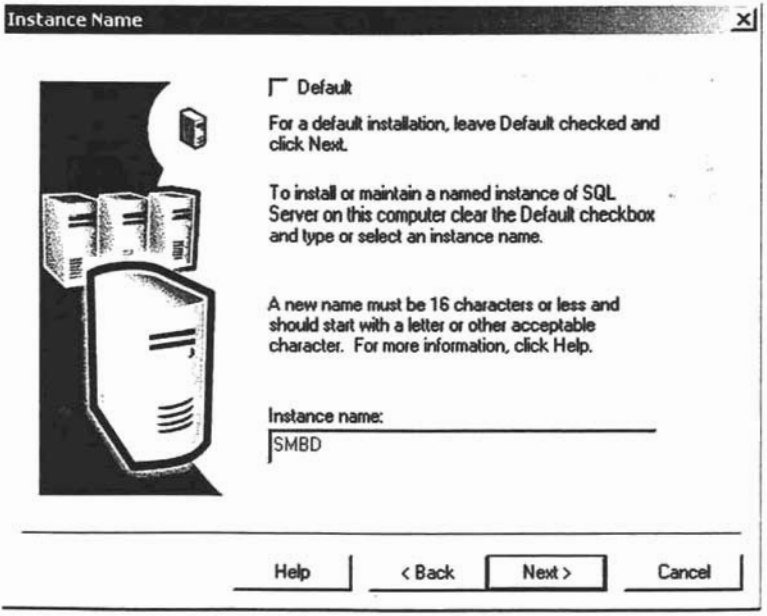

Figura E.6: Elección de nombre de una nueva instancia de SQL Server™ 2000.

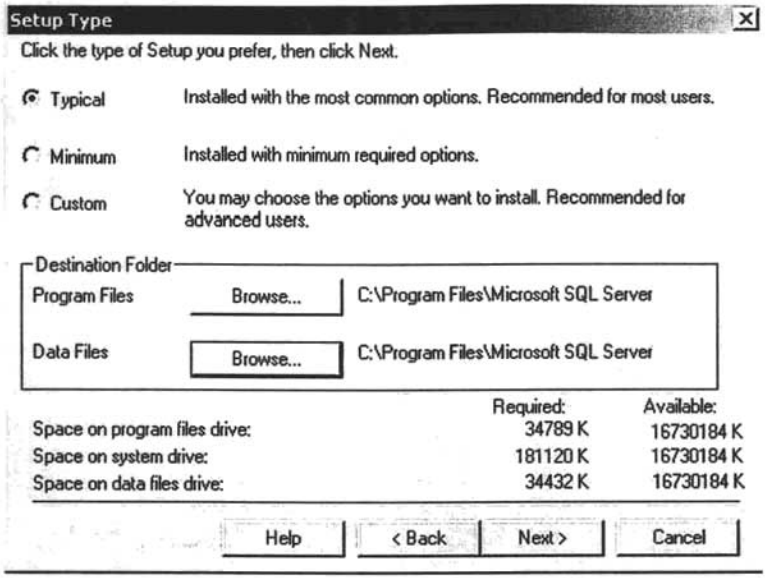

Figura E.7: Elección de una instalación personalizada.

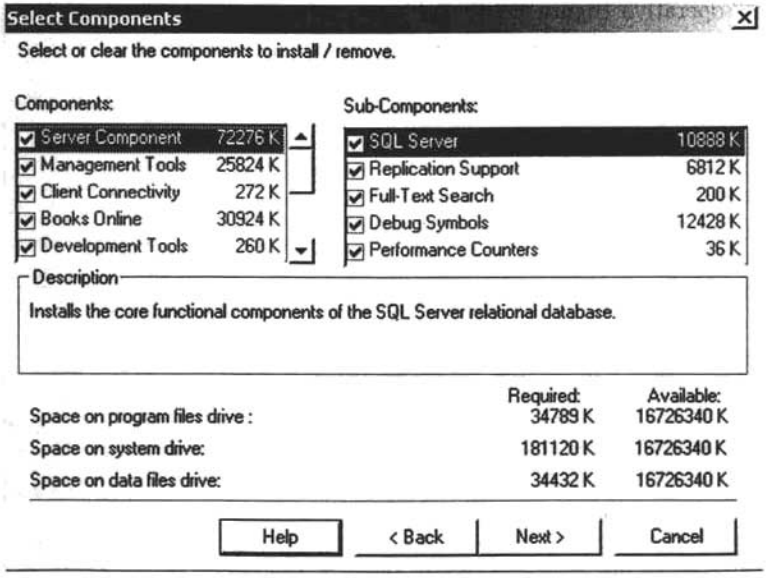

Figura E.8: Selección de todos los componentes de SQL Server™ 2000.

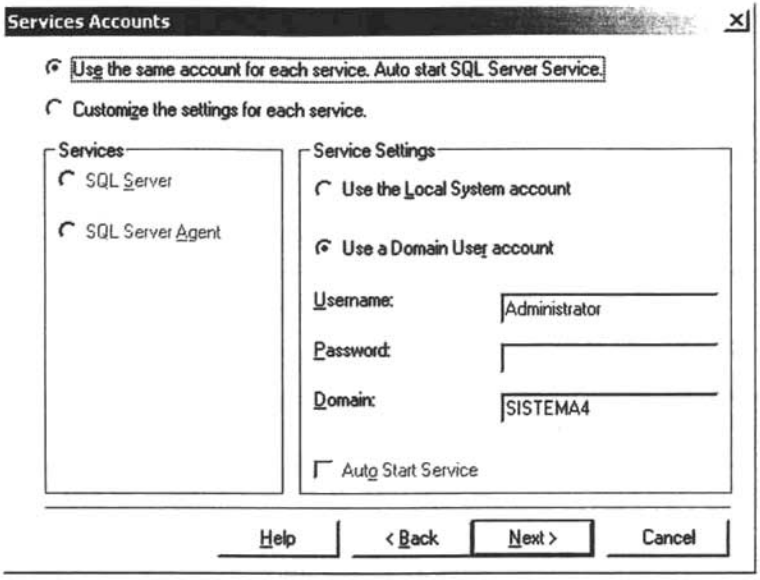

Figura E.9: Elección de características en los servicios de SQL Server™ 2000.

del sistema operativo (figura E.10). Adicionalmente hay que seleccionar el código de caracteres que utilizará la base de datos (figura E.11), las bibliotecas compartidas que utilizará el servidor para recibir peticiones externas y los puertos respectivos donde recibirá éstas peticiones (figura E.12).

Luego de instalar el SMBD, hay que instalar el SP4 para actualizar el servidor y evitar ciertos agujeros de seguridad (además es recomendable que luego de instalar este paquete de servicio, se actualice el sistema operativo). Posteriormente aparecerá dentro del menú de programas, el conjunto de herramientas de SQL bajo el nombre **"Microsoft© SQL Server™ 2000"**. Para iniciar la herramienta de administración basta con seleccionar **"Administrador Corporativo"** (figura E.13).

E.3. Servicios de Análisis (*'Analysis Services'*)

Los Servicios de Análisis de Microsoft© SQL Server™ 2000 están implementados por un servidor de nivel intermedio para procesos analíticos en línea (OLAP) y minería de datos. El sistema de los Servicios de Análisis incluye un servidor que administra cubos de datos de múltples dimensiones para analizarlos y proporciona un rápido acceso a la información de los cubos. Además organiza los datos de un datawarehouse en cubos con datos de agregaciones calculadas previamente para proporcionar respuestas rápidas a consultas analíticas complejas. Los Servicios de Análisis también permite crear

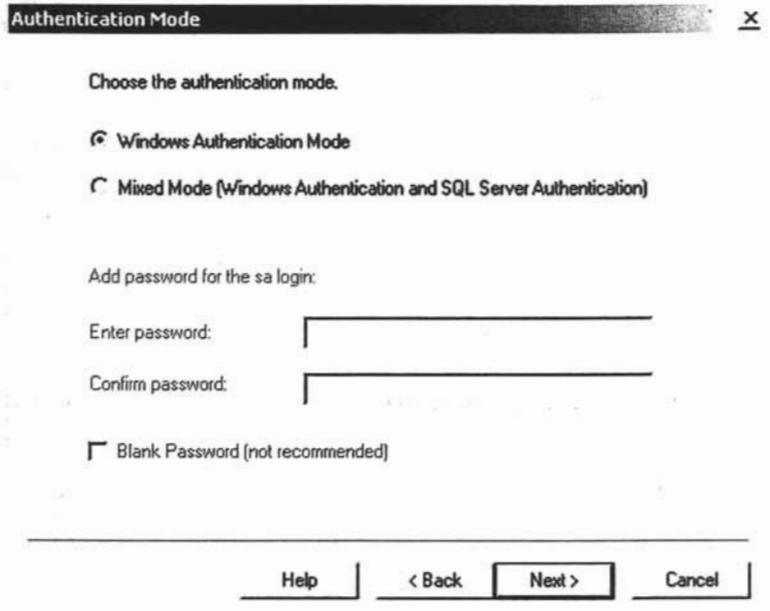

Figura E.10: Elección del método de autenticación.

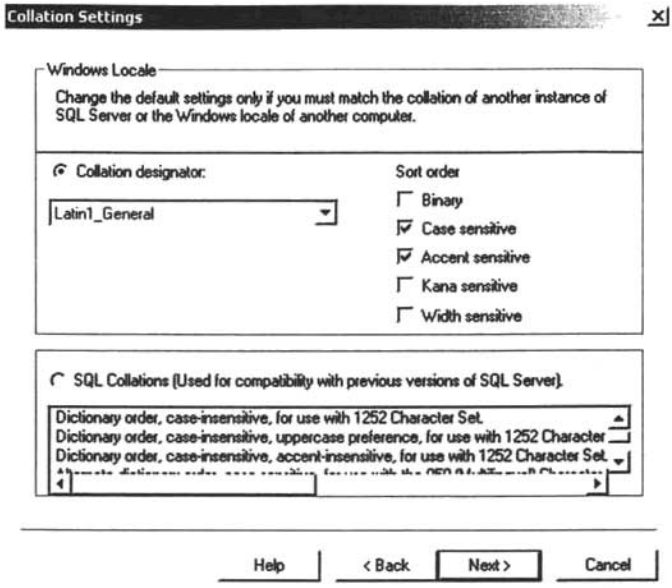

Figura E.11: Selección del código de caracteres de la base de datos.

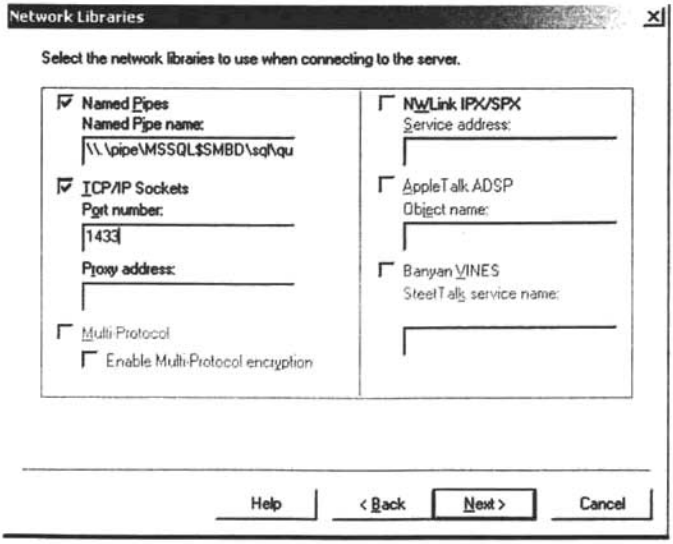

Figura E.12: Selección de las bibliotecas utilizadas para conectarse al servidor.

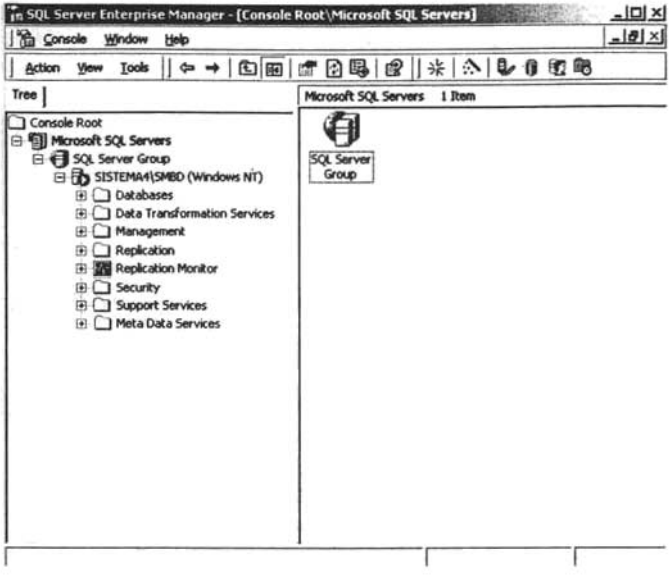

Figura E.13: El Administrador Corporativo de SQL Server™ 2000.

modelos de minería de datos de orígenes de datos de múltiples dimensiones (OLAP) y relacionales. Además se cuenta con un servicio llamado PivotTable© que se especializa en presentar esta información dentro de aplicaciones que permitan la incrustación de objetos OLE DB (por ejemplo hojas de calculo, procesadores de texto, presentaciones, etc.) y en la creación de cubos de datos locales para el análisis sin conexión.

Los Servicios de Análisis también proporcionan una arquitectura para obtener acceso a los datos de la minería de datos. Estos datos pueden enviarse al cliente en forma de datos relacionales o de múltiples dimensiones.

E.3.1. Instalación

El proceso de instalación es simple. Luego de aceptar los términos de la licencia de evaluación, la única pantalla que presenta cierta información relevante para nuestros fines consiste en la selección de componentes del Servidor de Análisis (figura E.14).

Al terminar la instalación se agregará de manera automática al menú de SQL Server™ 2000, el menú de **"Servicios de Análisis"**. Para iniciar la herramienta de administración basta con seleccionar **"Administrador de Análisis"** (figura E.15).

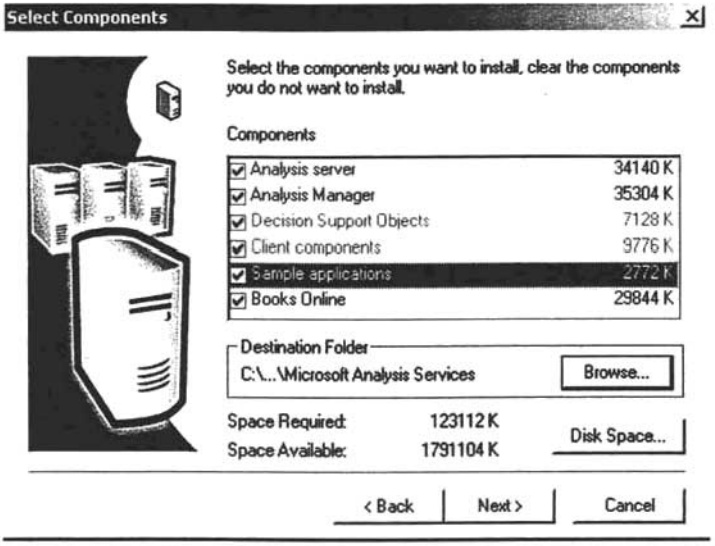

Figura E.14: Selección de componentes del Servidor de Análisis.

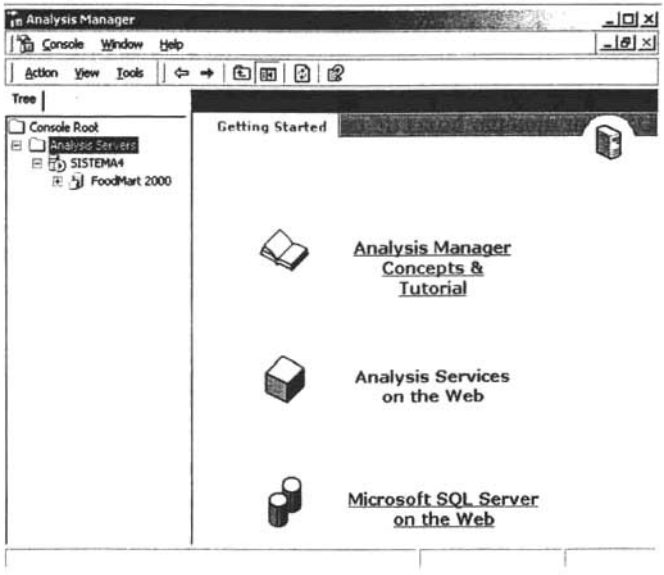

Figura E.15: El Administrador de Análisis de SQL Server™ 2000.

E.4. Fuentes de datos

En SQL Server™ 2000 se conoce como una fuente de datos a una ubicación que suministra datos mediante un proveedor OLE DB. En el presente trabajo se utilizará una fuente de datos ODBC. Para ello hay que descargar el controlador ODBC relativo a PostgreSQL de la siguiente página `http://gborg.postgresql.org/project/psqlodbc/projdisplay.php` . A continuación se debe agregar desde las fuentes de datos en el **"Panel de Control"** o **"Herramientas Administrativas"** una conexión con el controlador ODBC de PostgreSQL.

E.4.1. PostgresSQL para Windows©

PostgreSQL v 8.0 fue la primera versión que contó con soporte nativo para Windows©, y su utilización es semejante a la que se tiene dentro de Linux. Para nuestro trabajo se considera la instalación en este sistema operativo dada la necesidad de importar nuestros datos relativos de la base de datos del curso a SQL Server™ 2000. La versión de PostgreSQL para este sistema operativo se puede descargar de `http://www.postgresql.org/ftp/binary/v8.0.3/win32/` e incluye herramientas adicionales (por ejemplo pgAdmin III). Su instalación es relativamente más sencilla que en Linux, dado que se cuenta con un archivo ".msi" el cual se debe ejecutar. Solo hay que recordar que, al igual que en Linux se debe contar con un usuario distinto del administrador, pero con privilegios semejantes, para lograr con éxito el proceso de instalación.

Posterior a la instalación de PostgreSQL en Windows©, hay que exportar los datos desde PostgreSQL en Linux y transportar estos archivos para cargarlos posteriormente en Windows©. Esta tarea se facilita ampliamente con la utilización de pgAdmin III (ver Apéndice F), el procedimiento consiste en realizar una copia de la base de datos en PostgreSQL sobre Linux (utilizando pgAdmin III), copiar esa información en un medio de almacenamiento temporal, y por último cargar la base de datos dentro de PostgreSQL en Windows© (nuevamente con pgAdmin III). Esto se ilustra en la figura E.16.

Figura E.16: Exportar/Importar tablas de la base de datos con pgAdmin III.

Una vez cargada la base de datos del curso dentro de PostgreSQL, se debe agregar dentro de la instancia del servidor SQL Server™ creada inicialmente. En primer lugar hay que crear el esquema relacional de la base y posteriormente hay que cargar los datos provenientes de la conexión ODBC creada anteriormente utilizado los servicios de transformación de datos (DTS).

La creación del esquema relacional puede efectuarse con el asistente de creación de tablas que aparece al seleccionar **"New → Table"** en el menú contextual sobre la base de datos recién creada y de la misma forma se puede invocar el asistente para importar datos al seleccionar **"All Task → Import Data"** del menú contextual para cada una de las tablas creadas. Es importante recordar que ya teniendo los datos del archivo '.backup' creado por PostgreSQL estos también se pueden cargar dentro de la base de datos en SQL Server™ al ejecutar las sentencias SQL contenidas en él[2] .

[2] Para realizar esto, se deben utilizar el 'Query Analyzer'(**"Microsoft SQL Server™→ Query Analyzer"**).

APÉNDICE F

pgAdminIII

pgAdmin III es un sistema que permite interactuar con el SMBD PostgreSQL de manera visual y simplifica el proceso de administración de una base de datos. Es un programa que se distribuye libremente bajo los términos de la Licencia Artística (Artistic Licence).

Es importante remarcar que si bien pgAdmin III facilita gran parte de las tareas administrativas, siempre es necesario tener a mano la posibilidad de ejecutar los comandos directamente en una terminal puesto que aun cuando la herramienta provee estás facilidades, no siempre es garantía el contar con la misma (recordando que esta se distribuye independiente de PostgreSQL[1]).

F.1. Obtención e instalación

El paquete de pgAdmi III se pude descargar de la siguiente dirección electrónica `http://www.pgadmin.org/download.php` y su instalación resulta ser muy directa, ya que basta con descargar el archivo rpm y desde el intérprete de comandos ejecutar lo siguiente:

```
#bash=>rpm -Uvh pgadmin*.rpm
```

El listado de las funcionalidades que posee el sistema es grande, pero únicamente se presentan las herramientas básicas.

[1] Caso a parte es la ditribución de la versión para Windows©, que si incluye a pgAdmin III

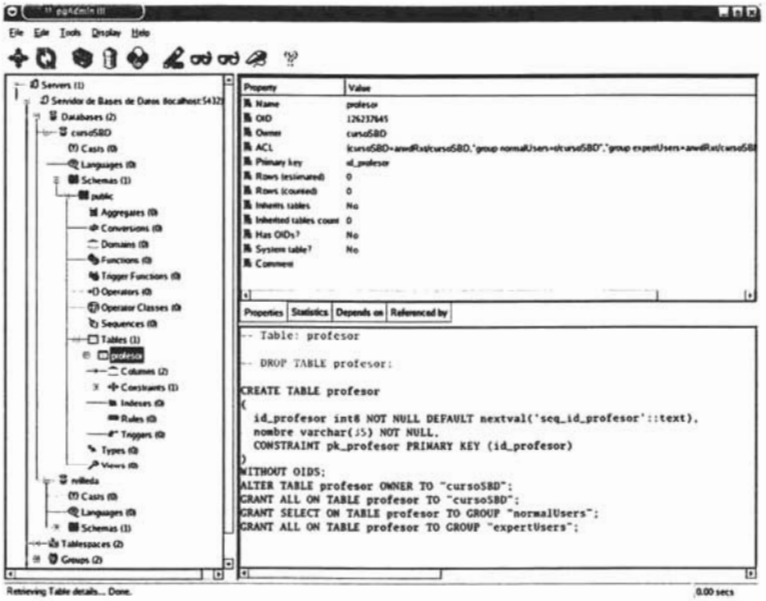

Figura F.1: Ventana principal de pgAdmin III.

F.2. La interfaz

La interfaz que maneja pgAdmin III es simple pero a la vez muy completa, consta principalmente de 4 elementos visuales.

En la pantalla principal se presenta la estructura de las bases de datos. Aquí es posible crear nuevos objetos, eliminarlos y editarlos para modificar algunas de sus características siempre y cuando el usuario en cuestión posea los privilegios para hacerlo dentro de la conexión a las bases de datos.

La parte izquierda de la pantalla presenta un árbol que contiene todos lo servidores a los cuales se puede conectar actualmente (de inicio no aparece alguno, se tiene que dar de alta como ve muestra posteriormente). La parte superior derecha muestra los detalles del objeto que se este seleccionando en el árbol de la izquierda. Algunos objetos pueden poseer características extras, y si es el caso se presentan en la pestaña de 'Statistics'.

La parte izquierda presenta una gran utilidad, que consiste en contener los guiones de SQL respectivos al objeto en cuestión. La linea de estado presenta información relativa al estado del servidor o de alguna operación que se encuentre realizando internamente así como del tiempo que tomo la última actividad del servidor. Se presenta una pantalla principal en la figura F.1.

Figura F.2: Ventana donde se agrega un servidor.

F.2.1. Agregando servidores

Antes de iniciar es necesario agregar un servidor de bases de datos, esta tarea es muy sencilla y se lleva a cabo por medio del menu contextual sobre el objeto 'Servers'(servidores) o mediante el menu general 'File'—→'Add server'.

Esta operación presenta la ventana para agregar el nuevo servidor y sus características. La ventana se presenta en la figura F.2. Luego de agregar el o los servidores deseados, cada uno se mostrará como un nodo a partir de la raíz del árbol 'Servers'.

Para abrir la conexión a un servidor, debe seleccionarse este y proceder con doble click. O utilizar del menu de 'Tools' la opción 'Connect'. La conexión se establecerá y las propiedades de los objeto más prioritarios se presentan. Si anteriormente se dio una conexión exitosa, pgAdmin III recuperará el estado anterior manteniendo la última selección.

Los datos que se deben suministrar son directos, solamente hay que tener especial atención en la dirección IP, el uso de SSL para las conexiones y el nombre de la base de datos inicial.

F.2.2. Herramienta de consultas de pgAdmin III

Esta herramienta permite ejecutar comandos SQL arbitrariamente. Todas las reglas expuestas se aplican aquí. Su ejecución se invoca a partir del icono con la imagen de un lápiz o del menu 'Tools'—→'Query Tool'. La ventana se presenta en la figura F.3.

La parte superior contiene la ventana de edición, donde se introducen los comandos. Estos, pueden leerse desde un archivo o escribirse a uno. Para ejecutar la consulta

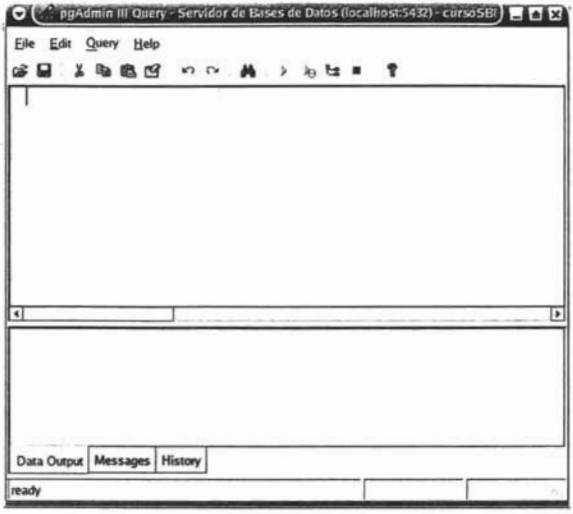

Figura F.3: Ventana principal del 'Query Tool'.

propiamente, debe seleccionarse 'Query'⟶'Execute' o presionar el botón de ejecución de la barra de herramientas o 'F5'. Se puede determinar ejecutar solamente cierta parte del código escrito en la ventana de edición, para ello hay que seleccionar el código deseado y ejecutarlo normalmente.

En caso de que una consulta demore demasiado, se puede cancelar en cualquier momento seleccionado del menú principal 'Query'⟶'Cancel'.

Si se desea ayuda acerca de un comando particular dentro de la ventana de edición basta con seleccionarlo y luego selecionar 'SQL Help' del menu de ayuda ('Help').

La parte inferior deplegará los resultados de la ejecución de la consulta. Estos resultados se pueden guardar utilizando la opción del menú 'File'⟶'Export'. También están las pestañas que permiten recuperar el historial de todos los comandos enviados así como la pestaña de que contiene los mensajes que regresa el sistema en relación a una consulta. La barra de estado muestra el tiempo de ejecución correspondiente a la última consulta ejecutada.

F.2.3. Herramientas administrativas

Mantenimiento

Esta herramienta permite dar mantenimiento a distintos objetos de las bases de datos. En general es recomendable realizar estas operaciones sobre la base de datos regularmente a modo de mantener las estadísticas lo más actuales posibles.

Las opciones de mantenimiento principal son `VACUMM`, el cual lee la base de datos

Figura F.4: Herramienta de mantenimiento.

o tabla en cuestión, tales que no se utilizan mas, la función general es similar a un recolector de basura; **ANALYZE** investiga los valores estadísticos acerca de la base de datos o tabla seleccionada, esto permite al optimizador escoger mejores planes de ejecución y con ello mejor rendimiento general. La última opción es **REINDEX** lo cual reconstruye los índices establecidos en la base de datos o en las tablas de la misma, actualizando valores y mejorando también su rendimiento.

Respaldo

El dialogo de respaldo presenta de manera simplificada una interfaz para la herramienta de PostgreSQL 'pg_dump'. Aquí se puede crear copias de respaldo de tablas, esquemas o incluso bases de datos completas. La extensión que tienen los archivos creados es '.backup' si el contenido de estos archivos se almacena en formato de texto sencillo o bien en archivos '.tgz' si se elige activar la compresión de datos. En ambos casos la información contenida son instrucciónes SQL.

La única restricción que presentan estos archivos consiste en no poder respaldar bases de datos mayores de 8 GB dentro de archivos con compresión ('.tgz').

Permisos

El asistente de permisos es una herramienta que permite definir los permisos que posee cada usuario dentro de la base de datos, que objetos puede utilizar y con que privilegios. El asistente permite asignación de un conjunto de privilegios a grupos y usuarios de múltiples objetos (tablas, secuencias, vistas y funciones) en una forma conveniente. La pestaña de selección permite seleccionar aquellos objetos a los cuales se les modifican los privilegios de acceso. La pestaña de privilegios define propiamente los privilegios que deben otorgarse.

Figura F.5: Herramienta para la creación de copias de respaldo.

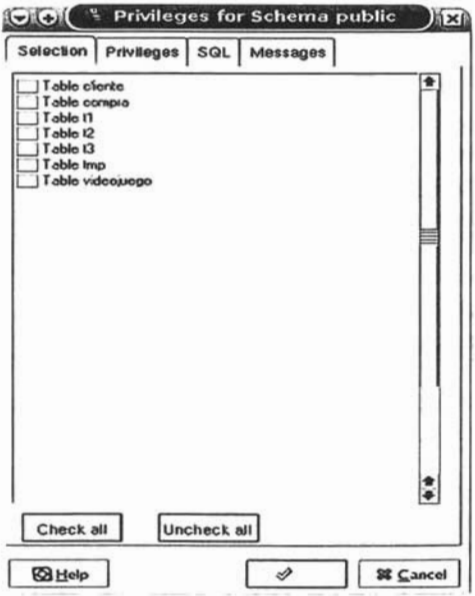

Figura F.6: Asistente de permisos.

Bibliografía

[1] Henry F. Korth Abraham Silberschatz and S. Sudarshan. *Database Systems Concepts.* McGraw-Hill, fourh edition, 2001.

[2] Scott W. Ambler. *Agile Database Techniques.* John Wiley & Sons, 2003.

[3] A. W. Burks, H. H. Goldstine, and J. von Neumann. Preliminary discussion of the logical design of electronic computing instrument. *Report prepared for the U.S. Army Ord. Dept.*, 1(1), 1946.

[4] Pelaez Valdez Canek. Prácticas de laboratorio para el curso 'introducción a ciencias de la computación'. `Tesis de Licenciatura.` 2002.

[5] M. C. José Galaviz Casas. Arquitectura de computadoras. *Vínculos Matemáticos - Facultad de Ciencias UNAM*, 1(10):100–101, 2002.

[6] Date C.J. *An Introduction to Database Systems.* Addison Wesley, 2003.

[7] Date C.J. and Darwen H. Into the great divide. *Relational Database Writings 1989-1991*, 1992.

[8] E. F. Codd. Further normalization of data base relational model. *Data Base Systems, Courant Computer Science Symposia*, Series 6, 1972.

[9] Kalen Delaney. How to conduct high-powered analyses with simple operations in sql server 6.5.
`http://www.windowsitpro.com/Articles/Print.cfm?ArticleID=5104.`

[10] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison Wesley, 2003.

[11] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database Systems: The Complete Book.* Prentice Hall, 2001.

[12] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *IBM Research*, 1(1), 1996.

[13] PostgreSQL Global Development Group. Loading the driver. `http://jdbc.postgresql.org/documentation/80/load.html`.

[14] PostgreSQL Global Development Group. Postgresql 8.0: Controlling the planner with explicit join clauses. `http://www.postgresql.org/docs/8.0/static/explicit-joins.html`.

[15] PostgreSQL Global Development Group. Postgresql 8.0: Pattern matching. `http://www.postgresql.org/docs/8.0/static/functions-matching.html`.

[16] PostgreSQL Global Development Group. Postgresql 8.0: Select. `http://borg.postgresql.org/docs/8.0/static/sql-select.html`.

[17] PostgreSQL Global Development Group. Postgresql 8.0: Sql conformance. `http://www.postgresql.org/docs/8.0/static/features.html`.

[18] PostgreSQL Global Development Group. Postgresql 8.0: Transaction isolation. `http://www.postgresql.org/docs/8.0/interactive/transaction-iso.html`.

[19] PostgreSQL Global Development Group. Running and installing postgresql on native windows faq. `http://pginstaller.projects.postgresql.org/FAQ_windows.html`.

[20] IBM. Open source : Jfs project web site. `http://jfs.sourceforge.net/`.

[21] Joe Knapka. Outline of the linux memory management system. `http://home.earthlink.net/~jknapka/linux-mm/vmoutline.html`.

[22] Victor M. Matos and Rebecca Grasser. A simpler (and better) sql approach to relational division. *Information Systems Education*, 13:85–88, 1992.

[23] Microsoft. Sql server home. `http://www.microsoft.com/sql/default.mspx`.

[24] NAMESYS. Reiserfs. `http://www.namesys.com/`.

[25] Carlos Ordonez and Javier Garcia-Garcia. Referential integrity quality metrics. Submitted.

[26] David A. Patterson, Garth Gibson, and Randy H Katz. A case for redundant arrays of inexpensive disks (raid). *Computer Science Division - University of California, Berkeley*, 1(1), 1987.

[27] Alfons González Pérez. *SQL Server - Programación y administración.* Alfaomega, first edition, 1999.

[28] Ramakrishnan R. and Gehrke J. *Database Management Systems 2nd Edition.* McGraw-Hill, 2000.

[29] Thomas Schwarz. Coen 180 - memory hierarchy. `http://www.cse.scu.edu/~tschwarz/coen180_04/LN/MemoryHierarchy.html`.

[30] SGI. Sgi - developer central open source — linux xfs. `http://oss.sgi.com/projects/xfs/`.

[31] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Optimizing join orders. 1993.

[32] Megan Thomas. University of california, berkeley - cs 186: Introduction to database systems.

[33] Stephen Tweedie. Ext3, journaling filesystem. `ftp://ftp.kernel.org/pub/linux/kernel/people/sct/ext3/`.

[34] Mark Allen Weiss. *Data Structures and Problem Solving Using Java.* Addison Wesley, 2005.

Índice alfabético

A

Autenticación de usuarios 91
métodos de 93

C

Consultas
anidamiento de 66
hacer ..... *ver* SQL - SELECT 45

D

DDS
Decision Support Systems 231
Dia 14

E

Expresiones regulares
POSIX 54

F

Formatos gráficos 15
eps 15
png 15

I

Indices
clasificación 160
concepto 160
Integridad
de dominio 77
de entidad 76
de usuario 79
definición 76
referencial 79

J

JDBC
acceso a bases de datos 113
controladores 114
definición 112
modelo de dos capas 113
modelo de tres capas 113
proceso general de conexión .. 115–120
Jerarquía de memoria 128
memoria virtual 128

L

Lenguaje de 24
Control de Datos ... *ver* SQL - dcl 24
Definición de Datos. *ver* SQL - ddl 24
expresiones de múliples dimensiones.. *ver* MDX 239
Manipulación de Datos *ver* SQL - dml 24
Llaves
foráneas 81

M

MDX
definición 239
Modelo Entidad-Relación 13
diagramas Entidad-Relación 13

N

Normalización

concepto 100

O

ODBC 111
OLAP
agregaciones 235
cubos de múltiples dimensiones ... 233
definición 233
drill-down 235
OLTP
definición 233

P

Perl 129
manejo de archivos 130–133
PostgreSQL 6
índices compuestos 176
índices en 161
niveles de aislamiento en 193
aislamiento de 202
bloqueos explícitos 205
nivel de tabla 206
nivel de tupla 207
comandos
createdb 9
createuser 9
concurrencia en 202
configuración URL - JDBC 115
deadlocks 207
instalación 6
parámetros de configuración 165
postmaster 7
psql 8
seguridad en 86
tipos de datos 25
wal 216
Procesamiento por lote 29

R

RAID
definición 138
nivel 0 138
nivel 1 139
nivel 5 139
por hardware 140
por software 140
response time 223

S

scaleup ..... *ver* aceleración 224
Sistemas de archivos
con journaling 152
definición 148
sin journaling
ext2fs 150
speedup ..... *ver* escalamiento 224
SQL 24
`ALTER GROUP` 88
`ALTER TABLE` 28
`ANALYZE` 162
`BEGIN TRANSACTION` 190
`CHECK` 78
`COMMIT` 190
`COPY` 37
`CREATE`
`GROUP` 88
`INDEX` 161
`TABLE` 25
`USER` 87
`VIEW` 69
`CUBE` 240
dcl 24
ddl 24
definición 24
`DELETE` 38
`DISTINCT` 47
división 67
dml 24
`DROP`
`INDEX` 162
`TABLE` 27
estándar 2003
conformación 24
`EXPLAIN` 163
`FOREIGN KEY` 81

FROM ..... *ver* SQL - SELECT 46
funciones escalares
avg() 50
count() 50
max() 50
min() 50
sum() 50
GRANT 89
GROUP BY 60
GROUPING 242
HAVING 61
INSERT 36
IS NULL 52
JOIN
CROSS JOIN 63
FULL OUTER JOIN 65
LEFT OUTER JOIN 65
NATURAL JOIN 63
RIGHT OUTER JOIN 65
LIKE 52
NULL 76
operaciones entre conjuntos
EXCEPT 51
INTERSECT 51
UNION 51
ORDER BY 49
REVOKE 90
ROLL UP 240, 243
ROLLBACK TO 191
SAVEPOINT 191
SELECT 46
SIMILAR TO 52
UNIQUE 76
UPDATE 39
vistas 68
WHERE 47
SQL Server™ 2000 234
componentes 278
historia 278
instalación 280

T

throughput 223
Transacciones
niveles de aislamiento 192
concepto 187
propiedades 188

U

UML 16
definición 16
diagramas 16
diagrama de clases 17
tipos de 16