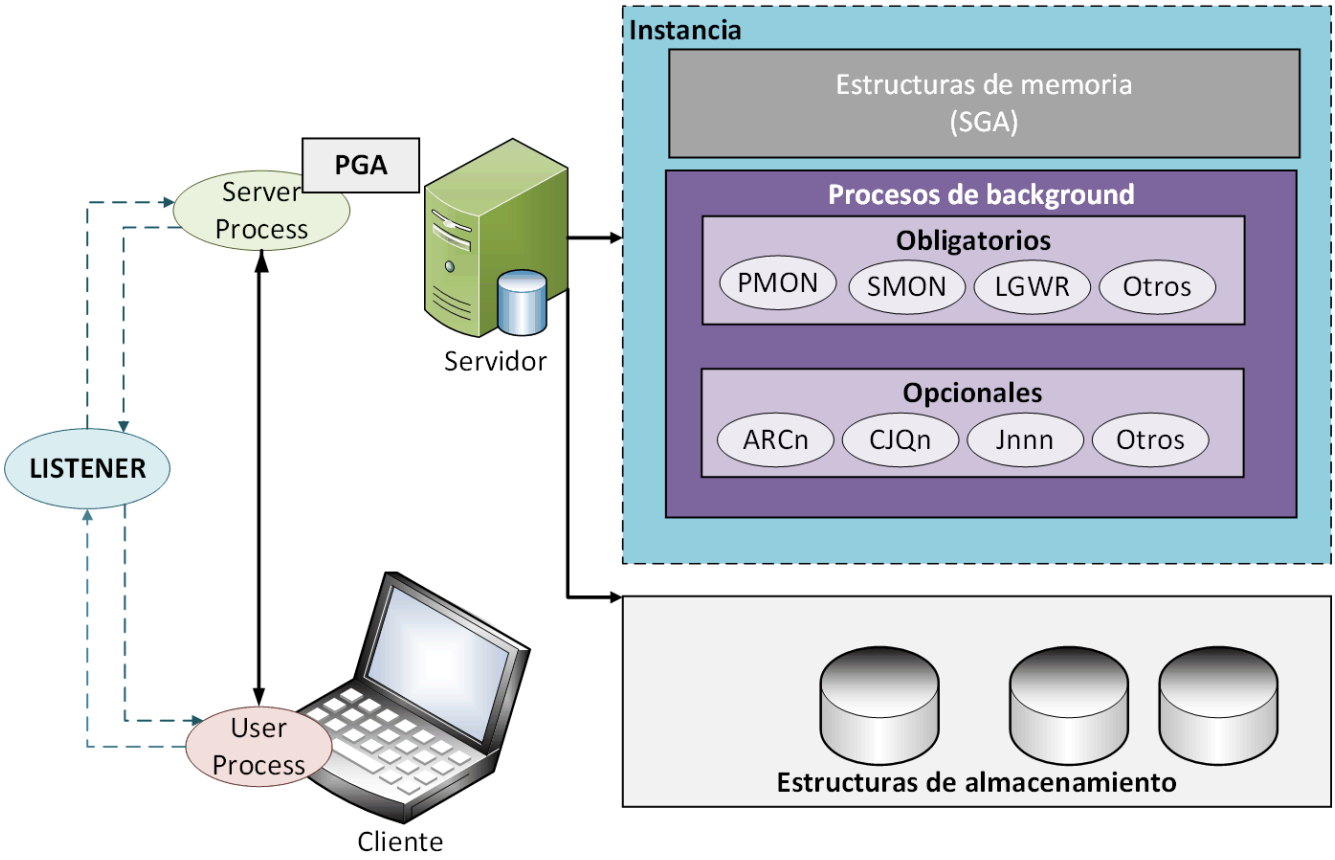
TEMA 5
Administración de procesos

5.1. ARQUITECTURA DE PROCESOS.



Los procesos en una base de datos Oracle pueden clasificarse dentro de las siguientes categorías:

- Procesos de usuario (*User processes*).
  - Es una aplicación o herramienta que se conecta con la base de datos de Oracle.
- Procesos de base de datos.
  - En esta categoría se encuentran:
    - *Server Processes*
      - *Dedicated Server Process (default)*
      - *Shared server process*
    - Procesos de background
- Demonios (Daemons)/ *Application Processes*.
  - Estos procesos no son específicos de una única base de datos, pueden ser compartidos por varias bases de datos en un mismo servidor. Entre ellos se encuentran:
    - Listeners
    - Grid

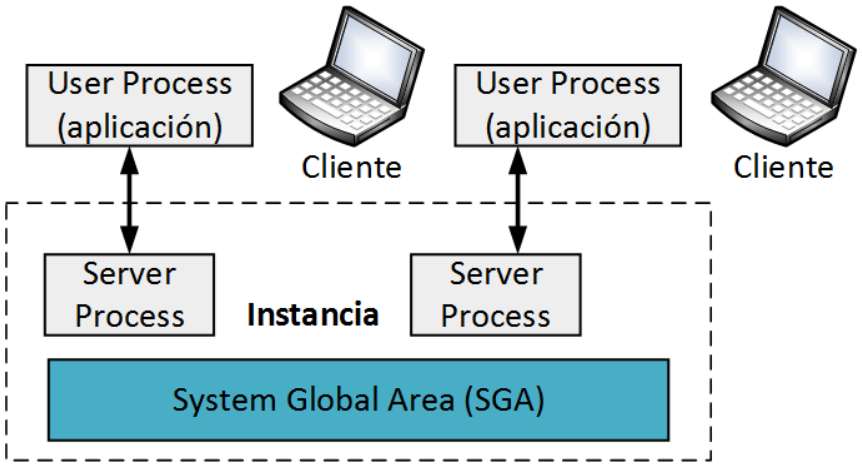
5.2. PROCESOS COMPARTIDOS Y DEDICADOS.

Una base de datos Oracle hace uso de diversos procesos que permite a múltiples usuarios conectarse a una instancia de forma simultánea.

- *Server process*: Creado por la instancia para manejar peticiones de usuarios (*user process*) conectados a la instancia.
- Existen 2 tipos de Server processes:
  - Dedicated server process (opción por default).
  - Shared server process (se debe configurar).

5.2.1. Dedicated Server process

- Se crea un nuevo server process por cada usuario conectado a la base de datos.
- Si 10 usuarios se conectan a la instancia, se crearán 10 server processes. Cada cliente se comunica con su server process.
- El server process estará exclusivamente dedicado y reservado para su usuario mientras dure la sesión.
- La información específica de cada server process se guarda en la UGA (área dentro de la PGA) del usuario.



5.2.1.1. ¿Cómo se crea un server process?

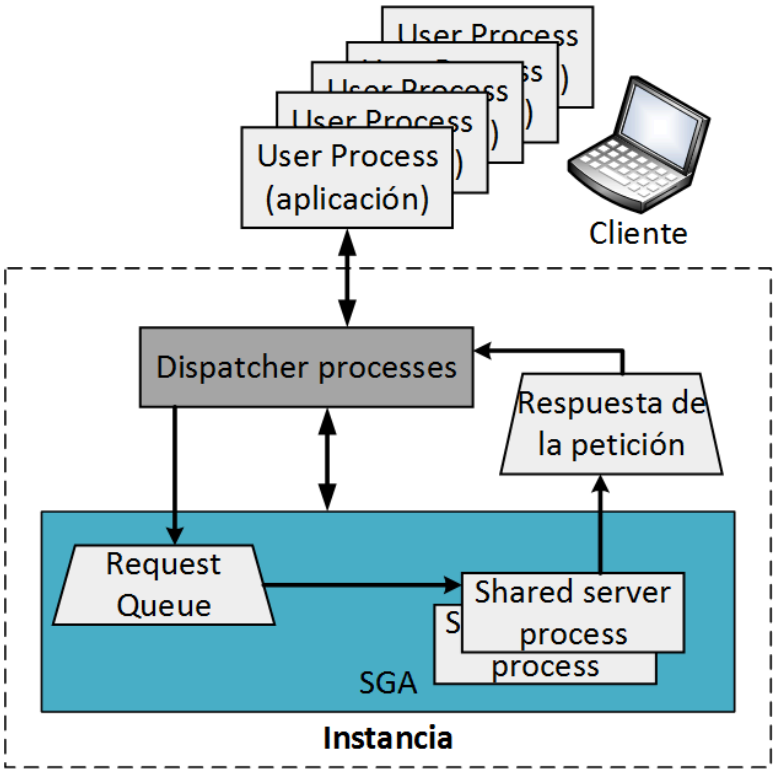
- Los server processes se crean de formas diferentes dependiendo el método de conexión:
  - Conexión desde una aplicación que crea el server process : SQL Plus (local al server), etc.
  - Net Listener. El cliente se conecta desde un listener.
  - Dedicated Broker: Componente que atiende a listener para crear server processes.

- Cuando un usuario se conecta a través de un listener hacia la instancia, ocurre lo siguiente:

1. El cliente solicita la conexión a través del uso de un listener o de un dedicated broker.
2. El listener inicia la creación de un nuevo server process
3. El sistema operativo crea un nuevo proceso.
4. La instancia inicializa varios componentes y notificaciones.
5. La instancia comienza con la administración y atención de la nueva sesión.

5.2.2. Shared Server process

- Ofrece beneficios con respecto a un *dedicated server process*.
- En esta arquitectura se cuenta con un *dispatcher* que se encarga de administrar las peticiones recibidas (user processes).
- La eficiencia radica en que el número de procesos requeridos para administrar a un conjunto de *user processes* es baja comparada con un *dedicated process.*
- En esta arquitectura, el cliente se conecta con un *dispatcher.*
- El dispatcher soporta múltiples conexiones simultáneas.
- A cada petición se le asigna parte de la memoria compartida llamada ***virtual circuit*** empleada por el dispatcher para atender a cada petición en la que se almacena información de las peticiones y respuestas del cliente
- El dispatcher envía a la petición a la llamada *cola compartida de peticiones* ubicada en la SGA (shared pool).
- En esta arquitectura, se crea un pequeño conjunto de procesos llamados *shared server process.*
- Se selecciona a un *shared server process* que se encuentre disponible para atender a la siguiente petición encontrada en la *cola de peticiones*.
- Cuando un *shared server process* está inactivo o no está siendo utilizado por alguna sesión, este puede reutilizarse para atender otras peticiones.



Esta configuración permite ahorrar recursos al evitar la creación de procesos dedicados por cada petición. Sin embargo, requiere que los clientes se conecten a la instancia a través de un servicio de red (Net Service, por ejemplo, a través del listener) esto sin importar que la petición sea local.

5.2.2.1. ¿Cuándo emplear shared servers?

- Esta funcionalidad puede incrementar escalabilidad, pero pudiera reducir performance.

- El tiempo de ejecución de las sentencias puede ser mayor.
- La actividad del CPU puede aumentar por la actividad de las colas.
- El beneficio es la escalabilidad mucho mayor número de peticiones por minuto, aunque estas sean lentas.
- Posible umbral para decidir usar esta técnica, más de 100-500 conexiones concurrentes.
- Se recomienda hacer uso de esta arquitectura cuando existe una cantidad de conexiones concurrentes mayor a la que el sistema operativo puede soportar.
- Para procesos en batch, *dedicated servers* es mejor (OLAP), puede existir contención en los dispatchers al transmitir grandes cantidades de resultados al *user process*.
- Para transacciones cortas donde interviene el usuario, y los tiempos de respuesta son con respecto a él, puede ser adecuado shared server (OLTP).
- Tareas de administración son más adecuadas con dedicated server (creación de índices, backups, recuperación).
- `shutdown` y `startup` no pueden lanzarse con un shared server por requerir de la SGA, se requiere dedicated server.
- Debido a que las acciones que realice un usuario dentro de su sesión pueden ser atendidas por diferentes shared processes, toda la información y datos de la sesión que comúnmente se almacenan en su correspondiente PGA, ahora deben almacenarse en un área de memoria compartida. En este caso estos datos se almacenan en una porción del *shared pool*, o en su defecto en el *large pool.*
- En algunas situaciones un *dedicated process* es la mejor opción:
    - Al ejecutar operaciones que implican procesamiento de datos en batch
    - Al emplear herramientas para realizar backups o recuperación de la base de datos (RMAN).
    - En general, sesiones en las que la gran parte del tiempo están activas, realizando múltiples operaciones dentro de la base de datos.

- Shared y dedicated servers pueden convivir en la misma instancia. Es decir, algunos usuarios pueden conectarse a través de un share process y otros a través de un dedicated server.

5.2.2.2. Configuración del modo compartido.

El modo compartido se habilita al configurar el parámetro `shared_servers` con un valor > 0, o con el parámetro `dispatchers` con un valor >0.

En resumen, los siguientes parámetros se relacionan con la configuración del modo compartido:

- `shared_servers`: Numero de shared servers a lanzar cuando arranca la instancia. Debe ser configurado con respecto al número máximo de peticiones concurrentes que se esperan.
    - Si se requieren más, se crean más *shared servers*, hasta llegar al valor del parámetro `max_shared` servers.
- `dispatchers`: Configura los dispatchers a lanzar cuando se inicia la instancia. Este parámetro es requerido.
- `max_dispatchers`: indica el valor máximo de dispatchers que se pueden lanzar. Si se requieren más, estos no serán creados a demanda. Se deberá ajustar el parámetro `dispatchers`.
- `circuits`: Número total de *virtual circuits* disponibles empleados para atender nuevas sesiones. Su valor se emplea para limitar la memoria compartida que será utilizada para atender peticiones.

Configuración por default:

`shared servers` =1 cuando el parámetro `dispatcher` es establecido.
`max_shared_servers` = 1/8 del valor del parámetro `processes`

Configuración típica:

- Un *shared process* por cada 10 conexiones.

Ejemplos:

```
--configura un solo dispatcher para protocolo TCP
alter system set dispatchers='(PROTOCOL=tcp)'

--configura 2 dispatchers para protocolo TCP
alter system set dispatchers='(dispatchers=2)(protocol=tcp)';

--configura 20 shared servers
alter system set shared_servers=20;
```

Importante hacer afinamiento de estos parámetros. Siempre deben existir shared servers disponibles y suficientes dispatchers para mejorar los tiempos de respuesta, pero cuidando en todo momento que el número de procesos sea el adecuado para evitar uso excesivo de recursos.

5.2.2.3. Configuración del listener y tnsnames.ora en modo compartido.

Tanto en listener como el archivo `tnsnames.ora` en el que se configuran los nombres de servicio requieren de especial atención al configurar el modo compartido.

- Una vez realizada la configuración se requiere registrar nuevamente los servicios con el listener para que pueda recibir peticiones y delegarlas a un dispatcher:

```
alter system register;
```

- Posteriormente, la siguiente instrucción muestra la lista de los servicios registrados en el listener, en la que se puede apreciar la configuración del modo compartido:

```
lsnrctl services

Service "cursobd.fi.unam" has 1 instance(s).
Instance "cursobd", status READY, has 3 handler(s) for this service...
Handler(s):
  "DEDICATED" established:1 refused:0 state:ready
  LOCAL SERVER
  "D001" established:0 refused:0 current:0 max:1022 state:ready
  DISPATCHER <machine: pc-dell-linux, pid: 12221>
  (ADDRESS=(PROTOCOL=tcp)(HOST=pc-dell-linux)(PORT=12669))
  "D000" established:0 refused:0 current:0 max:1022 state:ready
  DISPATCHER <machine: pc-dell-linux, pid: 4723>
  (ADDRESS=(PROTOCOL=tcp)(HOST=pc-dell-linux)(PORT=37394))
  Service "cursobdXDB.fi.unam" has 1 instance(s).
Instance "cursobd", status READY, has 0 handler(s) for this service...
```

- Observar que en la salida se muestran los procesos D000 y D001 que corresponden a cada uno de los dispatchers configurados. Notar que en este momento aún no se han establecido conexiones.
- Para comprobar esta configuración en el diccionario de datos:

```
select program,pid,pname
from v$process
where pname like'S0%'
or pname like 'D0%'
order by program;
```

PROGRAM	PID	PNAME
oracle@jrc-ora-pc.fi.unam (D000)	77	D000
oracle@jrc-ora-pc.fi.unam (D001)	79	D001
oracle@jrc-ora-pc.fi.unam (S000)	80	S000
oracle@jrc-ora-pc.fi.unam (S001)	81	S001
oracle@jrc-ora-pc.fi.unam (S002)	82	S002
oracle@jrc-ora-pc.fi.unam (S003)	83	S003

- Notar que los procesos asociados con los dispatchers (en este caso 2) inician con "`Dxxx`" y los shared servers (en este caso 4) con "`Sxxx`".

- Para el caso del archivo `tnsnames.ora`, existe un parámetro `(SERVER = DEDICATED)` y `(SERVER=SHARED)` que permiten indicar la forma en la que se establecerá la conexión para un usuario: modo dedicado o modo compartido. Esto permite que un mismo usuario pueda seleccionar el modo de conexión.

Ejemplo:

```
JRCBDA2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = jrc-ora-pc.fi.unam)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = jrcbd2.fi.unam)
    )
  )
```

De esta forma el usuario podría conectarse a través de sqlplus empleando la siguiente instrucción para hacer uso del modo compartido:

```
sqlplus usuario@jrcbda2
```

- Notar que en este ejemplo no se emplean los parámetros antes mencionados. Por default, si el modo compartido está habilitado, todas las conexiones vía listener harán uso del ***modo compartido***. Es posible agregar el parámetro `(SERVER=SHARED)` de forma opcional.
- Para permitir a un usuario conectarse en modo dedicado, podría agregarse una nueva entrada al listener con la siguiente configuración:

```
JRCBDA2_DE =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = jrc-ora-pc.fi.unam)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = jrcbd2.fi.unam)
      (SERVER=DEDICATED)
    )
  )
```

Y por lo tanto, la instrucción para acceder en modo dedicado:

```
sqlplus usuario@jrcbda2_de
```

5.2.3. Vistas del diccionario de datos asociadas con el modo shared server.

5.2.3.1. v$dispatcher

Muestra la información de los dispatchers configurados.

	NAME	NETWORK	PADDR	STATUS	ACCEPT	MESSAGES	BYTES
1	D000	(ADDRESS=(PROTOCOL=tcp)(HOST=jrc-ora-pc.fi.unam)(PORT=20540))	0000000072FAD148	WAIT	YES	70	11225
2	D001	(ADDRESS=(PROTOCOL=tcp)(HOST=jrc-ora-pc.fi.unam)(PORT=30861))	0000000072FAF438	WAIT	YES	78	11999

5.2.3.2. v$dispatcher_config

Muestra la configuración de los dispatchers.

CONF_INDX	NETWORK	DISPATCHERS	CONNECTIONS	SESSIONS	MULTIPLEX	LISTENER	SERVICE
0	(ADDRESS=(PARTIAL=YES)(PROTOCOL=tcp))	2	1022	1022	OFF	(ADDRESS=(PROTOCOL=TCP)(HOST=jrc-ora-pc.fi.unam)(PORT=1521))	jrcbd2.fi.unam

5.2.3.3. $queue

Muestra información de la cola de peticiones compartida.

PADDR	TYPE	QUEUED	WAIT	TOTALQ	CON_ID
00	COMMON	0	0	36	0
00	COMMON	0	0	38	0
0000000072FAD148	DISPATCHER	0	0	35	0
0000000072FAF438	DISPATCHER	0	0	39	0

5.2.3.4. v$shared_server

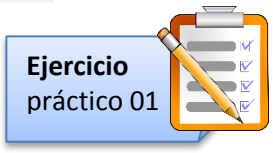
Muestra información de los shared servers.

NAME	PADDR	STATUS	MESSAGES	BYTES
S000	0000000072FB05B0	WAIT(COMMON)	0	0
S001	0000000072FB1728	WAIT(COMMON)	0	0
S002	0000000072FB28A0	WAIT(COMMON)	0	0
S003	0000000072FB3A18	WAIT(COMMON)	148	49678

5.2.3.5. v$circuit

Contiene información acerca de los *virtual circuits,* los cuales representan conexiones de los usuarios hacia la base de datos que hacen uso de dispatcher. Típicamente un registro por cada usuario.

CIRCUIT	DISPATCHER	SERVER	WAITER	SADDR	STATUS	QUEUE
0000000069F14310	0000000072FAD148	00	00	000000007245F6D8	NORMAL	NONE
0000000069407220	0000000072FAF438	00	00	0000000072455C78	NORMAL	NONE



Revisar el documento del ejercicio práctico 01. Configuración de la base de datos en modo compartido.

5.3. PROCESOS DE BACKGRUOUND.

- Los procesos de background inician junto con la instancia de base de datos.
- Interactúan entre ellos y el sistema operativo para manejar las estructuras de memoria, tareas de mantenimiento, escribir información en disco, entre otras funciones.
- Estos procesos ayudan a maximizar el rendimiento y dar cabida a muchos usuarios.
- Dentro de estos procesos se encuentran los que son de carácter obligatorio y los opcionales, estos últimos pueden o no ser utilizados; dependerá de las características de la base de datos que se encuentren en uso, por ejemplo, RAC.

Los procesos obligatorios son:

- *Process Monitor* (PMON)
- *Process Manager* (PMAN)
- *System Monitor* (SMON)
- *Listener Registration* (LREG)
- *Database Writer* (DBW*n*)
- *Log Writer* (LGWR)
- *Checkpoint* (CKPT)
- *Manageability Monitor* (MMON)
- *Manageability Monitor Lite* (MMNL)
- *Recoverer* (RECO)

Dentro de los procesos de background opcionales más comunes se encuentran:

- Archiver Processes (ARCn)
- Job Queue Processes (CJQ0 y Jnnn)
- Flashback Data Archive (FBDA)
- Space Management Coordinator (SMCO)
- dispatcher process (Dnnn)

5.3.1. PMON: Process Monitor

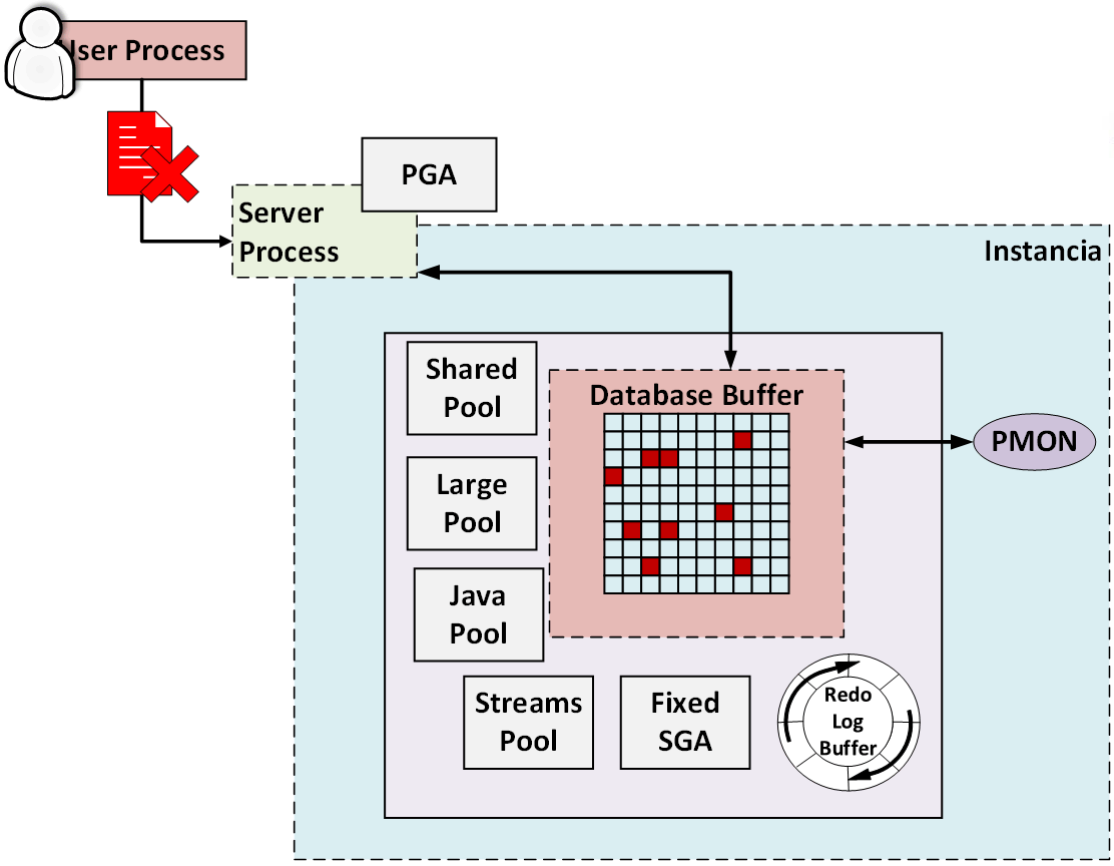
A partir la versión 12c de la base de datos, PMON se apoya en dos procesos adicionales. Este grupo, llamado *PMON group*, se conforma por:

- Process Monitor (PMON)
- Cleanup Main Process (CLMN)
- Cleanup Helper Processes (CL*nn*)

En términos generales, este grupo se encarga de la recuperación cuando falla un *user process*.

- Limpia el *database buffer cache*.
- Libera recursos que fueron utilizados por el proceso de usuario fallido o terminado. De esta forma garantiza que éstos puedan ser reusados por otros procesos. Por ejemplo:
  - Libera bloqueos que no son necesarios.
  - Elimina el ID del proceso de la lista de procesos activos.
- El grupo PMON despierta de forma periódica para verificar si es necesario. También puede ser llamado por otros procesos que lo requieran.
- Revisa el estatus otros procesos de background verificando que ninguno haya sido terminado de forma anormal. Si un *dispatcher* y *server process* ha detenido su ejecución, PMON los reinicia.

A nivel general PMON realiza las tareas de detección terminaciones anormales de procesos y delega las tareas de limpieza a CLMN el cual a su vez se apoya en CLnn



5.3.1.1. CLMN: Cleanup Main Process

Cuando PMON detecta que es necesario un trabajo de limpieza de un *user process*, lo delega a este proceso. De esta manera, CLMN se convierte en un *supervisor* responsable de esta tarea. Realiza de forma periódica, la limpieza de

- Procesos terminados.
- Sesiones terminadas.

- Transacciones.
- Conexiones de red.
- Sesiones inactivas.
- Transacciones desconectadas.
- Conexiones de red desconectadas que han excedido su tiempo inactivo.

5.3.1.2. CLnn: Cleanup Helper Processes

- CLMN delega el trabajo de limpieza a este conjunto de procesos auxiliares.
- Una actividad de limpieza puede provocar que el proceso sea bloqueado temporalmente.
- Para evitar serializar estas tareas, se crean varios procesos ***helpers*** que permiten paralelizar la limpieza, en especial cuando se requiera realizarla en varias partes de la BD.

- La terminación *–nn* se sustituye por un número de dos dígitos: CL*01*, CL*02*, etc.
- El número de estos procesos auxiliares es proporcional a la cantidad de limpieza que debe realizarse.
- Para monitorear el estado actual de los procesos de limpieza existen 2 principales vistas:

Vista	Descripción
`v$cleanup_process`	Proporciona información de los procesos PMON. `select name,state,dead_in_cleanup,cleanup_time, time_since_last_cleanup,num_cleaned from v$cleanup_process;` NAME | STATE | DEAD_IN_CLEANUP | CLEANUP_TIME | TIME_SINCE_LAST_CLEANUP | NUM_CLEANED PMON | IDLE | 00 | 0 | 6575 | 0 CLMN | IDLE | 00 | 0 | 6575 | 0 En este ejemplo se muestran la existencia de los procesos PMON Y CLMN. Ambos tienen status IDLE (sin actividad). En este ejemplo no existen procesos CL-nn creados.
`v$dead_cleanup`	Muestra los procesos muertos y sesiones terminadas (killed) desde que se inició una instancia, así como su status de limpieza: • CLEANUP_PENDING • IN PROGRESS • RESOURCES FREED • PARTIAL CLEANUP Revisar el siguiente ejemplo para mostrar datos en esta vista.

Ejercicio en clase 1

1. Entrar a sesión con un usuario cualquiera.

```
sqlplus jorge/jorge
```

2. Insertar un dato en alguna tabla sin hacer `commit`.

```
insert into test values(5);
```

3. En otra terminal entrar a sesión con usuario SYS y mostrar los datos de la sesión del usuario anterior.

```
sqlplus sys as sysdba

select sid,serial#,inst_id,machine,status
from gv$session where username='JORGE';
```

SID	SERIAL#	INST_ID	MACHINE	STATUS
74	18605	1	pc-jrc-ora.fi.unam	INACTIVE

4. Ejecutar una sentencia SQL para terminar la sesión del usuario anterior:

```
alter system kill session '74,18605,@1';
```

- Observar el valor de la cadena formada por los valores `'<sid>,<serial#>,@<inst_id>'`

5. Realiza una consulta a la vista `v$dead_cleanup` para mostrar la sesión que se acaba de terminar a través de `alter system kill session`.

```
select type,state,dead_time,cleanup_attempts,cleanup_process,
  cleanup_time,num_blocked
from v$dead_cleanup;
```

TYPE	STATE	DEAD_TIME	CLEANUP_ATTEMPTS	CLEANUP_PROCESS	CLEANUP_TIME	NUM_BLOCKED
KILLED SESSION	RESOURCES FREED	250	0	(null)	0	0

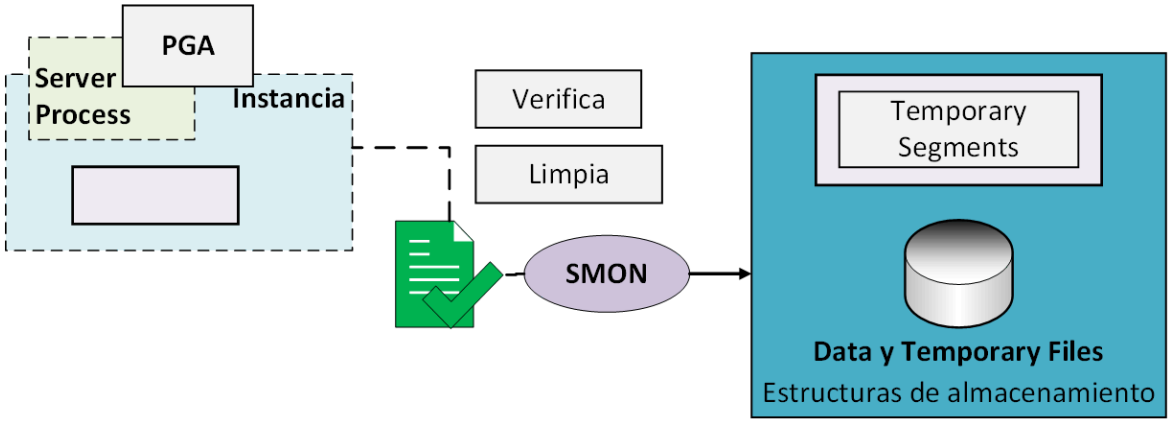
- Notar el registro de la consulta. Representa a la sesión que se acaba de terminar empleando el comando `alter system kill session`.
- `dead_time` representa el tiempo en segundos que han transcurrido desde que la sesión fue eliminada.
- `cleanup_attempts` representa el número de intentos que ha realizado PMON para realizar la limpieza.

5.3.2. PMAN: Process Manager

Supervisa, genera y detiene los siguientes tipos de procesos:

- *Dispatcher* y *shared server*.
- *Connection broker* y *server processes* agrupados en pools de conexiones residentes de las bases de datos.
- *Job queue processes*.
- Procesos de background reiniciables.

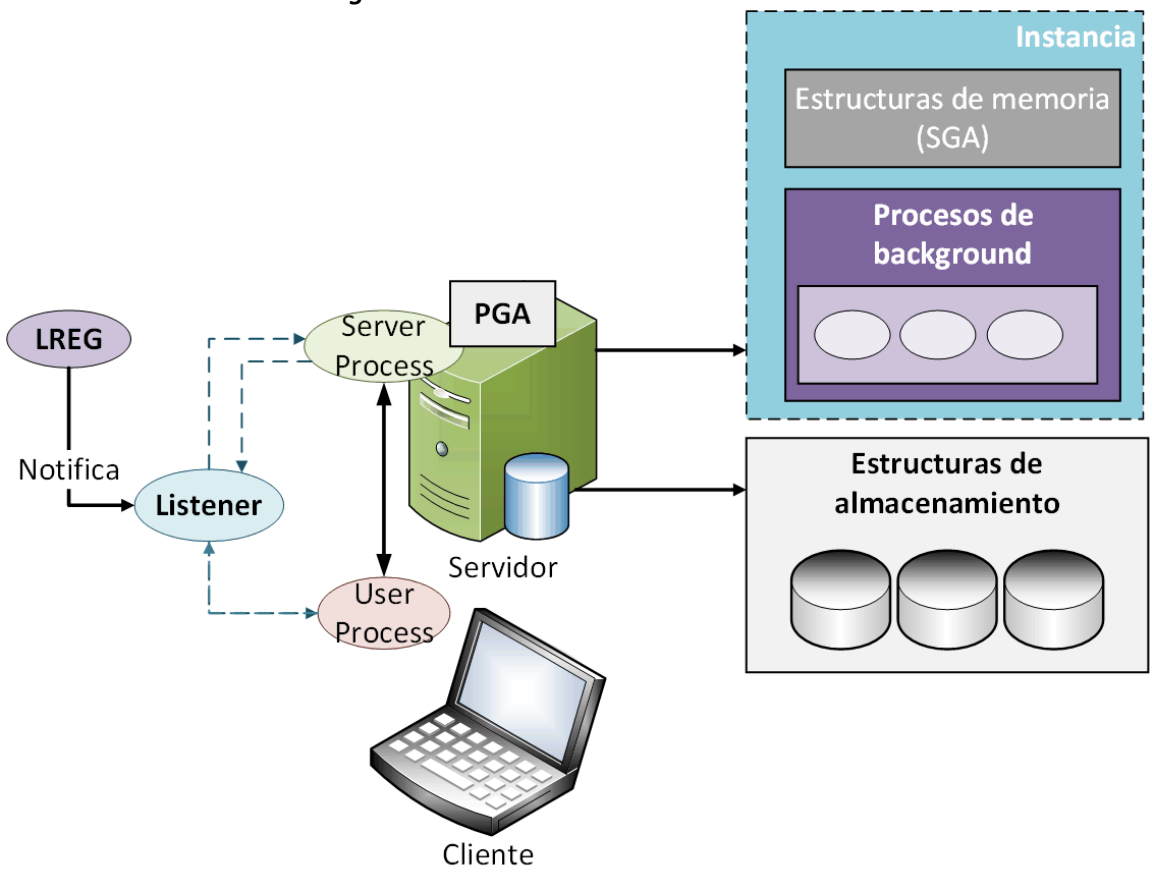
5.3.3. SMON: System Monitor



Este proceso es el encargado de realizar tareas de monitoreo del sistema. Entre sus principales labores se encuentran:

- Realizar la recuperación de la instancia durante el arranque. Esta tarea se realiza utilizando los archivos *redo log*.
- Recuperar los datos de transacciones incompletas (*uncommited*) que fueron omitidas durante la recuperación de la instancia, debido a errores de lectura de archivos de datos o errores de tablespaces fuera de línea.
    - SMON los recupera cuando estos dos elementos se encuentran en línea nuevamente.
- Limpiar segmentos temporales que no se utilizan.
- Juntar extensiones (*extents*) libres contiguas dentro del tablespace manejado por el diccionario.
- SMON despierta de forma periódica para verificar si es necesario. También puede ser invocado por otros procesos que lo requieran.

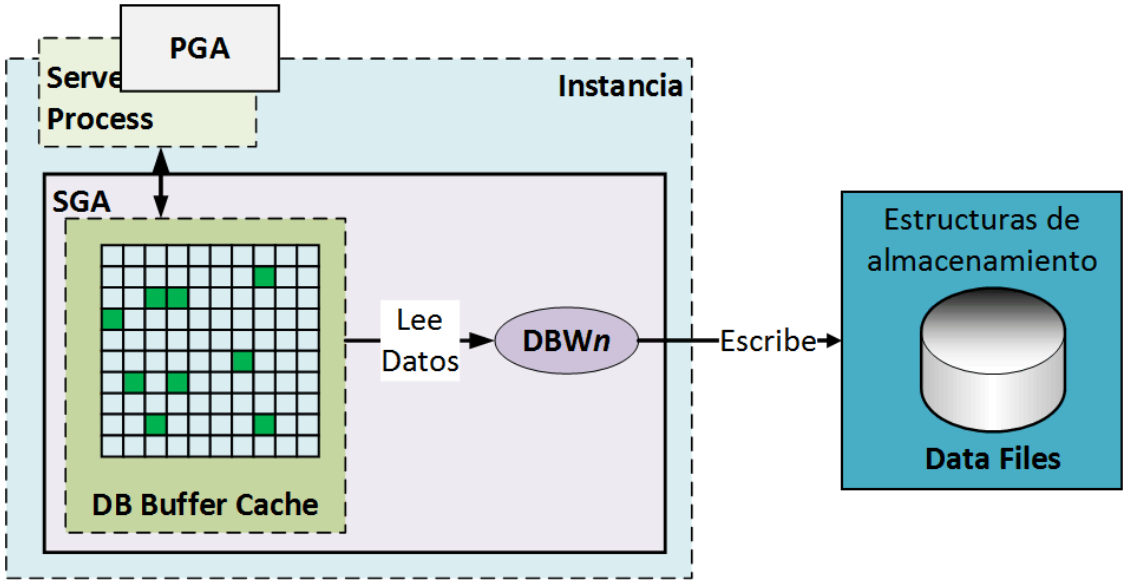
5.3.4. LREG: Listener Registration



LREG obtiene información sobre la instancia de la base de datos y de posibles *dispatcher processes* configurados y los ***registra*** o notifica al ***listener***.

- Le indica al listener si la instancia se encuentra arriba y lista para atender solicitudes de conexiones.
- Le proporciona al listener la siguiente información:
  - Los nombres de los servicios de la base de datos.
  - El nombre de la instancia asociada a los servicios y su carga máxima actual.
  - *Service handlers* (*dispatchers* y servidores dedicados) disponibles para la instancia. Incluye:
    - Tipo
    - Direcciones de protocolo
    - Carga actual y máxima.
- Al iniciar una instancia, este proceso de background verifica que el Listener esté ejecutándose.
  - Si este último se encuentra funcionado, LREG le pasa parámetros relevantes.
  - De lo contrario, LREG estará buscando contactarlo de forma periódica.
- En versiones anteriores a 12c, PMON efectuaba esta tarea.

5.3.5. DBWn: Database Writer



- DBW escribe el contenido de los buffers modificados (*dirty buffers*) de la *database buffer cache* en los data files, es decir, en disco.
- Escribir a los data files se considera como una operación ***costosa***. Por lo tanto DBW tratará de realizar esta operación con la menor frecuencia posible.
- Este proceso inicia junto con la instancia.
- DBW escribe lotes de bloques cuando es posible parar reducir los tiempos de escritura. El número de bloques escritos en un lote varía de acuerdo al sistema operativo.

La escritura se realiza bajo las siguientes 3 condiciones:

1) Cuando un server process no encuentra un buffer limpio que pueda utilizar tras un escaneo de un número determinado de buffers, o cuando el server process tarda demasiado en encontrar buffers limpios, le indica a DBW que escriba.

- Se escriben los buffers que no son utilizados con alta frecuencia.
- El DB buffer cache pudiera tener miles de buffers sucios, pero DBW solo escribirá unos cuantos cientos de buffers, solo aquellos que no han sido accedidos frecuentemente.

2) Existen demasiados buffers sucios. Existe un umbral interno para determinar esta condición.
3) Ocurre un ***checkpoint*** (se explica este concepto más adelante).

- A pesar de que un único proceso de este tipo (DBW0) es adecuado para la mayoría de los casos, es posible configurar procesos de escritura adicionales.
- Esto mejora rendimiento del sistema si se modifica un gran volumen de datos. El límite de procesos en total, es de 100.
- Los primeros 36 procesos se identifican como: DBW0 – DBW9 y DBWa–DBWz. Mientras que los siguientes procesos se identificarían como BW36–BW99.
- El parámetro `db_writer_processes` indica el número de procesos adicionales que se iniciarían.
- Configurar más de un proceso en un sistema con un único procesador no es recomendable.

5.3.5.1. Checkpoints.

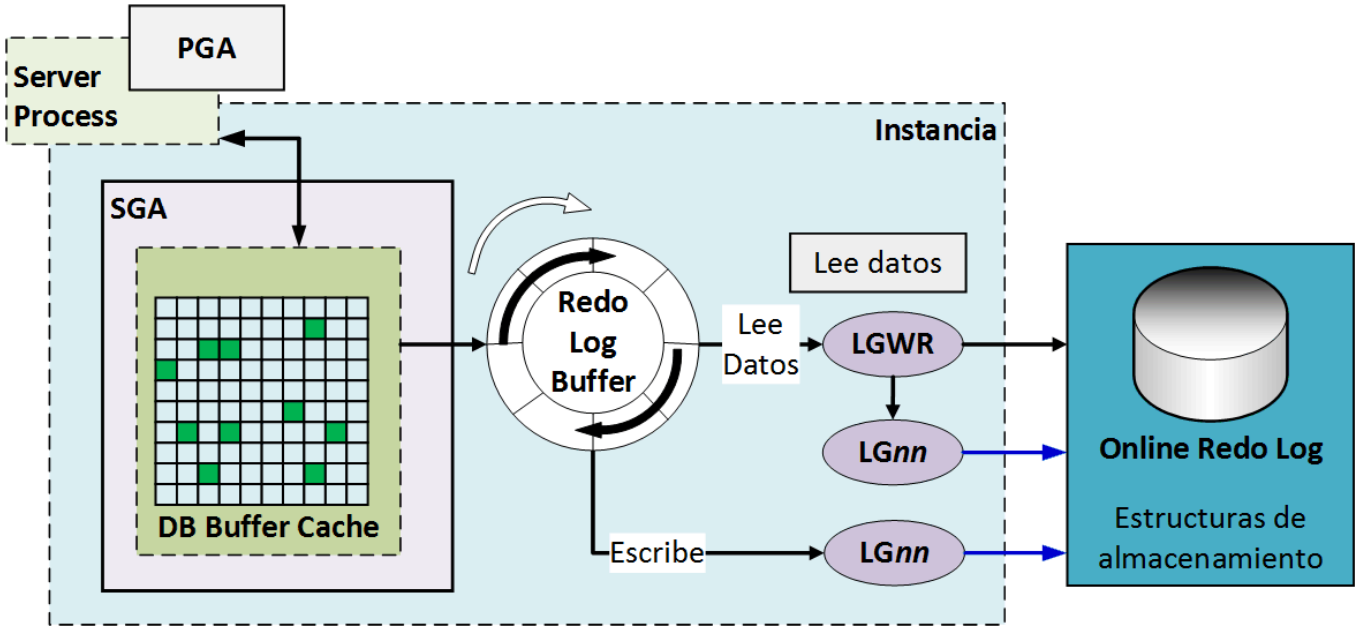
Este tema se revisará a detalle en el capítulo de Recovery. Por ahora, se muestran sus características principales:

- Un checkpoint es una señal que le indica a DBW escribir ***todos*** los buffer sucios que existan en el DB buffer cache.
- Lo anterior provoca una excesiva actividad a nivel disco y procesador lo que seguramente provocará un deterioro muy significante del desempeño en las sesiones conectadas.
- Existen ciertos eventos y operaciones que requiere una completa escritura de buffers sucios.
  - Checkpoint total: Escribe el 100% de todos los buffers sucios existentes
  - Checkpoint parcial: Escribe el 100% de todos los buffers sucios pero de un determinado data file
- De forma automática, un checkpoint ocurre únicamente al detener la instancia de forma ordenada. En este proceso, todos los archivos deben estar sincronizados antes de detener y cerrar la BD.
- De forma manual un checkpoint puede ocurrir en las siguientes condiciones:
  - Se ejecuta manualmente la instrucción `alter system checkpoint;`
  - Cuando se decide poner un tablespace o un data file en modo offline.
  - Cuando la BD se establece en modo de backup (se va a realizar un respaldo).
  - Cuando la BD se establece en modo read only.

<u>Ejercicio en clase 2</u>

A. Agregar un DBW adicional
B. Ejecutar la sentencia SQL correspondiente para comprobar que existe un nuevo DB writer, mostrar el id del proceso del sistema operativo, su nombre, y la ruta donde se encuentra su bitácora. Auxiliarse de la vista `v$process`

5.3.6. LGWR: Log Writer



Log Writer escribe una porción del contenido del Redo Log buffer en un *redo log file* en disco bajo las siguientes condiciones:

- Un usuario guarda los cambios de una transacción de forma permanente (`commit`).
- Cuando ocurre un *log switch*: el punto donde LGWR deja de escribir en el Redo Log activo y cambia al siguiente Redo Log disponible (cuando se encuentra lleno o se hace un cambio manual).
- Han transcurrido tres segundos desde la última vez que LGWR ha escrito.
- El Redo Log Buffer se encuentra un tercio lleno o contiene 1MB de datos.
- DBW*n* escribe buffers modificados en disco.
    - Es obligatorio que primero LGWR escriba antes que DBWn (*Write-ahead protocol*).
    - Si el DBW*n* detecta que no se han escrito los Redo Records, manda una señal a LGWR.

- LGWR inicia y coordina un conjunto de procesos auxiliares denominados Log Writer Slave (LG*nn*), que son utilizados para mejorar el rendimiento de los procesos de escritura en ambientes multiprocesamiento.
- Sólo puede existir un proceso LGWR en la base de datos.

5.3.6.1. LGWR y commits.

¿Por qué LGWR escribe al realizar `commit`?

- Recordando del tema anterior, cuando un usuario realiza operaciones DML, los cambios se almacenan en el *DB buffer cache*.
- El cambio debe respaldarse en el archivo *Online Redo Log* para situaciones en las que sea necesario reconstruir o rehacer (REDO) debido a la ocurrencia de una falla. A este respaldo se le conoce como ***Vector de cambios.***

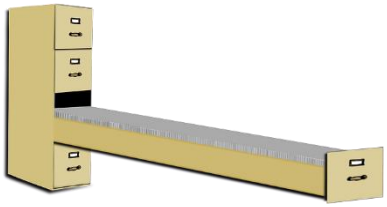
    - Por ejemplo, si se va la luz, el DB buffer cache se pierde.
    - Si los *vectores de cambios* están respaldados en el Online Redo Log, el *DB Buffer cache* se puede reconstruir.

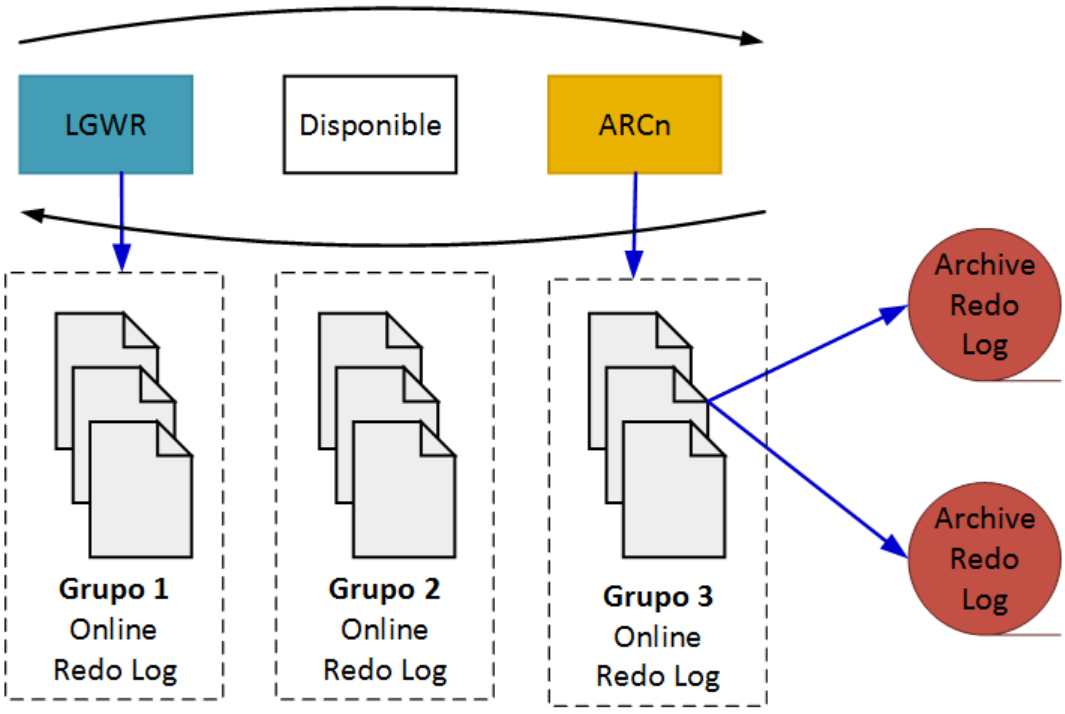
- Para mejorar desempeño, los vectores de cambios se escriben en el *Log Buffer* en lugar de ser escritos directamente al *Online Redo Log.*
- En una operación normal, pueden existir miles de vectores de cambios almacenados en el *Log Buffer.*
- LGWR es el encargado de sincronizar el contenido del *Log Buffer* con el *Online Redo Log* prácticamente en tiempo real.
- Notar que si ocurre una falla, el contenido del *Log buffer* al igual que el *DB buffer cache* podría perderse.
- Si el usuario no realiza `commit`, la BD no está obligada a recuperar los cambios en caso de una falla.
- Sin embargo, si el usuario realiza `commit`, la BD está obligada a recuperar sus cambios aunque exista una falla.
    - Para garantizar lo anterior, cuando un usuario ejecuta la instrucción `commit`, LGWR escribirá el contenido del Log buffer al Online Redo Log para garantizar que los vectores de cambios han sido escritos a disco y por lo tanto, será posible la recuperación en caso de falla.
    - La operación de `commit` no se considera exitosa hasta asegurarse que LGWR escriba los vectores de cambio al Online Redo Log. Es decir se realiza la escritura en tiempo real.

5.3.6.2. ARCn: Archiver processes, log switch

¿Qué es un log switch?

- En una BD productiva, contar con un solo Online Redo Log representa un riesgo latente. ¿Qué pasa si el archivo se daña?
- Para resolver esta situación existen 2 conceptos:
    - Grupos de Online Redo Logs
    - *Archive Redo Log.*

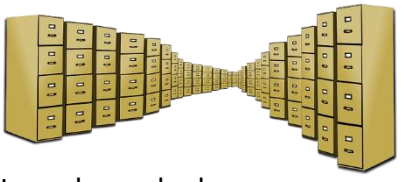




Grupos de Online Redo Logs.

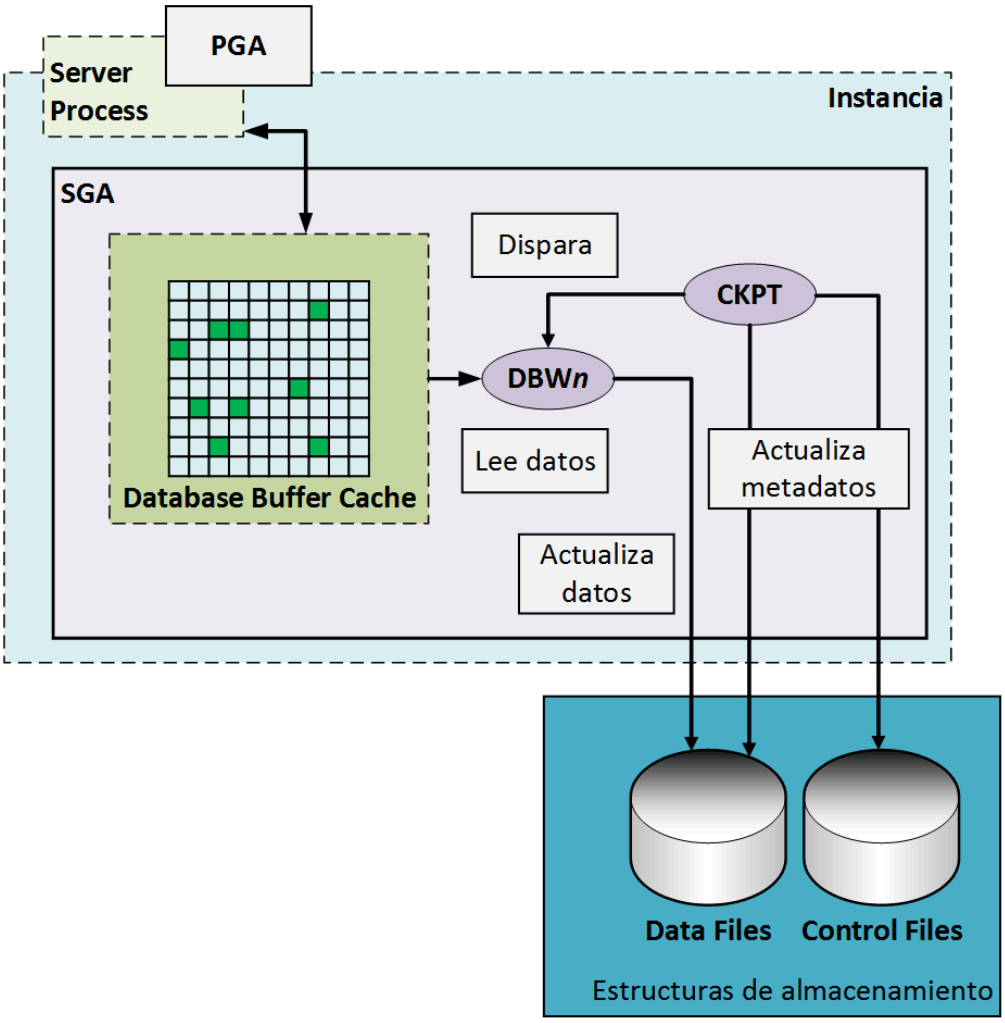
- Un grupo de archivos Online Redo Log contiene a un conjunto de Online Redo Logs.
- Todos los miembros del grupo tienen exactamente la misma información.
- Si uno de los miembros se pierde, el grupo puede seguir operando haciendo uso de los otros miembros.

Archivado de Online Redo Logs.



- La cantidad de vectores de cambios que se almacenan en los Online Redo Logs depende de su tamaño. Si la cantidad de operaciones DML es alta, esta capacidad de almacenamiento puede rebasarse en periodos de tiempo cortos.
- Si LGWR necesita almacenar más vectores de cambios, los Online Redo Logs comenzarán a sobrescribirse y por lo tanto los vectores anteriores ya no estarán disponibles.
- Esta condición evita que la BD pueda recuperarse con cambios realizados en periodos de tiempo más grandes.
- Para solucionar este inconveniente, antes de sobrescribir un grupo de Online Redo Logs, este deberá ser respaldado.
- El proceso de background ***ARCn Archive Process*** se encarga de copiar el contenido de un grupo Online Redo Log y respaldarlo en los llamados ***Archive Redo Logs.***
- Una vez que el respaldo ha concluido, el grupo de Online Redo Logs puede sobrescribirse sin peligro de perder datos.
- Notar en el diagrama que del lado izquierdo LGWR está escribiendo en un grupo 1 de Online Redo Logs (***active Redo Log***) y al mismo tiempo los procesos ARCn pudieran estar respaldando datos del grupo 3.
- Para poder contar con el paralelismo anterior se recomienda contar con 3 grupos de Archivos.
- Cuando el grupo 1 está por llenarse, LGRW realiza una última escritura y se produce un cambio de grupo de archivos llamado ***log switch.***
- Posterior a este evento LGWR comenzará a escribir en el grupo 2. Por otro lado, cuando ARCn termine de respaldar el grupo 3, este estará disponible para ser sobrescrito.
- La funcionalidad de Archivado de Online Redo Logs debe ser habilitada. Por default, esta funcionalidad no se habilita. Por esta razón los procesos ARCn son opcionales.
- Activar esta funcionalidad significa configurar a la base de datos en modo de archivado: ARCHIVEMODE

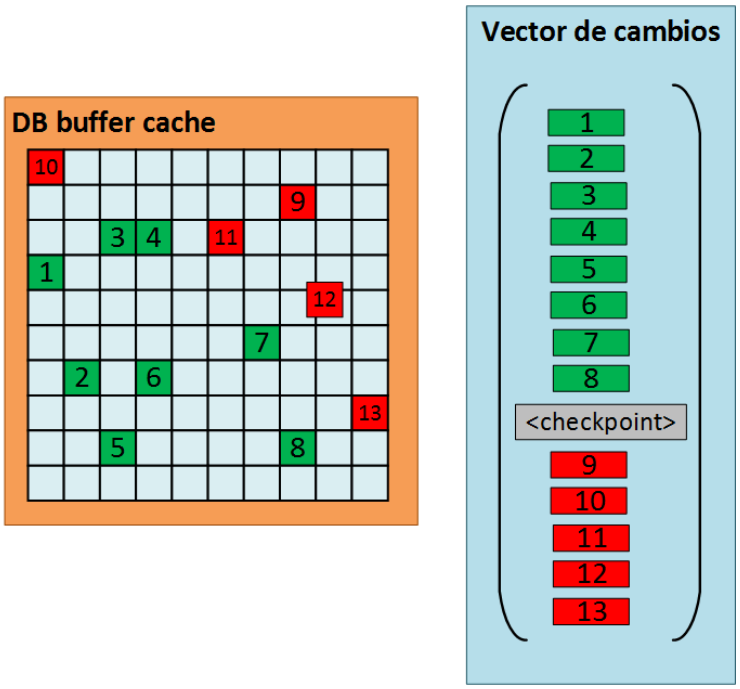
5.3.7. CKPT: Checkpoint.



- Como se mencionó anteriormente, una señal de checkpoint ordena a DBW escribir los buffers sucios a disco.
- En versiones anteriores de la base de datos, este proceso se disparaba en ciertos periodos de tiempo relativamente cortos y se realizaba la sincronización completa: ***full checkpoint.***

    - Con esta estrategia la cantidad de datos por sincronizar permanece relativamente baja en todo momento.
    - Sin embargo, la principal desventaja es que durante estas operaciones, el desempeño de la base de datos disminuye considerablemente ya que cada operación checkpoint implica sincronizar el 100% de los datos.

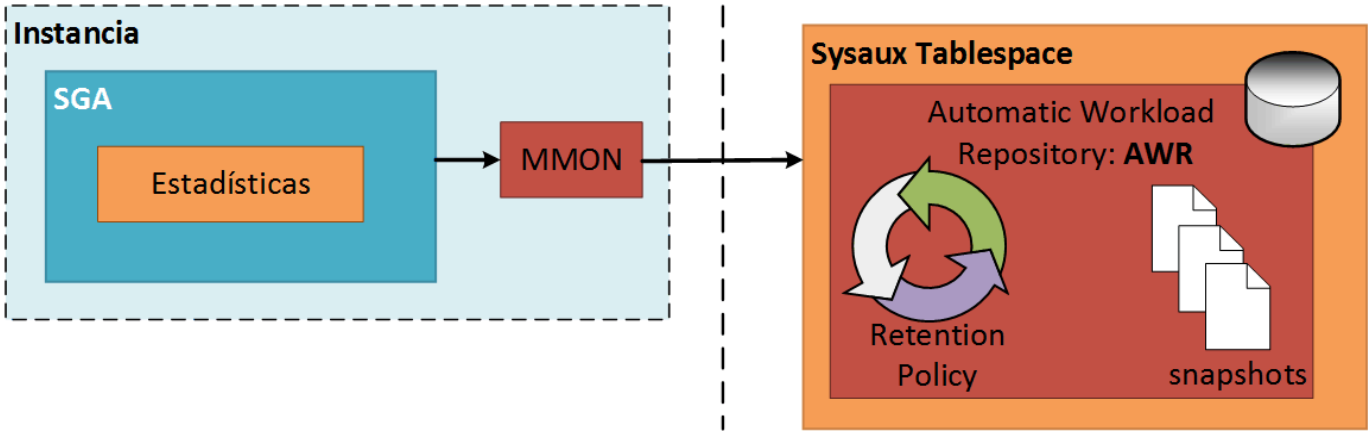
- Para resolver este problema de desempeño, en lugar de realizar *full checkpoints* se realizan ***incremental checkpoints***.
- En un *incremental checkpoint* no se realiza sincronización al 100%, solo una parte.
- Al seleccionar solo un subconjunto de buffers sucios para ser sincronizados provocará que existan más buffer sucios pendientes de sincronizar.



- El la figura anterior, los buffers 1 al 8 han sido seleccionados para ser sincronizados en disco.
- Esto implica que los buffers 9 al 13 están pendientes de ser sincronizados.
- CKPT instruye a DBW escribir los buffer 1 al 8.
- La siguiente vez que CKPT instruya a DBW escribir, deberá sincronizar los buffers 9 a 13. ¿Cómo saber que se debe sincronizar a partir del buffer 9?
- CKPT crea una especie de "marca" llamada *checkpoint position o RBA (Redo Byte Address).*
- Observar que esta marca se posiciona en el vector de cambios, es decir, en el Online Redo Log.
- Lo anterior implica que todos los cambios arriba de la marca ya han sido sincronizados por DBW, y los cambios debajo de la marca están pendientes por sincronizar.
- La información de esta marca se guarda en 2 lugares:
  - En el Control File
  - En los headers de los data files.
- La información que almacena contiene:
  - Posición del checkpoint.
  - SCN (System Change Number, se revisará este concepto más adelante).
  - Localización en los Online Rego Log para iniciar la recuperación.

- El único evento que provoca un *full checkpoint* es una operación de shutdown ordenado.

5.3.8. MMON: Manageability Monitor



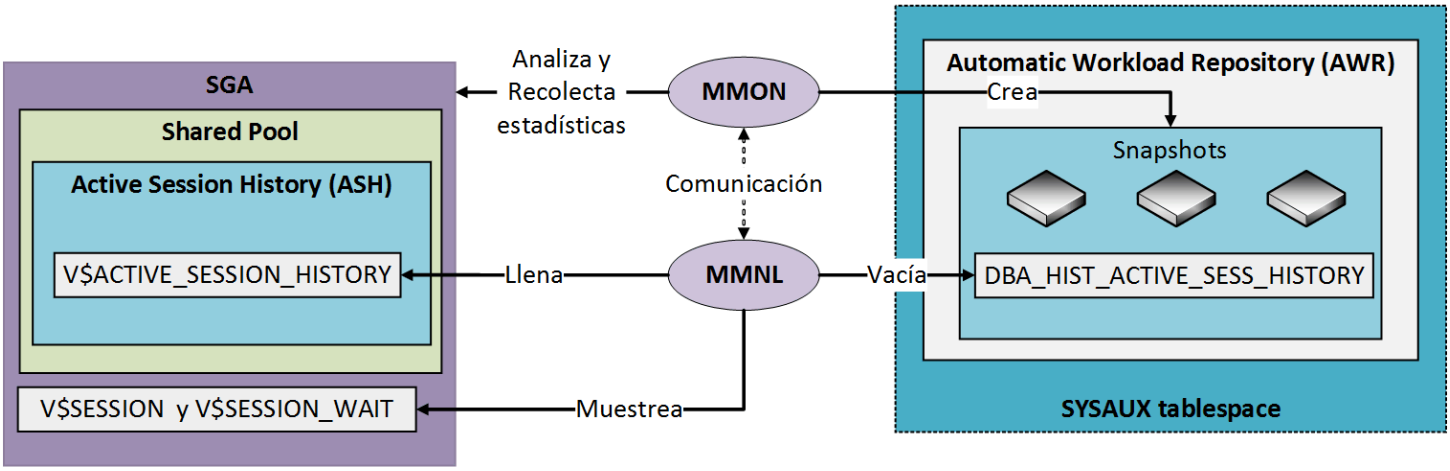
- La BD cuenta con funcionalidades que permiten mejorar el desempeño de forma automática:
  - Monitoreo automático de la base de datos
  - Capacidades de auto afinamiento
- Para poder implementar estas funcionalidades la BD recolecta una gran cantidad de ***estadísticas*** que contienen datos de la actividad y desempeño actual de la BD.
- Estas estadísticas son almacenadas en la SGA y pueden ser consultadas a través de sentencias SQL.
- Ejemplos de estadísticas son: Núm. De registros estimados por tabla, diversidad de valores de una columna, uso promedio del CPU, memoria y disco; etc.
- Adicionalmente se realizan cálculos, estimaciones, etc.
- Toda esta información debe almacenarse de forma permanente para realizar análisis futuros, en especial con estadísticas generadas a lo largo del tiempo.
- MMON es el proceso que se encarga de guardar o respaldar en disco el contenido de todas estas estadísticas en un repositorio llamado ***Automatic Workload Repository (AWR)***.
  - Este repositorio contiene datos históricos de desempeño y estadísticas acumulativas a diferentes niveles: nivel base de datos, nivel sesiones, Sentencias SQL, segmentos y servicios.
  - Las estadísticas que se almacenan en el AWR representan las bases para realizar *afinamiento* del desempeño de la BD así como la capacidad de *auto administrarse*.
- Por default MMON despierta cada hora y realiza la sincronización de estos datos con el AWR.
- La información que se recolecta en cada hora se le conoce como ***snapshot.***
- Por default estos datos se guardan de manera permanente solo por unos cuantos días (8 aprox.) en el tablespace `sysaux`. Su permanencia real depende de la configuración de retención de datos.
- ***AWR Baseline***: Conjunto de estadísticas tomadas en un periodo de tiempo cuando la BD se comporta correctamente con cargas de trabajo máximas (tiempos pico). Este conjunto de estadísticas puede ser formado por un conjunto de snapshots.
- EL AWR baseline puede ser comparado con estadísticas tomadas en un periodo donde la BD se comporta con bajo desempeño y poder de esta forma diagnosticar problemas.

5.3.8.1. ADDM (Automatic Database Diagnostic Monitor)

- Adicionalmente MMON lanza una herramienta llamada ***ADDM (Automatic Database Diagnostic Monitor).***
- El ADDM es un self-advisor que de forma automática y proactiva diagnostica el desempeño de la BD y determina como resolver problemas identificados.
- Dentro de sus actividades, ADDM identifica áreas dentro de la BD que consumen una gran cantidad de tiempo de procesamiento, realiza un análisis detallado para encontrar la causa raíz del problema.

- ADDM puede recomendar realizar cambios a nivel de hardware, configuración de la BD, cambios en configuración de esquemas, o aplicaciones. Si la aplicación se realiza, ADDM puede reportar el beneficio obtenido.
- Como resultado de ejecutar esta herramienta, MMON puede lanzar diversas ***alertas*** notificando la existencia de algún posible problema.

5.3.9. MMNL: Manageability Monitor Lite



- MMNL es un proceso que asiste a MMON.
- MMNL similar a MMON, obtiene datos estadísticos pero a nivel de sesión.
- Estos datos se almacenan en un buffer en la SGA llamado ***Active Session History (ASH) buffer***.

5.3.9.1. Active Session History (ASH)

- Realiza el muestreo de sesiones activas en la base de datos cada segundo.
- De forma específica, el muestreo se realiza desde las vistas `v$session` y `v$session_wait` y se almacenan en la vista `v$active_session_history`.
- Los datos que registra incluyen:
  - User ID.
  - Estado.
  - Sentencia SQL que está ejecutando.
- Es útil para diagnosticar problemas de rendimiento que no fueron detectados por ADDM, típicamente ocurridos en periodos de tiempo muy cortos.

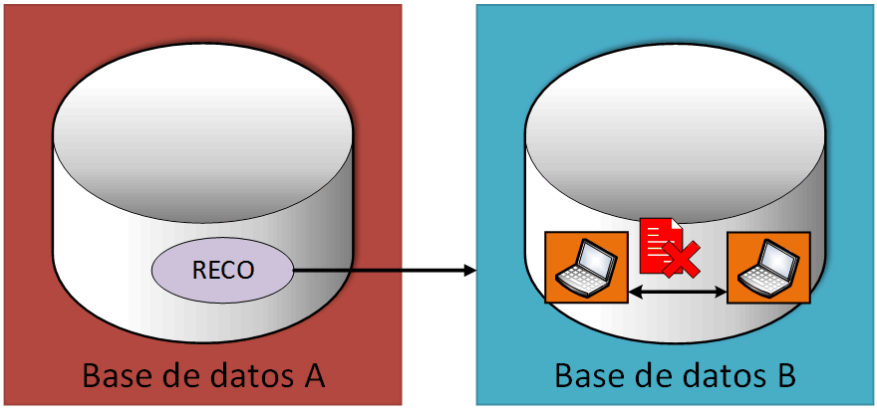
Ejercicio en clase 3

Considerando las vistas `v$session` , `v$sql`, resolver los siguientes puntos:

A. Abrir una terminal y entrar a sesión con el usuario `sys`. Abrir SQL developer y entrar a sesión con el usuario `sys`. Genere una sentencia SQL que muestre el usuario del sistema operativo, la máquina o host de donde se realizó la conexión y el status de la sesión de usuario `sys` conectado en SQL developer.
B. Mostrar el identificador y el texto o código SQL de la última sentencia que se ejecutó en la sesión obtenida en el punto anterior.

C. Generar una sentencia SQL que muestre la fecha y el código SQL (no siempre existe) de todas las veces que se ha realizado un muestreo a la sesión obtenida en el punto A. Usar `v$active_session_history`

5.3.10. RECO: Recovered Process



- Este proceso de background se utiliza en bases de datos distribuidas.
- RECO resuelve automáticamente transacciones fallidas o dudosas (in-doubt) debido a problemas de red o del sistema.
- El proceso RECO de un nodo se conecta de forma automáticamente a las bases de datos involucradas en la transacción en duda y reestablece la conexión.
- Elimina las filas de las tablas de transacciones pendientes de cada base de datos que corresponden a las transacciones resueltas.
- Si RECO no puede conectarse con un servidor remoto, el proceso intentará conectarse nuevamente dentro de un intervalo de tiempo. Este tiempo crecerá de forma exponencial antes de que RECO intente otra conexión.

Ejemplo:

Suponer que en la BD A se lanzan las siguientes sentencias:

```
update orden set importe = 100 where orden_id =15;
update orden@B set importe = 100 where orden_id =15;
commit;
```

- La primera instrucción se ejecuta de forma local en la BD A, pero la segunda sentencia se ejecuta en una BD remota identificada por la liga `@B`
- La instrucción `commit` ordena a las 2 Bases de Datos confirmar el cambio.
- Debido a que se trata de una transacción distribuida, se requiere hacer uso del protocolo ***Two Phase Commit (2PC).***
- La instrucción `commit` se ejecuta en cada base de datos y necesita coordinarse.
- Si una falla y la otra no, toda la transacción debe deshacerse a través de una operación `rollback`.
- El protocolo 2PC permite preparar a cada una de las Bases de Datos para instruir a cada BD realizar un `commit`. Esto implica que sus LGWRs escribirán los cambios en sus Online Redo Logs. A esta etapa se le conoce como ***Primera fase.***

- Si los commits locales se reportan como exitosos, la transacción se considera como exitosa y el `commit` de la tercera instrucción se considera exitoso. A esta etapa se le conoce como ***segunda fase.***
- RECO se encarga de cancelar e invocar operaciones de `rollback` en cada base de datos si algo sale mal durante este proceso.

5.4. OTROS PROCESOS DE BACKGROUND.

Nombre	Descripción
CJQ0, J000	Estos procesos administran Jobs que son configurados para ser ejecutados periódicamente a través del uso de un coordinador de colas de trabajos CJQ0. Los procesos Jnnn son los encargados de ejecutar las tareas.
D000	Dispatcher process que envía sentencias SQL a los share server processes Snnn. Esto aplica cuando el uso de Shared servers es habilitada (revisar tema 2).
DBRM	Database Resource Manager es el responsable de generar planes para realizar la administración de recursos que son otorgados a usuarios, aplicaciones y servicios. Los planes especifican la forma en la que los recursos son distribuidos a los consumidores de recursos.
DIA0	Diagnosability process zero Reponsable de la detección de procesos bloqueados y resolución de deadlocks.
DIAG	Realiza la ejecución de Dumps de diagnóstico así como la ejecución de comandos `oradebug`; herramienta empleada para detectar problemas con la instancia.
FBDA	Flashback Data Archiver Process realiza el archivado de registros históricos de tablas que son configuradas para hacer uso de la funcionalidad de recuperación de datos vía fastBack.
VKTM	Virtual Keeper of Time es un proceso encargado de sincronizar la fecha y hora en varios servidores de bases de datos, en especial cuando se hace uso de un Cluster. Por ejemplo, Oracle RAC.

Para una lista completa de procesos, revisar este enlace.

5.5. VISTAS DEL DICCIONARIO DE DATOS ASOCIADAS CON PROCESOS.

- La vista `v$process` contiene el detalle de todos los procesos de background conectados a una instancia

```
select sosid,pname,execution_type,username,tracefile,cpu_used
from v$process
where pname is not null
order by pname;
```

Muestra de un total de 85 procesos.

SOSID	PNAME	EXECUTION_TYPE	USERNAME	TRACEFILE	CPU_USED
3520	AQPC	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_aqpc_3520.trc	30288
3526	CJQ0	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_cjq0_3526.trc	1822776
3453	CKPT	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_ckpt_3453.trc	126340
3416	CLMN	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_clmn_3416.trc	27043
3439	DBRM	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_dbrm_3439.trc	413068
3449	DBW0	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_dbw0_3449.trc	390351
3447	DIA0	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_dia0_3447.trc	287964
3434	DIAG	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_diag_3434.trc	42024
3425	GEN0	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_gen0_3425.trc	65943
3431_3432	GEN1	THREAD	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_gen1_3431_3432.trc	109671
3455	LG00	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_lg00_3455.trc	294038
3459	LG01	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_lg01_3459.trc	187274
3451	LGWR	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_lgwr_3451.trc	142539
3467	LREG	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_lreg_3467.trc	47524
3482	M000	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_m000_3482.trc	677353
3611	M001	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_m001_3611.trc	2522486
3613	M002	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_m002_3613.trc	632331
3615	M003	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_m003_3615.trc	786521
3427	MMAN	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_mman_3427.trc	124386
3477	MMNL	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_mmnl_3477.trc	135999
3475	MMON	PROCESS	jorge	/u01/app/oracle/diag/rdbms/jrcbd2/jrcbd2/trace/jrcbd2_mmon_3475.trc	2818995

Revisar el documento del ejercicio práctico 02. Procesos de background.