proyecto2manualBDoracle Documentation

Publicación 0.0.1

Francisco

13 de June de 2016

Índice general

1. Home **3**

2. Arquitectura física **5**
- 2.1. **1. Estructura de memoria** . . . 5
- 2.2. **2. Procesos** . . . 7
- 2.3. **3. Archivos** . . . 8
- 2.4. **Estándar de documentación** . . . 13

3. Arquitectura lógica **15**
- 3.1. **Introducción** . . . 15
- 3.2. **Tipos de datos** . . . 15
- 3.3. **Almacenamiento en una Base de Datos Oracle** . . . 17
- 3.4. **Estándar de documentación** . . . 24

4. Modelo de datos **25**
- 4.1. **1. Modelo Entidad-Relación a Relacional** . . . 25
- 4.2. **3. Tipos de relaciones** . . . 27
- 4.3. **4. Conceptos sobre relaciones** . . . 29
- 4.4. **5. Restricciones de integridad asociadas a las relaciones de la base de datos** . . . 29
- 4.5. **6. Normalización** . . . 32
- 4.6. **Estándar de documentación** . . . 36

5. Componentes PLsql **37**
- 5.1. **Construyendo bloques de programas PL/SQL** . . . 37
- 5.2. **Conceptos Básicos** . . . 38
- 5.3. **Tipos de Datos** . . . 38
- 5.4. **Nombres en una base Oracle** . . . 39
- 5.5. **Procedimientos Almacenados** . . . 39
- 5.6. **Funciones Almacenadas** . . . 41
- 5.7. **Triggers** . . . 43
- 5.8. **Catálogo General** . . . 44

6. Recomendaciones **47**

Contenidos:

CAPÍTULO 1

Home

Escuela de Informática

Curso: Diseño e implementación de bases de datos

Proyecto II: Manual técnico base de datos Oracle

Estudiantes

- Francisco Sánchez Salas
- David Chacón Abarca
- Diego Jiménez Alvarado
- Daniel Quesada Sandí

Profesor: Johnny Villalobos Murillo

Universidad Nacional, Campus Benjamín Nuñez, Primer Ciclo, 2016

Estructura de una Base de Datos Oracle.

Una Base de Datos Oracle tiene una estructura física y una estructura lógica que se mantienen separadas, pero que son complemento una de la otra para poder trabajar. Las bases de datos de Oracle usan un modelo relacional.

- La estructura física se corresponde a los ficheros del sistema operativo: de datos (datafiles), de redo log (rollback) y de control (controlfiles).
- La estructura lógica está formada por los tablespace y los objetos de un esquema de BD (tablas, vistas, índices,...).

CAPÍTULO 2

Arquitectura física

Para la arquitectura física de la base de datos este manual se basará en la arquitectura física del gestor de bases de bases de datos Oracle esto porque se decidió que para la explicación de todo el diseño e implementación de la base de los datos se va a realizar con el gestor de bases de datos Oracle. A continuación se explica la forma en que se encontrará distribuida la arquitectura física, así como también pasos a seguir para crear nuevos archivos log iniciales. Además de explicar cómo realizar multiplicación y mantenimiento de los grupos y miembros redo log online. Finalmente se mencionan algunos problemas típicos relacionados a la arquitectura física así como también recomendaciones para los mismos.

La arquitectura física de Oracle tiene tres componentes básicos:

1. La estructura de memoria
2. Los procesos
3. Los archivos

2.1 1. Estructura de memoria

La estructura de la memoria de Oracle está formada por dos áreas de memoria llamada:

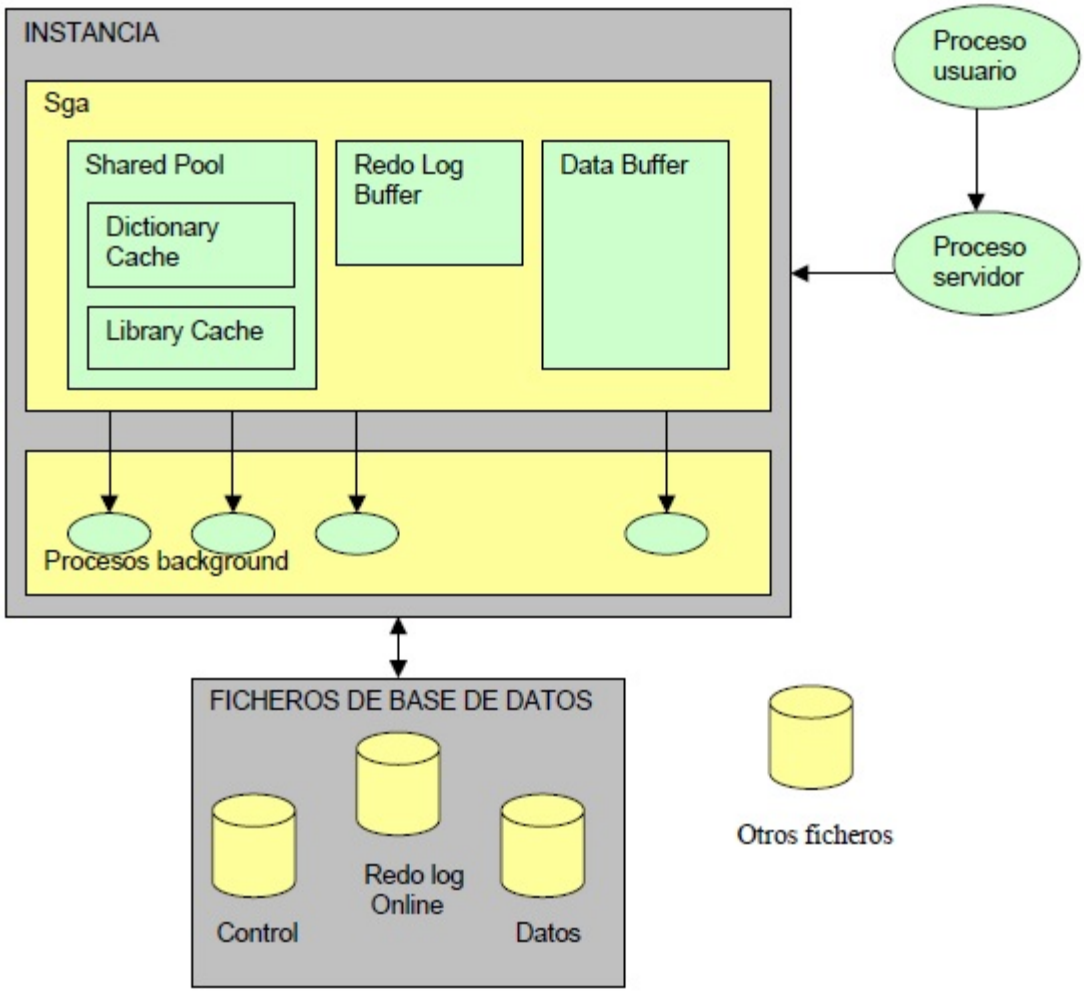
- SGA (Área Global del Sistema): Asignada al iniciar la instancia y componente fundamental de una instancia Oracle.
- PGA (Área Global de Programas): Asignada al iniciar el proceso de servidor SGA (System Global Area).

El SGA es un área de memoria compartida que se utiliza para almacenar información de control y de datos de la instancia. Se crea cuando la instancia es levantada y se borra cuando ésta se deja de usar (cuando se hace shutdown).

La información que se almacena en esta área consiste de los siguientes elementos, cada uno de ellos con un tamaño fijo:

- Shared Pool
- Database Buffer
- Redo Log Buffer

El tamaño de estos componentes del SGA pueden ser administrados manualmente o automáticamente. Si se escoge administrar manualmente estos componentes, se debe especificar el tamaño de cada uno de ellos aumentando o disminuyendo el tamaño de ellos según las necesidades de la aplicación. Si los estos componentes son administrados automáticamente, la instancia por si misma monitorea la utilización de cada componente de la SGA y ajusta sus tamaños. Independiente si la administración de la SGA es automática o manual, Oracle asigna o designa espacio dinámico


INSTANCIA
Sga
Shared Pool
Dictionary Cache
Library Cache
Redo Log Buffer
Data Buffer
Proceso usuario
Proceso servidor
Procesos background
FICHEROS DE BASE DE DATOS
Control
Redo log Online
Datos
Otros ficheros

dentro de la SGA dividiendo la SGA dentro de unidades llamadas granules. Dependiendo del sistema operativo donde reside el motor de base de datos Oracle, los granules pueden ser de 4 MB, 8 MB o 16 MB.

En principio podría, se podría subconfigurar la caché de buffers.

SHARED POOL

Almacena el DICCIONARIO DE DATOS (DATA DICTIONARY CACHE) y las sentencias SQL más recientemente utilizadas (SHARED SQL o LIBRARY CACHE). Es donde tiene lugar la fase de PARSING o ANALISIS de las sentencias SQL. Para definir su tamaño se realiza con el parámetro SHARED_POOL_SIZE del fichero de parámetros INIT.ORA y **no debería ser mayor al 50 % de la memoria disponible de la máquina**.:

```
ALTER SYSTEM SET SHARED_POOL_SIZE = 64 MB;
```

Database buffer

Almacena los datos más recientemente utilizados. Si al hacer una petición al sistema, los datos están ya en memoria por una petición anterior, se evita una entrada a disco, por lo que el rendimiento será mejor (compartir información). El tamaño del buffer se define en el parámetro `DB_BLOCK_BUFFERS`. También contiene los BUFFER DE ROLLBACK, que almacenan la imagen anterior de los datos, y que proporcionan consistencia en lectura.

Redo log buffer

Son de acceso secuencial y graban todos los cambios hechos a la base de datos por operaciones de insert, update, delte, creat, alter y drop con el mínimo de información necesaria. Su función es la de proporcionar seguridad, frente a una caída de la BD. Su tamaño se define con el parámetro `LOG_BUFFER`.

La SGA se gestiona mediante el algoritmo LRU, las sentencias y datos menos recientemente utilizados son eliminados de la memoria para permitir la entrada de nuevos datos.:

```
SQL > SHOW SGA;
SQL > SELECT * from V$SGA;
```

Si por circunstancias especiales debemos vaciar toda la SGA de sus estructuras, podemos hacerlo con la orden:

```
ALTER SYSTEM FLUSH SHARED POOL;
```

2.2 2. Procesos

Los procesos son programas que se ejecutan para permitir el acceso a los datos, se cargan en memoria y son transportados para los usuarios. Se clasifican en tres grupos:

Proceso de usuario:

Es un proceso que se crea cuando un usuario establece una conexión con la base de datos. Se encarga de ejecutar el código de aplicación del usuario y manejar el perfil del usuario con sus variables de ambiente. Estos procesos no se pueden comunicar directamente con la base de datos, por lo que la comunicación la establecen mediante procesos de servidores.

Oracle puede configurarse para variar el número procesos usuario por proceso servidor. En una configuración de servidor dedicado, un proceso servidor manipula las peticiones de un único proceso usuario. En una configuración de servidor multi-línea se permite a varios procesos de usuario compartir un pequeño número de procesos servidores, minimizando el número de procesos servidores y maximizando la utilización de los recursos del sistema disponibles.

Proceso servidor:

También se crea uno por conexión de usuario. Se crea en el servidor y se encarga de comunicar el proceso de usuario con la instancia de la base de datos. Estos procesos ejecutan las órdenes SQL de los usuarios y llevan los datos del buffer caché para que los procesos de usuario puedan tener acceso a los datos.

En algunos sistemas, los procesos de usuario y servidores están separados, mientras que en otros se combinan en un único proceso. Si un sistema utiliza el servidor multi-línea o, si los procesos de usuario y servidores corren en diferentes máquinas, ambos procesos deben estar separados.

Procesos background (Segundo plano).

Son procesos de sistema, encargados de ejecutar las funciones comunes que son necesarias para el servicio de respuesta a los usuarios. Dependen de la configuración del server.

Hay cinco obligatorios:

- Monitorización de procesos (PMON)

Recupera el proceso cuando hay un fallo en el proceso usuario. Libera la cache y libera los recursos asignados a ese proceso usuario.

- Monitorización del sistema (SMON)

Recupera la instancia (lee el controlfile). Limpia los segmentos temporales no usados y recupera las transacciones ante una caída del sistema. Compacta el espacio libre en los ficheros de datos.

- Escritura en la base de datos (DBWR)

Escribe los datos modificados del buffer de datos a los ficheros de datos.

- Escritura de la traza (LGWR)

Graba los cambios que se registran en el buffer de redo log a los ficheros de redo log.

- Punto de sincronismo (checkpoint o CKPT)

Modifica el estado de la información de la BD (cabeceras de ficheros), cuando hay un checkpoint o un log switch(llenado de redo log).

Entre los procesos opcionales en segundo plano están: ARCn, LMDN, QMNn, RECO, LMS, LCKn, LMON.

2.3 3. Archivos

Estos archivos guardan información tanto de los datos almacenados en la BD como la necesaria para gobernar la propia BD. Son 3 tipos de archivos o ficheros:

- Los Archivos de Datos (Datafiles)
- Archivos de Control (Control files)
- Archivos de Rehacer (redo log files)

Además, hay archivos fuera de línea (Archived files), que son archivos opcionales donde se pueda guardar información vieja de los archivos de rehacer, convenientes para respaldos de base de datos

3.1 Archivos de Datos (Datafiles):

Sirve para el almacenamiento físico del Diccionario de Datos, las tablas, índices, procedimientos y la imagen anterior de los bloques de datos que se han modificado en las transacciones (segmentos de rollback). Estos archivos son los únicos que contienen los datos de los usuarios de la base de datos. Se pueden tener sólo uno o cientos de ellos.

El número máximo de datafiles que pueden ser configurados está limitado por el parámetro de sistema `MAXDATAFILES`.

Si se decide que utilice varios datafiles, el administrador del sistema puede gestionar que éstos queden localizados en discos diferentes, lo que aumentará el rendimiento del sistema, principalmente por la mejora en la distribución de la carga de entrada / salida. Para obtener información de los Archivos de Datos, consultar la tabla `DBA_DATA_FILES`.

3.2 Archivos de Rehacer (redo log files):

Tienen los cambios que se han hecho a la base de datos para recuperar fallas o para manejar transacciones. Debe estar conformado por dos grupos como mínimo y cada grupo debe estar en discos separados. El principal propósito de estos archivos es de servir de respaldo de los datos en la memoria RAM.

Para establecer el tamaño apropiado de un archivo de este tipo deberá considerarse el tamaño del dispositivo que contendrá el respaldo del redo log.

Mínimo deben existir 2, aunque la instalación por defecto es de 3. Se recomienda trabajar con ficheros de Redo Log MULTIPLEXADOS en espejo, de forma que la información es escrita en varios ficheros, a ser posible en distintos discos.

Se puede configurar la base de datos para que mantenga copias de los archivos redo log online de forma que se evite la pérdida de información ante un fallo.

La base de datos contendrá grupos redo log online cuyos miembros son los archivos redo log online en sí. Los miembros de un grupo son copias idénticas de los archivos redo log online.

Un aspecto a tener en cuenta es el tamaño de los ficheros redo log. Si son muy pequeños, el LGWR deberá cambiar de ficheros demasiado frecuentemente, lo que reduce su rendimiento. Por otro lado, si los ficheros redo log son demasiado grandes, se necesitará mucho tiempo en las recuperaciones, ya que se tendrán que recuperar muchas transacciones. Oracle recomienda que se les dé un tamaño que permita que cada redo log tarde en llenarse unos 15 ó 20 minutos. Otro aspecto muy importante es la elección del número correcto de grupos, ya que disponer de demasiados pocos grupos puede acarrear problemas cuando estámos en modos ARCHIVELOG y tenemos una tasa de transacciones muy alta. Esto puede suponer que un grupo que todavía está archivando por el proceso ARCH se convierta en el grupo en el que el LGWR necesite escribir, lo que produciría que la BD se parara, ya que el LGWR tienen que esperar a que el grupo esté disponible, una vez que su contenido ha sido archivado. Para la mayoría de las implantaciones, tener entre 2 y 10 grupos puede ser suficiente. El número de grupos no puede exceder de `MAXLOGFILES`, ni el número de miembros puede ser mayor que `MAXLOGMEMBERS`.

3.2.1 Grupos redo log online

- Un grupo redo log online es un juego de copias idénticas de un archivo de redo log online.
- El proceso en segundo plano LGWR escribe simultáneamente en todos los ficheros redo log online de un grupo la misma información.
- El servidor necesita como mínimo dos grupos redo log online con un miembro cada uno para que el funcionamiento de la base de datos sea normal.

3.2.2 Miembros redo log online

- Cada archivo redo log online de un grupo se denomina miembro.
- Cada miembro de un grupo tiene números de secuencia de log idénticos y el mismo tamaño. El número de secuencia de log se asigna cada vez que el servidor comienza a escribir en un grupo de log para identificar cada archivo redo log de forma única. El número de secuencia se almacena en el archivo de control y en la cabecera de todos los archivos de datos.

3.2.3 Creación de archivos log iniciales

El juego inicial de grupos y miembros redo log online se crea durante la creación de la base de datos al lanzar la sentencia CREATE DATABASE. Ejemplo:

```
CREATE DATABASE newtest
          LOGFILE
                GROUP 1
```

```
                    ('diskb log1b.log',
                    'diskc log1c.log')
                      SIZE 5000K,
               GROUP 2
                      ('diskb log2b.log',
                       'diskc log2c.log')
                        SIZE 5000K,
          MAXLOGFILES 5 ........
```

Hay algunos parámetros que afectan a los grupos y miembros redo log online:

- `MAXLOGFILES`: en el comando `CREATE DATABASE`, especifica el máximo absoluto de grupos redo log online.
- `MAXLOGMEMBERS`: en el comando `CREATE DATABASE`, especifica el número máximo de miembros por grupo.
- `LOG_FILES`: parámetro de inicialización que define el número máximo actual de grupos redo log online que se pueden abrir en tiempo de ejecución (no puede exceder el parámetro `MAXLOGFILES`).

3.2.4 Multiplexación y mantenimiento de los grupos y miembros

Una vez creada la base de datos, puede ser necesario añadir nuevos grupos redo log online o añadir nuevos miembros a grupos ya existentes.

Cómo añadir un nuevo grupo redo log online

Para añadir un nuevo grupo de redo log online debe utilizar el siguiente comando SQL:

```
ALTER DATABASE [n_database]
ADD LOGFILE [GROUP n] espec_fichero
```

La identificación del grupo (n) se puede omitir y será el servidor el que asigne el identificador correspondiente. Deberá especificar al menos un miembro para el nuevo grupo añadido a la base de datos.

Cómo añadir un nuevo miembro redo log online

Otra posibilidad que le ofrece el servidor Oracle es añadir nuevos miembros redo log online a grupos redo log existentes. El comando SQL para realizar esta acción es la siguiente:

```
ALTER DATABASE [n_database]
ADD LOGFILE MEMBER espec_fichero [REUSE]
TO GROUP n
```

Si el archivo ya existe, debe utilizar la opción REUSE y debe ser del mismo tamaño que los otros miembros del grupo.

Cómo cambiar el nombre a un miembro redo log online

Para cambiar la ubicación de archivos redo log online basta con cambiar el nombre al archivo especificando la nueva ubicación. Antes de cambiar el nombre de un archivo redo log online, asegúrese de que el nuevo archivo exista en la ubicación correspondiente. Oracle sólo cambia los punteros a los archivos, no crea físicamente ningún archivo del sistema operativo. La sentencia SQL para cambiar el nombre a un archivo redo log online es:

```
ALTER DATABASE [n_database]
RENAME FILE 'espec_fichero'
TO 'espec_fichero'
```

3.2.5 Tamaño de los archivos redo log online

En primer lugar, un archivo redo log online debe ser como mínimo de 50KB. El tamaño máximo lo determina el sistema operativo. Los miembros de grupos diferentes podrán tener diferente tamaño pero esto no tiene ninguna ventaja.

Los siguientes aspectos pueden influir en la configuración de los archivos redo log online:

- Número de cambios de log y puntos de control.
- Número y cantidad de registros de redo.
- Cantidad de espacio en el medio de almacenamiento.

Debe procurar tener una configuración simétrica: mismo número de miembros por grupo y mismo tamaño.

3.3 Archivos de Control (Control files):

Tiene la descripción física y dirección de los archivos para el arranque correcto de la base de datos. Mantienen la información física de todos los ficheros que forman la BD, camino incluido; así como el estado actual de la BD. Son utilizados para mantener la consistencia interna y guiar las operaciones de recuperación. Son imprescindibles para que la BD se pueda arrancar y deben encontrarse siempre protegidos. Contienen:

- Infomación de arranque y parada de la BD.
- Nombres de los archivos de la BD y redo log.
- Información sobre los checkpoints.
- Fecha de creación y nombre de la BD.
- Estado online y offline de los archivos.

Los ficheros de control se crean durante la instalación del producto ORACLE y deben ser siempre accesibles cada vez que sea arrancada la Base de Datos.

Debe haber múltiples copias en distintos discos, mínimo dos, para protegerlos de los fallos de disco. La lista de los ficheros de control se encuentra en el parámetro `CONTROL_FILES`, que debe modificarse con la BD parada.

3.3.1 Gestionando los Ficheros de Control

El parámetro CONTROL_FILES del fichero contiene la lista de todos los ficheros de control.

Para protegerlos contra fallos de almacenamiento, se sugiere que al menos existan dos ficheros de control, cada uno en un disco diferente, aunque es buena idea mantener más copias en diferentes discos. Esto es una política de espejado que protege frente a fallos en disco. Si un disco falla y se pierden todos los ficheros en él, se puede seguir utilizando los ficheros de control de otros discos. Esto supone una pequeña sobrecarga al sistema, ya que cada vez que se produce un checkpoint o cambia el esquema de la BD, todos los ficheros de control son actualizados.

Si un fallo ha producido la pérdida de todas las copias de los ficheros de control habrá que recrearlos con el comando *create controlfile*. Si algunos de los parámetros `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY` , `MAXDATAFILES` y `MAXINSTANCES` varía habrá que utilizar también el comando `CREATE CONTROLFILE`.

3.3.2Multiplexación del archivo de control

Oracle permite tener múltiples e idénticos archivos de control de forma simultánea. Es aconsejable tener 3 copias del archivo y a ser posible en discos diferentes.

Puede realizar la multiplexación del archivo de control:

- Creando múltiples archivos de control al crear la base de datos.
- Agregando archivos de control una vez creada la base de datos.

Al crear la base de datos

Es el método más sencillo para tener múltiples archivos de control. La forma de hacerlo es indicar los diferentes archivos en el parámetro de inicialización `CONTROL_FILES` antes de crear la base de datos.

El servidor Oracle creará todos los archivos indicados en la ubicación correspondiente (al indicar los nombres de los ficheros hay que añadir la ruta completa). Una vez creada la base de datos

Para agregar un archivo de control una vez ya creada la base de datos, debe realizar los siguientes pasos:

1. Cerrar la base de datos.

2. Copiar el archivo de control desde el sistema operativo a las ubicaciones elegidas.
3. Modificar el archivo de parámetros añadiendo en el parámetro `CONTROL_FILES` los nuevos archivos...
4. Iniciar la base de datos.

Problemas tipicos y recomendaciones

- Los puntos críticos de fallos son el CONTROL FILE y los Ficheros de REDO LOG.
- Si se pierde el fichero de redo log en uso, sólo un backup completo de la base de datos la recuperaría hasta ese momento.
- Si falla otro de los ficheros de redo log, Oracle lo ignora y pasaría a otro, por lo que es importante que existan 3 o 4.
- El tamaño de los ficheros de redo log depende mucho de cada situación. Si es muy grande, en un recovery, tardaría mucho tiempo en arrancar.
- Oracle recomienda que se les dé un tamaño que permita que cada redo log tarde en llenarse unos 15 ó 20 minutos.
- Se pueden añadir redo log online (flexibilidad de Oracle).
- Respecto al control file es interesante tener una copia guardada en otro dispositivo. En caso de fallo, modificando en el init.ora el fichero de control con el debe arrancar, leería ese control file.

Ejemplo de una creación de una base de datos aplicando tópicos anteriormente mencionados sobre la arquitectura física

Antes de crear una base de datos debemos determinar dos variables importantes en el sistema operativo: `ORACLE_HOME` (directorio raíz para Oracle) y `ORACLE_SID` (Identificador de la base de datos para el sistema operativo, suele coincidir con el nombre que le vayamos a asignar a la base de datos).

En primer lugar, debemos arrancar la instancia de la base de datos. Para esto debemos conectarnos a SQLPlus con permisos de administrador de la siguiente forma:

```
C:\sqlplus / as sysdba
```

Una vez en el entorno de sqlplus, arrancaremos la instancia de la base de datos. Para esto tendremos que haber creado el fichero de parámetros `INITsid.ora`. Del archivo de parámetros debe especificar como mínimo los siguientes parámetros antes de iniciar la instancia:

- `DB_NAME`: Identificador de la base de datos de ocho o menos caracteres. No es necesario que coincida con la variable `ORACLE_SID` del sistema operativo aunque es recomendable que sí sean iguales. El parámetro `DB_NAME` debe coincidir con el nombre que se utilice posteriormente en la sentencia `CREATE DATABASE`.
- `CONTROL_FILES`: Especifica una lista de archivos de control. Como mínimo debería especificar dos nombres de archivo de control situados en discos diferentes si es posible. No es necesario que existan dichos archivos físicamente.
- `DB_BLOCK_SIZE`: Determina el tamaño de bloque de base de datos. Debe ser un múltiplo del tamaño de bloque del sistema operativo.

El comando para iniciar la instancia desde el entorno SQLPlus es:

```
Sqlplus> startup nomount pfile='fich_param'
```

A continuación hay que crear la base de datos. Una Base de Datos se crea mediante la sentencia *CREATE DATABASE*, después de arrancar una Instancia.:

```
CREATE DATABASE PRUEBA
[CONTROLFILE REUSE]
DATAFILE 'ORA1.DBF' SIZE 20M [REUSE]
```

```
LOGFILE 'LOG1.DBF' SIZE 500K [REUSE],
'LOG2.DBF' SIZE 500K [REUSE]
[MAXDATAFILES 100]
[MAXLOGFILES 16]
[MAXLOGMEMBERS 3]
[ARCHIVELOG | NOARCHIVELOG]
[CHARACTER SET WE8DEC]
[MAXLOGHISTORY 100]
```

- MAXLOGFILES: Número máximo de ficheros de Redo Log. Dependiente del sistema operativo.
- MAXDATAFILES: Número máximo de ficheros de Base de Datos. Dependiente del sistema operativo.
- REUSE: Si al crear la Base de Datos, existe ya alguno de los ficheros, los reutiliza perdiendo su contenido anterior.
- MAXLOGHISTORY: Número máximo de ficheros redo log que pueden ser recogidos en el registro histórico del archivo de control. Se recomienda dar el valor 0, para no activar el registro histórico., que se usa para la recuperación automática de media de un servidor paralelo. Su valor máximo es de 65535.
- Deben utilizarse múltiples archivos de control en discos distintos. Los nombres de los archivos de control se especifican por medio del parámetro CONTROL_FILES del INIT.ORA.

Cuando finaliza el comando CREATE DATABASE implícitamente se realiza la apertura de la base de datos.

Modificación de la Base de Datos

Primero se arranca la B.D. por pasos hasta realizar un:

```
ALTER DATABASE TEST MOUNT
```

- Funciones Adicionales:

  ```
  ALTER DATABASE ADD LOGFILE nombre_fichero [REUSE]
  ```

  ```
  ALTER DATABSE DROP LOGFILE nombre_fichero
  ```

  ```
  ALTER DATABASE RENAME FILE nom_antiguo TO nom_nuevo
  ```

  ```
  ALTER DATABASE ARCHIVELOG/NOARCHIVELOG
  ```

  ```
  ALTER DATABASE BACKUP CONTROLFILE TO TRACE
  ```

- Después de realizar la función necesaria:

```
ALTER DATABASE OPEN
```

- ALTER DATABASE DISMOUNT
- ALTER DATABASE CLOSE

2.4 Estándar de documentación

A continuación se dan las pautas para realizar la documentación de la arquitectura física.

1. Realizar un diagrama en donde se aprecie claramente la estructura física de la base de datos.
2. Especificar cada uno de los ficheros físicos junto con el espacio de la memoria con que cuentan cada uno de ellos.
3. Especificar los procesos de usuario y explicar en qué consiste cada uno de ellos.

4. Documentar los Grupos redo log online y en ellos especificar cada uno de sus miembros redo log. Así como también se debe actualizar esta parte cada vez que se dé la creación de un miembro redo log.
5. Documentar los archivos de control, su contenido y como fue designada la multiplexación de estos archivos. También si se crean archivos de control nuevos después de crear la base de datos estos deben ser documentados.
6. Documentar los archivos de datos y su contenido.
7. Finalmente en caso de agregar nuevos ficheros físicos documentar los cambios hechas para mantener está parte de la documentación de la base datos actualizada.

CAPÍTULO 3

Arquitectura lógica

3.1 Introducción

Una Base de Datos se divide en unidades de almacenamiento lógicas llamadas Tablespaces. Contienen distintos objetos relacionados, como las tablas de una misma base de datos. Cada base de datos está formada por uno o más tablespaces, donde al menos debe existir el tablespace SYSTEM (diccionario de datos).

Una base de datos Oracle contiene como mínimo un tablespace, SYSTEM se crea automáticamente. Un tablespace contiene uno o más segmentos, cada tablespace se corresponde con uno o más ficheros de datos.

Cada segmento está formado por extensiones, que son divisiones lógicas.

Una extensión está formada por bloques lógicos, así es como se asigna el espacio.

Un bloque es una unidad más pequeña para las operaciones de lectura y escritura.

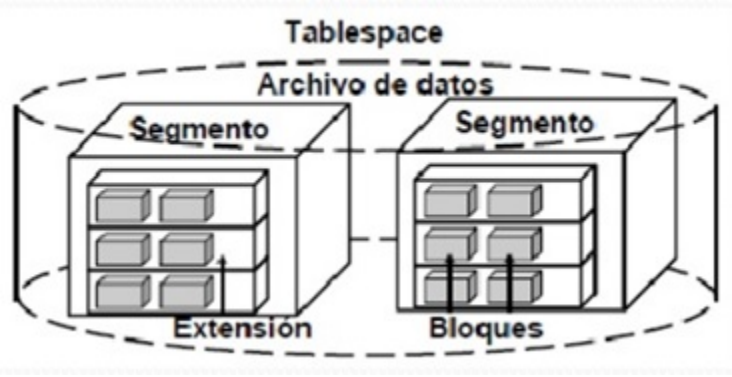


Imagen 01: representación gráfica de la distribución de la estructura lógica de las tablespace.

3.2 Tipos de datos

Los tipos de datos soportados por Oracle se agrupan en los siguientes conjuntos.

- Alfanuméricos

  `CHAR`

  `VARCHAR2`

  `VARCHAR`

  `NCHAR`

  `NVARCHAR2`

  `LONG`

- Numéricos

  `NUMBER`

  `FLOAT`

- Fecha

  `DATE`

- Bianarios

  `RAW`

  `LONG RAW`

  `BLOB`

  `CLOB`

  `BLOB`

  `BFILE`

- Otros

  `ROWID`

TIPOS DE DATOS ALFANUMÉRICOS

- Tipo de dato `CHAR(n)` : Almacena cadenas de caracteres de longitud fija.Su rango está entre 1 y 2.000 bytes de ocupación.
- Tipo de dato `VARCHAR2(n)` : Almacena cadenas de caracteres de longitud variable.
- Tipo de dato `VARCHAR(n)` : En Oracle8 es equivalente a VARCHAR2.
- Tipo de dato `NCHAR(n)` : Almacena un valor alfanumérico de longitud fija. Puede almacenar caracteres ASCII, EBCDIC, UNICODE...
- Tipo de dato `NVARCHAR2(n)` : Almacena un valor alfanumérico de longitud variable. Puede almacenar caracteres ASCII, EBCDIC, UNICODE...

TIPOS DE DATOS NUMÉRICOS

- Tipo de dato `NUMBER(p, s)` : Almacena valores numéricos en punto flotante que pueden estar entre 1.0 x 10-130 y 9.9...(38 nueves).
- Tipo de dato `FLOAT(N)` : Almacena un número en punto decimal sin restricción de dígitos decimales, donde n indica la precisión binaria máxima que puede moverse en el rango 1 a 126.

TIPOS DE DATOS FECHA

- Tipo de dato `DATE`: Almacena un valor de fecha y hora. Para un tipo de dato `DATE`, Oracle almacena internamente los siguientes datos:

  Siglo

  Año

  Mes

  Día

  Hora

  Minuto

  Segundo

El formato por defecto de las fechas es: `'DD-MON-YYYY'`

TIPO DE DATOS BINARIOS

- Tipos de datos binarios: Permiten almacenar información en formato "crudo", valores binarios tal y como se almacenan en el disco duro o como residen en memoria.
- Tipo de dato `LONG` : Almacena caracteres de longitud variable hasta 2Gb. En Oracle8 y siguientes versiones se deben usar los tipos de datos `CLOB` y `NLOB` para almacenar grandes cantidades de datos alfanuméricos.

OTROS TIPOS DE DATOS

- Tipo de dato `ROWID`: Representa una dirección de la base de datos, ocupada por una única fila. Este tipo de dato sirve para guardar punteros a filas concretas.

3.3 Almacenamiento en una Base de Datos Oracle

Oracle define "esquema" como la colección de estructuras lógicas que corresponden a los datos almacenados, y crea un nuevo esquema por cada usuario que crea objetos en la base de datos. No hay ninguna relación directa entre tablespace y esquema, objetos del mismo esquema pueden estar en diferentes tablespaces y un mismo tablespace puede almacenar distintos esquemas.

El control del uso del espacio del disco se obtiene mediante las estructuras lógicas de almacenamiento: bloque de datos, extensión y segmento. El nivel más pequeño de a nivel de la base de datos es el del bloque de datos.

El almacenamiento en Oracle se maneja con cinco conceptos:

- Tablespace
- Datafile
- Segment (segmento)
- Extend (extensiones)
- Data block (bloque de datos / paginas)

Tablespace

La distribución lógica de los datos está asociada directamente con la eficiencia de la base de datos, por eso es que se manejan varios tipos de datos de las tablespace. Se pueden tener tantos tablespace como sea necesario.

Las propiedades que se asocian a un tablespace cuando un objeto es creado dentro de un tablespace, hereda:

- Localización de los ficheros de datos

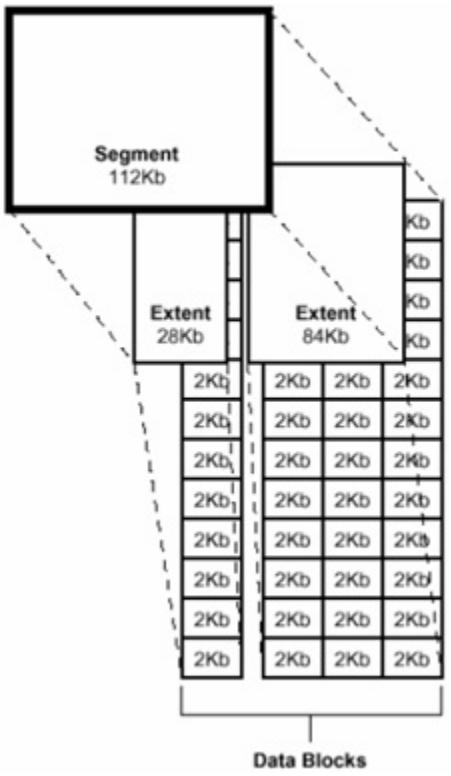


Figura 3.1: Imagen 02: Esquema extraído del Oracle8 Concepts. representación gráfica de la distribución lógica de la memoria por segmento, extensión y bloques de memoria.

- Especificación de consumo de disco.
- Control de disponibilidad de datos
- Backup de datos

A continuación se muestra la distribución de tablespace que se designa para la base de datos la cual es la siguiente.

- System (Diccionario de datos)
- Usuarios
- Temp
- Rollback
- Indices
- Datos

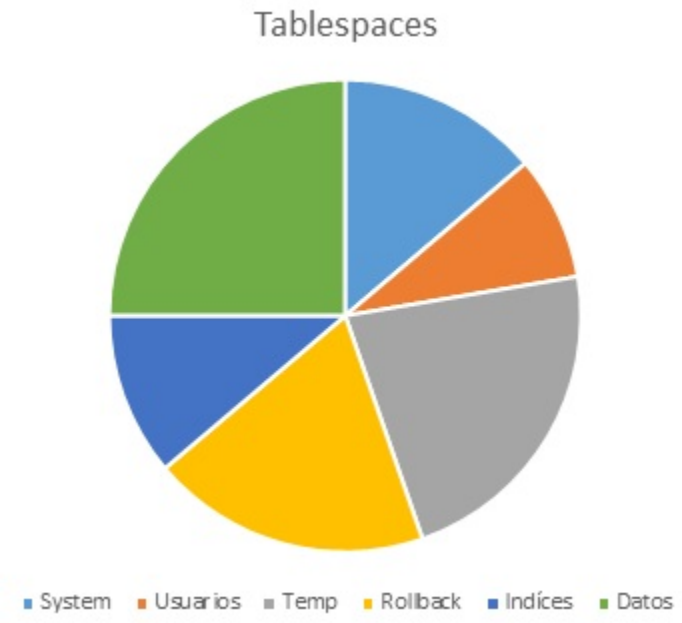


Imagen 03: posible distribución de las tablas en la memoria. Cada color representa tablas en distintos discos.

Si todos los datos están en el mismo espacio de memoria, podrían existir varios problemas, entre ellos problemas de seguridad, al tener todos los datos en el mismo disco, por ejemplo, el disco C. Si el sistema operativo falla, se cae la base de datos o hay problemas físicos con el disco, todos los datos se perderían, aun teniendo una tabla de respaldo, pues se encuentra en la misma distribución de la memoria. Una posible solución al problema de tener todo en System o en el mismo disco, es distribuir en diferentes memorias, una tabla podría ir en C:, otro en D:, otro en F:, sería más fácil respaldar la información así.

A continuación se explica con más detalle cada uno de los tablespaces designados así como su contenido.

System (Diccionario de datos)

El tablespace system será destinado para el diccionario datos el cual contiene toda la información de las tablas como nombres de columnas, cantidad de columnas, atributos, tipos de datos y otros datos importantes para la base de datos. Así como también serán almacenados catálogos de componentes PL/SQL creados para el control de los mismos, más adelante en el apartado de PL/SQL se explicará un poco más a fondo el contenido de estos catálogos de programación. Además, es importante tener en cuenta que el espacio de tablas SYSTEM esté al 50 % o 75 % de su espacio disponible. Finalmente, asegurarse de que los usuarios no tienen privilegios de escritura en el espacio de tablas SYSTEM, solo un usuario de tipo administrador de sistema debe poder modificar estos registros.

Usuarios

Este tablespace será utilizado para almacenar la información de todos los usuarios que utilicen las bases de datos, para así poder consultarlos y poder asígnales ciertos permisos, recursos o procedimientos.

Temp

Ciertas operaciones, necesitan de espacio en disco para poder realizarse. El espacio reservado se denomina "segmentos temporales". Se pueden crear segmentos temporales cuando:

- Se crea un índice.
- Se utiliza `ORDER BY`, `DISTINTC` o `GROUP BY` en un `SELECT`.
- Se utilizan los operadores `UNION`, `INTERSECT` o `MINUS`.
- Se utilizan `joins` entre tablas.
- Se utilizan subconsultas.

El tablespace temporal será destinado a todos aquellos datos temporales que ayudaran a mejorar el rendimiento de acceso a registros en la base de datos. Esto porque este espacio es de vital importancia para realizar operaciones en las que se lleve a cabo ordenamientos. También para la creación y análisis de índices para calcularles estadísticas. En los casos anteriores cuando el servidor no encuentre espacio suficiente libre en memoria utilizara el tablespace temp. También es importante mencionar que los rendimientos son muy superiores en comparación con otros tiempos de tablespaces normales. Esto dado que manera en que se utiliza este tablespace es orientado a objetos que crecen muy rápido y que a continuación rápidamente disminuyen su tamaño y desaparecen.

Rollback

El tablespace rollback será utilizado para almacenar todas las bitácoras correspondientes a las transacciones hechas en la base de datos esto con el fin de mantener una copia de seguridad en caso de fallo en la base de datos.

Índices (index)

Este tablespace será destinado a los índices de tablas. También es importante tomar en cuenta que los índices crecen en tamaño en mayor proporción que las tablas asociadas si los datos en la tabla son modificados frecuentemente. La gestión del espacio es mejor si se mantienen los índices de tablas grandes en espacios de tablas separados.

Datos

Este tablespace se utilizara para almacenar los registros de todas las tablas almacenadas en la base de datos. Dado el crecimiento de la cantidad de registros por tabla es recomendable prever un espacio grande para el datafile que se encuentre relacionado a este tablespace.

Cálculo de los tamaños de los tablespaces

Hay que considerar que los tablespace corresponden a una agrupación lógica, se refiere a cómo se distribuye una base de datos de manera lógica, la memoria que utiliza y cómo se maneja. Es bueno saber que tanto espacio va a ocupar un tablespace en la memoria, así que vamos a calcular el espacio que ocupa.

Primeramente, tenemos que tener un diagrama relacional.

Para este caso vamos a contar con 2 Entidades.

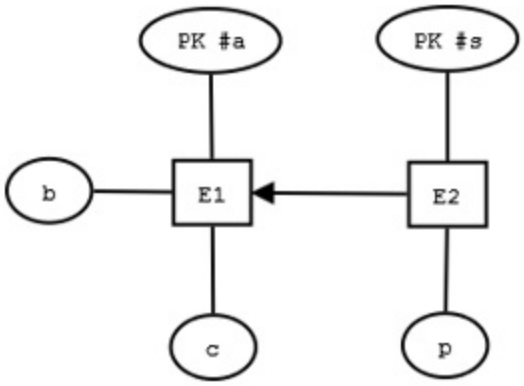
En este caso se cuenta con E1(a, b, c), donde son de tipo (Integer, Char, Char) respectivamente. E2(s, a), donde son de tipo (Integer, Integer) respectivamente.

El tamaño de un atributo tipo Integer es de 4 bytes y el tamaño de un atributo tipo Char es de 1 byte por carácter hasta 255 caracteres.

Lo primero que se debe hacer es calcular el tamaño de la tupla, eso se hace multiplicando cada dato por su tipo.

Ejemplo:

E1 (a = 4, b = 'ABC', c = 'QR') y E2 (s = 3, a = 4), ahora lo que hay que hacer es calcular el tamaño de la tupla.



E1		
Nombre	Tipo	Tamaño
#a	Integer	4 bytes
b	Char	1 byte por carácter
c	Char	1 byte por carácter

E2		
Nombre	Tipo	Tamaño
#s	Integer	4 bytes
a	Integer	4 bytes

Para E1: Er1 = (1*4) + (3*1) + (2*1) = 9 bytes

Para E2: Er2 = (1*4) + (1*4) = 8 bytes

Además, hay que considerar la tasa de transacción (T-T) de las entidades E1 y E2, donde es 25,000 transacciones por día de E1 y 38.000 transacciones por día de E2. Ahora se debe multiplicar las TrN con las tasas de transacción de cada entidad, con eso se calcula la cantidad de bytes que se usan por día.

Para E1: Tr1 -> 9 * 25,000 = 225000 bytes

Para E2: Tr2 -> 8 * 38,000 = 304000 bytes

Se deben sumar los tamaños de las transacciones totales para saber la cantidad de bytes que se consumen por día.

Tr1 + Tr2 = 529000 bytes

Además, se requiere también incluir las llaves primarias y foráneas de cada tabla de la relación. Hay que calcular para cada llave primaria y llave foránea su cantidad en bytes, la posición que es una dirección y la llave en si. Donde para E1 se tiene sólo una llave primaria que es 'a', su tipo es Integer lo cual representa 4 bytes y en la posición de memoria que ocupa, su apuntador es un long integer que ocupa 16 bytes. En el caso de E2 se tiene una primaria y foránea, para cada una su tipo es integer y su posición es long integer.

Ahora se calcula el tamaño de las llaves con sus posiciones en memoria (LM)

Ejemplo:

```
E1 → LlavesE1 (PK, posición) →  a(Integer) + long integer
       = 4 bytes + 16 bytes = 20 bytes
Total de E1 20 bytes.
E2 → LlavesE2 (PK, posición) → s(int) + long integer
       4 bytes + 16 bytes = 20 bytes
                                  → a(int) + long integer
       4 bytes + 16 bytes = 20 bytes
```

Como son 2 llaves ocupa 40 bytes en total.

Ya una vez calculados los tamaños de las transacciones y sus llaves, se debe hacer un estimado del espacio que puede llegar a ocupar, para este cálculo hay que considerar que las bases de datos relacionales de Oracle estiman que se debe aumentar en un 35 % del cálculo para ocupar mejor los recursos y recalcular periódicamente las transacciones.

La fórmula sería: (Tamaño de la tupla * T-T) + (LM * t.t * 0.35)

Para E1:

(9 * 25000) + (20 * 25000 * 0.35) = 400000 bytes

Para E2:

(8 * 38000) + (40 + 38000 * 0.35) = 317340 bytes

Estos resultados en total nos dan el espacio estimado de los tablespace que se consumiría por día.

400000 + 317340 = 717340 bytes / día.

A continuación algunos otros datos importantes en relación a los tablespaces.

Datafile

Un datafile son los "ficheros de datos" donde se almacena la información físicamente, este puede tener cualquier nombre y extensión, y puede estar localizado en cualquier directorio del disco duro, además está asociado a un solo tablespace y un tablespace está asociado a uno o varios datafiles.

Los datafiles tienen una propiedad llamada `AUTOEXTEND`, que en dado caso de que está activa, esta se encarga de que el datafile crezca automáticamente cada vez que se necesite espacio y no exista. Aunque el datafile este vacío, este tiene el tamaño en disco que ha sido indicado en su creación o el que el sistema le ha dado por defecto. Oracle hace

esto para direccionar espacio continuo en disco y evitar así la fragmentación. Cuando se vayan creando objetos en ese tablespace, se va ocupando el espacio a donde se direccionó la memoria.

Segment

Un segment (segmento) es aquel espacio direccionado por la base de datos dentro de un datafile para ser utilizado por un solo objeto. Así una tabla (o cualquier otro objeto) está dentro de su segment, y nunca podrá salir de él, ya que, si la tabla crece, el segment también crece. Se podría decir que el segment es la representación física de del objeto en base de datos.

Existen tres tipos de segments (principalmente):

- Segmentos de tipo `TABLE`: son aquellos que contienen tablas creadas en la base de datos.
- Segmentos de tipo `INDEX`: son aquellos que contienen índices, que contienen las direcciones de memoria donde se encuentran los datos, inicios de tablas, fin de tablas, etc.
- Segmentos de tipo `ROLLBACK`: son aquellos se usan para almacenar datos de información de la transacción realizada en la base para asi poder recuperar información o tener registros.
- Segmentos `TEMPORALES`: aquellos que se usan para realizar operaciones temporales que no pueden realizarse en memoria, tales como ordenaciones o agrupaciones de conjuntos grandes de datos.

Extend

Extent es un concepto que está relacionado con cualquier objeto que ocupe espacio de disco, es decir cualquier objeto que tenga un segment relacionado, que se direcciona una sola vez en el tiempo. El concepto de extent es un concepto físico, unos extent están separados de otros dentro del disco. Una vez aclarado que todo objeto tiene un segmento asociado, pero a su vez se compone de distintas extensiones. Un segmento, puede ser reservado una sola vez, en la creación se asignan 10MB de una vez, o de varias veces 5MB hoy y 5MB mañana.

Existen dos tipos de extends:

- `INITIAL` (extensiones iniciales): estas son las extensiones que se direccionan en el momento de la creación del objeto. Una vez que un objeto está creado, no se puede modificar su extensión inicial.
- `NEXT` (siguientes o subsiguientes extensiones): toda extensión reservada después de la creación del objeto. Si el `INITIAL EXTENT` de una tabla está llena y se está intentando insertar más filas, se intentará crear un `NEXT EXTENT` (siempre y cuando el datafile tenga espacio libre y tengamos cuota de ocupación suficiente).

Sabiendo que las extensiones se crean en momentos distintos de tiempo, es lógico pensar que unas extensiones pueden estar fragmentadas de otras. Un objeto de base de datos no está todo junto dentro del bloque, sino que se secciona en tantos bloques como extensiones tenga. Por eso es crítico definir un buen tamaño de extensión inicial, ya que, si es lo suficientemente grande, el objeto nunca estará fragmentado.

Si el objeto tiene muchas extensiones y éstas están muy separadas en disco, las consultas pueden tardar mucho más, ya que las cabezas lectoras tienes que dar saltos constantemente, por eso es que las secciones se crean con bytes de memoria que están consecutivos en el disco.

El tamaño de las extensiones (tanto las `INITIAL` como las `NEXT`), se definen durante la creación del objeto y no puede ser modificado después de la creación. Oracle recomienda que el tamaño del `INITIAL EXTENT` sea igual al tamaño del `NEXT EXTENT`.

La mejor solución es calcular el tamaño que tendrá el objeto ya sea una tabla o un índice, para esto se multiplica el tamaño de cada fila por una estimación del número de filas de la tabla. Este resultado es el que se debe utilizar para el tamaño de la extensión `INITIAL` y `NEXT`, evitando así la fragmentación en el disco. En caso de detectar más de 10 extensiones en un objeto es recomendable crear un nuevo objeto.

Data block

El concepto de Data block es un concepto físico, pero de una distribución lógica, donde el número de espacio de bytes contiguos de espacio físico en el disco, se maneja como bloques de memoria, esto es llamado paginación.

El soporte de paginación en las bases de datos Oracle aumenta el desempeño de las aplicaciones de base de datos de mucha memoria, especialmente en los casos en que el caché de buffer tiene varios gigabytes de tamaño, las CPU en el sistema podrán acceder más rápidamente a los buffers en la base de datos.

Ya que los bloques de datos representan la mínima unidad de almacenamiento que es capaz de manejar Oracle. En un disco duro no es posible que un fichero pequeño ocupe menos de lo que indique la unidad de asignación, así si la unidad de asignación es de 4 Kb, ya que así fue asignado por quien creó los ficheros, un fichero que ocupe 1 Kb en realidad ocupa 4 Kb.

3.4 Estándar de documentación

A continuación se dan las pautas para realizar la documentación de la arquitectura lógica

1. Iniciar en una introducción que contiene la documentación de la arquitectura lógica.
2. Realizar un gráfico de pastel para representar la división que se hizo de los tablespaces y asignar un color diferente para cada tablespace e indicar el nombre de cada tablespace.
3. Especificar cada tablespace, esta especificación debe contener sus nombres, cuales datos serán almacenados y también indicar a cual datafile se encuentra asociado o en caso de que sean varios indicar cuales.
4. Documentar como fueron calculados los espacios de disco la primera vez cuales fueron los criterios y fórmulas utilizadas para realizar los cálculos.
5. Documentar las fórmulas o estándar que se decida seguir utilizando después de implementado la base de datos para realizar las estimaciones de espacio.
6. Finalmente se recomienda estar actualizando la documentación de la arquitectura lógica cada vez que se cree un tablespace nuevo o se realiza cualquier otro cambio en la arquitectura lógica.

CAPÍTULO 4

Modelo de datos

Antes de comenzar con las indicaciones sobre cómo realizar el modelo de datos para la base de datos es importante tener bien claro que es un modelo de datos.

Un modelo de datos es un conjunto de herramientas conceptuales para la descripción de los datos y las relaciones entre ellos, su semántica y las restricciones de consistencia. Ahora bien, en este manual se utilizará el modelo de datos relacional esto dado que su simplicidad facilita el trabajo del programador en comparación con otros modelos. Sin embargo, primero se desarrollará un modelo basado en Entidad-Relación el cual después será trasformado finalmente en el Relacional, ya que E-R sirve para modelar el problema y reglas de negocio.

También a continuación se muestran algunos conceptos importantes relacionados con el diseño del modelo de bases de datos y también con respecto a integridad en las relaciones y base de datos antes de iniciar con el diseño del modelo

4.1 1. Modelo Entidad-Relación a Relacional

Este modelo es solamente un método que se aprovecha para diseñar los esquemas que posteriormente se deben implementar en la Base de datos. Este modelo se representa a través de diagramas y está formado por varios elementos que se analizarán a continuación.

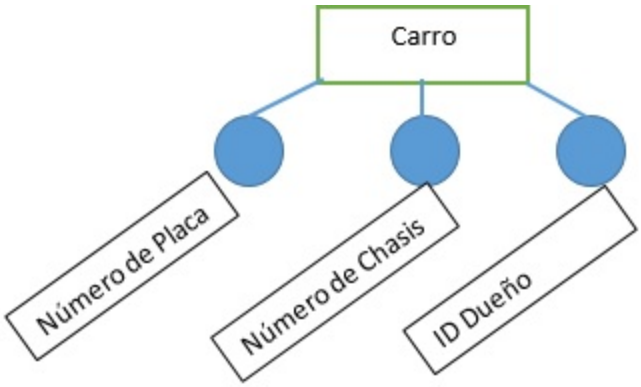
Este modelo además de tener un diagrama que ayuda a entender los datos y como se relacionan entre ellos, tiene que ser completado con la lista de los atributos y relaciones de cada elemento.

1.1. Entidad: Cada entidad representa cosas y objetos ya sean reales o abstractos que se diferencian entre sí. En un diagrama las entidades se representan con rectángulos.

1.2. Atributos: Los atributos definen las características de las entidades, son las propiedades de cada una. Cada entidad contiene distintos atributos, que dan información sobre esta entidad. Estos atributos pueden ser de distintos tipos (numéricos, texto, fecha...). Cada una de las columnas de una tabla. En un diagrama los atributos se representan con círculos que descienden de la entidad.

1.3.Tupla: La tupla son elementos formados por una fila de una tabla. Cada fila está de la entidad estaría compuesta por los atributos de la entidad correspondiente.

2 El concepto de relación



Una relación es lo que nos permite definir qué tipo de dependencia se debe dar entre entidades, es decir nos permite decir que ciertas entidades comparten ciertos atributos de manera necesaria.

Para manejarlo de una manera más gráfica se puede ver en un diagrama y en una tabla, los diagramas siempre se van a definir según las reglas del negocio.

Enunciado: los empleados de un taller tienen un cargo asociado.

Es decir, si un atributo de la entidad "Empleados" especifica que cargo tiene en el taller.

El cargo que ejerce debe coincidir con el que existe en la tabla de "cargos.

Tabla empleado

Nombre	Identificación	Cargo
Luis	115544223	001
Ana	223344556	002
Ignacio	998877665	001

Tabla Cargo de empleado

Cargo	Descripción
001	Mecánico
002	Gerente

3. Llaves en las relaciones

Una llave es un atributo de una entidad, a la cual hace distinguir a esta entidad de los demás registros. Primeramente, debemos saber cuáles son los tipos de llaves que existen en una entidad y como se pueden relacionar con otras entidades.

Una forma de representar en diagramas o textos la llave primaria es poniendo "#" como prefijo de la llave primaria (PK) o subrayando el nombre de la llave primaria.

Ejemplo:

- Estudiante (#ID, Nombre, Apellido)

- Estudiante (ID, Nombre, Apellido)

Llave primaria: Es un atributo (o columna) que restringe y distingue a las tuplas para que no se repitan en la misma entidad. La clave primaria es única. En algunos casos puede ser dos atributos lo cual conlleva al siguiente concepto.

Ejemplo:

Libros		
Código	Autor	Nombre
C2892-k	A	Libro1
M9899-L	B	Libro2
M8421-L	C	Libro3
A3430	D	Libro4

Llave primaria compuesta: Como su nombre lo dice es una llave primaria compuesta por varios atributos de una tabla generalmente una llave primaria compuesta está formada por dos llaves foráneas de la tabla.

Ejemplo:

Envíos			
Edificio	Sede	Oficina	Tipo
Correos CR	A3-SJ	3	10
Correos CR	A2-SJ	5	2
Correos CR	A5-H	1	1
Contraloría	A1-SJ	3	2

Llave foránea: es el atributo de una entidad que existe como dependencia en otra entidad, cuyos valores en las tuplas deben coincidir con valores de una llave que debe ser primaria de las tuplas de otra relación.

Libros		
Código	Autor	Nombre
C2892-k	880	Libro1
M9899-L	201	Libro2
M8421-L	201	Libro3
A3430	889	Libro4

En este caso la llave primara para identificar los libros es compuesta, El Código es propio del Libro y el Autor del Libro es la llave foránea, esta llave foránea es el código de Autor que es único en la tabla de Autor, pero en la tabla de Libros pueden repetirse, ya que hay libros que son escritos por un mismo autor.

4.2 3. Tipos de relaciones

Autor		
Código	Nombre	Apellido
880	Ana	Castro
201	Bruno	Díaz
225	Tony	Stark
889	Bruce	Banner

Existen diferentes tipos de relaciones según tengan parte las entidades en ellas. Las relaciones están dadas por el enunciado, como en el ejemplo anterior podemos analizar que cada empleado puede tener solamente un cargo, pero varios empleados pueden tener el mismo cargo.

La cardinalidad se miden con un mínimo y un máximo al lado de la entidad, en ese orden respectivamente. (min,max).

3.1 Relación uno a uno: una entidad se relaciona únicamente con otra entidad y viceversa. Por ejemplo, si tenemos una entidad chasis y otra con matrículas del chasis del auto, debemos determinar que cada chasis solo puede tener una matrícula y una matrícula solo puede existir en un chasis especifico.

Seleccionando el máximo en cada carnalidad se toma a 1 para esta representación, se leen las reglas de negocio como: Un chasis tiene una matrícula. Una Matricula única está en un solo chasis.

A y B son una entidad, R es la relación que tienen.

3.2 Uno a varios: una entidad puede estar asociada con varias entidades, pero en esta segunda entidad, la primera existir solo una vez.

Un ejemplo concreto sería:

Las reglas de negocio se leerían como: A un estudiante se le pueden prestar varios libros. Varios libros pueden ser prestados únicamente a un estudiante.

Al pasar de una relación de modelo entidad-relación a modelo relacional, se toma como llave primaria de la relación, la llave primaria de la relación de cardinalidad N.

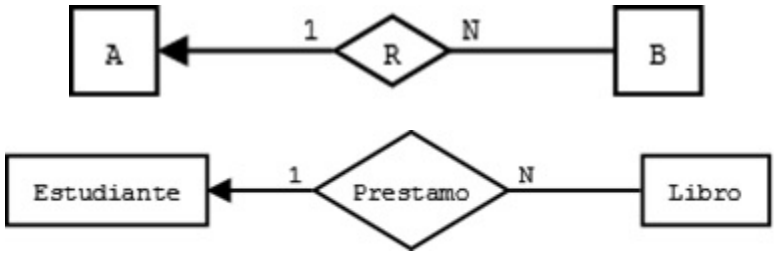
Si tenemos Estudiante(ID, Nombre, Apellido) y Libro(Código, Nombre, Autor), La llave primaria de la relación va a ser Código de libro.

3.3 Varios a varios: una entidad puede estar asociada con otra con ninguno o varias entidades y viceversa.

En este caso se pueden leer las reglas de negocio como: A varios estudiantes se les puede prestar varios libros. Varios libros pueden ser prestados a varios estudiantes.

Al pasar este modelo entidad-relación a las bases de datos relacionales, se deben cambiar las cardinalidades de las relaciones, los modelos relaciones soportan cardinalidades de 1 a N.

Al pasar de una relación de N a N a modelo relacional, la llave primaria de la relación es una llave primaria compuesta conformada por las llaves primarias de las relaciones.



Si tenemos Estudiante(ID, Nombre, Apellido) y Libro(Código, Nombre, Autor), La llave primaria de la relación va a ser (ID, Código) de libro.

4.3 4. Conceptos sobre relaciones

4.1. Relación R(Ai..An)

Es un un subconjunto del producto cartesiano D1×..×Dn , donde D son tablas del modelo de datos.

Es importante recordar que en una relación hay que distinguir dos aspectos:

4.2. Esquema de la relación: Los atributos A1...An

p.ej. Trabajadores (id_trabajador, nombre, tarifa_hr, tipo_de_oficio, id_supv).

4.3. Instancia de la relación: El conjunto de tuplas {(X1, x2,.., xn)} D1×D2×...×Dn que la componen en cada momento.

4.4. Restricción de integridad: Es una condición necesaria para preservar la corrección semántica de la base de datos.

4.5. Esquema de la base de datos: Colección de esquemas de relaciones junto con las restricciones de integridad que se definen sobre las relaciones.7

4.6 Dominio (en el contexto del modelo relacional) Conjunto de valores atómicos.

4.4 5. Restricciones de integridad asociadas a las relaciones de la base de datos

---

Una base de datos contiene datos que mayormente deben reflejar la realidad o bien la situación de una del mundo real dado por las reglas del negocio. En el caso de las bases de datos relacionales, esto significa que la extensión de las relaciones (es decir, las tuplas que contienen las relaciones) deben tener valores que reflejen la realidad correctamente.

Suele ser bastante frecuente que determinadas configuraciones de valores para las tuplas de las relaciones no tengan sentido, porque no representan ninguna situación posible del mundo real.

Restricciones de integridad asociadas a las tuplas de una relación:

Los datos que se ingresen a las tablas en las tuplas deben tener un sentido lógico representativo, algo de acuerdo con la realidad.

Ejemplo

En el caso de ingresar una edad, debe cumplir con las condiciones:

0 edad 120.

En el caso de ingresar los impuestos, debe cumplir con la condición:

impuestos sueldo.

Las restricciones de integridad de usuario: son condiciones específicas de una base de datos concreta; es decir, son las que se deben cumplir en una base de datos particular con unos usuarios concretos, pero que no son necesariamente relevantes en otra base de datos.

Ejemplo:

Dada una tabla de empleados con un atributo sueldo, según la cual los sueldos no podían ser negativos. Hay que tomar en cuenta que este caso en la base de datos tiene el atributo sueldo, al que se quería dar un valor; sin embargo, podría no ser necesaria en otra base de datos diferente donde, por ejemplo, no hubiese sueldos.

Las reglas de integridad de modelo: son condiciones más generales, propias de un modelo de datos, y se deben cumplir en toda base de datos que siga dicho modelo.

Ejemplo:

En el caso del modelo de datos relacional, habrá una regla de integridad para garantizar que los valores de una clave primaria de una relación no se repitan en tuplas diferentes de la relación. Toda base de datos relacional debe cumplir esta regla que, por lo tanto, es una regla de integridad del modelo.

Los SGBD deben proporcionar la forma de definir las restricciones de integridad de usuario de una base de datos; una vez definidas, deben velar por su cumplimiento, la manera de hacer esto es mediante restricciones en el lenguaje SQL.

Las reglas de integridad del modelo, no se deben definir para cada base de datos concreta, porque se consideran preestablecidas para todas las bases de datos de un modelo. Un SGBD de un modelo determinado debe velar por el cumplimiento de las reglas de integridad preestablecidas por su modelo.

Seguidamente vamos a estudiar con detalle las reglas de integridad del modelo relacional, reglas que todo SGBD relacional debe obligar a cumplir.

5.1. Regla de integridad de unicidad de la clave primaria

La regla de integridad de unicidad está relacionada con la definición de clave primaria. Concretamente, establece que toda clave primaria que se elija para una relación no debe tener valores repetidos.

Ejemplo:

Envíos			
Edificio	Sede	Oficina	Tipo
Correos CR	A3-SJ	3	10
Correos CR	A2-SJ	5	2
Correos CR	A5-H	1	1
Contraloría	A1-SJ	3	2

En este ejemplo el edificio y la sede son la llave primaria, puede haber varios edificios llamados “Correos CR” pero solo 1 asociado a una “Sede” especifica. Quiere decir que se establece que, si la llave primaria es compuesta o simple,

sea la clave primaria de una relación, entonces la extensión de esta no puede tener en ningún momento dos tuplas con la misma combinación de valores en la llave primaria.

5.2. Regla de integridad de entidad de la clave primaria

Los atributos de la clave primaria de una relación no pueden tener valores nulos.

Ejemplo

Tenemos la siguiente relación:

Envíos			
Edificio	Sede	Oficina	Tipo
Correos CR	A3-SJ	3	10
Correos CR	A2-SJ	5	2
Correos CR	A5-H	1	1
Contraloría	A1-SJ	3	2

En esta relación, puesto que la clave primaria está formada por edificio y sede, no hay ningún envió que tenga un valor nulo para edificio, ni tampoco para número. Esta regla es necesaria para que los valores de las claves primarias puedan identificar las tuplas individuales de las relaciones. Si las claves primarias tuvieran valores nulos, es posible que algunas tuplas no se pudieran distinguir.

5.3. Regla de integridad referencial

Esta regla se relaciona directamente con el concepto de llave foránea. Determina que todos los valores que toma una clave foránea deben ser valores nulos o valores que existen en la clave primaria que referencia.

La necesidad de la regla de integridad relacional proviene del hecho de que las llaves foráneas tienen por objetivo establecer una conexión con la clave primaria que referencian. Si un valor de una llave foránea no estuviese presente en la clave primaria correspondiente, representaría una referencia o una conexión incorrecta.

Un SGBD relacional tendrá que hacer cumplir esta regla de integridad. Deberá realizar las comprobaciones cuando se produzcan las siguientes operaciones:

1. Inserciones en una relación que tenga una clave foránea.
2. Modificaciones que afecten a atributos que pertenecen a la clave foránea de una relación.
3. Borrados en relaciones referenciadas por otras relaciones.
4. Modificaciones que afecten a atributos que pertenecen a la clave primaria de una relación referenciada por otra relación.

La integridad referencial mantiene las conexiones en las bases de datos relacionales. Es decir que si se refiere a una entidad con llave foránea, esta no se puede eliminar, si se modifica la llave se modifica en la tabla a la que se refiere.

Ejemplos:

imparte.NRP profesor.NRP

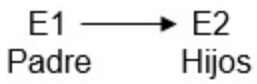
El profesor que imparte una asignatura debe existir en la tabla de profesores.

cuenta.sucursal sucursal.numero

Una cuenta tiene que pertenecerá una sucursal existente.

En este caso para mantener la integridad referencial no se debe eliminar el padre.

5.4. Regla de integridad de dominio



La regla de integridad de dominio está relacionada, como su nombre indica, con la noción de dominio. Esta regla establece dos condiciones.

- La primera condición consiste en que un valor no nulo de un atributo Ai debe pertenecer al dominio del atributo Ai; es decir, debe pertenecer a dominio(Ai).

Esta condición implica que todos los valores no nulos que contiene la base de datos para un determinado atributo de una tabla, deben ser del dominio declarado para dicho atributo.

Ejemplo Si en una relación de Empleados(ID,nombre,apellido,sueldo), se define el atributo ID como un entero de 9 digitos, no podemos ingresar en la tabla el valor "Ana", ya que no es un entero

- La segunda condición sirve para establecer que los operadores que pueden aplicarse sobre los valores dependen de los dominios de estos valores; es decir, un operador determinado sólo se puede aplicar sobre valores que tengan dominios que le sean adecuados.

Ejemplo Si en la relación EMPLEADOS(ID, nombre, apellido, edademp) se ha declarado que dominio(ID) es el dominio predefinido de los enteros, entonces no se permitirá consultar todos aquellos empleados cuyo ID sea igual a 'Ana' (DNI = 'Ana'). El motivo es que no tiene sentido que el operador de comparación = se aplique entre un DNI que tiene por dominio los enteros, y el valor 'Ana', que es una serie de caracteres.

4.5 6. Normalización

La normalización es un proceso que se realiza en las bases de datos para designar y aplicar una serie de reglas a las relaciones y restricciones ya estudiadas anteriormente, que se encuentren al pasar de un modelo Entidad-Relación a un modelo Relacional.

Una base de datos se normaliza por las siguientes razones:

- Proteger la integridad de los datos.
- Evitar que haya redundancia de datos.
- Para disminuir la probabilidad de que se den problemas de actualización de datos.

6.1 Dependencia funcional

Definición: Sean Xi y Yi subconjuntos de atributos de una relación. Yi tiene una dependencia funcional de Xi, también Xi determina a Yi, si cada valor de Xi tiene asociado siempre un único valor del subconjunto Yi.

De manera más simple, se puede ver como la conexión entre uno o más atributos de una entidad con otra. Se puede expresar como, un conjunto que puede ser expresado en otras funciones.

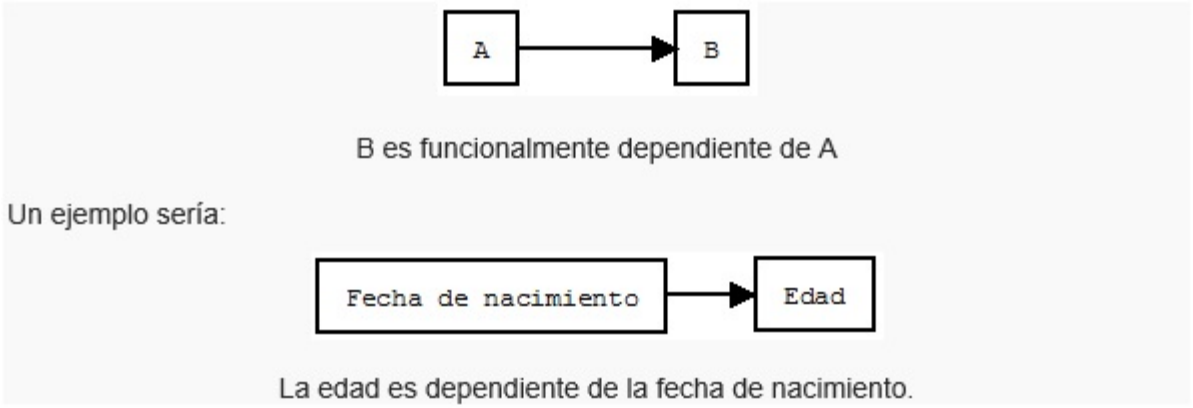
6.1.1 Tipos de Dependencias Funcionales

a) Dependencia Total: se dice que ocurre dependencia total si se tiene un "y" que solo dependa de un "x" y no de ningún otro atributo.

$x \rightarrow y$

b) Dependencia Parcial: esta dependencia ocurre cuando un "y" depende de manera parcial de 2 atributos un "xz" y un "z".

$xz \rightarrow y$



$z \rightarrow y$

A la hora de resolver esta dependencia generaría una solución de esta manera: {(x, y), (y, z)}.

c) Dependencia Transitiva: es cuando un "y" depende de un "x", pero que también hay un "z" que depende de ese "y"; por lo tanto, por medio de esta dependencia se dice que el "c" depende de "x".

$x \rightarrow y$

$y \rightarrow z$

Transitividad $x \rightarrow z$

Una tabla se da por normaliza cuando se cumple el teorema de "Descomposición de Producto sin Pérdida", el cual se puede definir de la siguiente manera:

> Sea R [x, y, z] y se verifica que "$y \rightarrow z$", entonces:

R = R1 [x, y] U R2 [y, z]

Para que una tabla esté normalizada tienen estar aplicadas al menos las 3 primeras formas normales, ya que se dice que una base de datos está normalizada si la base está en forma N, es decir que sus tablas están en forma normal N.

Primera Forma Normal (1FN): No debe tener grupo repetitivos.

Ejemplo:

> Analizaremos el caso donde en una tabla de estudiantes se presenta un grupo repetido.

Estudiante		
ID	Nombre	Carrera
112233	Ana	Comercio
445566	Juan	Educación Matemática

Aquí la tabla está bien pero no en 1FN, ya que Juan en la Carrera tiene 2 carreras en una misma celda, se podrían poner en una misma celda de caracteres, pero no quiere decir que este en forma relacional. La manera correcta de construir esta tabla es la siguiente:

Aquí las carreras de Juan están separadas en 2 tuplas. Lo que hace cumplir con la 1FN.

Segunda Forma Normal (2FN): Ocurre si la tabla está en 1FN y además cada atributo que no sea llave depende funcionalmente respecto de cualquiera de las llaves. Se da si no hay dependencia Parcial.

Estudiante		
ID	Nombre	Carrera
112233	Ana	Comercio
445566	Juan	Educación
445566	Juan	Matemática

Toda la llave primaria debe hacer dependiente al resto de atributos, si hay atributos que dependen solo de parte de la llave primaria, entonces esa parte de la llave y esos atributos formarán otra tabla.

Ejemplo:

Asistencia				
ID	Código de Curso	Nombre	Apellido	Nota
112233	44	Ana	Castro	10
112233	55	Ana	Castro	9
445566	87	Bruce	Banner	7
445566	30	Tony	Stark	8

Esta tabla está en 1FN, ahora vamos a pasarla a 2FN.

Suponiendo que el ID y Código de Curso formen una llave primaria para esta tabla, sólo la nota tiene dependencia funcional completa, el nombre y el apellido dependen de forma completa del ID, no de toda la llave primaria. Para solucionar esto se debe:

Ahora solo la nota depende totalmente de la llave primaria en la Tabla asistencia, la tabla profesor tiene el Nombre y Apellido, que existe una dependencia completa sobre el ID de profesor.

Tercera Forma Normal (3FN): Ocurre cuando una tabla está en 2FN y además ningún atributo que no sea clave depende transitivamente de las claves de la tabla. Es decir, no ocurre cuando algún atributo depende funcionalmente de atributos que no son clave. No debe haber dependencias entre los mismos atributos.

Ejemplo:

En esta tabla la provincia depende funcionalmente del código postal, lo que hace que no esté en 3FN. Para solucionarlo:

Ahora la provincia no está en la tabla de profesor y está en la tabla código postal, donde hay una depende completa con el Cód. Postal de la tabla.

Ejemplo:

Se desarrollará un ejemplo con atributos de una tabla de forma más genérica.

Sea T(a, b, c, d, e, f, g, h) la tabla con sus atributos.

f={ a→e, a→b,a→c, f→d, a→g, g→h },

```
T ( a , f , g )
T1 ( a, e , b , c , g )
T1.2 ( f , d )
T2 ( g , h )
```

Profesor		
ID	Nombre	Apellido
112233	Ana	Castro
112233	Ana	Castro
445566	Bruce	Banner
445566	Tony	Stark

Asistencia		
ID	Código de Curso	Nota
112233	44	10
112233	55	9
445566	87	7
445566	30	8

Profesor				
ID	Nombre	Apellido	Cód. Postal	Provincia
112233	Ana	Castro	10109	San José
112233	Ana	Castro	51098	Guanacaste
445566	Bruce	Banner	44080	Heredia
445566	Tony	Stark	60165	Puntarenas

Profesor			
ID	Nombre	Apellido	Cód. Postal
112233	Ana	Castro	10109
112233	Ana	Castro	51098
445566	Bruce	Banner	44080
445566	Tony	Stark	60165

Cod. Postal	
Cód. Postal	Provincia
10109	San José
51098	Guanacaste
44080	Heredia
60165	Puntarenas

De acuerdo a lo planteado anteriormente podemos llegar a la conclusión de que la normalización de los datos puede considerarse como un proceso durante el cual los esquemas de relación insatisfactorios se descomponen repartiendo sus atributos entre esquemas de relación más pequeños que poseen propiedades deseables sin dependencias que no son necesarias.

Ahora se sabe con certeza que para pasar de un modelo Entidad-Relacion, es necesario que la base de datos esté en 3FN, al cumplir con la tercena forma normal también garantiza un buen funcionamiento de la bese de datos

Una ventaja clara para las bases de datos relacionales al normalizarse es la optimización del espacio en la memoria física y un mejor manejo de datos a la hora de hacer consultas SQL estructurado y manejable, reduciendo posibles errores de datos como redundancia de datos.

4.6 Estándar de documentación

A continuación se indican las pautas a seguir para realizar la documentación del modelo de datos.

1. Iniciar con una pequeña introducción donde se indique los puntos que serán explicados en esta documentación.
2. Realizar los siguientes pasos para ambos diagramas primero para Entidad- Relación y luego para el Relacional a excepción del paso 6 ya que se trata del SQL utilizado que en este caso sería únicamente del relacional.
3. Identificar las entidades, indicando cuáles son sus atributos descriptivos y sus restricciones.
4. Identificar las relaciones, indicando sus atributos descriptivos, reglas de negocio y restricciones.
5. Realizar un gráfico del modelo de datos.
6. Adjuntar el SQL que se utilizó en la implementación en este caso sería el del modelo de datos relacional.
7. Finalmente es importante que esta documentación del modelo de datos siempre se encuentra actualizada cuando se realicen cambios en las entidades, atributos descriptivos, restricciones o cualquier otro cambio.

CAPÍTULO 5

Componentes PLsql

En esta sección del manual se definen algunos conceptos básicos pero muy importantes a la hora del desarrollo de los procedimientos almacenados, funciones, trigggers y demás componentes del PL/SQL. También algunos ejemplos de los componentes PL/SQL antes mencionados y su documentación dentro de un catálogo. La documentación de estos componentes es muy importante por lo tanto se recomienda realizar un trabajo ordenado para facilitar el mantenimiento de estos componentes en la base de datos. Otro aspecto importante es que los procesos y funciones almacenas son de gran ayuda en muchas empresas ya que les facilita el trabajo en cuanto se refiere a realizar algunos cálculos empresariales, además un aspecto aún más importantes es que muchos de estos resultados que muestran las bases de datos por medio de estos componentes de programación son vitales para la toma de decisiones en muchas empresas ya que pueden indicar si una empresa está teniendo un déficit de ventas por lo cuál sea necesario realizar un reajuste para mejorar dicha situación, esto para un empresa que se dedique a las ventas. Otro ejemplo es el uso del lanzamiento de triggers para realizar bitácoras lo cual es de bastante ayuda en casos de pérdida de información.También en una base de datos de un hospital puede ayudar a realizar estadísticas sobre enfermedades que ayuden en un futuro a combatirlas de una mejor manera. Es por ello que se hace el énfasis en realizar un buen trabajo en esta parte de la base de datos.

PL/SQL es el lenguaje de programación que proporciona Oracle para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

5.1 Construyendo bloques de programas PL/SQL

PL/SQL es un lenguaje estructurado con bloques. Un bloque PL/SQL es definido por las palabras clave `DECLARE`, `BEGIN`, `EXCEPTION`, y `END`, que dividen el bloque en tres secciones

1. Declarativa: sentencias que declaran variables, constantes y otros elementos de código, que después pueden ser usados dentro del bloque

2. Ejecutable: sentencias que se ejecutan cuando se ejecuta el bloque

3. Manejo de excepciones: una sección especialmente estructurada para atrapar y manejar cualquier excepción que se produzca durante la ejecución de la sección ejecutable

Unidades que se pueden programar en una base de datos Oracle:

- Procedimientos almacenados
- Funciones
- Triggers

5.2 Conceptos Básicos

• **DELIMITADOR:** Es un símbolo, simple o compuesto, que tiene una función especial en PL/SQL:

- Operadores Aritméticos
- Operadores Lógicos
- Operadores Relacionales

• **IDENTIFICADOR:** Son empleados para nombrar objetos de programas en PL/SQL, así como a unidades dentro del mismo:

- Constantes
- Cursores
- Variables
- Subprogramas
- Excepciones
- Paquetes

• **LITERAL:** Es un valor de tipo numérico, carácter, cadena o lógico no representado por un identificador (es un valor explícito).

• **COMENTARIO:** Es una aclaración que el programador incluye en el código. Son soportados dos estilos de comentarios, el de línea simple y de multilínea, para lo cual son empleados ciertos caracteres especiales.

5.3 Tipos de Datos

PL/SQL proporciona una variedad de tipos de datos para especificar el formato de almacenamiento, restricciones y rango de valores válidos de constantes y variables.A continuación una lista de los tipos de datos más comunes:

- `NUMBER` (numérico): Almacena números enteros o de punto flotante..
- *CHAR* (carácter): Almacena datos de tipo carácter con un tamaño máximo de 32.767 bytes y cuyo valor de longitud por defecto es 1.
- `VARCHAR2` (carácter de longitud variable): Almacena datos de tipo carácter empleando sólo la cantidad necesaria aun cuando la longitud máxima sea mayor.
- `BOOLEAN` (lógico): Almacena valores TRUE o FALSE.
- `DATE` (fecha): Almacena datos de tipo fecha. Las fechas se almacenan internamente como datos numéricos, por lo cual se puede realizar operaciones aritméticas con ellas.
- Atributos de tipo. Un atributo de tipo PL/SQL es un modificador que puede ser usado para obtener información de un objeto de la base de datos. El atributo `%TYPE` permite conocer el tipo de una variable, constante o campo de la base de datos. El atributo `%ROWTYPE` permite obtener los tipos de todos los campos de una tabla de la base de datos, de una vista o de un cursor.

5.4 Nombres en una base Oracle

Estas son las reglas para construir identificadores válidos en una base Oracle:

- El largo máximo es de 30 caracteres.
- El primer caracter debe ser una letra, pero cada caracter después del primero puede ser una letra, un número (0 a 9), un signo de pesos ($), un guión bajo (_), o un numeral (#). Todos los siguientes son identificadores válidos.

```
hola_mundo
hola$mundo
hola#mundo
```

pero estos son inválidos:

```
1hola_mundo
hola%mundo
```

- PL/SQL es case-insensitive (no es sensitivo a mayúsculas y minúsculas) con respecto a los identificadores. PL/SQL trata todos los siguientes como el mismo identificador

```
hola_mundo
Hola_Mundo
HOLA_MUNDO
```

Para ofrecer más flexibilidad, Oracle permite evitar las restricciones de la segunda y tercera regla, encerrando al identificador entre comillas dobles. Un quoted identifier (identificador encerrado entre comillas) puede contener cualquier secuencia de caracteres imprimibles excluyendo las comillas dobles; las diferencias entre mayúsculas y minúsculas serán además preservadas. Así, todos los siguientes identificadores son válidos y distintos:

```
"Abc"
"ABC"
"a b c"
```

Estas mismas reglas aplican a los nombres de los objetos de base de datos como tablas, vistas y procedimientos, con una regla adicional: a menos que se encierren entre comillas los nombres de estos objetos, Oracle los mantendrá en mayúsculas.

También se buscará la consistencia en el empleo de mayúsculas y minúsculas, lo que facilitará la elaboración de herramientas de administración. En general, todo el código se escribirá con mayúsculas con excepción a los textos entre comillas.

5.5 Procedimientos Almacenados

A continuación se muestran algunos procesos almacenados básicos pero que en en este caso funcionan para ejemplificar como se debería dar la documentación de los mismos. A la hora de crear los procesos almacenados es muy importante la identificación de los mismos es preferible que se utilice un prefijo en el identificador similar para todos los procesos almacenados, así como también para funciones y triggers esto por ayudará en el mantenimiento de la base de datos ya que facilitará las búsquedas de los mismos cuando se requieran hacer auditorias o sea necesario reparar un problema que se esté dando. Para estos ejemplos se utilizará el e siguiente prefijo SP el cual ira seguido por un consecutivo por ejemplo `SP001` y `SP002` y así sucesivamente claro está que esta escogencia del prefijo quedará a cargo de los creadores de la base de datos.

La siguiente es una tabla llamada T1 la cual se usará para ejemplificar el desarrollo de algunos procedimientos almacenados los cuales a su vez modificarán el contenido de sus registros.

```
Create table  t1(
a int,
b int,
c int   );
```

Insertar

El siguiente método se utilizará para insertar registros dentro base de datos, en este caso específicamente en la tabla t1 , se utiliza la recomendación anteriormente mencionada sobre la asignación del identificador utilizado para el proceso almacenado en este caso se utiliza `SP001` más adelante en los siguientes procesos se podrá ver como se continua con un prefijo de nombre similar para los demás procesos.

```
CREATE OR REPLACE PROCEDURE SP001 (X INT , Y   INT  , Z INT)
AS
BEGIN
INSERT INTO T1 (A,B,C) VALUES(X,Y,Z);
COMMIT;
END;
/
```

Es importante no olvidar que después de escribir el procedimiento almacenado para finalizar la operación se coloca un / y para ejecutar un proceso se utiliza execute nombre del Proceso (variable tipo). Ejemplo:

```
EXECUTE  SP001(1,2,3);
```

Actualizar

El siguiente método se utilizará para realizar una actualización algún registro dentro base de datos, en este caso específicamente al atributo a con el valor igual a 1 de la tabla T1 para este caso el nombre que se le asigna es `SP002` continuando así con la secuencia de nombres asignados.

```
CREATE OR REPLACE PROCEDURE SP002 ( X INT )
AS
BEGIN
UPDATE T1 SET A = X
WHERE A = 1;
COMMIT;
END;
```

Eliminar

El siguiente método se utilizará para eliminar algún dato dentro base de datos, en este caso específicamente al atributo a con el valor igual a la variable x la cuál le es enviada por parámetro. También como en los casos anteriores se le asigna el mismo prefijo pero con diferente consecutivo `SP003`.

```
CREATE OR REPLACE PROCEDURE SP003 ( X INT )
AS
BEGIN
DELETE FROM T1 WHERE A = X;
COMMIT;
END;
```

A continuación se dará una referencia de como documentar los procedimientos almacenados el cual también se podrá usar para documentar otros componentes como funciones almacenadas y triggers mediante un catálogo de objetos para este primer caso será un catálogo de procedimientos almacenados. Además de documentar de manera digital este catálogo también se debe documentar a nivel de la base datos mediante la creación de una tabla específicamente hecha para esta documentación de los procesos almacenados.

Catálogo de Procedimientos Almacenados

Paquete	Identificador	Parámetros	Versión	Responsable	Detalles
BD1	SP001	x int , y int , z int	1	Pablo	Insertar datos en T1
BD1	SP002	X int	1	Pablo	Actualizar datos en T1
BD1	SP003	X int	1	Pablo	Eliminar registros de T1

Ahora se procederá a explicar cada una de las columnas de este recuadro y los valores que contienen las filas.

- Paquete: Indica el nombre del paquete donde se van a guardar subprogramas y otros objetos de la base de datos (los procedimientos y funciones).
- Identificador: Es el nombre asignado al proceso almacenado para identificarlo.
- Parámetros: Contiene los parámetros del proceso así como el tipo de cada uno.
- Versión: Lo cual indica como su nombre lo dice que versión del procedimiento.
- Responsable: La columna responsable muestra quien es el responsable de dar mantenimiento a este procedimiento.
- Detalles: Detalla un poco más fondo cuál es la función principal del procedimiento almacenado.

5.6 Funciones Almacenadas

Ahora se procederá a explicar un poco del proceso de la creación y documentación de funciones almacenadas en la base datos el cuál es muy similar al anterior.

Para explicar este procedimiento se debe crear la siguiente.

```
CREATE TABLE  T2(
D INT,
E INT,
F INT
);
```

Función Sumatoria

La siguiente función es una función almacenada que se encarga de realizar una sumatoria de la multiplicación hecha entre dos valores de dos tablas en este caso de las tablas t1 y t2, esta es una función muy utilizada en bases de datos que requieren cálculos diarios en bases de datos de ventas de muchas empresas. Retomando también un poco lo que es el identificador de la función se le asignó SF_001 en donde más adelante se podrá ver como se seguirá utilizando el prefijo `SF_` para otra función almacenada.

```
CREATE OR REPLACE FUNCTION SF_001(X INTEGER)
RETURN FLOAT
```

```
IS
RESULTADO FLOAT;
BEGIN
SELECT SUM(T1.A * T2.D)
INTO RESULTADO
FROM  T1, T2
WHERE T1.A= X ;
RETURN RESULTADO;
END;
/
```

Función Suma Valor

La siguiente función es una función almacenada que se encarga de realizar una sumatoria de un valor en la tabla t2, esta función es similar a la anterior con la diferencia de que es una sumatoria de la suma del valor de la columna e de la tabla t2 respectivamente, lo cual es una práctica utilizada para realizar cálculos por ejemplo de la suma de las ventas por un vendedor en un mes dado. De manera similar a la función anterior el identificador de la función que se le asignó es SF_002 manteniendo la política de nombres.

```
CREATE OR REPLACE FUNCTION SF_002(X INTEGER)
RETURN FLOAT
IS
RESULTADO FLOAT;
BEGIN
SELECT SUM(T2.E)
INTO RESULTADO
FROM  T2
WHERE T2.D= X ;
RETURN RESULTADO;
END;
/
```

A continuación se dará una referencia de como documentar las funciones almacenadas en un catálogo de funciones. Además de documentar de manera digital este catálogo también se debe documentar a nivel de la base datos mediante la creación de una tabla específicamente hecha para esta documentación de las funciones almacenadas.

Catálogo de Funciones Almacenadas

Paquete	Identificador	Parámetros	Versión	Responsable	Detalles
BD1	SF_001	x int	1	Pablo	Sumatoria de la multiplicación de dos valores
BD1	SF_002	X int	1	Pablo	Sumatoria de valores de columna e tabla t2

5.7 Triggers

En este apartado se mostrará como crean un trigger y también la manera recomendada de documentarlos.

La siguientes tablas se crearán para ayudar al desarrollo y almacenamiento respectivamente en el trigger que será explicado en el siguiente punto.

```
Créate table t3(
X int ,
Y int
);


create table bitacora(
oldX int,
oldY int,
newX int,
newY int,
fechahora date,
accion varchar(20)
);


Trigger  Bitacora TR_001
```

El siguiente trigger tiene como función insertar la información necesaria en la tabla bitácora. Así con dicha información en la tabla bitácora será de gran ayuda para un rollback en la base de datos o también una función aún más importante la de realizar un redo log o copia de seguridad de las transacciones hechas en la base de datos y así colaborar en una eventual pérdida de información. Para los triggers el prefijo utilizado es `TR_` seguido por un consecutivo.

```
CREATE OR REPLACE TRIGGER TR_001
AFTER INSERT OR UPDATE OR DELETE
ON T3
FOR EACH ROW
BEGIN
 IF INSERTING THEN
  INSERT INTO BITACORA ( OLDX , OLDY , NEWX   ,  NEWY , FECHAHORA , ACCION) VALUES (:OLD.X  ,  :OLD.Y
ELSIF UPDATING THEN
 INSERT INTO BITACORA ( OLDX , OLDY , NEWX ,   NEWY , FECHAHORA , ACCION) VALUES (:OLD.X ,   :OLD.Y ,
 ELSE
  INSERT INTO BITACORA ( OLDX , OLDY , NEWX  ,  NEWY , FECHAHORA , ACCION) VALUES(:OLD.X,   :OLD.Y,
 END IF;
END;
/
```

A continuación se dará una referencia de como documentar los triggers en un catálogo. Además de documentar de manera digital este catálogo también se debe documentar a nivel de la base datos mediante la creación de una tabla específicamente hecha para esta documentación de los triggers creados.

Los procedimientos almacenados y funciones se guardaran en paquetes (`package`). Es necesario distribuir nuestros procesos funciones y procedimientos en paquetes para dar una mejor estructura al proyecto en el que estamos trabajando, para empezar un paquete tiene dos niveles (Declaración y Cuerpo), en la declaración solo declaramos los procedimientos y funciones con sus parámetros. En el cuerpo desarrollamos el código de nuestros procedimientos y funciones. Por ejemplo:

Catálogo de Triggers

Paquete	Identificador	Parámetros	Versión	Responsable	Detalles
BD1	TR_001	x int	1	Pablo	Realiza inserción de información de transacciones a la tabla bitácora

```
CREATE OR REPLACE PACKAGE BD1 AS
  PROCEDURE SP001 (X INT, Y int  , Z INT);
  PROCEDURE SP002 ( X INT);
  PROCEDURE SP003 ( X INT);

  SF_001(X INTEGER);
  SF_002(X INTEGER);
END BD1;
/
```

Ahora veremos como llamar a los procedimientos o funciones dentro de dicho paquete:

```
BD1.SP001(3,4,5);
BD1.SF_001(7);
```

5.8 Catálogo General

En este último apartado se hará un ejemplo de cómo se podría realizar un catálogo general que contenga la información general de los catálogos de PL/SQL con los que cuenta la base de datos. Así mismo para ayudar a la identificación de los mismo se recomienda la utilización de un prefijo similar para todos por ejemplo en este caso se utilizará `ctbd1_` seguido por el nombre de catálogo. También es importante como en los catálogos anteriores aparte de mantener este catálogo.

La idea de crear este catálogo general de los componentes PL/SQL nace a partir del punto de simular una estructura similar al diccionario de datos, sin embargo la función de este catálogo es la de facilitar el acceso de información a los administradores de bases de datos el acceso a esta información ya que por ejemplos si necesitan saber el número de catálogos de los componentes PL/SQL creados para la base de datos basta con realizar la consulta a la base de datos.

Catálogo de componentes PL/SQL

Nombre del Catálogo	Descripción
Ctdb1_Procedures	Este catálogo contiene toda la información concerniente a los procesos almacenados creados.
Ctdb1_Functions	Este catálogo contiene toda la información concerniente a las funciones almacenadas.
Ctdb1_Triggers	Este catálogo contiene toda la información concerniente a los triggers creados.

CAPÍTULO 6

Recomendaciones

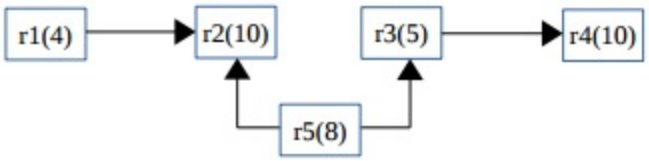
A continuación para finalizar el manual se dan algunas recomendaciones básicas para ser tomadas en cuenta finalizado el trabajo de diseño e implementación de la base de datos, sin embargo es importante tomar en cuenta que lo siguiente son solo algunas sugerencias ya que la principal función de este manual es guiar en el diseño, implementación y documentación de una base de datos pero se considera importante dar algunas referencias para ser tomadas en cuenta en un futuro. También se ha venido hablando en las secciones anteriores del manual la importación de velar por la seguridad de la información que se guarda en una base de datos la cual es muy importante para la empresa o institución en la que se encuentra implementada y un acceso no deseado a la base de datos por agentes externos o internos podrían causar grandes daños a la compañía o inclusive la decaída en la economía de un país entero. Por ejemplo instituciones como bancos, hospitales, compañías de seguros entre otros contienen datos que son muy valiosos, los cuales deben ser resguardados de la mejor manera. Sin embargo las siguientes recomendaciones son solamente algunas de las muchas que se deben considerar para mantener la seguridad en una base datos y este es un manual para el diseño y implementación de una base de datos. A continuación se enumeran las recomendaciones junto con una breve explicación:

1. El administrador de la base datos es el único que debe tener acceso a la base de datos.
2. El administrador de la base de datos es quién se encarga de ingresar los SQL, se recomienda utilizar una base datos a escala de la base de datos principal para verificar la eficiencia de las sentencias SQL en esta base antes de ingresarlos en la base de datos principal.
3. Solo los administradores de la base de datos deben tener acceso a la documentación de la misma.
4. Cuando se realiza algún cambio en la base de datos a nivel físico, lógico o en el modelo de datos dicho cambio debe ser actualizado en la documentación de la misma, esto con el fin de que la documentación siempre este actualizada.
5. Finalmente como última recomendación es importante fomentar el uso de sentencias SQL optimizadas a continuación se explica un poco de que es optimización y un pequeño ejemplo de optimización.

Optimización

La optimización es el mejoramiento de consultas en SQL. Esto es una práctica muy utilizada en bases de datos con el fin de obtener datos mediante consultas SQL en el menor tiempo posible. Es muy importante tomar en cuenta lo anterior porque hoy en día la velocidad en el acceso a los datos puede marcar diferencia en una compañia, por lo cual es muy importante tomar en cuenta está práctica ya que por ejemplo en bases de datos de grandes empresas un acceso más rápido a la información puede hacer esta diferencia, esto porque en bases de datos que almacenan millones de datos, si las consultas SQL no están optimizadas pueden consumir demasiado tiempo, con esto se refiere a horas para obtener algún dato. Por eso importante tomar en cuenta la recomendación de fomentar el uso de sentencias optimizadas. A continuación se explica brevemente un pequeño ejemplo de optimización en donde es importante destacar que se hace el uso de algebra relacional.

A continuación un esquema de lo que sería la base datos y adjunto la consulta que se desea realizar que posteriormente será optimizada. En el siguiente modelo cada relación (r), tiene esquema (k,a,b) siendo k llave primaria o foránea.



Consulta

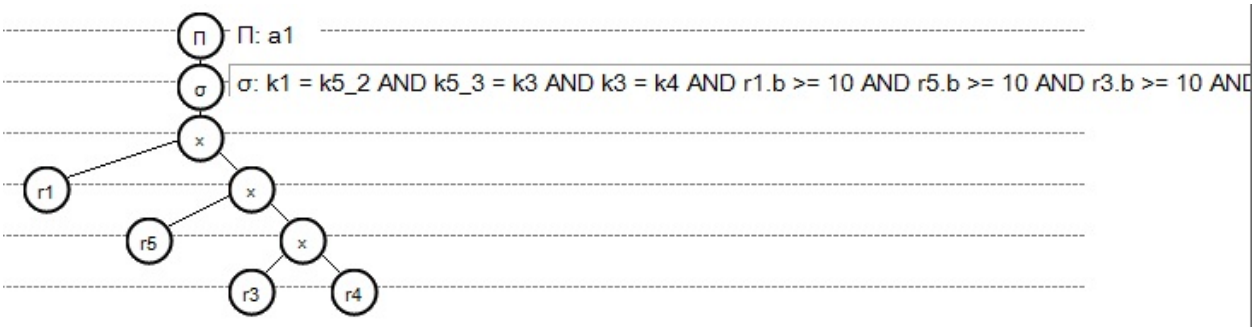
Calcular de atributos (a) de r1 relacionados con atributos (a) de r4. Considerando que los valores b > 10 en cada relación se consideran datos erróneos y no pueden ser tomados en cuenta. El valor 4 en r1 indica la cantidad de registros que contiene.

1. El primer paso es realizar la sentencia SQL sin optimizar.

```
select a1
from r1,r5,r3,r4
where k1 = k5_2
and k5_3 = k3
and k3 = k4
and r1.b >= 10
and r5.b >= 10
and r3.b >= 10
and r4.b >= 10;
```

A partir de esta sentencia SQL se hará la optimización.

2. Ahora se procede a realizar el árbol de la sentencia no óptima



3. A partir del árbol anterior se realiza el álgebra relacional de la sentencia SQL no optimizada.

```
Π{a1}(σ{k1=k5_2  k5_3=k3  k3=k4  r1.b10  r5.b10  r3.b10  r4.b10 }(r1 × r5 × r3  × r4))
```

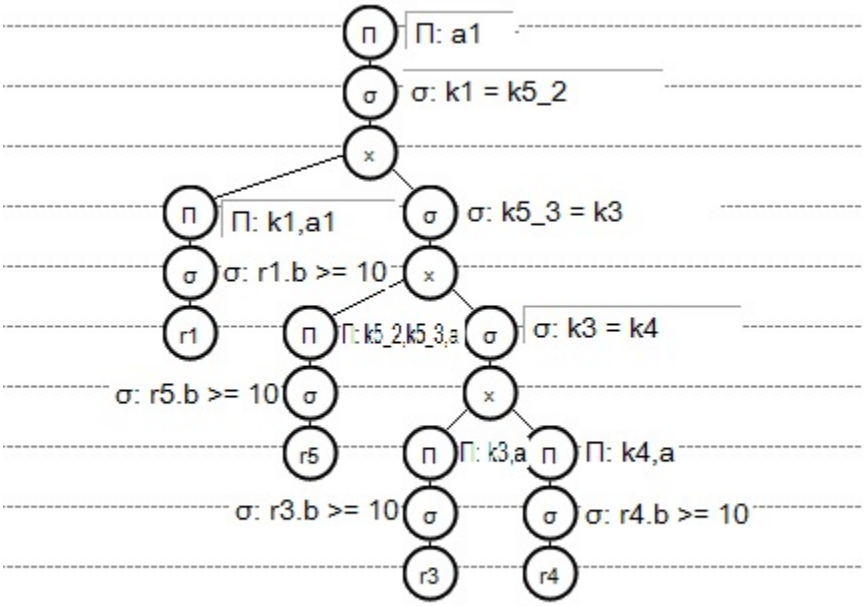
4. Ahora se procede a realizar un árbol mejorado partiendo también del árbol del punto 2 esto analizando los costos de datos que consumía el árbol del paso 2.

5. A partir del árbol que se realizó en el paso 5 se hace el álgebra relacional.

```
R1 ←σ{k3=k4 }(Π{k3,a }(σ{r3.b10}(r3)) ×  Π{k4,a}(σ{r4.b10}(r4)))
R2 ←σ{k5_3=k3}(Π{k5_2,k5_3,a}(σ{r5.b10} (r5)) × R1)
R3 ←Π{a1}(σ{k1=k5_2}(Π{k1,a1}(σ{r1.b10} (r1)) × R2))
```

6. Finalmente está sería la sentencia SQL optimizada

```
SELECT a1 FROM (SELECT k1,a1 FROM r1 WHERE r1.b >= 10)alias1 ,
 (SELECT * FROM (SELECT k5_2,k5_3,a FROM r5 WHERE r5.b >= 10)alias2 ,
    (SELECT * FROM (SELECT k3,a FROM r3 WHERE r3.b >= 10)alias3 ,
        (SELECT k4,a FROM r4 WHERE r4.b >= 10)alias4
```




```
          WHERE k3 = k4
        )alias5
        WHERE k5_3 = k3
)alias6
WHERE k1 = k5_2;
```

Nota: Para la optimización se utilizó el software RAT (Relational Algebra Translator), el cual es muy recomendado para generación de la sentececia en álgebra relacional a partir de una consulta. El software se puede obtener en el siguiente enlace http://www.slinfo.una.ac.cr/rat/rat.html