ADMINISTRACION DE BASE DE DATOS

III. Configuración y administración del espacio en disco

3. Configuración y administración del espacio en disco

TEMAS

- 3.1. Definición de espacio de almacenamiento
- 3.2. Definición y creación del espacio asignado para cada base de datos
- 3.3 Asignación de cuotas de espacio para usuarios
- 3.4. Espacios para objetos de la base de datos
- 3.5 Roles

Competencia

- Planear, diseñar e implementar la organización del espacio en disco.
- Definir las fases de las instancias de un SGBD.
- Crear espacios de almacenamientos dinámicos

3. Configuración y administración del espacio en disco

Espacio en disco. Es la cantidad de espacio en disco que hace falta para los archivos de la base de datos. Normalmente, el diseñador querrá minimizar este espacio.

3. Configuración y administración del espacio en disco

Las bases de datos suelen ser creadas para almacenar grandes cantidades de datos de forma permanente. Por lo general, los datos almacenados en éstas suelen ser consultados y actualizados constantemente.

La mayoría de las bases de datos se almacenan en las llamadas memorias secundarias, especialmente discos duros, aunque, en principio, pueden emplearse también discos ópticos, memorias flash, etc.

Las razones por las cuales las bases de datos se almacenan en memorias secundarias son:

- En general, las bases de datos son demasiado grandes para entrar en la memoria primaria.
- La memoria secundaria suele ser más barata que la memoria primaria (aunque esta última tiene mayor velocidad).
- La memoria secundaria es más útil para el almacenamiento de datos permanente, puesto que la memoria primaria es volátil.

3. Configuración y administración del espacio en disco

Gestión de almacenamiento

Los datos que maneja un SGBD son almacenados físicamente en soporte físico que utilice (disco, unidad externa, etc.).
El administrador de bases de datos define el modelo de datos y es el SGBD el que se ocupa de guardar esos datos físicamente. Para ello, cuenta un componente informático de alta complejidad que se encarga de ello.

Las principales funciones de un gestor de almacenamiento son estas:

- Convertir consultas del usuario en operaciones lógicas en sistema de archivos físico.
- Controlar el búfer de la memoria principal, es decir, gestionar la memoria RAM del sistema para mejorar el rendimiento.
- Asegurarse de que las restricciones de integridad se cumplen.
- Sincronizar acciones simultáneas de varios usuarios
- Controlar los sistemas de copia de seguridad y recuperación.

3. Configuración y administración del espacio en disco

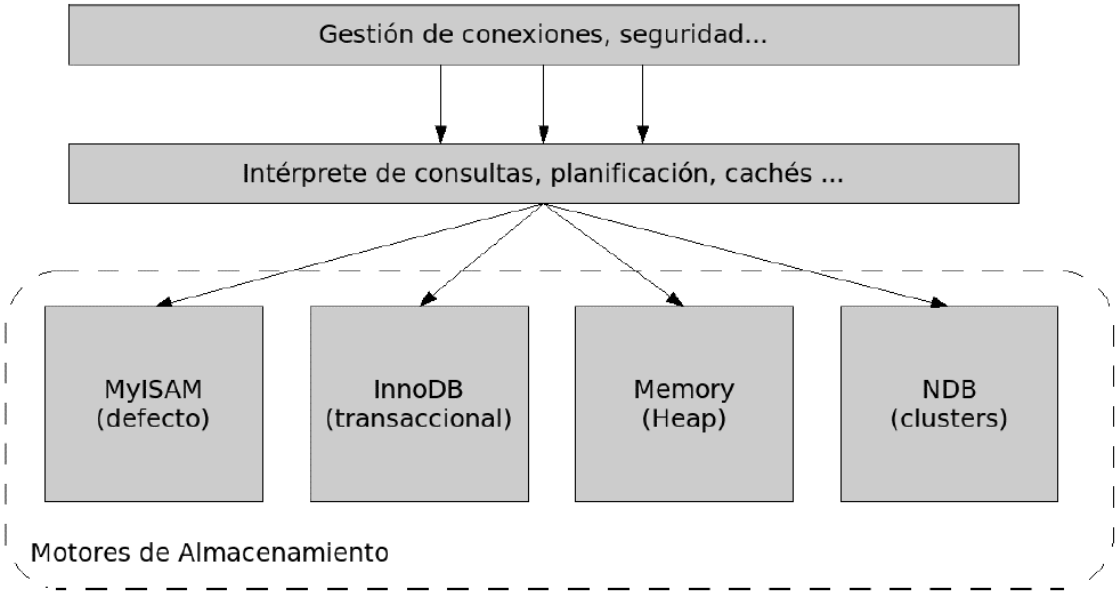
Podemos imaginar la arquitectura interna de MySQL dividida en tres capas. Se trata de una división lógica, que no coincide necesariamente con la división interna del código, pero nos ayudará a entender los conceptos.

Las tres capas son:

1. **Capa de Conexión:** En la que reside la funcionalidad que conecta MySQL con otros sistemas y lenguajes (APIs, sockets, ODBC, etc.)
2. **Capa de Lógica:** En la que reside la lógica para procesar consultas SQL (parseo de sentencias, planificación, ejecución, cachés, etc.)
3. **Capa de Almacenamiento:** En la que reside la lógica para almacenar y acceder a los datos de las tablas.

3. Configuración y administración del espacio en disco

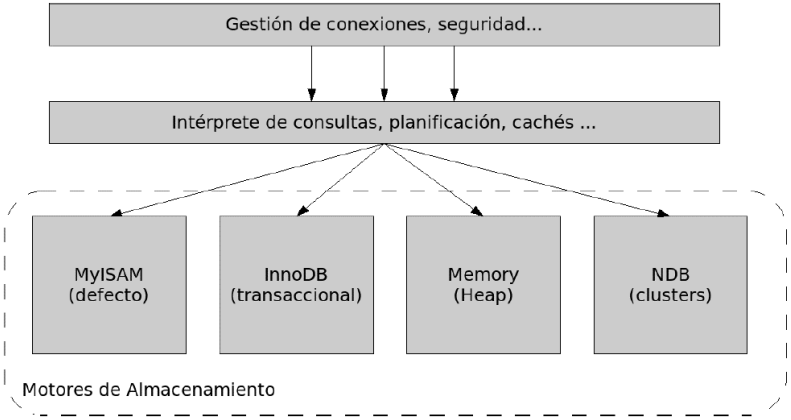
Una característica de MySQL es que puede utilizar distintos motores de almacenamiento. Es decir, que la capa de almacenamiento dispone de varios subniveles, uno por cada tipo de motor soportado. La siguiente figura nos aclarará un poco más las cosas:



3. Configuración y administración del espacio en disco

Podemos ver que MySQL soporta cuatro motores de almacenamiento básicos.
Hay que destacar que una base de datos puede almacenar sus datos utilizando una combinación de distintos motores. Es decir, parte de los datos pueden estar en una tabla MyISAM y otra parte en una tabla InnoDB.

Por lo que es posible utilizar distintos motores en una misma base de datos, incluso pueden ser utilizados en una misma transacción.



3. Configuración y administración del espacio en disco

Motores de Almacenamiento

Al diseñar una base de datos con MySQL será necesario decidir qué motores de almacenamiento vamos a utilizar. MySQL dispone, actualmente, de los siguientes motores de almacenamiento:

- **MyISAM:** Es el motor por defecto. Es muy rápido pero no transaccional.
- **InnoDB:** Es transaccional, incluyendo integridad referencial.
- **Memory (Heap):** Es una tabla MyISAM, pero almacenada en memoria, no en disco. Es todavía más rápida.
- **Archive:** Es una tabla MyISAM, pero comprimida y de sólo lectura.
- **MRG_MyISAM:** Es una agregación de tablas MyISAM. Las tablas agregadas deben ser exactamente iguales.

3.1. Definición de espacio de almacenamiento

- Los **archivos** se organizan lógicamente como secuencias de registros. Estos registros se corresponden con los bloques del disco. Los archivos se proporcionan como un instrumento fundamental de los sistemas operativos, por lo que se supondrá la existencia de un **sistema de archivos** subyacente. Hay que tomar en consideración diversas maneras de representar los modelos lógicos de datos en términos de archivos.

- Aunque los bloques son de un tamaño fijo determinado por las propiedades físicas del disco y por el sistema operativo, los tamaños de los registros varían. En las bases de datos relacionales las tuplas de las diferentes relaciones suelen ser de tamaños distintos.

3.1. Definición de espacio de almacenamiento

LOS ARCHIVOS EN ORACLE

Tipo de Archivos

Los Archivos de Datos(DAtafiles): Son los únicos que contienen los datos de los usuarios de la base de datos, sirven para el almacenamiento físico de las tablas, índices o agrupamientos y procedimientos.

Espacio de Tablas(Tablespaces): Unidades lógicas para el almacenamiento de los datos, permiten manejar o controlar el espacio en los discos.

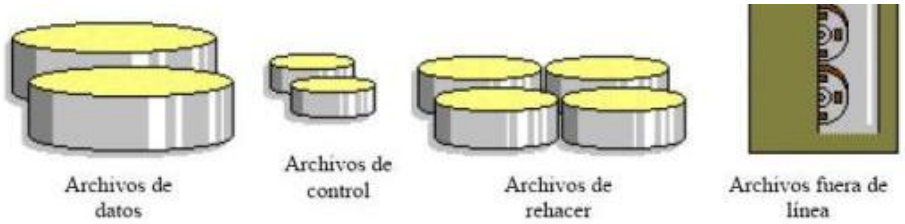


Ilustración 2. Tipos de archivos

3.1. Definición de espacio de almacenamiento

Archivos de control (COntrol Files)
Descripción física y dirección de los archivos de la BD y de los archivos de rehacer, en estos archivos se especifica que datafiles conforman la BD para poder tener el acceso a los datos o para recuperar la base de datos ante una falla. Los archivos de control se crea automaticamente con la orden CREATE DATABASE y no son editables, se actualizan automáticamente.

Archivos de rehacer(redo log files), posee los cambios hechos a la base de datos para la recuperación ante fallas o manejo de transacciones, el próposito es de servir de respaldo de los datos en la memoria RAM, debe estar conformado por dos grupos mínimoy cada grupo debe estar almacenado en discos separados.
El DBMS va sobrescribiendo sobre la información más vieja cuando se agota el espacio en estos grupos de archivos.

Archivos fuera de linea(archived files), son archivos opcionales donde se guarda información vieja de los archivos de rehacer son convenientes para los respaldos de la base de datos.

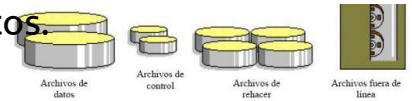


Ilustración 2. Tipos de archivos

3.1. Definición de espacio de almacenamiento

- Las bases de datos de SQL Server tienen tres tipos de archivos, tal como se muestra en la tabla siguiente.

Archivo	Descripción
Primario	**El archivo de datos principal incluye la información de inicio de la base de datos y apunta a los demás archivos de la misma**. Los datos y objetos del usuario se pueden almacenar en este archivo o en archivos de datos secundarios. Cada base de datos tiene un archivo de datos principal. La extensión recomendada para los nombres de archivos de datos principales es .mdf.
Secundario	Los archivos de datos secundarios **son opcionales, están definidos por el usuario y almacenan los datos del usuario**. Se pueden utilizar para distribuir datos en varios discos colocando cada archivo en una unidad de disco distinta. Además, si una base de datos supera el tamaño máximo establecido para un archivo de Windows, puede utilizar los archivos de datos secundarios para permitir el crecimiento de la base de datos. La extensión de nombre de archivo recomendada para archivos de datos secundarios es .ndf.
Registro de transacciones	Los archivos del registro de transacciones **contienen la información de registro que se utiliza para recuperar la base de datos.** Cada base de datos debe tener al menos un archivo de registro.La extensión recomendada para los nombres de archivos de registro es .ldf.

3.1. Definición de espacio de almacenamiento

Todo el sistema de permisos de acceso al servidor, a las bases de datos y sus tablas, MySQL lo almacena en una tabla llamada **mysql**, que como todas estará en el directorio **/data**, a menos que hallamos especificado otro directorio.
En Windows esta tabla se crea con la instalación, pero en Linux/Unix debemos crearla con:
/usr/local/mysql/bin/mysql_install_db

Los directorios **/include y /lib** contiene los fichero *.h y las librerias necesarias, en **/bin** estan los ficheros ejecutables y en **/data** encontraremos como subdirectorio cada una de las bases de datos que hayamos creado.

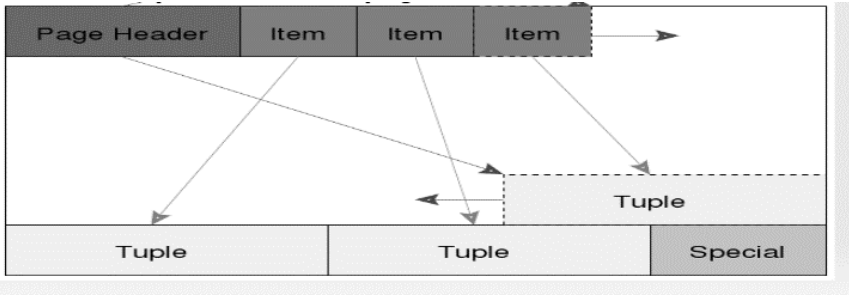
Para cada base de datos que nostros creamos, MySQL crea un directorio con el nombre que le hemos asignado a la bade de datos. Dentro de este directorio, por cada tabla que definamos MySQL va ha crear tres archivos: **mitabla.ISD, mitabla.ISM, mitabla.frm**

El archivo con extensión **ISD**, es el contine los datos de nuestra tabla, el **ISM** contiene información acerca de las claves y otro datos que MySQL utiliza para buscar datos en el fichero **ISD**. Y el archivo **frm** contine la estructura de la propia tabla.

3.1. Definición de espacio de almacenamiento

Estructura Lógica
– Bases de Datos
– Tablespaces
– Roles de grupo
– Roles de login

El tamaño de una página en PostgreSQL puede ser tan pequeño como 8k (por defecto) hasta un máximo de 32k y no se permite que un tupla pueda ser mas grande que una página de tamaño.
Cuando se necesita guardar data muy grande la data es comprimida y partida en pequeñas “filas” que se guardan en una tabla paralela, esto es transparente para el usuario.
Las páginas contienen “items” los cuales apuntan a tuplas o entradas de índices junto con metadata.
Para el caso de PostgreSQL las operaciones de R/W primero se consulta al Buffer Manager (memoria RAM) si contiene la página.

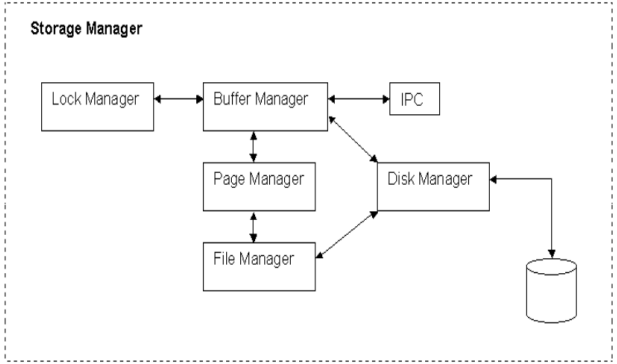


3.1. Definición de espacio de almacenamiento

PostgreSQL posee un solo “Storage Manager” PostgreSQL siempre esta añadiendo data, la data modificada o borrada realmente no se modifica o se borra, las páginas donde ellas están almacenadas se marca como “no visible” y se inserta un nuevo registro completo con un clon de toda la data. Esto hace que la base de datos ocupe mucho espacio y afecta el “tiempo de acceso” a la data.

Los módulos se programaron bajo 3 lineamientos bien claros:

- Manejar transacciones sin necesidad de escribir código complejo de recuperación en caso de caídas.
- Mantener versiones históricas de la data bajo el concepto de “graba una vez, lee muchas veces”.
- Tomar las ventajas que ofrece el hardware especializado como multiprocesadores, memoria no volátil, etc.



3.2. Definición y creación del espacio asignado para cada base de datos

En general, el almacenamiento **de los objetos** de la base de datos (tablas e índices fundamentalmente) no se realiza sobre el archivo o archivos físicos de la base de datos, sino que se hace a través de **estructuras lógicas de almacenamiento** que tienen por debajo a esos archivos físicos, y que independizan por tanto las sentencias de creación de objetos de las estructuras físicas de almacenamiento. Esto es útil porque permite que a esos "espacios de objetos " les sean asociados nuevos dispositivos físicos (es decir, más espacio en disco) de forma dinámica cuando la base de datos crece de tamaño más de lo previsto. **Posibilita además otra serie de operaciones como las siguientes:**

- ✓ Asignar cuotas específicas de espacio a usuarios de la base de datos.
- ✓ Controlar la disponibilidad de los datos de la base de datos, poniendo fuera de uso alguno de esos espacios de tablas individualmente.
- ✓ Realizar copias de seguridad o recuperaciones parciales de la base de datos.
- ✓ Reservar espacio para almacenamiento de datos de forma cooperativa entre distintos dispositivos.

3.2. Definición y creación del espacio asignado para cada base de datos

El administrador de la base de datos puede crear o borrar nuevos espacios lógicos de objetos, añadir o eliminar ficheros físicos de soporte, utilizados como espacio temporal de trabajo, definir parámetros de almacenamiento para objetos destinados a ese espacio de datos, todos los gestores relacionales siguen esta filosofía.

Un enfoque de la correspondencia entre la base de datos y los archivos es utilizar varios y guardar los registros de cada una de las diferentes longitudes fijas existentes en cada uno de esos archivos. Los archivos con registros de longitud fija son más sencillos de implementar que los archivos con registros de longitud variable. Muchas de las técnicas utilizadas para los primeros pueden aplicarse al caso de longitud variable.

3.2. Definición y creación del espacio asignado para cada base de datos

Dado un conjunto de registros, la pregunta siguiente es la manera de organizarlos en archivos. Existen diferentes formas de organizaciones de archivos que determinan la forma en que los registros de unos archivos se colocan físicamente en el disco y, por lo tanto, cómo se accede a éstos.

Archivos de montículos (o no ordenados)

Archivos ordenados (o secuenciales)

3.2. Definición y creación del espacio asignado para cada base de datos

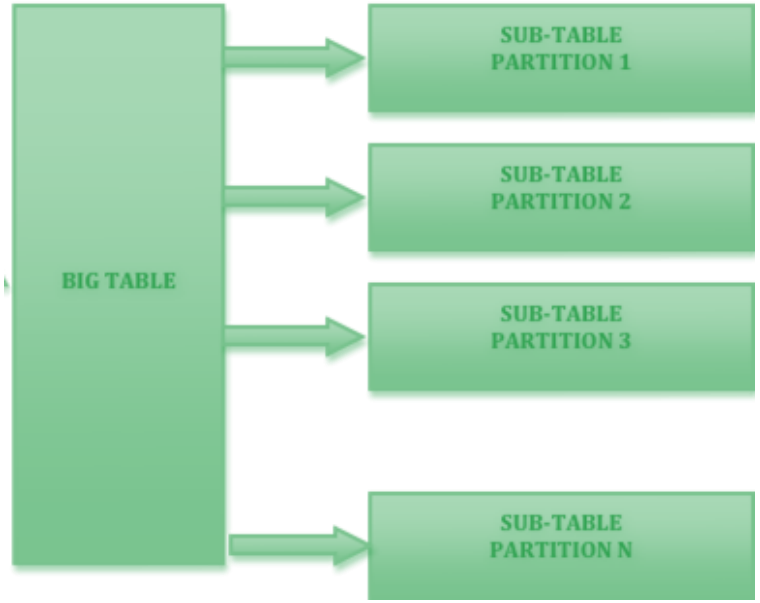
Archivos de montículos (o no ordenados): Esta técnica coloca los registros en el disco sin ningún orden particular, añadiendo nuevos registros al final del archivo. En esta organización se puede colocar cualquier registro en cualquier parte del archivo en que haya espacio suficiente. No hay ninguna ordenación de los registros. Generalmente sólo hay un archivo por cada relación.

Archivos ordenados (o secuenciales): En esta organización los registros se guardan en orden secuencial, basado en el valor de la clave de búsqueda de cada registro.
Los archivos secuenciales están diseñados para el procesamiento eficiente de los registros de acuerdo con un orden basado en una clave de búsqueda. Una clave de búsqueda es cualquier atributo o conjunto de atributos; no tiene por qué ser una clave primaria, ni siquiera una superclave. Para permitir la recuperación rápida de los registros según el orden de la clave de búsqueda, los registros se vinculan mediante punteros. El puntero de cada registro apunta al siguiente registro según el orden indicado por la clave de búsqueda.

3.2. Definición y creación del espacio asignado para cada base de datos

Cuando alguna de las tablas de una base de datos llega a crecer tanto que el rendimiento empieza a ser un problema, es hora de empezar a conocer algo sobre **optimización**.

Los SGBD, permiten la realiza con de particiones de las tablas lo que facilita rotar la información de nuestras tablas en diferentes particiones, consiguiendo así realizar consultas más rápidas y recuperar espacio en disco al borrar los registros.



3.2.1 Particiones

Crear particiones de datos puede ofrecer una serie de ventajas. Por ejemplo, se puede aplicar para:

- Mejorar la escalabilidad. El escalado de un sistema de base de datos única permitirá alcanzar finalmente un límite de hardware físico. Dividir datos en varias particiones, cada una de las cuales se encuentra hospedada en un servidor independiente, permite al sistema escalar de manera casi indefinida.
- Mejorar el rendimiento. Las operaciones de acceso a datos en cada partición se realizan en un volumen de datos menor. Siempre que se cree una partición de los datos de forma adecuada, esto resulta mucho más eficaz. Las operaciones que afectan a más de una partición pueden ejecutarse en paralelo. Cada partición puede estar cerca de la aplicación que la utiliza para minimizar la latencia de red.
- Mejorar la seguridad. Según la naturaleza de los datos y de cómo estén particionados, es posible separar los datos confidenciales y no confidenciales en particiones distintas y, por tanto, en diferentes servidores o almacenes de datos. Por lo tanto, es posible optimizar la seguridad de manera específica para los datos confidenciales.

3.2.1 Particiones

- Mejorar la disponibilidad. Separar los datos en varios servidores evita un punto único de error. Si se produce un error en un servidor o están realizando realizando tareas de mantenimiento planeada, solo estarán disponibles los datos de esa partición. Las operaciones en otras particiones pueden continuar. Aumentar el número de particiones reduce el impacto relativo de un error de servidor único reduciendo el porcentaje de los datos que no van a estar disponibles. Replicar cada partición puede reducir todavía más la posibilidad de que se produzca un error en una única partición que afecte a las operaciones. También permite la separación de datos críticos que deben presentar una alta disponibilidad de manera continua desde los datos de valor bajo (por ejemplo, archivos de registro) que presentan requisitos de disponibilidad inferiores.

3.2.1 Particiones

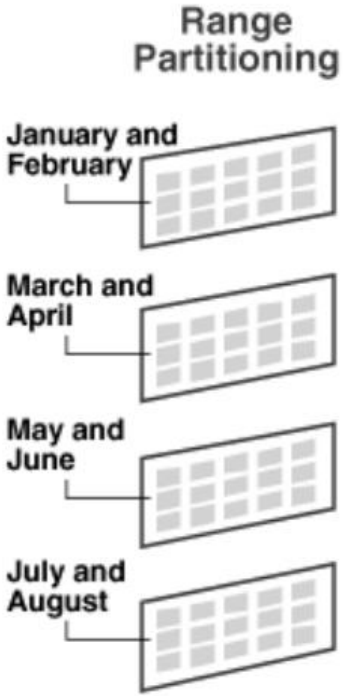
- **Proporcionan flexibilidad operativa.** La creación de particiones ofrece muchas oportunidades para ajustar con precisión las operaciones, maximizar la eficacia administrativa y minimizar los costes. Algunos ejemplos son la definición de distintas estrategias de administración, supervisión, copias de seguridad y restauración y otras tareas administrativas en función de la importancia de los datos de cada partición.

- **Adaptación del almacén de datos al patrón de uso.** La creación de particiones permite la implementación de cada partición en un tipo de almacén de datos diferente en función de costo y las características integradas que ofrece el almacén de datos. Por ejemplo, podrían almacenarse datos binarios de grandes dimensiones en un almacén de datos blob, aunque se podrían mantener datos más estructurados en una base de datos de documentos.

3.2.1 Particiones

PARTICIONAMIENTO POR RANGO

Los datos se distribuyen de acuerdo con el **rango de valores** de la clave de particionamiento. La distribución de datos es continua.
Se requiere que los registros estén identificado por un “partition key” relacionado por un predefinido rango de valores, el valor de las columnas “partition key” determina la partición a la cual pertenecerá el registro.



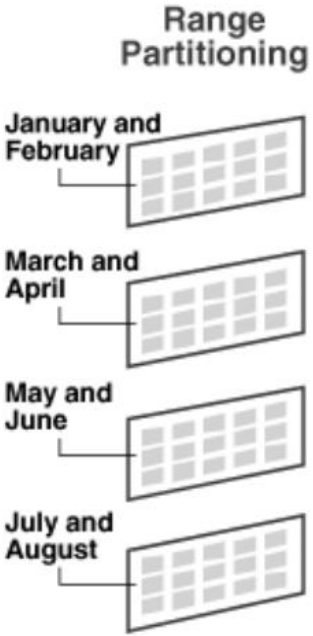
1. **Particionamiento por Rango [IMAG.04]**[1]

3.2.1 Particiones

PARTICIONAMIENTO POR RANGO

Se deben considerar las siguientes reglas:

- Cada partición se define con la clausula VALUES LESS THAN, la que indica el límite superior no inclusive para las particiones, cualquier valor de la clave de la partición igual o superior, es añadida a la próxima partición.
- Todas las particiones, excepto la primera, tienen un límite inferior, especificado en la clausula VALUES LESS THAN de la partición previa.
- Un literal MAXVALUE puede ser definido para la última partición; representa un valor virtual de infinito.



1. **Particionamiento por Rango [IMAG.04]**[1]

3.2.1 Particiones

PARTICIONAMIENTO POR RANGO

Ejemplo:

```
create table ESTUDIANTES  (
  CEDULA            VARCHAR2(10)              not null,
  NOMBRES           VARCHAR2(40),
  APELLIDOS         VARCHAR2(40),
  SEXO              CHAR(1),
   ID_ESCUELA         INTEGER,
  constraint PK_ESTUDIANTES primary key (CEDULA)
)
  PARTITION BY RANGE (id_escuela)

  (PARTITION EISIC VALUES LESS THAN (2) TABLESPACE TEISIC,
   PARTITION EITEX VALUES LESS THAN (3) TABLESPACE TEITEX,
            PARTITION CIME VALUES LESS THAN (4) TABLESPACE TCIME,
            PARTITION CIERCOM VALUES LESS THAN (5) TABLESPACE TCIERCOM;
```

3.2.1 Particiones

PARTICIONAMIENTO POR RANGO

Ejemplo:

TABLA ESTUDIANTES (sin partición)

CEDULA	NOMBRES	APELLIDOS	SEXO	ID_ESCUELA
1001766284	JUANA NARCISA	BUITRON DOMINGUEZ	F	2
1002222915	MARTHA CENEIDA	LOMAS NIETO	F	1
0401067871	FLOR DEL ROCIO	PABON POZO	F	3
1002673828	MARIA TATIANA	ACOSTA CAICEDO	F	2
1001645009	LOURDES ZENEIDA	VALLES FIERRO	F	4
1704464633	CARLOS VICENTE	PINEDA PALACIOS	M	2
1001607702	LUIS ALFREDO	AVENDAÑO DIAZ	M	4
0801518648	ELISA ESIL	ZAMBRANO DEL VALLE	F	4
1201467113	ARTURO NARCISO	UBE LUCES	M	1
1302112907	EUCLIDES OJILVIE	MENENDEZ LOOR	M	4
1001537321	SILVIA SUSANA	RIVERA GOMEZ	F	3
1600355794	BLADIMIR RUBEN	TERAN NARVAEZ	M	2
0802729202	MARIUXI BETSY	BONE GARRIDO	F	1
0801303348	WILVER ALEXANDER	SALAS GUERRERO	M	4
1002828588	ANGELICA CAROLA	MORAN ACOSTA	F	2
1302871817	PAULA HENOES	SANCHEZ MEZA	F	3
1304148750	JOSE RENSON	ZAMBRANO MOREIRA	M	1
1305584227	SONIA ESTHER	MACIAS LOOR	F	2
1306687565	ANTONIO GEOVANNY	ALCIVAR CHAVEZ	M	2
1307507135	GIANI GUILLERMO	CEDEÑO GARCIA	M	1

Tabla Estudiantes Particionada por Rango (EITEX)

CEDULA	NOMBRES	APELLIDOS	SEXO	ID_ESCUELA
1001766284	JUANA NARCISA	BUITRON DOMINGUEZ	F	2
1002673828	MARIA TATIANA	ACOSTA CAICEDO	F	2
1704464633	CARLOS VICENTE	PINEDA PALACIOS	M	2
1600355794	BLADIMIR RUBEN	TERAN NARVAEZ	M	2
1002828588	ANGELICA CAROLA	MORAN ACOSTA	F	2
1305584227	SONIA ESTHER	MACIAS LOOR	F	2
1306687565	ANTONIO GEOVANNY	ALCIVAR CHAVEZ	M	2

Tabla Estudiantes Particionada por Rango (CIME)

CEDULA	NOMBRES	APELLIDOS	SEXO	ID_ESCUELA
0401067871	FLOR DEL ROCIO	PABON POZO	F	3
1001537321	SILVIA SUSANA	RIVERA GOMEZ	F	3
1302871817	PAULA HENOES	SANCHEZ MEZA	F	3

Tabla Estudiantes Particionada por Rango (CIERCOM)

CEDULA	NOMBRES	APELLIDOS	SEXO	ID_ESCUELA
1001645009	LOURDES ZENEIDA	VALLES FIERRO	F	4
1001607702	LUIS ALFREDO	AVENDAÑO DIAZ	M	4
0801518648	ELISA ESIL	ZAMBRANO DEL VALLE	F	4
1302112907	EUCLIDES OJILVIE	MENENDEZ LOOR	M	4
0801303348	WILVER ALEXANDER	SALAS GUERRERO	M	4

3.2.1 Particiones

PARTICIONAMIENTO POR LISTA

El particionamiento por lista no soporta claves de particionamiento formada por varios atributos.
La clave de particionado es una lista de valores, que determina cada una de las particiones, la distribución de datos se define por el listado de valores de la clave de partición. El valor DEFAULT sirve para definir la partición donde irán el resto de registros que no cumplen ninguna condición de las diferentes particiones.

2. Particionamiento por Lista [IMAG.05][2]

3.2.1 Particiones

PARTICIONAMIENTO POR LISTA

Ejemplo

```
create table MATERIAS  (
  ID_MATERIA          INTEGER   not null,
  MATERIA             VARCHAR2(50),
  ID_ESCUELA          INTEGER,
  MATERIA_ANTERIOR    INTEGER,
  ID_NIVEL            INTEGER,
  constraint PK_MATERIAS primary key (ID_MATERIA)
)
PARTITION BY LIST (id_escuela)
  (
    PARTITION EISIC values(1)TABLESPACE TEISIC,
    PARTITION EITEX values(2)TABLESPACE TEITEX,
    PARTITION CIME values(3)TABLESPACE TCIME,
    PARTITION CIERCOM values(4)TABLESPACE TCIERCOM,
    PARTITION ESDYM values(5)TABLESPACE TESDYM,
    PARTITION ESIIN values(6)TABLESPACE TESIIN)
```

3.2.1 Particiones

PARTICIONAMIENTO POR LISTA

Ejemplo

TABLA MATERIAS (sin partición)

ID_MATERIA	MATERIA	ID_ESCUELA	MATERIA_ANTERIOR	ID_NIVEL
1	Analisis Matematico I	1		1
2	Analisis Matematico II	1	1	2
3	Ecuaciones Diferenciales	1	2	3
4	Matematicas Aplicadas	1	3	4
5	Modelacion y Simulacion de Computadoras	1		5
6	Fibrologia	2		1
7	Introduccion al Hilado	2		2
8	Quimica Organica I	2		3
9	Quimica Organica II	2	8	4
10	Control de Procesos I	2		5
11	Introduccion a la Ingenieria	3		1
12	Dibujo Mecanico I	3		2
13	Dibujo Mecanico II	3	12	3
14	Maquinas Herramientas	3	13	4
15	Emprendimiento e Innovacion Tecnologica	3		5
16	Tecnicas de Aprendizaje	4		1
17	Expresion Oral y Escrita	4		2
18	SistemasOperativos	4		3
19	Base de Datos	4	18	4
20	Programacion de Sistemas Multimedia	4	19	5

Tabla Materias Particionada por LIST (EISIC)

ID_MATERIA	MATERIA	ID_ESCUELA	MATERIA_ANTERIOR	ID_NIVEL
1	Analisis Matematico I	1		1
2	Analisis Matematico II	1	1	2
3	Ecuaciones Diferenciales	1	2	3
4	Matematicas Aplicadas	1	3	4
5	Modelacion y Simulacion de Computadoras	1		5

Tabla Materias Particionada por LIST (EITEX)

ID_MATERIA	MATERIA	ID_ESCUELA	MATERIA_ANTERIOR	ID_NIVEL
6	Fibrologia	2		1
7	Introduccion al Hilado	2		2
8	Quimica Organica I	2		3
9	Quimica Organica II	2	8	4
10	Control de Procesos I	2		5

Tabla Materias Particionada por LIST (CIME)

ID_MATERIA	MATERIA	ID_ESCUELA	MATERIA_ANTERIOR	ID_NIVEL
11	Introduccion a la Ingenieria	3		1
12	Dibujo Mecanico I	3		2
13	Dibujo Mecanico II	3	12	3
14	Maquinas Herramientas	3	13	4
15	Emprendimiento e Innovacion Tecnologica	3		5

Tabla Materias Particionada por LIST (CIERCOM)

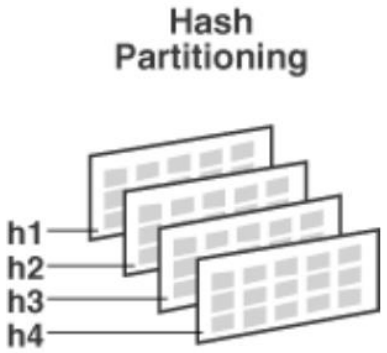
16	Tecnicas de Aprendizaje	4		1
17	Expresion Oral y Escrita	4		2
18	SistemasOperativos	4		3
19	Base de Datos	4	18	4
20	Programacion de Sistemas Multimedia	4	19	5

3.2.1 Particiones

PARTICIONAMIENTO POR HASH

La correspondencia entre las filas y las particiones se realiza a través de una función de hash. Es una opción útil cuando:

- Se desconoce la correspondencia en función de los rangos o no hay unos criterios de particionado claros.
- El rango de las particiones difiere sustancialmente o es difícil balancearla manualmente.



3. **Particionamiento por Hash [IMAG.06][3]**

La clave de particionado es una función hash, aplicada sobre una columna, que tiene como objetivo realizar una distribución equitativa de los registros sobre las diferentes particiones. La funcion hash devuelve un valor automatico que determina a que partición irá el registro. Es una forma automática de balancear el particionado. Se puede definir la partición sin indicar los nombres de las particiones, solo poniendo el número de particiones deseadas.

3.2.1 Particiones

PARTICIONAMIENTO POR HASH

Ejemplo

```
create table DOCENTES  (
   CEDULA_DOCENTE      VARCHAR2(12)                not null,
   NOMBRES             VARCHAR2(40),
   APELLIDOS           VARCHAR2(40),
   ID_TITULOS          INTEGER,
   constraint PK_DOCENTES primary key (CEDULA_DOCENTE)
)
 PARTITION BY HASH (cedula_docente)
    partitions 4 store in (tdocente1, tdocente2,tdocente3,tdocente4);
```

El sistema creará automáticamente los nombres asignando cada partición a un diferente tablespace. Igualmente, se pueden indicar los nombres de cada particion individual o los tablespaces donde se localizaran cada una de ellas:

3.2.1 Particiones

PARTICIONAMIENTO POR HASH

Ejemplo

Ejemplo:

```
create table DOCENTES  (
   CEDULA_DOCENTE      VARCHAR2(12)                not null,
   NOMBRES             VARCHAR2(40),
   APELLIDOS           VARCHAR2(40),
   ID_TITULOS          INTEGER,
   constraint PK_DOCENTES primary key (CEDULA_DOCENTE)
)  PARTITION BY HASH (cedula_docente)
    (
        PARTITION DOCENT1 TABLESPACE tdocente1,
        PARTITION DOCENT2 TABLESPACE tdocente2,
        PARTITION DOCENT3 TABLESPACE tdocente3,
        PARTITION DOCENT4 TABLESPACE tdocente4);
```

3.2.1 Particiones

PARTICIONAMIENTO POR HASH

Ejemplo

TABLA DOCENTES (sin partición)

CEDULA_DOCENTE	NOMBRES	APELLIDOS	ID_TITULOS
1001	JAIME	AGUAS	1
1002	HUMBERTO	BRAVO	1
1003	CATALINA	RAMIREZ	2
1004	'ENDERSON	LARA	2
1005	PAUL	ANDRADE	2
1006	IRVING	REASCOS	1
1007	JORGE	CARAGUAY	1
1008	RODRIGO	NARANJO	1
1009	MARCELO	JURADO	1
1010	HUGO	IMBAQUINGO	4
1011	WIDMAR	AGUILAR	1
1012	DANIEL	JARAMILLO	1
1013	MIGUEL	ORQUERA	1
1014	JORGE	PORRAS	1
1015	JAIME	ALVARADO	1
1016	NANCY	CERVANTES	1
1017	JOSE	HUACA	1
1018	LUIS	ROMAN	5
1019	IVAN	GARCIA	1
1020	CARPIO	PINEDA	1

Tabla Docentes Particionada por HASH (DOCENTE1)

CEDULA_DOCENTE	NOMBRES	APELLIDOS	ID_TITULOS
1001	JAIME	AGUAS	1
1002	HUMBERTO	BRAVO	1
1003	CATALINA	RAMIREZ	2
1004	'ENDERSON	LARA	2
1005	PAUL	ANDRADE	2

Tabla Docentes Particionada por HASH (DOCENTE2)

CEDULA_DOCENTE	NOMBRES	APELLIDOS	ID_TITULOS
1006	IRVING	REASCOS	1
1007	JORGE	CARAGUAY	1
1008	RODRIGO	NARANJO	1
1009	MARCELO	JURADO	1
1010	HUGO	IMBAQUINGO	4

Tabla Docentes Particionada por HASH (DOCENTE3)

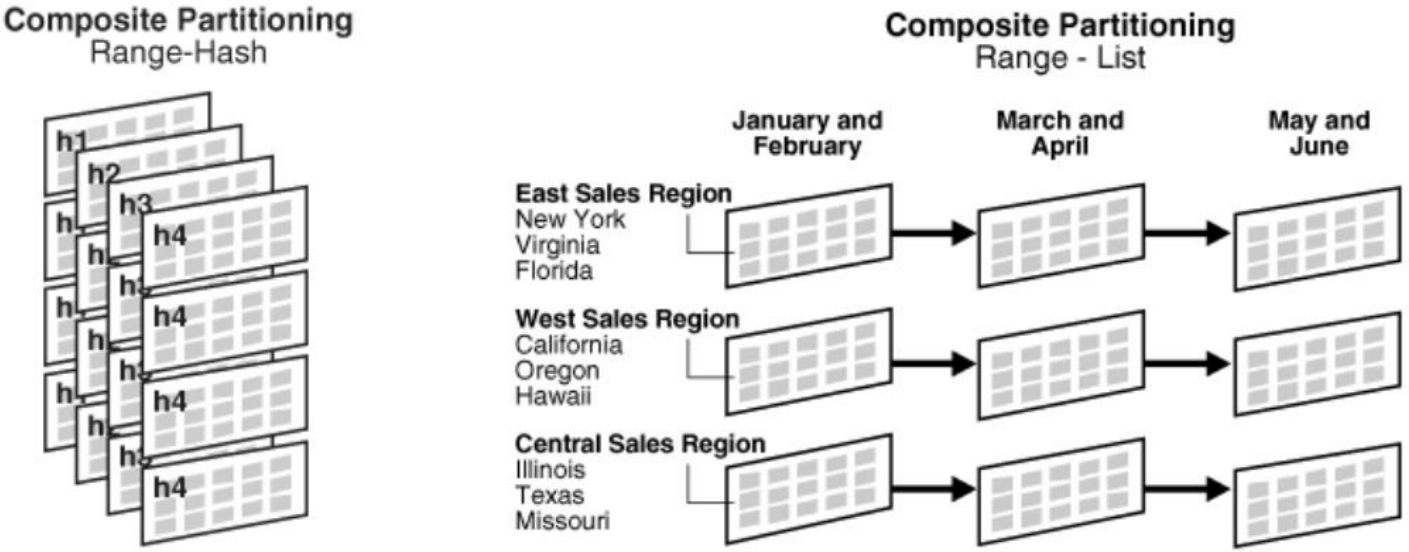
CEDULA_DOCENTE	NOMBRES	APELLIDOS	ID_TITULOS
1011	WIDMAR	AGUILAR	1
1012	DANIEL	JARAMILLO	1
1013	MIGUEL	ORQUERA	1
1014	JORGE	PORRAS	1
1015	JAIME	ALVARADO	1

Tabla Docentes Particionada por HASH (DOCENTE4)

CEDULA_DOCENTE	NOMBRES	APELLIDOS	ID_TITULOS
1016	NANCY	CERVANTES	1
1017	JOSE	HUACA	1
1018	LUIS	ROMAN	5
1019	IVAN	GARCIA	1
1020	CARPIO	PINEDA	1

3.2.1 Particiones

PARTICIÓN POR COMPOSICIÓN



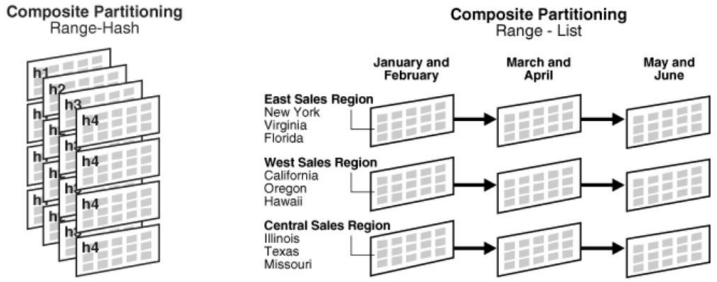
4. Particionamiento por Composición [IMAG.07][4]

3.2.1 Particiones

PARTICIÓN POR COMPOSICIÓN

Oracle nos permite utilizar métodos de particionado compuestos, ya que se conjuga el uso de dos particionados a la vez. Primero la tabla se particiona con un primer método de distribución de datos y luego cada partición se vuelve a dividir en subparticiones utilizando un segundo método de distribución de datos.
Las técnicas de partición compuesta disponibles son:

- **Rango - Hash**
- **Rango - List**
- **Rango - Rango**
- **List - Rango**
- **List - List**
- **List – Hash**



4. **Particionamiento por Composición [IMAG.07][4]**

Las particiones más generales se hacen con el método de rango, cada partición se sub-particiona con el método de hash o por lista.

3.2.1 Particiones

Una característica de MySQL son las particiones.
Particionar tablas en MySQL nos permite rotar la información de nuestras tablas en diferentes particiones, consiguiendo así realizar consultas más rápidas y recuperar espacio en disco al borrar los registros. El uso más común de particionado es según la fecha.

El motor de almacenamiento InnoDB mantiene las tablas en un espacio que puede ser creado a partir de varios ficheros. Esto permite que una tabla supere el tamaño máximo individual de un fichero. Este espacio puede incluir particiones de disco, lo que permite tablas extremadamente grandes. El tamaño máximo del espacio de tablas es 64TB.

3.2.1 Particiones

Se puede particionar una tabla de 5 maneras diferentes:

Por Rango: para construir las particiones se especifican rangos de valores.

```
ALTER TABLE contratos
PARTITION BY RANGE (YEAR (fechaInicio)) ( PARTITION partDecada80 VALUES LESS THAN (1990), PARTITION
partDecada90 VALUES LESS THAN (2000),
PARTITION partDecada00 VALUES LESS THAN (2010), PARTITION partDefault VALUES LESS THAN MAXVALUE );
```

La última partición (partDefault) tendrá todos los registros que no entren en las particiones anteriores. De esta manera nos aseguramos que la información nunca dejará de insertarse en la tabla.

Por Listas: para construir nuestras particiones especificamos listas de valores concretos.

```
ALTER TABLE contratos
PARTITION BY LIST (YEAR (fechaInicio)) (
    PARTITION partDecada80 VALUES IN (1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989),
    PARTITION partDecada90 VALUES IN (1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999),
    );
```

3.2.1 Particiones

Por Hash: MySQL se encarga de distribuir las tuplas automáticamente usando una operación de módulo. Sólo hay que pasarle una columna o expresión que resulte en un entero (el hash) y el número de particiones que queramos crear.

```
ALTER TABLE contratos
PARTITION BY HASH (YEAR (fechaInicio))
PARTITIONS 7;
```

Por Clave: similar a la partición por hash, pero en este caso no necesitamos pasarle un entero; MySQL utilizará su propia función de hash para generarlo. Si no se indica ninguna columna a partir de la que generar el hash, se utiliza la clave primaria por defecto.

```
ALTER TABLE contratos
PARTITION BY KEY ()
PARTITIONS 7;
```

Compuesta: podemos combinar los distintos métodos de particionado y crear particiones de particiones

3.2.1 Particiones

Borrar Particiones
Lo bueno de trabajar con particiones es que podemos borrar rápidamente registros sin tener que recorrer toda la tabla e inmediatamente recuperar el espacio en disco utilizado por la tabla.

Por ejemplo si queremos borrar la partición más antigua simplemente ejecutamos:

```
ALTER TABLE reports DROP PARTITION p201111;
```

Consultar Particiones
Para consultar información de particiones creadas en una tabla así como también los registros que contiene cada una ejecutamos:

```
SELECT PARTITION_NAME, TABLE_ROWS FROM information_schema.PARTITIONS WHERE TABLE_NAME='reports';
```

3.2.1 Particiones

La partición de una tabla permite dividir una tabla de grandes dimensiones en varias tablas. Cada una de estas subtablas es más pequeña que la tabla inicial y, por lo tanto, más fácil de gestionar para SQL Server. Es la unión de los datos presentes en todas estas subtablas lo que se corresponde con la tabla de origen.

Crear una función de partición (PARTITION FUNCTION), en esta se definirá el rango que cada partición va a almacenar. Para este caso existen dos métodos de Rangos, Izquierda (left) o Derecha (right), la diferencia del uso de este método, es la manera de realizar el análisis de rangos. Un ejemplo de la sentencia para cada método serían los siguiente

Syntax

```
CREATE PARTITION FUNCTION
partition_function_name (
input_parameter_type ) AS RANGE [
LEFT | RIGHT ] FOR VALUES ( [
boundary_value [ ,...n ] ] ) [ ; ]
```

ALTER PARTITION FUNCTION

3.2.1 Particiones

Syntax

```
CREATE PARTITION FUNCTION partition_function_name ( input_parameter_type ) AS
RANGE [ LEFT | RIGHT ] FOR VALUES ( [ boundary_value [ ,...n ] ] ) [ ; ]
```

Argumentos

partition_function_name: Es el nombre de la función de partición. Los nombres de las funciones de partición deben ser únicos dentro de la base de datos y cumplir con las reglas para los identificadores.

Input_parameter_type: Es el tipo de datos de la columna utilizada para la partición. Todos los tipos de datos son válidos para su uso como columnas de partición, excepto text, ntext, image, xml, timestamp, varchar(max), nvarchar(max), varbinary(max), tipos de datos de alias o tipos de datos CLR definidos por el usuario.

La columna real, conocida como columna de partición, se especifica en la instrucción CREATE TABLE o CREATE INDEX.

boundary_value: Especifica los valores límite para cada partición de una tabla o índice particionado que utiliza partition_function_name. Si valor_límite está vacío, la función de partición asigna toda la tabla o el índice utilizando partition_function_name en una sola partición. Solo se puede utilizar una columna de partición, especificada en una sentencia CREATE TABLE o CREATE INDEX.

boundary_value: es una expresión constante que puede hacer referencia a variables. Esto incluye variables de tipo definidas por el usuario o funciones y funciones definidas por el usuario.

3.2.1 Particiones

Crear una función de partición (PARTITION FUNCTION), en esta se definirá el rango que cada partición va a almacenar. Para este caso existen dos métodos de Rangos, Izquierda (left) o Derecha (right), la diferencia del uso de este método, es la manera de realizar el análisis de rangos. Un ejemplo de la sentencia para cada método serían los siguiente

```
CREATE PARTITION FUNCTION [pfDatos] (datetime)
AS RANGE RIGHT FOR VALUES
('20100101','20100201','20100301');
GO
```

CREATE PARTITION FUNCTION pfAnualR(int) AS RANGE RIGHT FOR VALUES(2008,2009,2010,2011)	CREATE PARTITION FUNCTION pfAnualL(int) AS RANGE LEFT FOR VALUES(2008,2009,2010,2011)
Primera Partición < 2008	Primera Partición <= 2008
Segunda Partición >= 2008	Segunda Partición > 2008
Tercera Partición >= 2009	Tercera Particion > 2009
Cuarta Partición >= 2010	Cuarta Particion > 2010
Quinta Partición >= 2011	Quinta Particion > 2011

3.2.1 Particiones

Crear un esquema para la partición (PARTITION SCHEME) , en esta se definirán los**FileGroups**donde se almacenara cada partición, un ejemplo es el siguiente (para la ejecución correcta de este script, es necesario que se creen o existan 5 FileGroups llamados FG01, FG02, FG03, FG04, FG05):

```
CREATE PARTITION SCHEME psAnualR AS PARTITION
pfAnualR TO ([FG01], [FG02], [FG03], [FG04], [FG05])
```

```
CREATE PARTITION SCHEME psAnualL AS PARTITION
pfAnualL TO ([FG01], [FG02], [FG03], [FG04], [FG05])
```

3.2.1 Particiones

En la creación de la Tabla, la diferencia es que en lugar de definir un **FileGroup** donde se guardara el objeto, se define la Función de partición y la columna eje. Ejemplo

```
CREATE TABLE tblEstadosR (Ano int, Mes int, Dia int, Tipo varchar(30), Importe float) ON psAnualR(Ano)
```

```
CREATE TABLE tblEstadosL (Ano int, Mes int, Dia int, Tipo varchar(30), Importe float) ON psanualL(Ano)
```

3.2.1 Particiones

Una vez realizados estos pasos, se procede a realizar el llenado de las tablas particionadas

Para saber en qué partición se encuentra la data se utilizara el siguiente comando Transac-SQL**$PARTITION**, este regresa el número de partición en el cual se encuentra la data. Para el ejemplo que estamos realizando, el nombre de la Base de Datos utilizada es **BDPrueba**
SELECT Ano, **BDPrueba.$Partition.pfAnualR(Ano)** FROM tblEstadosR
SELECT Ano, **BDPrueba.$Partition.pfAnualL(Ano)** FROM tblEstadosL

ALTER PARTITION FUNCTION
Altera una función de partición dividiendo o fusionando sus valores límite. La ejecución de una instrucción ALTER PARTITION FUNCTION puede dividir una partición de tabla o un índice que usa la función de partición en dos particiones. La declaración también puede fusionar dos particiones en una partición menos.

```
ALTER PARTITION FUNCTION partition_function_name() {
SPLIT RANGE ( boundary_value ) | MERGE RANGE (
boundary_value ) } [ ; ]
```

3.2.1 Particiones

Las **particiones** de tablas de **PostgreSQL** proporcionan un marco para el manejo de alto rendimiento de la entrada de datos y la generación de informes. Utilice **particiones** para bases de datos que requieren una entrada muy rápida de grandes cantidades de datos.

Los dos tipos de particionado mas comunes en postgresql son:

1. **Particionar por rangos.** La tabla es particionada mediante rangos definidos en base a la columna de llave primary o cualquier columna que no se solape entre los rangos de valores asignados a diferentes tablas hijas.

2. **Particionar por lista.** La tabla es particionada listando los valores de cada una de las llaves en cada particion.

3.4. Espacios para objetos de la base de datos

Los DBMS se basan en archivos para almacenar datos, y estos archivos, o conjuntos de datos, residen en medios de almacenamiento, o dispositivos. Una buena parte del trabajo del DBA implicará la planificación para el almacenamiento real de la base de datos.

• El rendimiento de la base de datos depende de la entrada y salida a disco. La cantidad de datos almacenados es mayor que nunca antes, y los datos son almacenados por más tiempo.

- Una base de datos está hecha de varios componentes u objetos:
  - tablas, consultas, formas, reportes.

3.4. Espacios para objetos de la base de datos

Son sumamente importantes para la velocidad de acceso y recuperación de datos.
Las técnicas dependen del tipo de almacenamiento, el uso que se le da o se le dará a la base de datos, la estructura de la misma, el SGBD empleado, etc.

Esta dependencia no significa necesariamente que haya que cambiar la estructura de la base de datos si se cambian las técnicas empleadas. Las técnicas de almacenamiento son independientes de la base de datos, pero, de todas maneras, las mejores técnicas muchas veces pueden determinarse viendo la estructura de la base de datos, entre otras características.

3.4. Espacios para objetos de la base de datos

Los encargados de elegir estas técnicas son los diseñadores y administradores de bases de datos, y dependen también de las capacidades del SGBD. En general, el SGBD ofrece diferentes opciones y técnicas para organizar los datos.

3.4. Espacios para objetos de la base de datos

Existen diferentes formas de organizaciones primarias de archivos que determinan la forma en que los registros de un archivo se colocan físicamente en el disco y, por lo tanto, cómo se accede a éstos.

Las distintas formas de organizaciones primarias de archivos son:

Archivos de montículos (o no ordenados): esta técnica coloca los registros en el disco sin un orden específico, añadiendo nuevos registros al final del archivo.
Archivos ordenados (o secuenciales): mantiene el orden de los registros con respecto a algún valor de algún campo (clave de ordenación).
Archivos de direccionamiento calculado: utilizan una función de direccionamiento calculado aplicada a un campo específico para determinar la colocación de los registros en disco.
Árboles B: Se vale de la estructura de árbol para las colocaciones de registros.

3.5. ROLES

Los Roles, que son simplemente un conjunto de privilegios que se pueden otorgar a un usuario o a otro Rol. De esa forma se simplifica el trabajo del DBA en esta tarea.

Utilice roles de base de datos para gestionar con mayor facilidad los privilegios de los grupos de usuarios.

Los roles de base de datos simplifican el proceso de gestión de privilegios, ya que se pueden otorgar privilegios a un rol y luego otorgar el rol a usuarios. Cuando desee revocar privilegios para un usuario, simplemente tiene que revocar la autorización de rol del usuario, en vez de revocar cada privilegio individual.

Roles
Conjunto de privilegios agrupados.

Privilegios
Es la capacidad de un usuario dentro de la base de datos a realizar determinadas operaciones o acceder a determinados objetos de otros usuarios.

3.5. ROLES

Privilegios sobre los objetos
Nos permite acceder y realizar cambios en los datos de otro usuario. Ejemplo: El privilegio de consultar la tabla de otro usuario es un privilegio sobre objetos.

Privilegios del sistema
Dan derecho de ejecutar un tipo de comando SQL o a realizar alguna opción sobre objetos de un tipo especificado.

3.5 Roles

Roles de usuarios en oracle

El rol de un usuario determina el acceso a las funciones de la interfaz gráfica de usuario y de SCI

Administrator (Administrador): un administrador de biblioteca. Este rol tiene acceso a prácticamente todas las funciones y puede gestionar a otros usuarios.

User (Usuario): el rol de los operadores diarios de la biblioteca. Este rol puede llevar a cabo la mayoría de las acciones en la biblioteca, pero algunas están fuera de alcance, como la configuración de particiones o la creación de usuarios.

Operator (Operador) : un rol con menores privilegios que el rol User (Usuario). Este rol solo tiene acceso a un pequeño subconjunto de acciones, como la visualización de datos y la operación de los CAP.

Viewer (Visor): rol de solo lectura. Los usuarios con este rol pueden ver, pero no modificar lo que se muestra en la biblioteca.

Service (Servicio) : un rol especial para los técnicos de servicio. Este rol puede tomar paquetes de servicio, ejecutar diagnósticos, cambiar los ajustes de configuración, etc.

Advanced Service (Servicio avanzado) :rol de servicio avanzado con acceso adicional para diagnóstico y reparación de la biblioteca.

Escalation (Escalada) : el rol de nivel de servicio superior con acceso extensivo para reparación de la biblioteca.

Installer (Instalador) ; un rol especial que se usa durante la instalación y la configuración iniciales de la biblioteca. Este rol solo está disponible antes de entregar la biblioteca al cliente.

3.5 Roles

Para administrar con facilidad los permisos de las bases de datos, SQL Server proporciona varios roles* que son las entidades de seguridad que agrupan a otras entidades de seguridad. Son como los grupos del sistema operativo Microsoft Windows. Los roles de nivel de base de datos se aplican a toda la base de datos en lo que respecta a su ámbito de permisos.

Para agregar y quitar usuarios en un rol de base de datos, use las opciones ADD MEMBER y DROP MEMBER de la instrucción ALTER ROLE .

Existen dos tipos de roles en el nivel de base de datos: los roles fijos de base de datos que están predefinidos en la base de datos y los roles de base de datos definidos por el usuario que el usuario puede crear.
Los roles fijos de base de datos se definen en el nivel de base de datos y existen en cada una de ellas. Los miembros de los roles de base de datos db_owner pueden administrar la pertenencia a roles fijos de base de datos. También hay algunos roles de base de datos con fines especiales en la base de datos msdb.
Puede agregar cualquier cuenta de la base de datos y otros roles de SQL Server a los roles de nivel de base de datos.
Los permisos de los roles de base de datos definidos por el usuario se pueden personalizar con las instrucciones GRANT, DENY y REVOKE

3.5 Roles

Roles fijos de base de datos

No se pueden cambiar los permisos asignados a los roles fijos de base de datos.

Nombre del rol fijo de base de datos	Descripción
db_owner	Los miembros del rol fijo de base de datos **db_owner** pueden realizar todas las actividades de configuración y mantenimiento en la base de datos y también pueden quitar la base de datos en `drop`SQL Server. (En SQL Database y Azure Synapse, algunas actividades de mantenimiento requieren permisos de nivel de servidor y los roles **db_owners** no las pueden realizar).
db_securityadmin	Los miembros del rol fijo de base de datos **db_securityadmin** pueden modificar la pertenencia a roles únicamente para roles personalizados y administrar permisos. Los miembros de este rol pueden elevar potencialmente sus privilegios y se deben supervisar sus acciones.
db_accessadmin	Los miembros del rol fijo de base de datos **db_accessadmin** pueden agregar o quitar el acceso a la base de datos para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server .
db_backupoperator	Los miembros del rol fijo de base de datos **db_backupoperator** pueden crear copias de seguridad de la base de datos.
db_ddladmin	Los miembros del rol fijo de base de datos **db_ddladmin** pueden ejecutar cualquier comando del lenguaje de definición de datos (DDL) en una base de datos.
db_datawriter	Los miembros del rol fijo de base de datos **db_datawriter** pueden agregar, eliminar o cambiar datos en todas las tablas de usuario.
db_datareader	Los miembros del rol fijo de base de datos **db_datareader** pueden leer todos los datos de todas las tablas y vistas de usuario. Los objetos de usuario pueden existir en cualquier esquema, excepto *sys* e *INFORMATION_SCHEMA*.
db_denydatawriter	Los miembros del rol fijo de base de datos **db_denydatawriter** no pueden agregar, modificar ni eliminar datos de tablas de usuario de una base de datos.
db_denydatareader	Los miembros del rol fijo de base de datos **db_denydatareader** no pueden leer datos de las tablas y vistas de usuario dentro de una base de datos.

3.5 Roles

Un rol de MySQL es una colección de privilegios con nombre. Al igual que las cuentas de usuario, los roles pueden tener privilegios otorgados y revocados.

A una cuenta de usuario se le pueden otorgar funciones, lo que otorga a la cuenta los privilegios asociados con cada función. Esto permite la asignación de conjuntos de privilegios a las cuentas y proporciona una alternativa conveniente para otorgar privilegios individuales, tanto para conceptualizar las asignaciones de privilegios deseadas como para implementarlas.

3.5 Roles

La siguiente lista resume las capacidades de gestión de funciones proporcionadas por MySQL:

CREATE ROLE y DROP ROLE crean y eliminan roles.
GRANT y REVOKE asignan privilegios para revocar privilegios de cuentas y roles de usuario.
SHOW GRANTS muestra las asignaciones de roles y privilegios para cuentas y roles de usuario.
SET DEFAULT ROLE especifica qué roles de cuenta están activos de forma predeterminada.
SET ROLE cambia los roles activos dentro de la sesión actual.
La función CURRENT_ROLE() muestra los roles activos dentro de la sesión actual.

Las variables de sistema obligatorio_roles y activar_todos los roles_en_login permiten definir roles obligatorios y la activación automática de roles otorgados cuando los usuarios inician sesión en el servidor.
A menos que se especifique lo contrario, las instrucciones SQL que se muestran aquí deben ejecutarse utilizando una cuenta MySQL con suficientes privilegios administrativos, como la cuenta **root**.

3.5 Roles

PostgreSQL utiliza un sistema de **roles** para gestionar los permisos de base de datos. Se utilizan **roles** para otorgar un conjunto de **privilegios** a un usuario individual o a un grupo de usuarios. Puede determinar los **roles**, grupos y **privilegios** para todos los **roles** del despliegue utilizando el mandato \du de **psql** .

Gestión de roles

Como ya se menciono, PostgreSQL gestiona los permisos de acceso a la base de datos utilizando el concepto de roles y que un rol puede considerarse como un usuario de la base de datos o un grupo de usuarios de esta misma.

En PostgreSQL es posible otorgar la membresía en un rol a otro rol, es por ello que el termino de roles subsume los conceptos de “usuario” y “grupos”.

Atributos para roles

Un rol puede contener una serie de atributos que definen sus privilegios.

LOGIN, SUPERUSER, CREATEDB, CREATEROLE, REPLICATION LOGIN, PASSWORD

3.5 Roles

Crear un rol usuario

Para crear un nuevo rol de tipo usuario podemos usar tanto CREATE ROLE como CREATE USER. La diferencia entre ambos es que si usamos la opción de USER añadimos automáticamente el atributo LOGIN. Este atributo es el que nos va a servir para indicar si es un rol usuario o un rol de grupo.

Los roles pueden contener atributos (LOGIN, SUPERUSER, CREATEDB, CREATEROL…) que van a definir sus privilegios.

```
CREATE ROLE demouser WITH
ENCRYPTED PASSWORD '123456'
LOGIN
VALID UNTIL 'INFINITY';
```

CREATE ROLE *name*;
DROP ROLE *name*;

CREATEUSER *name*
DROPUSER *name*