PostgreSQL

Patricio Denzer

23 de octubre de 2002

Resumen

La idea de este trabajo es introducir los conceptos fundamentales acerca del diseño y modelado de bases de datos usando PostgreSql, adquirir los conocimientos teóricos y prácticos en manejo de bases de datos relacionales, y el lenguaje SQL. Además pretende proporcionar la documentación necesaria para iniciar el estudio de bases de datos sin conocimiento previo acerca del tema, presentando algunas situaciones claves que permitan dar una visión amplia y analizar las ventajas y dificultades que presenta este sistema en comparación con sus pares, para poder decidir al momento de iniciar el estudio en esta cada día más necesaria e importante área.

1. Introducción

Entre los sistemas de bases de datos existentes hoy en día, PostgreSQL juega un papel muy importante ya que es un sistema que tiene muchas cualidades que lo hacen ser una muy buena alternativa para instalar sistemas en empresas, universidades y una gran cantidad de otras aplicaciones. Este documento está pensado como un material práctico de introducción a los sistemas de bases de datos relacionales basados en PostgreSQL y no profundiza mayormente en los conceptos, aunque aborda una gran parte de los temas necesarios para iniciar el iniciar de buena forma el estudio, sin descuidar ningún aspecto. Además la información se ilustra por medio de varios ejemplos que permiten entender más a fondo los conceptos. Gran parte de la información que aquí se encuentra fué obtenida de la documentación oficial de PostgreSQL, aunque también de algunos libros y manuales, de donde se trató de obtener la mayor cantidad de ideas y ponerlas en un documento que fuera fácil de entender y que lograra el objetivo de dar una visión global acerca del sistema de bases de datos y en un tamaño reducido.

2. ¿Que es PostgreSQL

PostgreSQL es un avanzado sistema de bases de datos relacionales basado en Open Source. Esto quiere decir que el código fuente del programa está disponible a cualquier persona libre de cargos directos, permitiendo a cualquiera colaborar con el desarrollo del proyecto o modificar el sistema para ajustarlo a sus necesidades. PostgreSQL está bajo licencia BSD.

Un sistema de base de datos relacionales es un sistema que permite la manipulación de acuerdo con las reglas del álgebra relacional. Los datos se almacenan en tablas de columnas y renglones. Con el uso de llaves, esas tablas se pueden relacionar unas con otras.

2.1. Ideas Básicas acerca del funcionamiento

En la jerga de bases de datos, PosgreSQL usa el modelo cliente/servidor. Una sesión en PostgreSQL consiste en ejecución de los siguientes procesos.

- El servidor, que maneja archivos de bases de datos, acepta conecciones a las aplicaciones cliente, y realiza acciones en la base de datos. El programa servidor de bases de datos se conoce como `postmaster`
- La aplicación cliente, que necesita realizar operaciones en la base de datos. Las aplicaciones cliente pueden ser de la más diversa naturaleza: pueden ser aplicaciones de texto en una consola, aplicaciones gráficas, un servidor web que accede a la base de datos para mostrar una página, o herramientas especializadas de mantenimiento de bases de datos.

Como es habitual en las aplicaciones cliente/servidor, el cliente y el servidor pueden estar en diferentes máquinas. En este caso, estos se comunican sobre una conexión de red TCP/IP.

El servidor PostgreSQL puede manejar múltiples conecciones concurrentes de los clientes. Para esto inicia un nuevo proceso ("fork") para cada conexión llamado `backend`. Con esto, el cliente y el nuevo proceso del servidor se comunican sin la intervención del proceso original del `postmaster`. Así, el `postmaster` está siempre corriendo, esperando por conexiones de parte de los clientes Todo esto por supuesto es invisible para el usuario y se menciona acá solo como un comentario.

¿Que es una base de datos relacional?
Una base de datos relacional desde el punto de vista del usuario podemos decir que es como una colección de tablas interrelacionadas que permiten almacenar información para que esta pueda ser utilizada posteriormente, y se basa en el modelo de datos relacional para la manipulación de las tablas, el que a su vez se basa en elementos de la teoría de conjuntos para establecer las relaciones.

¿Que es una consulta?
Una consulta es una petición de información que se hace a la base de datos, la que se implementa de acuerdo a ciertas reglas e instrucciones que provee el lenguaje SQL y que permite ver y manipular datos que se encuentran en el sistema.

3. PostgreSQL v/s sus pares

A continuación se muestra una tabla con algunas características de tres importantes sistemas de Bases de Datos. Aunque no son las versiones más recientes, casi la totalidad de las características que allí aparecen concuerdan con lo que son las últimas versiones, y es una buena referencia para conocer aspectos de los tres sistemas.

Sistema	MySQL	PostgreSQL	SAP DB
Versión	Mysql-3.23.41	PostgreSQL 7.1.3	SAP DB Version 7.3
Licencia	GPL	BSD	GPL
Cumplimiento con estándar SQL	Media	Alta	-
Velocidad	Media/Alta	Media	-
Estabilidad	Alta / Muy Alta	Alta	-
Integridad de datos	NO	Si	Si
Seguridad	Alta	Media	-
Soporte de LOCKING y CONCURRENCIA	Media	Alta	-
Soporte de Vistas	No (Planeada v4.2)	Si	Si
Soporte Subconsultas	No (Planeada v4.1)	Si	Si
Replicacion	Si	Si	-
Procedimientos almacenados	No	Si	Si
Soporte Unicode	NO	Si	-
Soporte Disparadores	No	Si	Si
Integridad referencial	No	Si	Si
Interfaces de programación	ODBC, JDBC, C/C++, OLEDB, Delphi, Perl, Python, PHP	ODBC, JDBC, C/C++, SQL embebido (en C), Tcl/Tk,Perl, Python, PHP	ODBC ,JDBC, C/C++, Precompilado(SQL Embebido),Perl, Python, PHP
Tipos de Tablas alternativas	ISAM, MYISAM, BerkeleyDB, InnoDB, HEAP, MERGE, Gemini	PostgreSQL mantiene su propio sistema de tipos de tablas	-
Transacciones	si	Si	-
Claves foráneas	NO (Planeado v4.0)	Si	-
Backups en caliente	Si	Si	-

3.1. Ventajas de PostgreSQL

PostgreSQL se caracteriza por ser un sistema estable, de alto rendimiento, gran flexibilidad ya que funcionar la mayoría de los sistemas Unix, además tiene características que permiten extender fácilmente el sistema. PostgreSQL puede ser integrada al ambiente Windows permitiendo de esta manera a los desarrolladores, generar nuevas aplicaciones o mantener las ya existentes. Permite desarrollar o migrar aplicaciones desde Access, Visual Basic, Foxpro, Visual Foxpro, C/C++ Visual C/C++, Delphi, etc., para que utilicen a PostgreSQL como servidor de BD; Por lo expuesto PostgreSQL se convierte en una gran alternativa al momento de decidirse por un sistema de bases de datos.

4. Instalación de PostgreSQL

Los requerimientos mínimos con que debe cumplir una máquina para poder instalar PostgreSQL son:

- 8 megabytes de Memoria RAM
- 30 megabytes de espacio en disco duro para el cogido fuente
- 5 megabytes de espacio en disco duro para la instalación de los ejecutables
- 1 megabyte extra para las bases de datos básicas
- 3 megabytes de espacio en disco duro para el tarball con el codigo fuente

Para chequear el espacio en disco podemos usar el comando: df -k

Lo primero que debemos hacer es crear la cuenta del superusuario de PostgreSQL, normalmente se usa por defecto como nombre de usuario "postgres". Este usuario debe ser un usuario común del sistema, sin privilegios de root, esto reduce considerablemente los riesgos de inseguridad. En la secuencia de abajo se detalla el procedimiento para esto:

```
# adduser postgres
# passwd postgres
```

Toda la instalación del software y configuración se debe hacer desde la cuenta postgres.

El proceso de compilación es idéntico a cualquier otro programa. En esta caso la documentación recomienda una secuencia algo distinta pero con el mismo resultado.

```
# gmake all >& make.log &
# tail -f make.log
```

Para instalar los binarios debemos realizar lo siguiente:

```
# cd /usr/src/pgsql/src
# gmake install >& make.install.log &
# tail -f make.install.log
```

Luego de esto debemos instalar la documentación desde el directorio pgsql/postgresql-6.5/docτ ejecutar:

```
# make install
```

4.1. PostgreSQL y las distribuciones de Linux

Cuando PostgreSQL se instala junto con una distribución de Linux, por lo general vienen hechas casi totas las tareas de configuración, las carpetas del sistema ya están creadas y el `Superuser` también, por lo general con el nombre `postgres` como mencionamos anteriormente.

5. Empezando con PostgreSQL

El *Administrador de la base de datos* es el usuario que instaló el software, creó los directorios e inició el proceso postmaster que como mencionamos anteriormente es el demonio que permite a los usuarios interactuar con el sistema. Este usuario no tiene que ser el administrador del sistema operativo ó superusuario, aunque a veces en algunos documentos aparece como `Superuser`, refiriéndose el que es el superusuario del sistema de bases de datos, nó el super usuario de sistema operativo. Este `Superuser` no tiene permisos especiales en el sistema operativo. Varios de los pasos para usar el sistema los puede realizar cualquier usuario, pero otros los debe realizar el administrador de la base de datos. El nombre de este usuario suele ser `postgres` ó `pgsql`.

5.1. Configurando el entono

Como ya dijimos, postgreSQL es una aplicación cliente/servidor, y el usuario sólamente necesita tener acceso a la parte cliente, por ejemplo el programa `psql` que es el que nos permite interactuar con el sistema. Vamos a asumir que Postgres se instaló en el directorio `/var/lib/pgsql`, y todos los programas de Postgres se instalarán en este caso en el directorio `/var/lib/pgsql/bin`. Este último directorio debemos agregarlo al `PATH` de nuestro sistema. Si estamos usando batch, ksh ó sh debemos agregar lo siguiente a nuestro archivo `.profile`:

```
PATH=/usr/local/pgsql/bin:$PATH
export PATH
```

5.2. Administrando una Base de datos

Para empezar a trabajar, vamos a suponer que se ha iniciado correctamente el proceso `postmaster`, y que se ha creado previamente el usuario `postgres`. La mayoría de las aplicaciones Postgres asumen que el nombre de la base de datos, si no se especifica, es el mismo que el de su cuenta en el sistema.

Lo primero que vamos a hacer es ingresar como el usuario `postgres`.

```
[postgres@localhost pgsql]# su postgres
```

5.3. Creación de una base de datos

Si queremos crear una base de datos llamada `mydb` hacemos lo siguiente desde la consola unix:

```
[postgres@localhost pgsql]# createdb mydb
```

notar que aún no hemos ingresado al sistema, es decir, no hemos ejecutado el monitor interactivo `pgsql`. Los nombres de las bases de datos pueden contener hasta 32 caracteres y deben comenzar por un caracter alfabético. Con PostgreSQL podemos crear un número ilimitado de bases de datos y el usuario que

las crea automáticamente será el administrador de las bases de datos que creó. No todos los usuarios están autorizados para ser administradores de bases de datos.

5.4. Accediedo a una base de datos

Existen dos formas de acceder a una base de datos:

- Mediante el programa de monitorización de Postgres llamado `psql`, o algún otro programa de monitorización.
- Mediante un programa en C, usando la librería de subrutinas `LIBPQ`, que permite enviar y recibir instrucciones SQL desde el programa creado por el usuario.

Si queremos acceder a la base de datos que creamos anteriormente mediante el programa de monitorización `psql`, hacemos lo siguiente:

```
[postgres@localhost pgsql]# psql mydb
```

al hacer esto veremos lo siguiente:

```
Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT for copyright terms of POSTGRESQL

    type \? for help on slash commands
    type \q to quit
    type \g or terminate with semicolon to execute query
 You are currently connected to the database: template1

mydb=#
```

ahora el programa está listo para recibir instrucciones SQL. `psql` responde a los códigos de escape que empiezan con el caracter `\`. Los comandos que aparecen al inicio de la sesión en `psql` son:

`\copyright`	muestra los términos de la distribución
`\h`	muestra la ayuda acerca de los comandos SQL
`\g`	termina la ejecución de una consulta. Equivale a usar ';'
`\q`	para salir del programa.

Para eliminar una base de datos usamos lo siguiente:

```
[postgres@localhost pgsql]# dropdb mydb
```

6. El Lenguaje de consultas de PostgreSQL

El lenguaje de consultas de PostgreSQL es una variación del lenguaje SQL estándar SQL, y son extensiones que propias de Postgres. Para ver los comandos propios de Postgres debemos usar `\?`.

Como Postgres es orientado al obejeto, la idea fundamental es la de una *clase*, donde todas las instancias de esa clase tienen los mismos atributos y cada atributo es de un tipo específico. Además, cada instancia posee un *identificador de objeto* único. La relación que se hace entre SQL y el modelo de programación orientada a objetos es como sigue: una tabla corresponde a una clase, una fila corresponde a una instancia de una clase y las columnas a los atributos.

6.1. Creación de una nueva Clase

Para crear una clase debemos especificar el nombre de la clase, además de los nombre de los atributos y sus tipos de la siguiente forma:

```
CREATE TABLE guitarras(
   marca         varchar(20),
   precio        int,
   num_frets     int,
   num_cuerdas   int,
   origen        varchar(30)
);
```

debemos cuidarnos del hecho de que el sistema distingue entre mayúsculas y minúsculas. Los tipos que soporta Postgres son los siguientes: int, float, real, smallint, char(N), varchar(N), date, time y timestamp además de otros de propósito general y otros con tipos geométricos. En realidad, la forma de escribir esto en el monitor interactivo de PostgreSQL es en una sola linea

```
CREATE TABLE guitarras(marca varchar(20),precio int, ...
num_frets int, num_cuerdas int, origen varchar(30));
```

6.2. Llenar una Clase con instancias

La declaración `insert` es para llenar una clase con instancias, es decir, desde el punto de vista de SQL es llenar una tabla con datos. Su sintaxis es la siguiente:

```
INSERT INTO guitarras
   VALUES('Gibson', 650000, 22, 6, 'USA');
```

Para cargar cantidades de datos mayores desde archivos ASCII podemos usar el comando `copy` lo que es mucho más rápido porque se cargan en una tabla todos los datos de una sola vez desde el archivo. Lo mismo para la escritura hacia un archivo. Esto se realiza de la siguiente forma:

```
COPY guitarras FROM '/var/lib/pgsql/archivo.txt'
    USING DELIMITERS '|';
```

Se poner cuidado en el hecho de que se debe especificar la ruta y nó solo en nombre del archivo. Además, la ruta del archivo debe ser accesible para el proceso `backend` que se encuentra en el servidor donde se está ejecutando Postgres, porque es él quién va a acceder al archivo.

6.3. Consultar una Clase

Para realizar las consultas de una clase usamos la función `select` que se explica en la sección SQL de en la página 19 del APENDICE en la página 15. Ilustraremos esto con algunos ejemplos.

1. Ver toda la tabla

   ```
   SELECT * FROM guitarras;
   ```

   El signo `*` significa que debe entregar todos los datos de la tabla `guitarras`. El resultado de esto es el siguiente

   ```
        marca      | precio | num_frets | num_cuerdas | origen
   ----------------+--------+-----------+-------------+--------
    Samick         | 120000 |        21 |           6 | Taiwan
    Ibanez         | 780000 |        24 |           6 | USA
    Fender         | 200000 |        22 |           6 | USA
    Gibson Les Paul | 650000 |        22 |           6 | USA
   (4 rows)
   ```

2. Especificando una expresión. La siguiente tabla nos entrega la marca de la guitarra y el precio en dólares.

   ```
   SELECT marca,precio/750 AS us,origen FROM guitarras;
   ```

   al usar `precio/750 AS us` estamos diciendo que queremos que divida el atributo `precio` de la tabla `guitarras` por 750 y que lo entregue ahora bajo el nombre de us (se refiere a $US). El resultado es el siguiente

   ```
         marca      |  us  | origen
   -----------------+------+--------
    Samick          |  160 | Taiwan
    Ibanez          | 1040 | USA
    Fender          |  266 | USA
    Gibson Les Paul |  866 | USA
    (4 rows)
   ```

3. También podemos usar operadores lógicos en nuestras consultas. Supongamos que queremos una guitarra proveniente de USA, pero que cueste menos de $300.000. Entonces hacemos lo siguiente:

```
SELECT marca,precio,origen FROM guitarras WHERE origen='USA'
       and precio<=300000;
```

   El resultado de esto es el siguiente

```
marca   | precio | origen
--------+--------+--------
 Fender | 200000 | USA
(1 row)
```

6.4. Redireccionamiento de Consultas SELECT

Una característica muy útil de PostgreSQL es la posibilidad de redireccionar una consulta a una nueva clase. Es decir, en pocas palabras la salida de una consulta la ponemos en una nueva tabla de la siguiente manera:

```
SELECT * INTO TABLE nueva_tabla FROM tabla;
```

Ahora podemos realizar cualquier tipo de operación de las que hacemos normalmente sobre la clase `nueva_tabla`.

6.5. Joins entre Clases

Las Joins o uniones son un tipo de consulta que accede a múltiples instancias de las mismas o diferentes clases a la vez, donde todas ellas son procesadas al mismo tiempo. Por ejemplo, si queremos obtener todos los registros que están dentro de un cierto rango dado de otros registros.

- Supongamos que tenemos 2 tipos de Guitarras, las eléctricas que son las que hemos estado usando hasta ahora, y las eléctroacústicas, que tienen las siguientes tablas respectivamente.

```
      marca      | precio  | num_frets | num_cuerdas | origen
-----------------+---------+-----------+-------------+--------
 Samick          |  120000 |        21 |           6 | Taiwan
 Ibanez          |  780000 |        24 |           6 | USA
 Fender          |  200000 |        22 |           6 | USA
 Gibson Les Paul |  650000 |        22 |           6 | USA
 Ibanez Jem777   | 1200000 |        24 |           7 | USA
 Maxtone         |  100000 |        21 |           6 | Taiwan
(6 rows)
```

```
  marca   | precio | num_frets | num_cuerdas | origen
----------+--------+-----------+-------------+--------
 Ovation  | 850000 |        21 |           6 | USA
 Takamine | 480000 |        21 |           6 | USA
 Yamaha   | 300000 |        20 |           6 | Taiwan
(3 rows)
```

Ahora vamos a elegir todas las guitarras que tengan 21 espacios en su fingerboard (frets), mediante la siguiente entrada

```
SELECT guitarra.marca, guitarra.precio,
guitarra.num_frets AS num_frets, electroacusticas.marca,
electroacusticas.precio,  electroacusticas.num_frets AS num_frets
FROM guitarra, electroacusticas
WHERE guitarra.num_frets= electroacusticas.num_frets;
```

Notar que lo anterior es una sola línea. El resultado de obtenemos es el siguiente

```
  marca  | precio | num_frets |  marca   | precio | num_frets
---------+--------+-----------+----------+--------+-----------
 Samick  | 120000 |        21 | Ovation  | 850000 |        21
 Samick  | 120000 |        21 | Takamine | 480000 |        21
 Maxtone | 100000 |        21 | Ovation  | 850000 |        21
 Maxtone | 100000 |        21 | Takamine | 480000 |        21
(4 rows)
```

6.6. Actualizaciones

`update` nos permite realizar actualizaciones de instancias que ya existen. Por ejemplo, si el precio de la guitarra Ibanez Jem777 bajó a $1.120.000, entonces podemos actualizar el precio de la siguiente manera

```
UPDATE guitarras SET precio=1120000 WHERE marca='Ibanez Jem777';
```

obteniendo como resultado

```
      marca      | precio  | num_frets | num_cuerdas | origen
-----------------+---------+-----------+-------------+--------
 Samick          |  120000 |        21 |           6 | Taiwan
 Ibanez          |  780000 |        24 |           6 | USA
 Fender          |  200000 |        22 |           6 | USA
 Gibson Les Paul |  650000 |        22 |           6 | USA
 Maxtone         |  100000 |        21 |           6 | Taiwan
 Ibanez Jem777   | 1120000 |        24 |           7 | USA
(6 rows)
```

6.7. Borrar

Para borrar registros de una tabla usamos el comando `DELETE`. Se debe tener especial cuidado en las consultas del tipo:

```
DELETE FROM clase
```

porque al hacer esto estamos borrando todas las instancias de la clase `clase` dejándola vacía, y no se pedirá confirmación para realizar esta operación.

6.8. Funciones de Conjuntos

Las funciones de conjuntos permiten obtener resultados a partir de múltiples filas de entrada, como por ejemplo contar, sumar, obtener un promedio, obtener un valor máximo o mínimo, etc. Las funciones SQL `WHERE` Y `HAVING` permiten filtrar los datos por columnas y filas respectivamente. Veamos algunos ejemplos de funciones de conjuntos

- obtener el precio de la guitarra eléctrica más cara

  ```
  SELECT max(precio) FROM guitarras;
  ```

  de donde obtenemos

  ```
     max
  ---------
   1120000
  (1 row)
  ```

- Obtengamos ahora el precio de la guitarra más económica proveniente de USA.

  ```
  SELECT min(precio) FROM guitarras WHERE origen='USA';
  ```

  lo que obtenemos es lo siguiente

  ```
    min
  --------
   200000
  (1 row)
  ```

7. Características SQL Avanzadas de Postgres

En esta sección veremos características avanzadas que distinguen a PostgreSQL del resto de los gestores de bases de datos.

7.1. Herencia

Cuando tenemos una clase y creamos otra clase derivada de ella, se dice que esta última *hereda* los atributos de la clase base. Para ilustrar esto crearemos dos clases, la clase guitarras" que es la clase base y la clase guitarra eléctrica" que sería su derivada, la cuál tendrá los atributos específicos que le correspondan, además de los que hereda de la clase base. Para crear una clase que deriva de otra hacemos lo siguiente

```
CREATE TABLE guitarra_electrica(num_capsulas int,
color varchar(20),
microafinacion char(10)) INHERITS (guitarras);
```

ahora si examinamos la tabla completa de la clase `guitarra_electrica` veremos los siguiente

```
 marca | precio | num_frets | num_cuerdas | origen | num_capsulas |
-------+--------+-----------+-------------+--------+--------------+...

    color | microafinacion
...-------+---------------

(0 rows)
```

Aparece la clase vacía como recién creada, pero además aparecen dentro de sus atributos los de la clase base `guitarra`.

7.2. Valores No Atómicos

Dentro de los principios del modelo de datos relaciónal que aparece en la página 15 se tiene que los atributos de una relación son atómicos. Esto quiere decir que la unidad mínima de datos es un atributo de una k-tupla. PostgreSQL no tiene esa restricción ya que los atributos pueden tener sub-valores, por ejemplo, podemos tener atributos que sean vectores que contengan algún tipo de dato base.

7.2.1. Vectores

En PostgreSQL los atributos pueden ser vectores multidimensionales de cualquiera de los tipos de datos existentes y su longitud puede ser fija o variable. Los elementos de un vector se agregan usándo paréntesis de llaves { } y separándolos mediante comas. Además, por defecto los elementos de los vectores comienzan en 1. Para acceder a los elementos de un vector se usan paréntesis cuadrados [ ].

Conclusiones

Al finalizar este trabajo se puede decir que se ha adquirido los conceptos fundamentales acerca del trabajo con bases de datos relacionales mediante el uso de PostgreSQL, se han aprendido aspectos de instalación, configuración, creación y administración de sistemas de bases de datos, así como también aspectos teóricos acerca de los fundamentos del modelo de datos relacional, sobre el cuál está basado el sistema de bases de datos y acerca de la forma en que interactúa el sistema de bases de datos con el sistema operativo. Después de haber realizado la investigación acerca del tema, existe mucha más claridad acerca de los aspectos a considerar al momento de decidirse a utilizar un sistema de bases de datos y de las ventajas y limitaciones que tiene este sistema y algunos de sus pares.

8. Bibliografía

1. PostgreSQL Introduction and Concepts, Bruce Momjian, Addison-Wesley

2. The PostgreSQL Tutorial Introduction , The Tutorial , Editado por Thomas Lockhart, 1998-10-01, The PostgreSQL Global Development Group.

3. The PostgreSQL Administrator's Guide , The Administrator's Guide ,Editado por Thomas Lockhart, 1998-10-01, The PostgreSQL Global Development Group.

4. The PostgreSQL Reference Manual.

Apéndice - Introducción a SQL

SQL es la abreviación de *Structured Querry Language*, que significa *Lenguaje de Consulta Estructurado* y es hoy en dia el lenguaje de consulta relacional más usado en bases de datos. A continuación veremos una introducción al modelo de datos relacional y luego su definición formal.

8.1. El Modelo de Datos Relacional

Como mencionamos anteriormente, una *base de datos relacional* es una base de datos que desde el punto de vista del usuario parece una colección de tablas. Una tabla consiste en filas y columnas, en las que cada fila representa un registro, y cada columna representa un atributo del registro contenido en la tabla. A continuación ilustraremos esto con un ejemplo.

SNO	SNAME	CITY
1	Smith	London
2	Jones	Paris
3	Adams	Vienna
4	Blake	Rome

Cuadro 1: SUPPLIER

PNO	PNAME	PRICE
1	Tornillos	10
2	Tuercas	8
3	Cerrojos	15
4	Levas	25

Cuadro 2: PART

SNO	PNO
1	1
1	2
2	4
3	1
3	3
4	2
4	3
4	4

Cuadro 3: SELLS

las tablas de PART y SUPPLIER podemos decir que son *entidades* y SELLS la *relación* que existe entre un articulo y su proveedor.

8.2. El Modelo de Datos Relacional. Definición Formal

El modelo de datos relacional se basa en el concepto matemático de *relación* perteneciente a la teoría de conjuntos. Una relación se dice que es un subconjunto del producto cartesiano entre una lista de dominios. Un dominio es un conjunto de valores, como por ejemplo, los números enteros, los números que están entre cero y diez, etc.

El producto cartesiano entre los dominios D_1 y D_2 es el conjunto de las k-tuplas v_1, v_2, $\ldots v_k$ tales que $v_1 \in D_1$, $v_2 \in D_2$, $\ldots v_k \in D_k$.

Ejemplo:
Sea $k = 2$, y los dominios:
$D_1 = \{0, 1\}$
$D_2 = \{a, b, c\}$
entonces el producto cartesiano entre los 2 dominios es:
$D_1 \times D_2 = \{(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)\}$

- Una relación es cualquier subconjunto del producto cartesiano entre los dominios. En el ejemplo anterior, una relación puede ser $\{(0, a), (0, b), (0, c)\}$
- Los miembros de una relación se llaman tuplas.
- Cada relación de algún producto cartesiano $D_1 \times D_2 \times \ldots \times D_k$ se dice que tiene nivel k y de este modo es un subconjunto de k-tuplas.

Ahora, si regresamos a la idea inicial de relación representada como una tabla, tenemos que las filas de la tabla representan las tuplas, y cada columna corresponde a una componente de la tupla. Se dice que las columnas de la tabla contienen los atributos.

Un *esquema relacional* R es un conjunto finito de atributos A_1, A_2, $\ldots A_k$. Hay un dominio D_i, para cada atributo A_i, $1 \leq i \leq k$, de donde se toman los valores de los atributos. Entonces escribimos es esquema relacional como $R(A_1, A_2, ...A_k)$.

Un *esquema relacional* es sólo un juego de plantillas mientras que una relación es un ejemplo de un esquema relacional. La relación consiste en las tuplas (y pueden ser vistas como una tabla); no así el esquema relacional.

Hasta ahora en los sistemas de bases de datos hemos hablado más de tipo de datos que de dominios. Cuando creamos una tabla, debemos decidir qué atributos vamos a incluir, y qué tipo de datos vamos a almacenar en los valores de los atributos. Al seleccionar el tipo de datos también estamos seleccionando un dominio para el atributo.

8.3. Operaciones en el Modelo de Datos Relacional

Ahora veremos qué podemos hacer con las tablas de un modelo de datos relacional, para recuperar algo desde una base de datos. Existen dos formas diferentes de notaciones para expresar las operaciones entre relaciones.

- El *Álgebra Relacional* es una notación algebraica, en la cuál las consultas se realizan aplicando operadores especializados a las relaciones.
- El *Cálculo Relacional* es una notación lógica, donde las consultas se expresan formulando algunas restricciones lógicas que las tuplas de la respuesta deban satisfacer.

8.3.1. Álgebra Relacional

El álgebra realacional consiste en una serie de operaciones con las relaciones.

- `SELECT`(σ): extrae de una relación las tuplas que satisfagan una restricción dada. Sea R una tabla que contiene un atributo A, $\sigma_{A=a}(R) = \{t \in R | t(A) = a\}$ donde t denota la tupla de R
- `PROJECT`(π): extrae *atributos* (columnas) específicos de una relación. Sea R una relación que contiene un atributo X, $\pi_X(R) = \{t(X) | t \in R\}$, donde $t(X)$ denota el valor del atributo X de la tupla t.
- `PRODUCT`($\times$): construye el producto cartesiano de dos relaciones. Sea R una tabla de rango (arity) k_1 y sea S una tabla con rango (arity) k_2. $R \times S$ es el conjunto de las $k_1 + k_2$-tuplas cuyos primeros k_1 componentes forman una tupla en R y cuyos últimos k_2 componentes forman una tupla en S.
- `UNION`($\cup$): supone la unión de la teoría de conjuntos de dos tablas. Dadas las tablas R y S (y ambas deben ser del mismo rango), la unión $R \cup S$ es el conjunto de las tuplas que están en R, S ó en las dos.
- `INTERSECT`($\cap$): Construye la intersección de la teoría de conjuntos de dos tablas. Dadas las tablas R y S, $R \cap S$ es el conjunto de las tuplas que están en R y en S. De nuevo requiere que R y S tengan el mismo rango.
- `DIFFERENCE`($-$ ó $\setminus$): supone el conjunto diferencia de dos tablas. Sean R y S de nuevo dos tablas con el mismo rango, $R - S$ es el conjunto de las tuplas que están en R pero no en S.
- `JOIN`(Π): conecta dos tablas por sus atributos comunes. Sea R una tabla con los atributos A,B y C y sea S una tabla con los atributos C,D y E. Hay un atributo común para ambas relaciones, el atributo C.

  $$R \Pi S = \Pi_{R.A,R.B,R.C,S.D,S.E}(\sigma_{R.C=S.C}(R \times S))$$

  ¿Qué estamos haciendo aquí?. Primero calculamos el producto cartesiano $R \times S$. Entonces seleccionamos las tuplas cuyos valores para el atributo común C sea igual ($\sigma_{R.C=S.C}$). Ahora tenemos una tabla que contiene el atributo C dos veces y lo corregimos eliminando la columna duplicada.

8.3.2. Cálculo Relacional de Tuplas

Existen 2 tipos de cálculo relacional, el cálculo relacional de dominios DRC y el cálculo relacional de tuplas TRC. Solo veremos este último porque es el que utilizan la mayor parte de los lenguajes relacionales. Las consultas utilizadas en TRC se realizan con el siguiente formato:

$$\{x(A)|F(x)\}$$

donde x es una variable de tipo tupla, A es un conjunto de atributos y F es una fórmula. La relación resultante está compuesta de todas las tuplas $t(A)$ que satisfacen $F(t)$.

8.3.3. Algebra Relacional v/s Cálculo Relacional

El Álgebra Relacional y el Cálculo Relacional son equivalentes en cierto sentido, ya que cualquier operación que se realice mediante el Algebra Relacional también puede realizarse mediante el Cálculo Relacional. Está demostrado que cualquier expresión del Cálculo Relacional puede ser expresada mediante el Algegra Realcional (E.F.Cood 1972).

El Cálculo Relacional es de más alto nivel que el Algebra Relacional, porque el álgebra especifica el orden de las operaciones y el cálculo nó, ya que es un compilador el responsable de determinar el orden adecuado en este último.

8.4. El Lenguaje SQL

SQL es un lenguaje basado en el Cálculo Relacional de Tuplas, pero ademas de esto existen algunas capacidades extra del lenguaje SQL que no son parte de las operaciones del modelo de datos relacional.

- Comandos para insertar, modificar y borrar datos
- Operaciones Aritméticas y comparaciones.
- Asignación y comandos de Impresión.
- Funciones agregadas, como promedio, suma, máximo, mínimo, etc, que se pueden aplicar a las columnas de una relación.

Algunos comandos SQL

- `SELECT` : es el comando más usado en SQL y se utiliza para recuperar datos. La sintaxis es la siguiente

```
SELECT [ALL|DISTINCT]
       { * | expr_1 [AS c_alias_1] [, ...
             [, expr_k [AS c_alias_k]]]}
FROM table_name_1 [t_alias_1]
     [, ... [, table_name_n [t_alias_n]]]
[WHERE condition]
[GROUP BY name_of_attr_i
          [,... [, name_of_attr_j]] [HAVING condition]]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
[ORDER BY name_of_attr_i [ASC|DESC]
          [, ... [, name_of_attr_j [ASC|DESC]]]];
```

  Para ver ejemplos de `SELECT` ver la sección consultar una clase en la página 8.

- `JOINS` : Las uniones, desde el punto de vista de SQL permiten unir dos tablas de acuerdo a los datos que ellas tienen en común. Para un ejemplo de unión, ver Joins entre clases en la página 9.

- Operadores Agregados.
  El lenguaje SQL tiene algunos operadores agregados que no son parte del modelo de datos relacional, como son la suma, el promedio, los valores máximo y mínimo, y que permiten obtener un resultado aplicando el operador a todos los valores de una columna. Por ejemplo, si tenemos una base de datos con marcas de monitores de computador, sus precios, y otros datos y queremos calcular cuál es el precio promedio de un monitor, de acuerdo a los valores que están en la base de datos, hacemos lo siguiente.

```
SELECT agv(precio) FROM monitores;
```

donde suponemos que `monitores` es la tabla que contiene los datos y `precios` es el atributo de la tabla que contiene los valores de venta.