

Administración de Sistemas Gestores de Bases de Datos

Iván López Montalbán
M.º Jesús Castellano Pérez
John Ospino Rivas

Índice general

1. Instalación y configuración de un sistema gestor de base de datos	1
1.1. Los SGBD y sus funciones. Componentes. Tipos	2
1.1.1. Funciones avanzadas de Oracle	3
1.1.2. Funciones avanzadas de DB2	5
1.1.3. Funciones avanzadas de MySQL	7
1.2. Arquitectura de un SGBD en tres capas	8
1.3. Instalación y configuración de MySQL	9
1.3.1. Instalación desde el gestor de paquetes apt-get	9
1.3.2. Instalación desde el gestor de paquetes synaptic	10
1.3.3. La postinstalación	11
1.3.4. Instalación compilando el código fuente	12
1.3.5. Arranque del servidor mysql	14
1.3.6. Configuración del autoarranque del servidor mysql	16
1.4. Instalación y configuración de DB2 Express	18
1.5. Instalación y configuración de Oracle	32
1.5.1. Pasos previos a la instalación de Oracle	32
1.5.2. La instalación de Oracle	34
1.5.3. La postinstalación	42
1.6. Los ficheros LOG	44
1.6.1. Ficheros Log en Oracle	45
1.6.2. Ficheros Log en DB2	46
1.6.3. Ficheros Log en MySQL	46
1.7. Gestión del espacio de almacenamiento	48
1.7.1. Sentencias SQL para la gestión de almacenamiento	49
1.8. Prácticas	51
1.9. Amplia tus conocimientos	54
2. Acceso a la información	55
2.1. El acceso a la información	56
2.2. Las vistas	56
2.3. Los usuarios	58
2.4. Los privilegios	60
2.4.1. El sistema de privilegios de MySQL	60
2.4.2. El sistema de privilegios de Oracle	64
2.4.3. El sistema de privilegios de DB2	70
2.5. El catálogo de metadatos	72
2.5.1. El catálogo de MySQL	72
2.5.2. El catálogo de Oracle	74
2.5.3. El catálogo de DB2	75
2.6. Prácticas	77

2.7. Amplia tus conocimientos	80
3. Automatización de tareas. Scripts de administración	81
3.1. Herramientas para creación de guiones	82
3.1.1. Herramientas gráficas	82
3.1.2. Herramientas no gráficas	84
3.1.3. Métodos de ejecución de guiones	86
3.1.4. Elección del Método de ejecución adecuado	91
3.2. Planificación de tareas de administración con scripts	92
3.2.1. Estructuras de control de flujo	93
3.2.2. Comandos Unix	103
3.3. Ejemplo de scripts de automatización	112
3.3.1. Ejemplo sencillo	112
3.3.2. Ejemplo avanzado	114
3.4. Disparadores de sistema	119
3.5. Eventos de sistema	120
3.6. Excepciones servererror	122
3.7. Prácticas	123
3.8. Amplia tus conocimientos	125
4. Optimización del rendimiento: monitorización y optimización	127
4.1. Introducción	128
4.2. Herramientas de monitorización de un SGBD	128
4.2.1. Consolas gráficas: Enterprise Manager, Grid Control, Toad.	129
4.2.2. Vistas dinámicas	129
4.2.3. dbms_monitor	131
4.2.4. La traza 10046: <i>alter session, oradebug, sql_trace</i>	134
4.2.5. Informes AWR	135
4.2.6. OVO, NetCool, Tivoli, Patrol.	138
4.3. Elementos y parámetros susceptibles de ser monitorizados	139
4.3.1. Caso 1	140
4.3.2. Caso 2	143
4.4. Optimización en Oracle	144
4.4.1. Instancia	147
4.4.2. Recursos	162
4.4.3. Estadísticas	169
4.4.4. Particionamiento y paralelización	173
4.4.5. Consultas	178
4.5. Herramientas y sentencias para la gestión de índices	190
4.6. Herramientas para la creación de alertas de rendimiento	196
4.7. Prácticas	198
4.8. Amplia tus conocimientos	203

5. BBDD distribuidas y replicadas	205
5.1. BBDD y SGBD distribuidos	206
5.1.1. Componentes de una BBDD distribuida	206
5.1.2. Otras clasificaciones	207
5.2. Técnicas de fragmentación	209
5.3. Consultas distribuidas	210
5.3.1. DB Links	211
5.3.2. Ejecución de consultas distribuidas	213
5.4. Transacciones distribuidas	215
5.4.1. TWO-PHASE COMMIT	215
5.4.2. Ejemplo de transacción distribuida	216
5.5. Optimización de consultas sobre bases de datos distribuidas	219
5.5.1. Optimización mediante consultas derivadas	220
5.5.2. Optimización mediante hints	220
5.6. Replicación	223
5.7. Ejemplo de replicación. MySQL	224
5.7.1. Replicación en DB2 y Oracle	228
5.8. Prácticas	229
5.9. Amplía tus conocimientos	232

INSTALACIÓN Y CONFIGURACIÓN DE UN SISTEMA GESTOR DE BASE DE DATOS

Contenidos

- » Funciones del sistema gestor de base de datos (SGBD). Tipos
- » Arquitectura del SGBD. Arquitectura ANSI/SPARC
- » Instalación y configuración de un SGBD. Parámetros relevantes
- » Herramientas clientes de un SGBD
- » Configuración de los parámetros relevantes
- » Ficheros LOG
- » Organización del espacio de almacenamiento

Objetivos

- » Implantar SGBD analizando sus características y ajustándose a los requerimientos del sistema
- » Verificar el correcto funcionamiento de la instalación
- » Resolver incidencias en el proceso de instalación
- » Configurar las herramientas y software cliente del sistema gestor
- » Gestionar la organización del espacio de almacenamiento

En este capítulo se describe el proceso detallado de instalación y configuración de un SGBD y de sus herramientas clientes, así como la configuración del sistema conforme a los requisitos establecidos.

1.1. Los SGBD y sus funciones. Componentes. Tipos

Un Sistema Gestor de Base de Datos (SGBD) es el conjunto de herramientas que facilitan la consulta, uso y actualización de una base de datos. Hoy en día estas herramientas se utilizan en cualquier tipo de industria para dar soporte a sus bases de datos. Todas las industrias, por muy diferente que sea su naturaleza, necesitan almacenar, consultar y analizar datos, y en este punto, los SGBD son herramientas tecnológicamente punteras para conseguir este objetivo.

Tradicionalmente, los SGBD pueden tratar múltiples tipos de bases de datos, generalizándose en las últimas décadas los *SGBD relacionales* o *RDBMS (Relational Database Management System)*, cuya base es la lógica relacional para la manipulación y almacenamiento de la información.

Existen multitud de SGBD, los que gozan de más fama en el mercado actualmente son entre otros:

DB2 Producto de IBM. Incorpora numerosas versiones: desde versiones comerciales de pago para grandes servidores hasta versiones gratuitas para ordenadores personales con sistemas operativos Windows (DB2 Express).

Oracle El servidor de Oracle es famoso por ser uno de los más completos del mercado, competencia directa de DB2 tanto en funciones como en precio.

MySQL Alternativa libre a los grandes SGBD, propiedad de Oracle tras su compra en 2009, junto con Sun Microsystems. Muy extendido en entornos web con aplicaciones de tipo XAMPP¹. MySQL posee una licencia dual, con la posibilidad de comprar una licencia comercial o someterse a los términos de la licencia pública GNU.

Access Gestor de base de datos ofimáticas de Microsoft. Muy sencillo de utilizar, con un entorno gráfico muy usable. Indicado para usuarios no expertos que deseen crear sus aplicaciones con informes y formularios muy vistosos.

SQL Server La competencia ofrecida por Microsoft a los grandes gestores de bases de datos corporativas. Incorpora un lenguaje de programación (T-SQL) muy potente y versátil, y se integra fácilmente con aplicaciones del entorno de desarrollo .NET.

Todos estos SGBD responden a una serie de necesidades de las empresas privadas y públicas, gobiernos y todo tipo de organizaciones. Estas necesidades se cubren aportando diferentes funciones, y en base a lo numerosas y a la calidad de cada una de estas funciones, el precio de la licencia de uso del SGBD es más o menos cara.

Aparte de las funciones básicas de almacenamiento e integridad de la información, seguridad y concurrencia, la gran competencia existente en el sector ha desencadenado que cada gestor de base de datos incorpore nuevas funciones y aplicaciones increíbles para hacer más grande su lugar en el mercado. A continuación se enumeran las siguientes:

¹XAMPP es una arquitectura libre para el desarrollo y la publicación de sitios web con licencia GNU: X=Cualquier sistema Operativo, A=Apache, M=MySQL, P=Perl, P=PHP

1.1.1. Funciones avanzadas de Oracle

Oracle cuenta con multitud de herramientas que se incorporan como parte del gestor. Las versiones 10g y 11g incorporan las siguientes funciones en forma de aplicaciones que, o bien forman parte de las características del propio gestor, o bien se instalan de forma independiente, o se pueden parametrizar en el momento de la instalación del gestor:

ACID con REDO y Automatic UNDO Esta función hace referencia a las propiedades básicas de una base de datos (Atomicity - Consistency - Isolation - Durability), atomicidad, consistencia, aislamiento y durabilidad.

1. El *REDO* es un mecanismo de manejo de una serie de sentencias DML capaces de regenerar la base de datos en caso de una caída, dejándola de forma consistente.
2. El *Automatic UNDO* es un módulo que se encarga de guardar la información de imágenes de los datos cuando se inician las transacciones.

ASMM Automatic Shared Memory Management se encarga del dimensionamiento automático de la memoria compartida que usa Oracle, redistribuyendo el espacio que Oracle reserva con el objetivo de dar prioridad según la necesidad de su uso.

DATAGUARD Solución de arquitectura de máxima disponibilidad, donde se tiene un entorno en Standby a la espera de una posible caída (por ejemplo incendios, terremotos, etc.) Se usa para largas distancias. En la versión 11g existe la posibilidad de usar ese entorno Standby para consultas.

ASSM Automatic Segment Space Management se encarga de gestionar y distribuir el espacio de los segmentos en los tablespaces, teniendo en cuenta los datafiles y el número de peticiones de extensiones.

RAC Real Application Cluster. Solución de alta disponibilidad en la que una base de datos tiene instancias en 2 o más nodos.

RAC ONE NODE Solución de alta disponibilidad en la que una base de datos de una instancia puede realizar switchover (cambio de papeles entre la base de datos primaria y alguna de las que están en standby) o failover (la BD principal falla y una de las standby pasa a ser la principal).

RMAN Herramienta para hacer de forma sencilla tanto BACKUPS como RECOVERYS. Fríos o calientes, de tablespaces específicos, de datafiles, de archivelogs, etc. Hacer recuperaciones incompletas a un determinado punto en el tiempo, recuperaciones de bloques corruptos, etc. En la actualidad desde la versión 10g, existe un proceso que optimiza el BACKUP incremental, llamado BLOCK CHANGE TRACKING.

FLASHBACK Oracle cuenta con tres tipos de flashback: Query, Table y Database. En caso de error Oracle puede "retroceder en el tiempo" y retomar el punto perdido. Con el FLASHBACK QUERY se puede consultar el estado de los datos en un punto

anterior al tiempo. Con el FLASHBACK TABLE en caso de borrado, actualización o inserción de información errónea, Oracle puede regresar a un punto anterior en el tiempo. Y FLASHBACK DATABASE permite que toda la base de datos regrese a un punto anterior en el tiempo, sin necesidad del antiguo INCOMPLETE RECOVER, que podría ser más engorroso.

ROLLING UPGRADE En los entornos en los que se tiene DATAGUARD Oracle permite realizar UPGRADE de versiones sin pérdida de servicio.

ADVISORS Oracle cuenta con todo tipo de herramientas de diagnóstico con el objetivo de mejorar la vida a los DBA 2.0, para que puedan realizar más tareas y tener mejor control sobre sus bases de datos. Estos ADVISORS están a todos los niveles: consultas pesadas, segmentos fragmentados, necesidad de índices y necesidades en memoria.

SCHEDULER Oracle cuenta con todo un módulo de programación de tareas en las que además se pueden ejecutar SHELL SCRIPTS de Unix o programas en Windows y se pueden reprogramar automáticamente. Adicionalmente, se pueden crear ventanas de mantenimiento para tareas especiales.

RESOURCE MANAGER Permite poder administrar los recursos como CPU, UNDO, TIEMPO, PARALELISMO (por usuario o grupos de usuarios) con el objetivo de priorizar las necesidades de los usuarios más importantes.

ASM (10g y 11g) / ACFS (11g) Automatic Storage Management/ ASM Cluster File Systems es uno de los proyectos estrella en Oracle, porque abarca hasta el almacenamiento. Oracle desde la versión 10g se encarga del almacenamiento. Administra directamente la creación del espacio usado por la base de datos. Además en la versión 11g se puede no solo almacenar la información propia de la base de datos, sino también los binarios y más.

OCFS Oracle Cluster File System. Oracle posee una herramienta para poder almacenar información en Cluster File System. Estos filesystem se usan para almacenar la información de bases de datos en RAC y guardar ficheros propios de Clusterware.

AUDIT Oracle provee unas herramientas para auditar fácilmente tanto el acceso como la modificación de los datos que sea necesario auditar, con la posibilidad de guardar esta información tanto en la propia BBDD como fuera de ella. Desde la 10g existe la posibilidad de guardar los ficheros en XML, otorgando nuevas posibilidades de manejo de la información de auditoría.

DATABASE REPLAY Desde la versión 10g, Oracle cuenta con un módulo que almacena la carga creada en un punto en el tiempo, con el objetivo de poder reproducir pruebas de estrés en entornos de preproducción o aceptación, y con la posibilidad de probar cambios sin que afecten al rendimiento de los entornos de producción.

RAT Real Application Testing, desde la versión 11g Oracle provee a los DBA de varias características en las que es posible observar los posibles efectos que tendría realizar alteraciones en los parámetros de la BBDD, sin que llegue a afectar el rendimiento.

PARTITIONING Oracle puede particionar y subparticionar las tablas y segmentos con el objetivo de mejorar las búsquedas, actualizaciones, inserciones y borrado de información. Permite poder particionar las tablas por RANGE, HASH y LIST. Dependiendo de cómo se acceda y qué maneje la tabla.

DOP (11g) Auto Degree of Parallelism: Oracle calcula automáticamente según la necesidad, si la consulta o algún DML precisa que se ejecute en paralelo, determinando la cantidad de procesos necesarios en la ejecución en tiempo real.

-
- **Actividad 1.1:** Busca en Internet a qué denomina Oracle un DBA 2.0.
-

1.1.2. Funciones avanzadas de DB2

DB2 se presenta en varias ediciones en función de la complejidad del entorno donde vaya a ser instalado (dispositivos móviles, servidores convencionales, servidores MPP-Massively parallel processors), el sistema operativo sobre el que resida (Windows, Unix...), y del uso que quiera dársele (número de usuarios y cpu's, clusters...): Everyplace, Personal Edition, Express-C y Express, Workgroup, Enterprise, Data Warehouse Base, Enterprise with Database partitioning feature, Data Warehouse Enterprise.

Cada uno se licencia de una manera particular, existiendo la posibilidad de añadir al paquete que se compra, funcionalidad aparte, aunque debe tenerse en cuenta que no todas las funcionalidades son compatibles con todas las ediciones.

Además de esto también existe una amplia gama de productos del DB2 Client para la conexión de las aplicaciones cliente, y del DB2 Connect para la conexión hacia el Host.

Las funciones se agrupan de la siguiente forma:

- Alta Disponibilidad: Mediante TSA (Tivoli System Automation), equiparable al Clusterware de Oracle, HADR (High Availability Disaster Recovery), equivalente al Dataguard de Oracle y XKoto: como el RAC de Oracle, pero no es un producto IBM.
- Performance Optimization: para disponer de tablas multidimensionales (MDC- Multidimensional Clustering), consultas materializadas (MQTs- Materialized Query Tables), y paralelismo de consultas. Aumenta/controla el rendimiento de la base de datos (incluye el DB2 Performance Expert y el DB2 Query Patroller).
- Workload Management: posibilita la definición de perfiles y umbrales de decisión para limitar o ampliar el consumo de recursos en la base de datos (incluye el DB2 Governor).
- Advanced Access Control (LBAC): proporciona una arquitectura de alta seguridad de acceso a los datos basándose en un sistema de etiquetas y roles asignadas a los propios datos.

- Spatial Extender: permite almacenar, manejar y analizar datos espaciales: puntos, líneas, polígonos...
- Geodetic Data Management Feature: muy útil en el modelado de datos esféricos y espaciales en aquellas aplicaciones que necesitan tener en cuenta la curvatura de la Tierra.
- Storage Optimization Feature: compresión a nivel de fila y de los backups que aumenta significativamente la velocidad de acceso a los datos y minimiza el coste de almacenamiento.
- Pure XML: introduce un nuevo tipo de dato e índice, una ingeniería híbrida que permite manipular y almacenar datos jerárquicos junto con datos relacionales, definiéndose para su manipulación un nuevo concepto de consulta: las XQuery.
- XML Extender: permite trabajar con documentos XML almacenándolos como ficheros externos a la base de datos, o descomponiéndolos en tablas relacionales que serán guardadas dentro de la base de datos.
- Net Search Extender: muy útil para mejorar el rendimiento de los negocios que precisan máxima velocidad en las búsquedas de información en sus bases de datos.
- Pure Scale: ofrecido como alternativa, puramente IBM, al RAC de Oracle, disponible desde la versión 9.7.

En cuanto a herramientas para facilitar la administración/gestión/explotación de las bases de datos, las básicas son:

- Control Center que permite manejar a golpe de click de ratón todas las bases de datos de las instancias registradas, incluyendo un sinfín de asistentes para generar DDL, catalogar, configurar, hacer backups, restaurar, crear objetos...
- Configuration Assistant para llevar la cuenta de los valores de las variables de entorno, las instalaciones, las comunicaciones...
- Advisors, Event Monitor, Health Center para definir umbrales de alertas, chequear automáticamente el estado de la base de datos, tomar decisiones de tuning, pasar estadísticas...
- El moderno DB2 Data Studio en Eclipse, que tiene como objetivo englobar prácticamente todo lo anterior para simplificar la administración, añadiendo incluso más funcionalidad.

Además de distintas herramientas que también se pueden encontrar aparte (muchas de ellas se licencian separadamente) existen otras como Data Archive Expert, DB2 High Performance Unload, Performance Expert...

1.1.3. Funciones avanzadas de MySQL

Hay que tener en cuenta que MySQL Server es un producto de Oracle Corporation desde que esta adquirió Sun Microsystems. Desde entonces, continuando con la filosofía *open source* del producto, actualmente se ofrecen las versiones MySQL Standard Edition, MySQL Enterprise Edition y MySQL Cluster para Web y aplicaciones de comercio electrónico. Cada versión incorpora nuevas herramientas para la gestión avanzada del gestor. Por ejemplo:

MySQL Replication Disponible en todas las versiones, permite dotar al servidor de base de datos de la escalabilidad y fiabilidad necesaria para una gran aplicación con grandes requisitos de rendimiento. Una herramienta específica de la versión de MySQL para Cluster es la MySQL Geo Replication, para replicar bases de datos en cluster. Grandes aplicaciones como Facebook o ebay implementan replicación mediante MySQL Geo Replication para poder dar servicio a millones de usuarios.

MySQL Partitioning Disponible en las versiones Enterprise y Cluster. Permite realizar divisiones sobre las tablas y distribuirlas a través de un sistema de ficheros en base a unas reglas. Incorpora nuevas características de SQL para su gestión.

MySQL Enterprise Backup Permite realizar todo tipo de backups en caliente sin bloquear las tablas de las bases de datos. Implementa también backups incrementales y múltiples opciones de restauración de forma completa o parcial. Esta herramienta solo se puede encontrar en las versiones Enterprise y Cluster.

MySQL Enterprise Monitor Herramienta para monitorizar constantemente el rendimiento del servidor y resolver potenciales problemas que puedan ocurrirle al gestor. Consta de complementos como *Query Analyzer*, herramienta visual para analizar las consultas enviadas al SGBD, permitiendo buscar problemas en el rendimiento de las mismas. Disponible únicamente en las versiones Enterprise y Cluster.

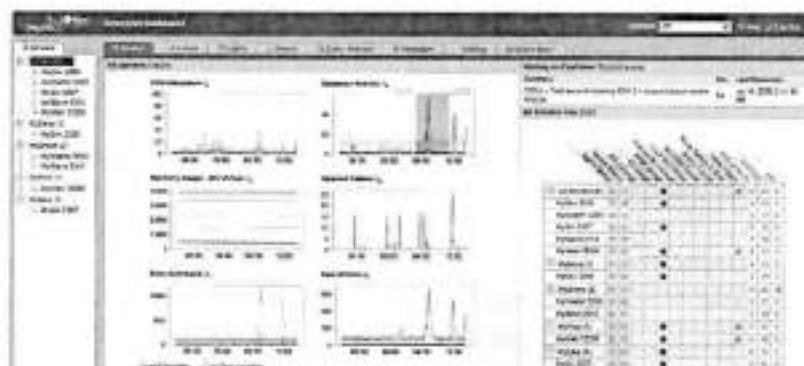


Figura 1.1: MySQL Monitor.

MySQL Cluster Manager Herramienta que simplifica la creación y el mantenimiento de BBDD en cluster automatizando muchas tareas de mantenimiento.

NDB Motor de almacenamiento para clusters. Se conecta a un cluster de nodos ofreciendo alta disponibilidad a través de redundancia, alto rendimiento a través del *partitioning* y escalabilidad mediante una combinación de ambos métodos. Solo disponible en la versión Cluster.

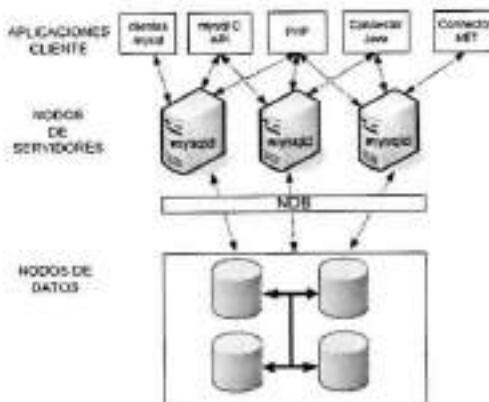


Figura 1.2: Cluster de MySQL con NDB.

1.2. Arquitectura de un SGBD en tres capas

Todos los gestores mencionados hasta ahora cumplen una arquitectura estándar denominada *ANSI-SPARC*, acrónimo de *American National Standards Institute, Standards Planning and Requirement Committee*. Aunque este estándar nunca se formalizó, contiene un modelo tan abstracto que se puede aplicar a cualquier gestor de base de datos corporativas. Este modelo separa la visión que tiene el usuario de una base de datos de cómo está implementada a nivel de software. De este modo, se aisla la complejidad del sistema de almacenamiento de datos y el usuario puede entender la forma de trabajar con los datos de forma más intuitiva. Para aislar esta complejidad, el software se divide en 3 capas:

La capa interna o física Describe cómo los datos son almacenados en los dispositivos hardware del ordenador.

La capa conceptual Representa una forma de describir la información que se almacena en la base de datos y sus relaciones. En esta capa no se especifica cómo se almacenan en el ordenador.

La capa externa Esta capa constituye una colección de vistas de usuarios. Una vista de usuario muestra la información relevante para un usuario en concreto.



Figura 1.3: Arquitectura ANSI-SPARC.

1.3. Instalación y configuración de MySQL

En esta sección se afronta la instalación del sistema gestor de base de datos MySQL desde dos puntos de vista: la instalación en un sistema operativo de tipo Linux usando un gestor de paquetes (en nuestro caso, el gestor de paquetes Synaptic de la distribución Ubuntu en modo gráfico y el gestor de paquetes apt-get en modo texto), y compilando directamente el código fuente para poder instalar MySQL en cualquier versión de sistema operativo basado en Unix.

1.3.1. Instalación desde el gestor de paquetes apt-get

Advanced Packaging Tool, o APT, es un sistema de gestión de paquetes creado por el proyecto Debian. APT simplifica en gran medida la instalación y eliminación de programas en los sistemas GNU/Linux.

Para instalar mysql mediante este gestor de paquetes hay que abrir una consola de comandos (terminal) e introducir el comando:

```
sudo apt-get install mysql-server
```

La ejecución del comando comenzará la descarga de los paquetes necesarios y solicitará una contraseña para el usuario administrador de la base de datos (root):

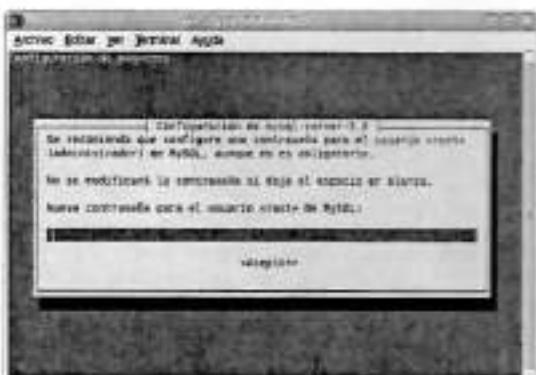


Figura 1.4: Instalación de mysql desde apt-get.

◦ **Actividad 1.2:** En una máquina virtual con Linux, instala mysql-server desde los repositorios con la utilidad apt-get. Conéctate al servidor con el cliente desde la línea de comandos y el usuario root y responde a las siguientes preguntas:

1. ¿Qué versión has instalado?
2. ¿En qué idioma está instalado el servidor?
3. ¿En qué ubicación se encuentra la base de datos mysql?
4. Cambia las passwords de los usuarios root con el comando `UPDATE mysql.user set password=PASSWORD('nueva_password') WHERE user='root'.`
5. Desinstala el servidor haciendo uso del comando `sudo apt-get remove mysql-server`.

1.3.2. Instalación desde el gestor de paquetes synaptic

Synaptic es un software que permite la gestión de paquetes instalados en un sistema operativo de distribución Debian o derivados, como por ejemplo Ubuntu. El uso de este tipo de paquetes facilita la labor de instalación de cualquier software como se comprobará a continuación.

Es posible ejecutar synaptic desde el menú de inicio, seleccionando la opción Sistema, Administración, Gestor de paquetes synaptic, o desde un terminal, escribiendo el comando `synaptic`.

A continuación, pulsar el botón de buscar paquetes, y escribir `mysql-server`. Seleccionar la versión que se desea instalar, o la versión más actual (current version) y marcar la casilla de verificación para instalar. El gestor de paquetes informará de que necesita instalar también paquetes adicionales. Tras pulsar el botón Aplicar comenzará la descarga

de los paquetes de mysql. Posteriormente el SGBD solicitará confirmación para iniciar la instalación, se pulsa en aceptar y se lanza el proceso de instalación.



Figura 1.5: Instalación de mysql desde apt-get.

1.3.3. La postinstalación

La instalación a través de APT configura automáticamente la ubicación de los ficheros del SGBD:

1. Directorio de instalación: /usr/bin.
2. Directorio de datos: /var/lib/mysql.
3. Fichero de configuración: /etc/mysql/my.cnf.

El directorio de instalación indica la ruta de los ejecutables del gestor de mysql. Entre otros se encuentran los siguientes:

1. mysqld: Este programa lanza el proceso *demonio*² de mysql server. Es decir es la aplicación servidora, el software gestor de base de datos propiamente dicho. Es el software que se encarga de organizar los datos y almacenarlos en dispositivos de almacenamiento persistente. Es el encargado de recibir las consultas enviadas por el cliente y ejecutarlas, devolviendo los resultados pertinentes.
2. mysql: Este proceso es la aplicación cliente. En el cliente se escriben las consultas que se van a enviar al gestor y una vez obtenidos los resultados, se muestran por pantalla al usuario de base de datos.

²la última d es precisamente del término (en inglés *daemon*)

3. `mysqld_safe`: Es un script que se encarga de arrancar el gestor de base de datos (`mysqld`). El gestor de base de datos siempre se debe arrancar con este script. También es el encargado de rearrancar el gestor si por cualquier motivo se detuviera.

El directorio de datos contiene los archivos físicos (datafiles) donde se almacenan los datos.

También hay que conocer el fichero de configuración, puesto que es donde se establecen los parámetros de funcionamiento tanto del servidor como de los programas clientes locales.

-
- **Actividad 1.3:** En una máquina virtual con la distribución Ubuntu de Linux, instala desde el gestor de paquetes synaptic la versión más reciente de `mysql-server`. Si no aparece nada en la búsqueda es posible que sea necesario actualizar los repositorios. Al terminar, comprueba qué versión se ha instalado y en qué directorios se encuentran los binarios de la instalación y los archivos de las bases de datos.
-

1.3.4. Instalación compilando el código fuente

La instalación desde un gestor de paquetes es totalmente automática y muy poco configurable. Para poder parametrizar la instalación se debe recurrir a la instalación desde el propio código fuente.

Para poder instalar mysql desde el código fuente, hay que descargar el archivo de código fuente comprimido de la página de mysql. Es posible descargar numerosas versiones, en este libro se ha optado por el archivo '`mysql-5.1.31.tar.gz`' correspondiente a la versión del servidor mysql 5.1.31, y los pasos que se van a seguir corresponden a la instalación en un ubuntu 10.04. Se puede extraer la instalación de cualquier servidor de mysql en cualquier máquina linux cambiando unos simples detalles.

Para comenzar, se descarga el fichero en un directorio del servidor, el mejor sitio para descargarlo es el directorio '`home`' (casa) del usuario con el que se haya entrado al sistema.

De forma genérica, la instalación de cualquier software desde el código fuente en linux se debe realizar fundamentalmente en 4 pasos:

1. Convertirse en el superusuaria del sistema (root) con el comando '`sudo su`' y crear el directorio donde se va a instalar con el comando `mkdir`.
2. Descomprimir el código fuente con el comando `tar -zxvf fichero.tar.gz`.
3. Configurar el entorno con el comando `./configure --prefix=/ubicacion_instalacion`.
4. Compilar el código fuente con el comando `make`.³
5. Instalar los ficheros binarios que se han generado en la compilación con el comando '`make install`'.

³La utilidad `make` y las herramientas de compilación deben estar previamente instaladas, si no reconoce el comando, es posible instalar el paquete '`build-essential`' desde synaptic

Antes de comenzar se debe planificar dónde se desea que se instale el servidor y dónde las bases de datos:

1. Los binarios se instalarán en el directorio /usr/mysql.
2. Las bases de datos irán en el directorio /usr/mysql/data.

Para instalar el software servidor de base de datos, se han de seguir los mismos pasos:

```
#paso 0: Convertirse en root y crear el directorio de instalación
sudo su
mkdir /usr/mysql

#paso 1: Descomprimir el fichero
tar -zvxf mysql-5.1.31.tar.gz

#paso 2: Cambiar al directorio donde se ha descomprimido
cd mysql-5.1.31

#paso 3: Configurar el entorno (tardará un rato)
# la opción --prefix indica al comando dónde se van a instalar los binarios
# la opción --localstatedir indica dónde se guardarán las bases de datos
# la opción --with-named-curses-libs=/lib/libncurses.so.5
#   indica la ruta a una librería necesaria
./configure --prefix=/usr/mysql --localstatedir=/usr/mysql/data
               --with-named-curses-libs=/lib/libncurses.so.5

#paso 4: Compilar (tardará unos minutos)
make

#paso 5: Instalar
make install
```

Durante el proceso pueden aparecer errores por falta de librerías o paquetes instalados, en este caso habría que identificar qué librerías o qué paquetes faltan e instalarlos.

De igual modo, se podría haber agregado al comando *configure* algunos parámetros para el soporte de características adicionales, como por ejemplo, el soporte de conexiones cifradas con ssl (secure sockets layer), agregando la opción *--with-openssl* (cuyo paquete debería estar instalado).

Para terminar, se delega la tarea de arrancar y parar el gestor de base de datos a un usuario especial (que no sea el superusuario root), que se encargará de administrar la base de datos, y al que se llamará 'mysql'. Se puede crear este usuario con el gestor de usuarios de ubuntu, accediendo desde Sistema ⇒ Administración ⇒ Usuarios y Grupos.



Figura 1.6: Usuarios y grupos.

Se pulsa el botón añadir creando así el nuevo usuario:



Figura 1.7: Nuevo usuario.

Finalmente, hay que cambiar los permisos de los ficheros instalados para que el propietario sea el nuevo usuario creado:

```
#cambiar propietario del directorio de instalación  
$chown -R mysql:mysql /usr/mysql  
#cambiar de usuario para las posteriores acciones  
su mysql
```

1.3.5. Arranque del servidor mysql

Para arrancar por primera vez el servidor de base de datos, es necesario tener dos cosas, por un lado el catálogo de metadatos creado y el fichero de configuración 'my.cnf' preparado, este fichero se puede copiar de los códigos fuentes descomprimidos.

El catálogo de metadatos se crea ejecutando el script `mysql_install_db` que está ubicado en el directorio bin dentro de la instalación de mysql:

```
#creación del catálogo de metadatos  
/usr/mysql/bin/mysql_install_db --user=mysql
```

El fichero de configuración puede tener varias ubicaciones, por ejemplo el directorio de datos /usr/mysql/data.

En el directorio 'support-files' de los códigos fuente, existen varias ejemplos de ficheros de configuración ya preparados para sistemas grandes, pequeños y medianos. Se considerará que el sistema es mediano copiando el fichero de configuración mediano al directorio de datos:

```
#Generar el fichero my.cnf  
cd /usr/mysql-5.1.31  
cp support-files/my-medium.cnf /usr/mysql/data/my.cnf  
#Cambiar los permisos  
cd /usr/mysql  
chown -R root .  
chown -R mysql data  
chgrp -R mysql .
```

Después de copiarlo hay que editararlo para comprobar ciertos parámetros:

```
vi /usr/mysql/data/my.cnf  
  
user          = mysql  
pid-file     = /usr/mysql/data/mysqld.pid  
socket        = /usr/mysql/data/mysqld.sock  
port          = 3306  
basedir       = /usr/mysql  
datadir       = /usr/mysql/data  
tmpdir        = /tmp
```

A continuación, ya se puede arrancar el servidor manualmente con el comando mysqld_safe:

```
/usr/mysql/bin/mysqld_safe --user=mysql &
```

Para comprobar que está arrancado el servidor se puede ejecutar el comando ps:

```
ps -ef|grep mysqld
root  2419      1  0 09:08 ?  00:00:00 /bin/sh /usr/bin/mysqld_safe
mysql  2461  2419  0 09:08 ?  00:00:02 /usr/sbin/mysqld --basedir=/usr
      --datadir=/var/lib/mysql --user=mysql
      --pid-file=/var/run/mysqld/mysqld.pid
      --skip-external-locking --port=3306
      --socket=/var/run/mysqld/mysqld.sock
```

Se pueden observar dos procesos, mysqld_safe y mysqld, el primero mantiene 'vivo' al segundo, que es el servidor de bases de datos.

1.3.6. Configuración del autoarranque del servidor mysql

A continuación, para poder configurar el sistema para que arranque mysql al iniciar, hay que copiar el fichero mysql.server del código fuente y reconfigurarlo para adaptarlo al sistema, para ello hay que seguir los siguientes pasos:

```
#ir al directorio de código fuente->support-files
cd /usr/mysql/mysql-5.1.31/support-files

#copiar en /etc/init.d/ el Fichero mysql.server con el nombre mysql
cp mysql.server /etc/init.d/mysql

#hacer ejecutable el fichero
chmod +x /etc/init.d/mysql

#editar lo
vi /etc/init.d/mysql

#rellenar adecuadamente basedir, datadir y pid_file para que
#sean los que hemos usado al instalar:
basedir=/usr/mysql          #directorio base
datadir=/usr/mysql/data      #directorio datos
pid_file=/usr/mysql/data/mysqld_5.1.31.pid  #fichero pid
#comprobar que el usuario con el que se va a arrancar es mysql
user=mysql

#crear un enlace simbólico al directorio del nivel de arranque 2:
ln -s /etc/init.d/mysql /etc/rc2.d/S50mysql

#este comando en linea, acepta las opciones start,stop y restart
#Se arranca manualmente, se prueba la conexión y se detiene el servidor
/etc/init.d/mysql start
mysql -u root
/etc/init.d/mysql stop
```

Finalmente, habría que dotar de password a los usuarios anónimos y a los usuarios 'root' para que no se pueda conectar ningún usuario indeseable.

◦ **Actividad 1.4:** Sigue los pasos descritos en el capítulo para instalar la última versión de mysql-server desde el código fuente. Asegúrate primero que tienes instaladas las herramientas de compilación y enlazado, ejecutando el comando make en un terminal. Si el sistema no reconoce el comando, instala el paquete 'build-essential' antes de proceder con la instalación.

◦ **Actividad 1.5:** Por defecto, mysql da los mensajes de error en inglés:

```
mysql> use aaa;  
ERROR 1049 (42000): Unknown database 'aaa'
```

Cambia los mensajes a español con los siguientes pasos: en el fichero my.cnf hay que añadir la siguiente entrada en la sección [mysqld]:

```
language=/usr/mysql/share/mysql/spanish
```

Para comprobarlo hay que reiniciar el servidor:

```
/etc/init.d/mysql restart
```

Después, pruébalo:

```
mysql> use aaa;  
ERROR 1049 (42000): Base de datos desconocida 'aaa'
```

◦ **Actividad 1.6:** Crea una cuenta en mysql para poder acceder y publicar en los foros de mysql. <http://dev.mysql.com/register/>

1.4. Instalación y configuración de DB2 Express

DB2 no es posible instalarlo desde el código fuente, puesto que no es código libre, es propiedad de IBM. Sin embargo, su versión DB2 Express-C es gratuita, y aunque no es libre en el sentido GNU, se puede usar y distribuir libremente, tanto en sistemas operativos Windows como Unix, de ahí viene la *C* de DB2 Express-*C* que significa *Community*. Para recibir asistencia técnica y soporte o funcionalidades añadidas, sí que es necesario pagar una suscripción anual.

La instalación de DB2 Express-C está repleta de opciones de configuración que hacen de su instalación un proceso muy sencillo. A continuación se describe paso a paso el proceso de instalación y configuración para su puesta en marcha.

El primer paso de la instalación consiste en descargar el software de la página web de IBM <http://www-01.ibm.com/software/data/db2/>:

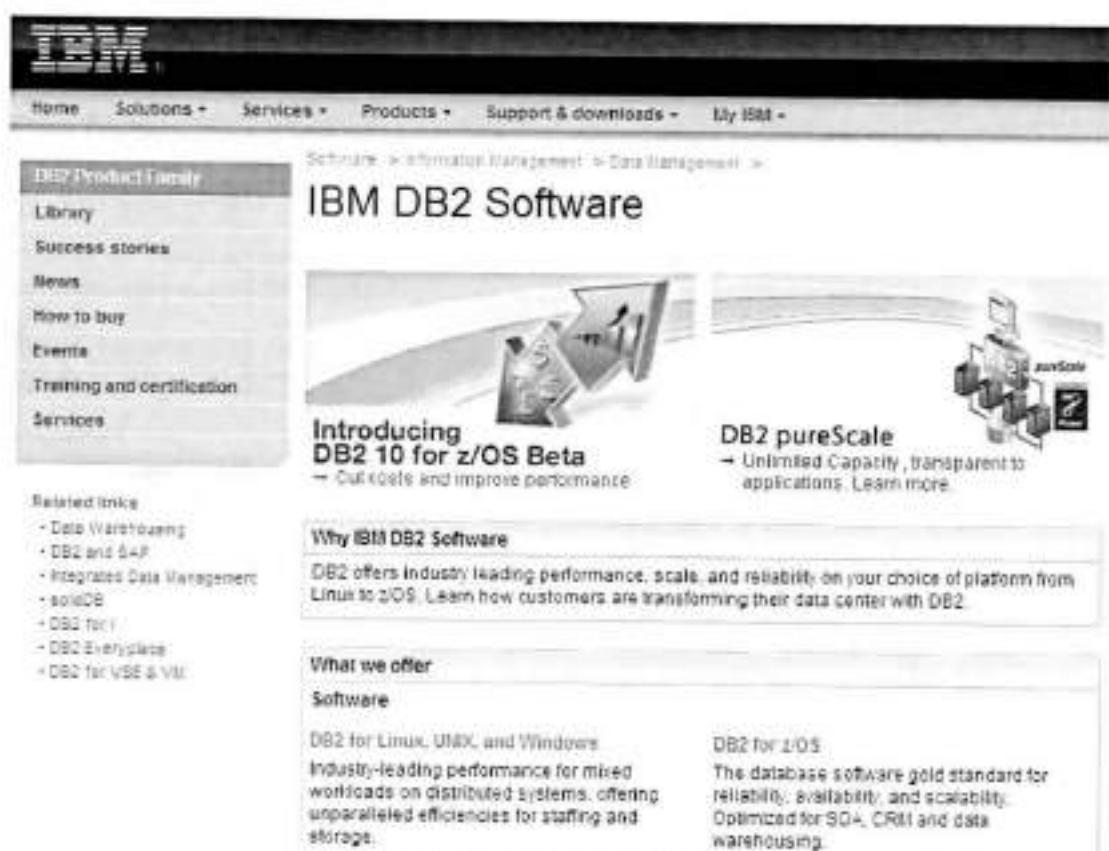


Figura 1.8: Descarga de DB2. Paso 1.

Después, hay que pinchar en *DB2 for Linux, UNIX and Windows* (o DB2 LUW), a continuación *Express-C* y finalmente *Download* (Descarga).

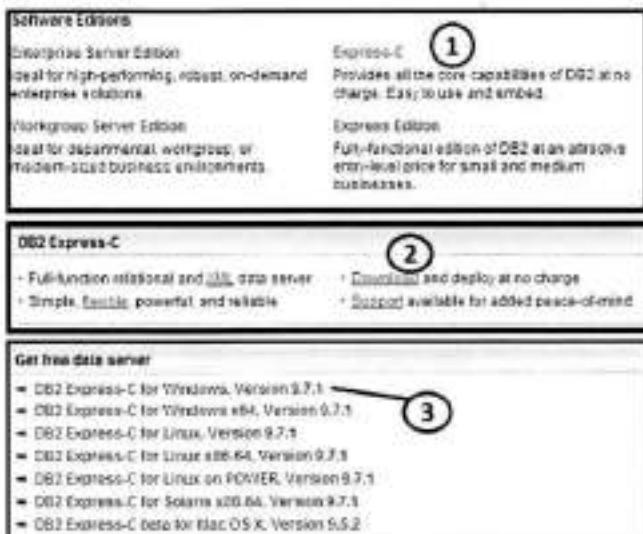


Figura 1.9: Descarga de DB2. Paso 2.

Después de pinchar sobre *DB2 Express-C for Windows, Version 9.7.1* pide un usuario, se debe dar de alta uno nuevo con una dirección de correo y aceptar las condiciones de la licencia. Se debe leer exhaustivamente los términos de la licencia, puesto que en el caso de IBM, el código no es libre y podría incurrirse en alguna ilegalidad si se hace un uso inapropiado de la licencia. Seleccionar los tres productos db2exc_971_WIN_x86.zip, ibm_data_studio_standalone_win.zip y db2exc_vsai_971_WIN_x86.exe.

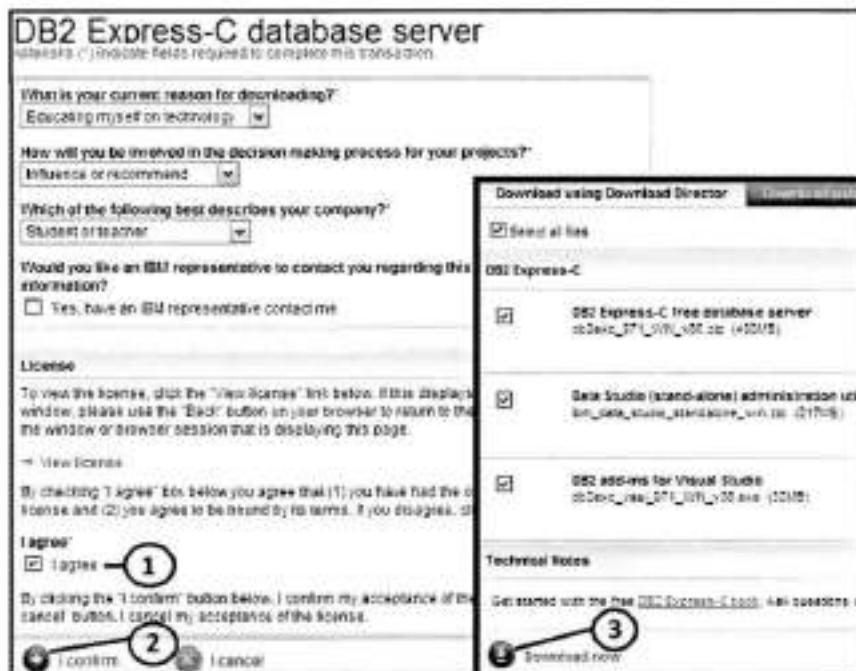


Figura 1.10: Descarga de DB2. Paso 3.

Después de bajar el software, IBM mandará al correo del usuario dado de alta un mensaje agradeciendo la confianza depositada y ofreciendo manuales y más información sobre el DB2 Express C.

Para arrancar el asistente de la instalación, pulsar sobre:
db2exc_971_WIN_x86.zip\EXPC\image\setup.exe

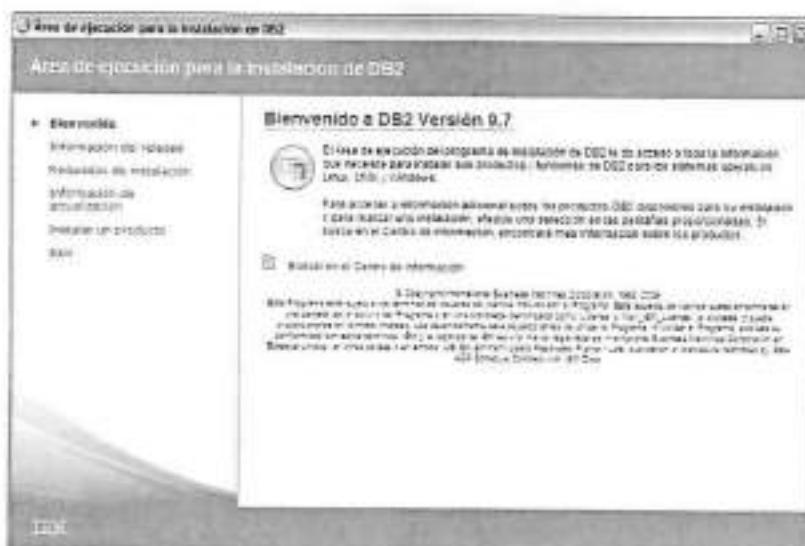


Figura 1.11: Instalación de DB2. Paso 1.

Si se tiene acceso a internet, a través de las distintas opciones que ofrece la instalación, se podrá acceder a información de interés para la instalación y para el DB2 en general, gracias al servidor del *Centro de Información de DB2*.

Figura 1.12: Centro de información de DB2.

Para llevar a cabo la instalación se recomienda tener 1Gb de RAM, si bien, con 256Mb sería suficiente, o 512Mb si se pretenden incorporar las herramientas gráficas.

Para proceder a la instalación, en el menú de la izquierda pulsar sobre *Instalar un producto*:

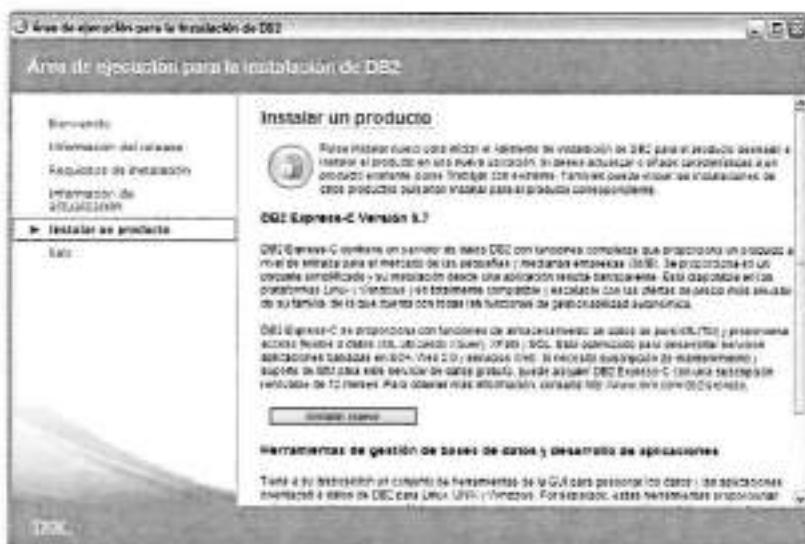


Figura 1.13: Instalación de DB2. Paso 2.

Seleccionar *Instalar nuevo* y a continuación pulsar en *Siguiente*

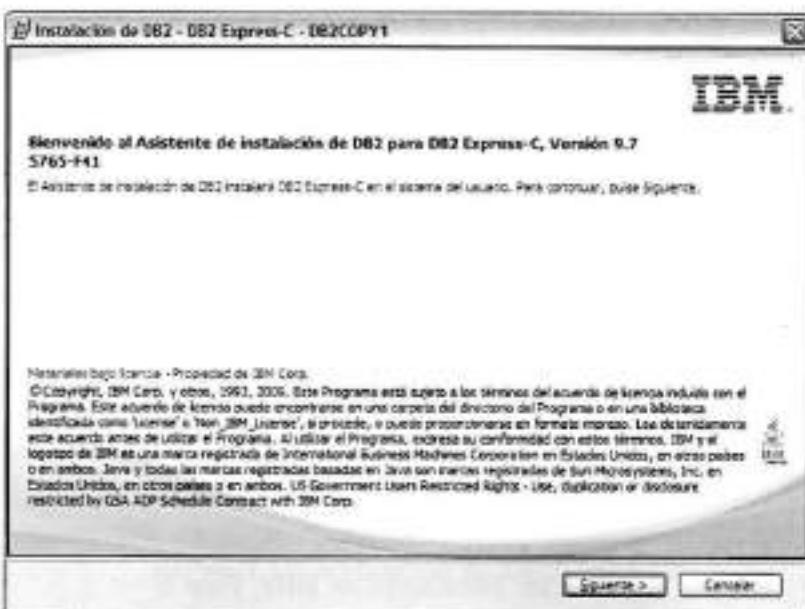


Figura 1.14: Instalación de DB2. Paso 3.

Marcar *Acepto los términos del acuerdo de licencia* y pulsar *Siguiente*



Figura 1.15: Instalación de DB2. Paso 4.

Dejar señalada la opción *Típica*. En *Ver opciones* se puede observar qué es lo que se va a instalar. Pulsar *Siguiente*.



Figura 1.16: Instalación de DB2. Paso 5.

Cambiar la ubicación del fichero de respuestas si se desea (sería muy útil para hacer instalaciones masivas). Pulsar *Siguiente*.



Figura 1.17: Instalación de DB2. Paso 6.

La carpeta de instalación es donde se copiarán los binarios del software de gestión de bases de datos DB2. Pulsar *Siguiente*.



Figura 1.18: Instalación de DB2. Paso 7.

El servidor de administración de bases de datos o *DAS* es una instancia que se crea como intermediaria entre la instancia de base de datos donde se va a trabajar y las herramientas gráficas del DB2. Escribir una contraseña y su confirmación y pulsar *Siguiente*.



Figura 1.19: Instalación de DB2. Paso 8.

Además, se crea automáticamente una instancia que se llama DB2.



Figura 1.20: Instalación de DB2. Paso 9-1.

En la pantalla de *Configurar instancias de DB2*, pulsar *Siguiente*, pero antes, a través del botón *Configurar* se puede acceder a la información del puerto, que servirá para habilitar la conexión remota a la instancia, así como las opciones de arranque automático de la misma una vez que se inicie el Windows del PC.



Figura 1.21: Instalación de DB2. Paso 9-2.

Para terminar la configuración de la instalación se muestra un resumen con las opciones escogidas.



Figura 1.22: Instalación de DB2. Paso 10.

Tras unos minutos copiando archivos e instalando DB2, el asistente nos informa del proceso completado.

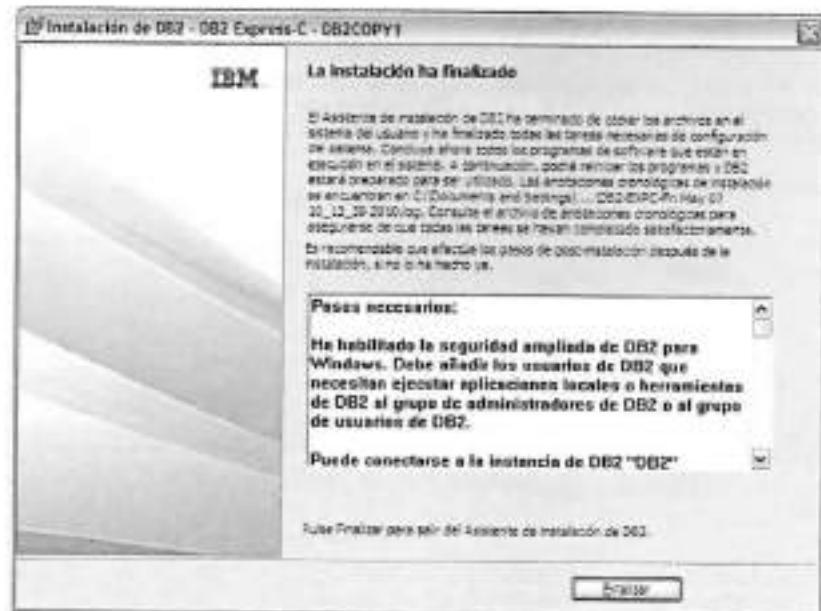


Figura 1.23: Instalación de DB2. Fin.

Puesto que la instalación no ha arrojado ningún mensaje de error, es de esperar que haya salido todo bien, no obstante, para verificar que la instalación se ha realizado correctamente se debe hacer uso del comando db2val. Para ejecutarlo basta con abrir una *Ventana de mandatos* desde el nuevo acceso *IBM DB2*, que habrá creado la instalación en el menú de Windows Inicio/Todos los Programas.

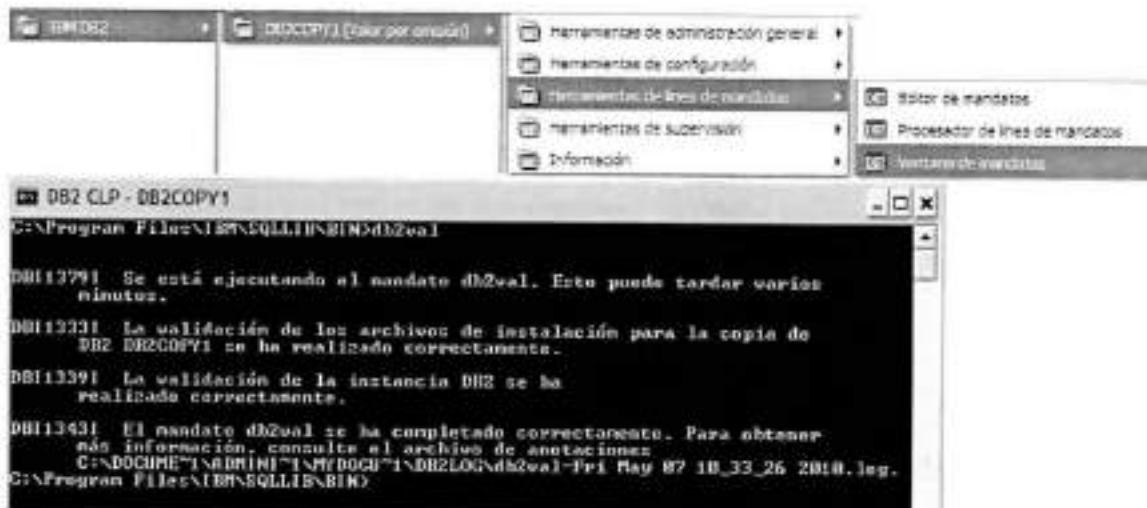


Figura 1.24: Verificación de la instalación. Paso 1.

Otra comprobación para asegurar que la instalación es correcta es la creación de una base de datos. Antes de eso, lo primero es comprobar la instancia DB2 creada está levantada, para ello, desde el menú de Windows Inicio/Ejecutar, ejecutar el programa services.msc

para ver los servicios levantados:

DB2 - DB2COPY1 - DB2	Permite que las aplicaciones...	Started	Automatic	./db2admin
DB2 Governor (DB2COPY1)	Recopile estadísticas pa...	Manual	Automatic	./db2admin
DB2DAS - DB2DA500	Da soporte a peticiones ...	Started	Automatic	./db2admin

Figura 1.25: Verificación de la instalación. Paso 2.

Efectivamente, con la instalación del DB2, se han creado tres nuevos servicios, de ellos, el más importante es el primero. Si estuviera parado, bastaría con pinchar sobre él con el botón derecho y seleccionar *Iniciar*, o bien, desde la anterior ventana de mandatos (desde donde se ejecutó el db2val), escribir db2start:

```
C:\Program Files\IBM\SQLLIB\BIN>db2start
SQL1063N El proceso de DB2START ha sido satisfactorio.
```

Figura 1.26: Verificación de la instalación. Paso 3.

Con el comando db2ilist, se pueden ver las instancias creadas en la máquina:

```
C:\Program Files\IBM\SQLLIB\BIN>db2ilist
DB2
```

Figura 1.27: Verificación de la instalación. Paso 4.

Y ejecutando set db2, gracias a la variable de sistema DB2INSTANCE se sabrá con qué instancia está trabajando la ventana de mandatos abierta:

```
C:\Program Files\IBM\SQLLIB\BIN>set db2
DB2CLP=0B20PADE
DB2INSTANCE=DB2
DB2PATH=C:\Program Files\IBM\SQLLIB
```

Figura 1.28: Verificación de la instalación. Paso 5.

Con el comando db2 list db directory se pueden mostrar las bases de datos configuradas en la instancia objeto de estudio:

```
C:\Program Files\IBM\SQLLIB\BIN>db2 list db directory
SQL1031N El directorio de la base de datos no se encuentra en el sistema de
archivos que se ha especificado. SQLSTATE=58031
```

Figura 1.29: Verificación de la instalación. Paso 6.

En este caso, todavía no existe ninguna. Para crear una base de datos de ejemplo, basta con ejecutar el comando db2sampl:

```
C:\Program Files\IBM\SQLLIB\BIN>db2sampl
Starting the DB2 instance...
Creating database "SAMPLE"...
Connecting to database "SAMPLE"...
Creating tables and data in schema [REDACTED]...
Creating tables with XML columns and XML data in schema [REDACTED]...
Stopping the DB2 instance...
'db2sampl' processing complete.
```

Figura 1.30: Verificación de la instalación. Paso 7.

Expresamente aparece borrado el nombre del schema donde se han creado las tablas, ese nombre será el del usuario que abrió la sesión del Windows.

Y ahora sí:

```

DB2 CLP - DB2COPY1
Stopping the DB2 instance...
'db2sampl' processing complete.

C:\Program Files\IBM\SQLLIB\BIN>db2 list db directory
Directorio de bases de datos del sistema
Número de entradas en directorio = 1
Entrada 1 de base de datos:
Alias de base de datos = SAMPLE
Nombre de base de datos = SAMPLE
Directorio de bases datos locales = C:
Nivel release de base de datos = 6.00
Comentario =
Tipo de entrada del directorio = Indirecto
N.º de partición base datos de catálogo = 0
Nomb. syst. real. serve. alternativo =
N.º puerto de servidor alternativo =

```

Figura 1.31: Verificación de la instalación. Paso 8.

Es decir, que la instancia DB2 (que es el nombre al que apunta la variable de entorno DB2INSTANCE) tiene creada una base de datos en local (eso es lo que quiere decir el valor *Indirecto* en la linea *Tipo de entrada del directorio*) que se llama *SAMPLE* y que reside directamente sobre la unidad C: del disco. De hecho, al navegar con el explorador, se ve fácilmente que existe un directorio C:\DB2\NODE000\SAMPLE, donde DB2 es el nombre de la instancia, NODE000 indica que corresponde al número de partición 0 de esta base de datos (en este caso, la base de datos no está particionada, luego tendrá solo un nodo que es este NODE000), y SAMPLE es el nombre de la base de datos.

La prueba definitiva para comprobar que está todo bien, es realizar una conexión a la base de datos, pero previamente a ello, puesto que en la creación de la base de datos salió el mensaje *Stopping the DB2 instance...*, habrá que levantar la instancia, y luego intentar la conexión con el comando *db2 connect to sample*:

```

C:\Program Files\IBM\SQLLIB\BIN>db2start
SQL1053N El proceso de DB2START ha sido satisfactorio.

C:\Program Files\IBM\SQLLIB\BIN>db2 connect to sample
   Información de la conexión con la base de datos
   Servidor bases datos = DB2/NT 9.7.1
   ID autorización SQL = [REDACTED]
   Alias base datos local = SAMPLE

```

Figura 1.32: Verificación de la instalación. Conexión a la BBDD (1).

El *ID autorización SQL* (que aquí aparece borrado) será el usuario con el que se inició la sesión de Windows. Para conectar con cualquier otro usuario, por ejemplo el db2admin que es el propietario del motor de base de datos, basta con añadir la cláusula user: *db2 connect to sample user db2admin* que pedirá la password por linea de comando, o escribiendo la

password directamente: *db2 connect to user db2admin using la_password_que_sea*

```
C:\Program Files\IBM\SQLLIB\BIN>db2 connect to sample user db2admin
Escriba la contraseña actual de db2admin:
Información de la conexión con la base de datos
Servidor bases datos = DB2/NT 9.7.1
ID autorización SQL = DB2ADMIN
Alias base datos local = SAMPLE
```

Figura 1.33: Verificación de la instalación. Conexión a la BBDD (2).

En todo momento, se puede acceder a la ayuda en línea para conocer la sintaxis adecuada de cada comando, basta con escribir *db2 ? el_comando_que_sea*, o, si se desean ver todos los comandos que se pueden ejecutar: *db2 ?*

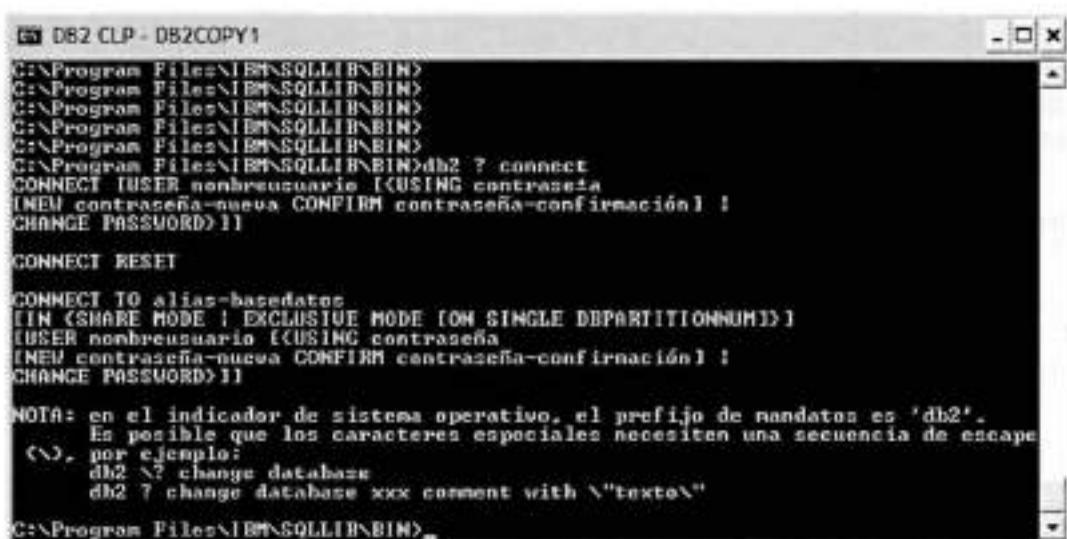


Figura 1.34: Ayuda de DB2.

Nótese que los comandos que se ejecutan sobre la instancia van precedidos del prefijo *db2*, si se quiere evitar esto, se puede abrir un *Procesador de línea de mandatos*. También se puede conseguir esto desde la ventana de mandatos de siempre, ejecutando simplemente *db2*.

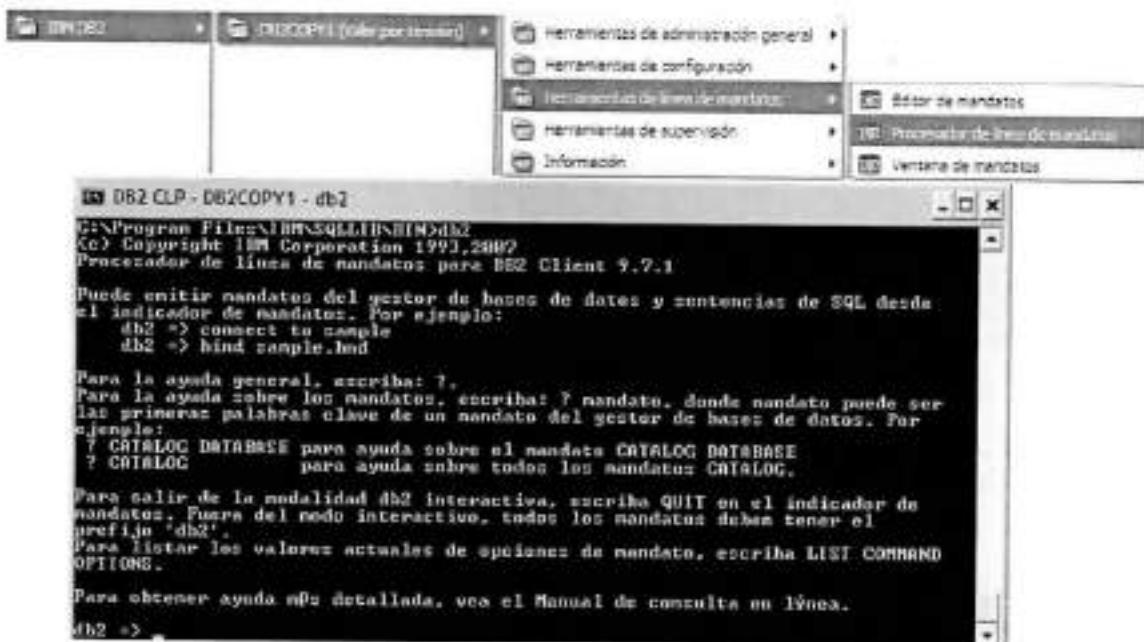


Figura 1.35: Ejecución de comandos en db2 (1).

De modo que ya no será necesario usar el prefijo db2 para escribir los comandos de instancia o base de datos. Por ejemplo, para ver qué conexión está abierta, se ejecutaría directamente *connect*.

```
db2 => connect  
Información de la conexión con la base de datos  
Servidor bases datos = DB2/NT 9.7.1  
ID autorización SQL = [REDACTED]  
Alias base datos local = SAMPLE
```

Figura 1.36: Ejecución de comandos en db2 (2).

Para salir del procesador de linea de mandatos, se usa el comando *quit*.

```
db2 => quit  
DB200001 El mandato QUIT ha finalizado satisfactoriamente.  
C:\Program Files\IBM\SQLLIB\BIN>
```

Figura 1.37: Ejecución de comandos en db2 (3).

Quizá una forma más amigable de lanzar los comandos sea a través del *Editor de mandatos*:



Figura 1.38: Ejecución de comandos en db2 (4).

En este caso se escribe el comando en la ventana superior y para ejecutarlo se pulsa sobre el triángulo verde de la barra de herramientas. Por ejemplo, para ver qué hora es:

The screenshot shows the DB2COPY1 command editor interface. The menu bar includes 'Editor de mandatos Seleccionado', 'Editar', 'Ver', 'Herramientas', and 'Ayuda'. The toolbar contains icons for copy, paste, find, and other operations. A dropdown menu 'Herramientas' is open, showing options like 'Herramientas de administración general', 'Herramientas de configuración', 'Herramientas de línea de mandatos', 'Herramientas de supervisión', and 'Información'. Below the menu is a sub-menu 'Herramientas de línea de mandatos' with options 'Editor de mandatos', 'Procesador de líneas de mandatos', and 'Ventana de mandatos'. The main window displays the command being run:

```

connect to sample;
select current_timestamp from sysibm.sysdummy1;

  
```

Below the command window, the results are shown:

```

Servidor bases datos: DB2/HT 9.7.1
ID autorización TSI: ===
Alias base datos local: *SAMPLE

select current_timestamp from sysibm.sysdummy1

1
-----
2010-08-07-10.31.17.275000

1 registros(s) seleccionados.

Una conexión de JDBC con el destino ha resultado satisfactoria.
  
```

Figura 1.39: Ejecución de comandos en db2 (5)

- **Actividad 1.7:** Instala DB2 Express siguiendo el proceso de instalación de DB2 descrito en esta sección con la última versión disponible en la página web de IBM.

1.5. Instalación y configuración de Oracle

Aunque el software servidor de Oracle dispone de un instalador automático basado en java, sus exigencias en requisitos de instalación y software hacen que la instalación no sea sencilla.

La instalación que se va a describir es del software Oracle 11g R2 para Ubuntu 10.10, y aunque Oracle no soporta la instalación sobre Ubuntu de forma nativa, se pueden utilizar algunos trucos para completar la instalación en esta distribución de linux.

1.5.1. Pasos previos a la instalación de Oracle

Para preparar el entorno hay que crear un usuario delegado que posteriormente será el que utilice el administrador de base de datos para las tareas diarias. Ese usuario pertenecerá a dos grupos, el grupo dba y el grupo oinstall que también hay que crear. Para crearlos se puede hacer con el gestor de usuarios y grupos de ubuntu:

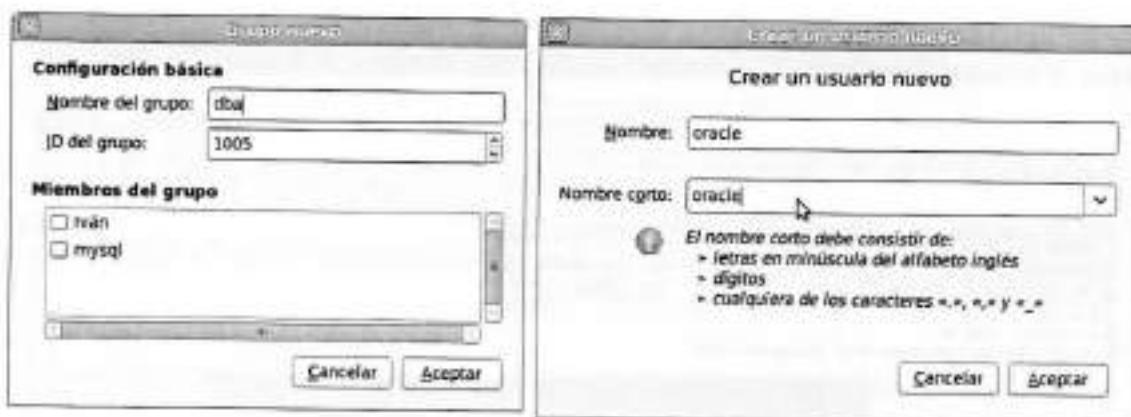


Figura 1.40: Creación de usuarios y grupos.

Después hay que editar el fichero /etc/security/limits.conf para establecer ciertos límites en la cantidad de procesos a crear y número de ficheros. Además, hay que asignarle al usuario oracle como grupo principal el grupo dba y como grupo secundario oinstall. Todos estos primeros pasos hay que hacerlos con el usuario root.

```
#Convertirse en
root sudo su

#Crear copia del fichero original:
cp /etc/security/limits.conf /etc/security/limits.conf.original
#Añadir estas líneas al fichero /etc/security/limits.conf
echo "#Límites para Oracle 11g R2 :" >> /etc/security/limits.conf
echo "oracle soft nproc 2048" >> /etc/security/limits.conf
echo "oracle hard nproc 16384" >> /etc/security/limits.conf
echo "oracle soft nofile 1024" >> /etc/security/limits.conf
echo "oracle hard nofile 65536" >> /etc/security/limits.conf
```

```
#hacer que el usuario oracle pertenezca al grupo de instalación:  
usermod -g oinstall -G dba oracle
```

A continuación, para completar la creación del usuario oracle, hay que declarar las siguientes variables de entorno:

```
#Editar /home/oracle/.bashrc y agregar las siguientes líneas  
vi /home/oracle/.bashrc  
export PATH=$PATH:/var/oracle/product/11.2.0/dbhome_1/bin  
export ORACLE_HOME=/var/oracle/product/11.2.0/dbhome_1  
export ORACLE_BASE=/var/oracle  
export ORACLE_SID=asir  
export NLS_LANG=spanish_spain.we8iso8859p1;
```

Después se crean los directorios donde van a instalarse el software de Oracle:

```
#Crear el directorio de la instalación:  
mkdir -p /var/oracle  
#Hacer al usuario oracle propietario del directorio (recursivo)  
chown -R oracle:dba /var/oracle
```

A continuación, hay que instalar una serie de paquetes para satisfacer los requisitos software del SGBD:

```
#paquetes instalables con apt-get  
apt-get install build-essential  
apt-get install sysstat  
apt-get install libelf-dev  
apt-get install libaio1  
apt-get install libaio-dev  
apt-get install unixODBC  
apt-get install unixODBC-dev  
apt-get install elfutils  
apt-get install lsb-cxx  
apt-get install pdksh  
apt-get install expat  
  
#descargar e instalar un par de paquetes de c++ y trucarlos para que no  
#de problemas de enlazado  
cd /tmp  
wget http://mirrors.kernel.org/ubuntu/pool/universe/g/gcc-3.3/libstdc++5_3.3.6-17ubuntu1_amd64.deb  
dpkg-deb -x libstdc++5_3.3.6-17ubuntu1_amd64.deb ia64-libs  
wget http://mirrors.kernel.org/ubuntu/pool/universe/l/ia32-libs/ia32-libs_2.7ubuntu6.1_amd64.deb  
dpkg-deb -x ia32-libs_2.7ubuntu6.1_amd64.deb ia32-libs
```

```
cp ia64-libs/usr/lib/libstdc++.so.5.0.7 /usr/lib64/
cp ia32-libs/usr/lib32/libstdc++.so.5.0.7 /usr/lib/
cd /usr/lib64/
ln -s libstdc++.so.5.0.7 libstdc++.so.5
cd /usr/lib
ln -s libstdc++.so.5.0.7 libstdc++.so.5
rm /tmp/*_deb
rm -r /tmp/ia64-libs
rm -r /tmp/ia32-libs
```

También hay que crear unos enlaces simbólicos para engañar al instalador y dejar el sistema con la estructura de directorios estilo RedHat.

```
mkdir /etc/rc.d
for i in 0 1 2 3 4 5 6 5 ; do ln -s /etc/rc$1.d /etc/rc.d/rc$1.d; done
ln -s /usr/bin/awk /bin/awk
ln -s /usr/bin basename /bin basename
```

Finalmente, hay que añadir unas líneas en el fichero /etc/sysctl.conf para cambiar unos parámetros en el kernel:

```
#Hacer una copia del fichero original
cp /etc/sysctl.conf /etc/sysctl.original
#Modificar los parámetros del kernel
echo "#" >> /etc/sysctl.conf
echo "#Entradas para Oracle 11gR2" >> /etc/sysctl.conf
echo "fs.aio-max-nr=1048576" >> /etc/sysctl.conf
echo "fs.file-max=6815744" >> /etc/sysctl.conf
echo "kernel.shmall=2097152" >> /etc/sysctl.conf
echo "kernel.shmmni=4096" >> /etc/sysctl.conf
echo "kernel.sem=250 32000 100 128" >> /etc/sysctl.conf
echo "net.ipv4.ip_local_port_range=9000 65500" >> /etc/sysctl.conf
echo "net.core.rmem_default=262144" >> /etc/sysctl.conf
echo "net.core.rmem_max=4194304" >> /etc/sysctl.conf
echo "net.core.wmem_default=262144" >> /etc/sysctl.conf
echo "net.core.wmem_max=1048586" >> /etc/sysctl.conf
#shmmmax tiene que ser igual al tamaño total de la memoria del servidor
echo "kernel.shmmmax=2147483648" >> /etc/sysctl.conf
#cargar nuevos parámetros
sysctl -p
```

1.5.2. La instalación de Oracle

Suponemos que el software de Oracle se halla en una carpeta compartida de otro equipo, por ejemplo, en un servidor Windows donde está insertado el DVD de Oracle o los ficheros

descargados de internet. Lo primero entonces será montar en un directorio la carpeta compartida con el software de oracle y editar el fichero /etc/fstab:

```
#Convertirse en root  
sudo su  
#Crear directorio de montaje  
mkdir /mnt/oracle  
#editar fichero fstab  
vi /etc/fstab  
//192.168.1.33/oracle11g /mnt/oracle smbfs defaults 0 0
```

La dirección IP 192.168.1.33 puede sustituirse por la IP del servidor, y el resto de la ruta con el nombre de la carpeta compartida. Se ha de instalar el paquete smbfs con el comando apt-get install smbfs para poder montar el directorio compartido de windows en el directorio /mnt/oracle. Y si fuera necesario, indicar usuario y contraseña para acceder a la carpeta, añadir username=xxx en lugar de defaults. Después:

```
#montar el directorio:  
mount /mnt/oracle  
#puede introducirse la contraseña en blanco o la de autenticación
```

Como último paso a realizar con root ejecutar:

```
#Permitir acceso a terminal X a todos los usuarios  
xhost +  
#Cambiar a usuario oracle  
su oracle
```

Posteriormente, habiendo cambiado al usuario oracle, se cambia al directorio del software de Oracle y se inicia la instalación:

```
#cambiar al directorio database donde se encuentra el instalador  
cd /mnt/oracle/database/  
# y ejecutarlo:  
.runInstaller -ignoreSysPrereqs
```

La opción ignoreSysPrereqs es para que no compruebe si la distribución linux actual es redhat y así permita continuar. Si todo ha ido bien, se obtendrá una ventana que lanzará el proceso de instalación, el cual consta de 12 pasos:

Paso 1: Si se desea, se puede escribir una dirección de correo electrónico para recibir información de oracle sobre problemas de seguridad.

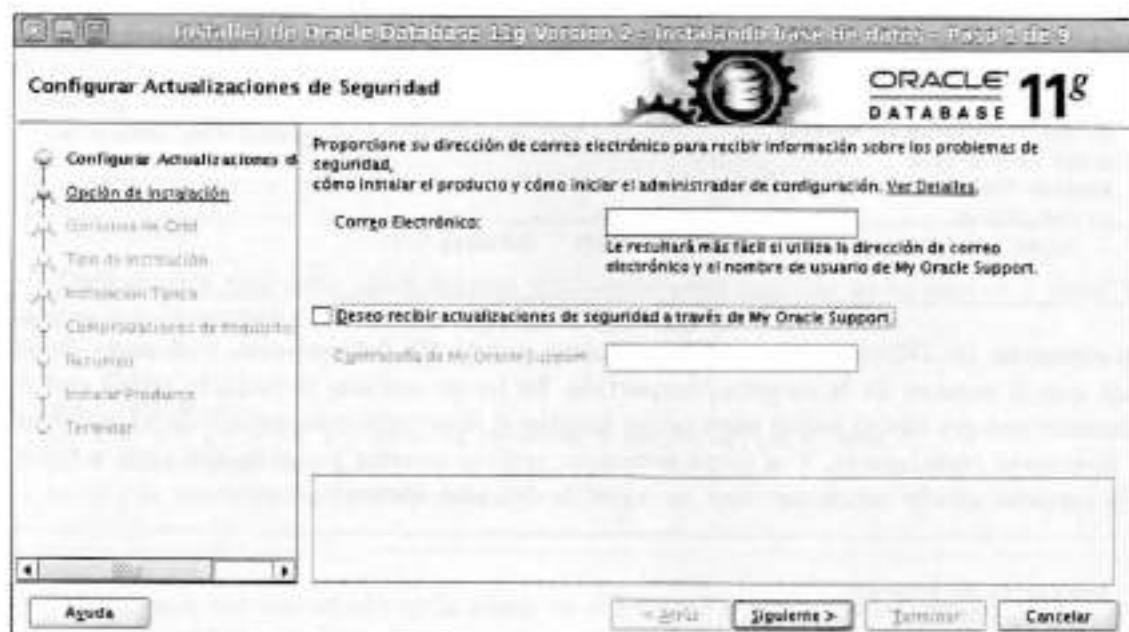


Figura 1.41: Instalación de oracle. Paso 1.

Paso 2: Seleccionar *Instalar solo software de base de datos*.

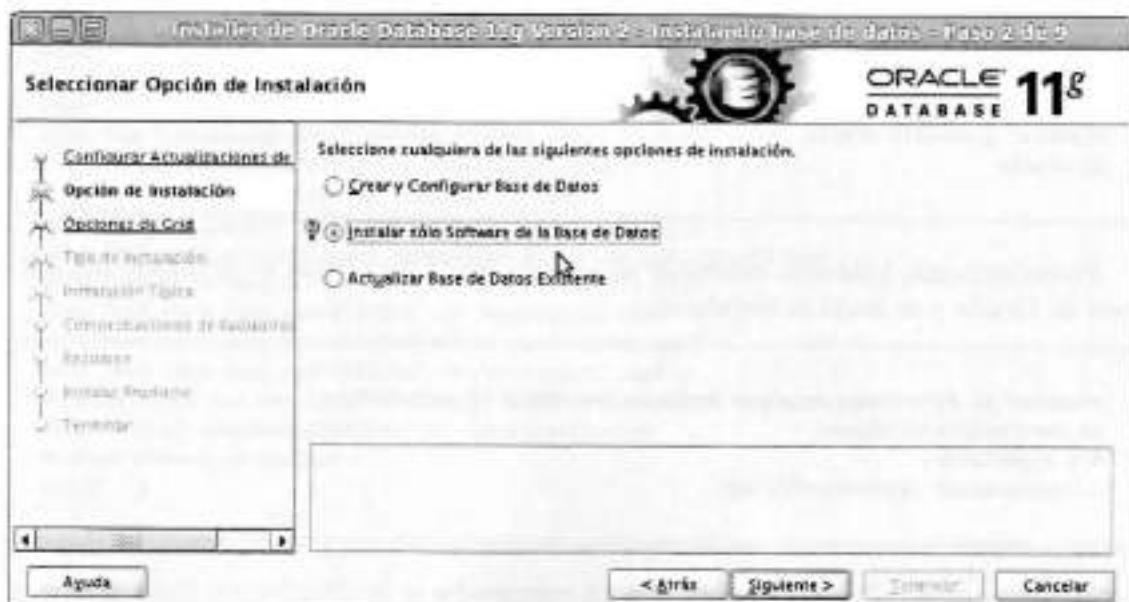


Figura 1.42: Instalación de oracle. Paso 2.

Paso 3: Seleccionar *Instalar base de datos de instancia única*



Figura 1.43: Instalación de oracle. Paso 3.

Paso 4: Seleccionar los idiomas de la instalación.



Figura 1.44: Instalación de oracle. Paso 4.

Paso 5: Seleccionar *Seleccionar Standard Edition One*, la instalación menos exigente en requisitos:



Figura 1.45: Instalación de oracle. Paso 5.

Paso 6: Directorios de instalación conforme a \$ORACLE_BASE y \$ORACLE_HOME.



Figura 1.46: Instalación de oracle. Paso 6.

Paso 7: Configuración del directorio de inventario. Modificar a /var/oracle/oraInventory.

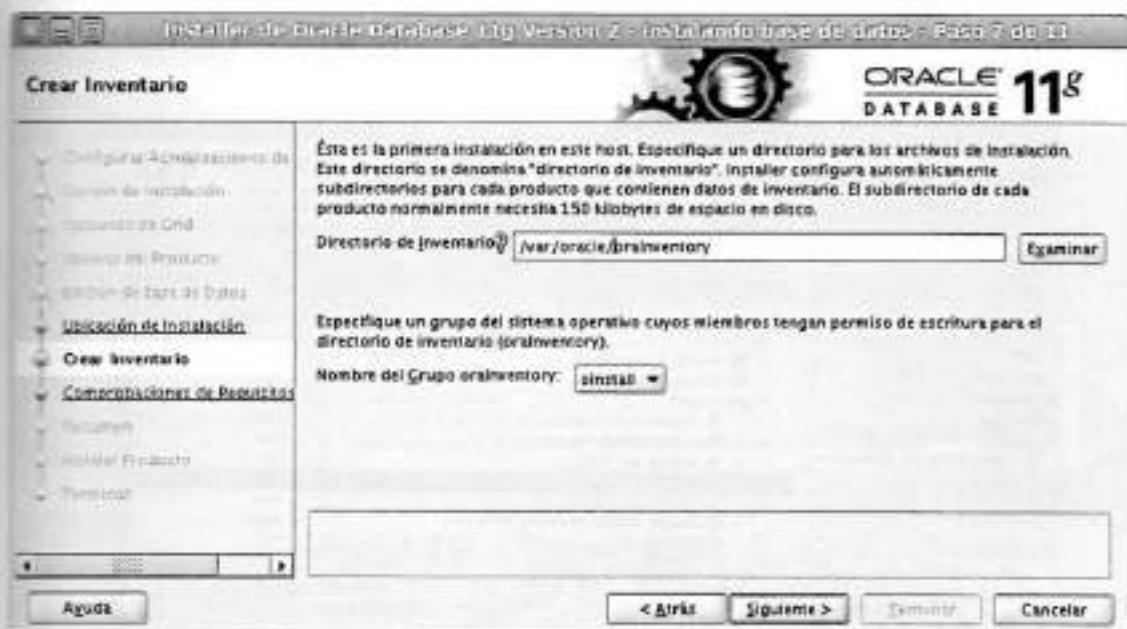


Figura 1.47: Instalación de oracle. Paso 7.

Paso 8: Asignar grupos dba y oinstall.

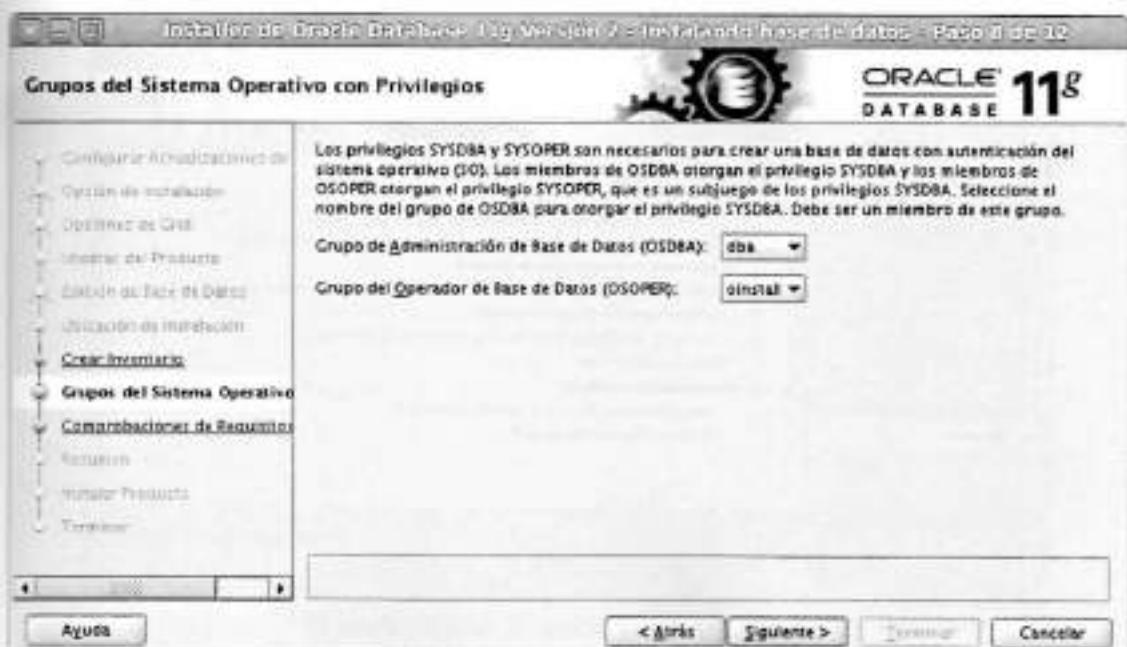


Figura 1.48: Instalación de oracle. Paso 8.

Pasos 9: Verificación de requisitos. Oracle indicará que hay ciertos requisitos que no se cumplen. En realidad, si que se cumplen, puesto que las versiones de los paquetes que se han instalado son más modernas y compatibles con los paquetes que exige oracle, por tanto, se selecciona la casilla *Ignorar Todo* para poder continuar.



Figura 1.49: Instalación de oracle. Paso 9.

Paso 10: Resumen de los parámetros de la instalación.

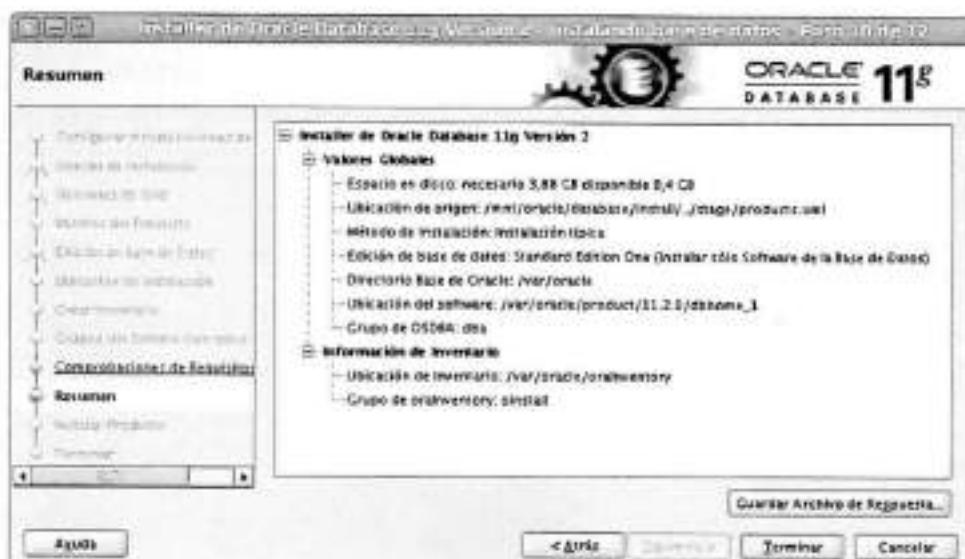


Figura 1.50: Instalación de oracle. Paso 10.

Paso 11: Instalación del software, puede durar varios minutos.



Figura 1.51: Instalación de oracle. Paso 11.

Paso 12: Fin de la instalación.



Figura 1.52: Instalación de oracle. Paso 12.

1.5.3. La postinstalación

Una vez terminada la instalación del software, Oracle nos pedirá la ejecución como root de dos shell scripts.

```
/var/oracle/oraInventory/orainstRoot.sh  
/var/oracle/product/11.2.0/dbhome_1/root.sh
```

Después, habrá que ejecutar el comando netca desde un terminal, *Net configuration Assistant*, para configurar un listener, es decir, un proceso servidor que escucha a través de los protocolos tcp-ip o ipc las peticiones de los procesos clientes. Se puede escoger, por ejemplo, vía TCP-IP a través del puerto estándar 1521.



Figura 1.53: Configuración de un listener en Oracle con netca.

Ahora, ya es posible configurar la instancia que contendrá la base de datos. Para hacerlo de forma automática, es posible ejecutar el comando dbca desde un terminal, *Database configuration Assistant*.

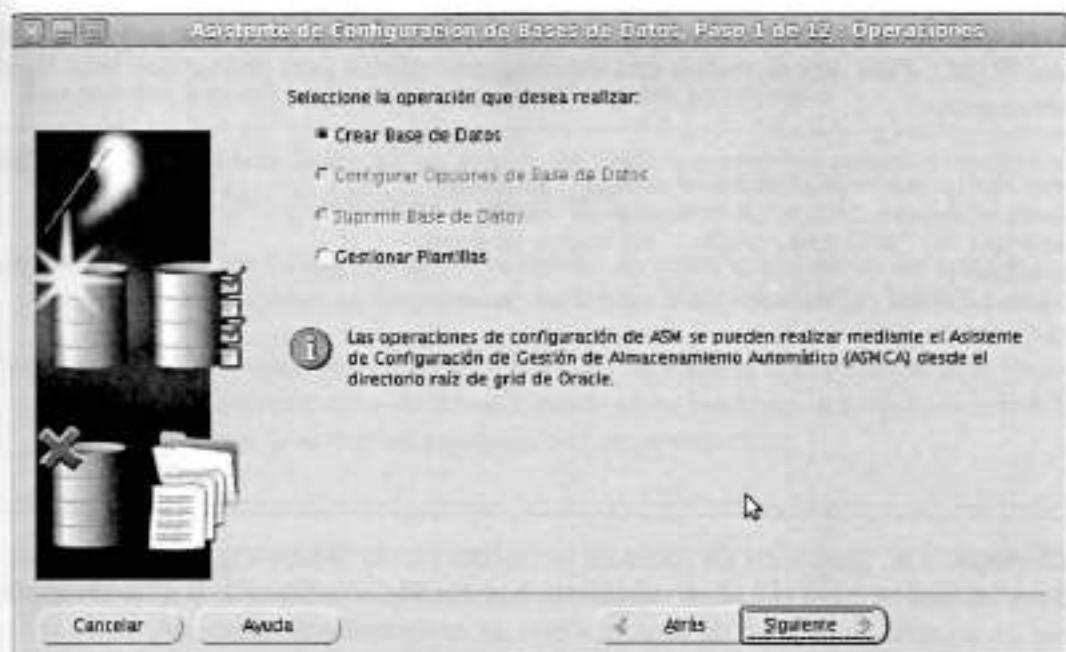


Figura 1.54: Configuración de la instancia con dbca.

Esta utilidad nos permite, además, crear el esquema para el repositorio de Enterprise Manager:

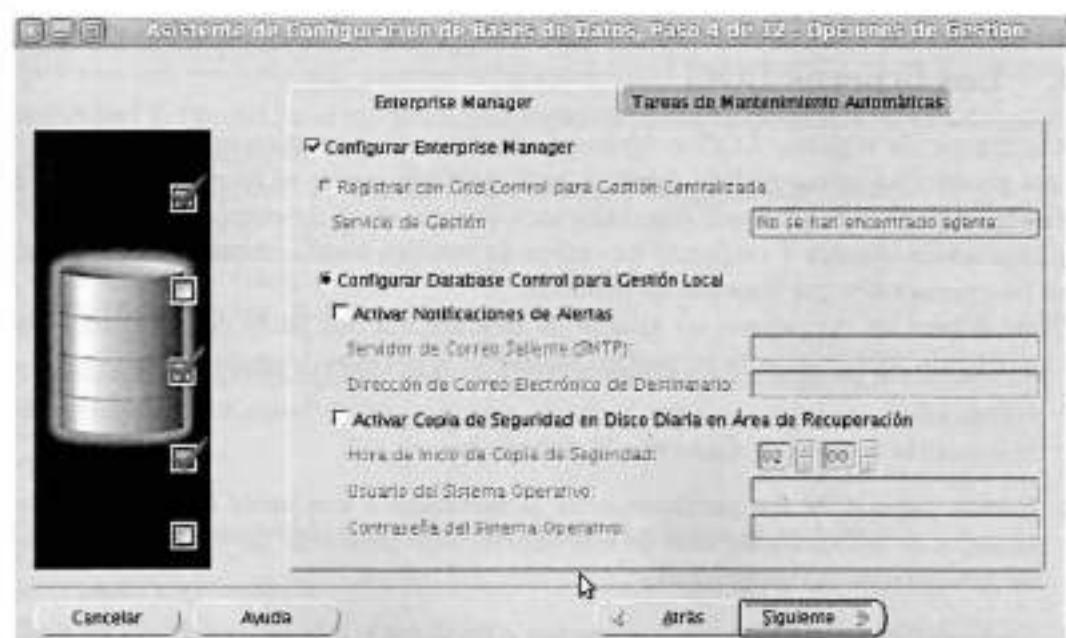


Figura 1.55: Configuración de Enterprise Manager con dbca.

Tras esperar unos minutos de ejecución del dbca, podemos comenzar a trabajar con nuestro SGBD. Para ello, se realiza una conexión con sqlplus para probar que todo ha ido correctamente:

```
oracle@unix:/mnt/oracle/database$ sqlplus / as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Lun Dic 6 15:14:35 2010
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Conectado a:
Oracle Database 11g Release 11.2.0.1.0 - Production
SQL> select * from dual;

D
-
X
```

- **Actividad 1.8:** Siguiendo los pasos de la instalación de Oracle 11g R2 descritos en el capítulo, instálalo en una máquina virtual con una distribución Ubuntu. Dota a la máquina virtual de un mínimo de 1GB de memoria para su correcto funcionamiento.

¿Sabías que ... ? Se puede consultar cuántas instancias hay de Oracle en un servidor Unix ejecutando el comando ps y filtrando por el proceso pmon (Oracle Process Monitor).

```
ps -ef|grep pmon
```

1.6. Los ficheros LOG

Un fichero de registro, LOG o bitácora, almacena información sobre los numerosos eventos producidos en un SGBD, tanto en su instalación, como en su explotación y administración. Así, estos ficheros son esenciales para conocer el comportamiento de un SGBD y ayudan al administrador a averiguar las causas de posibles malfuncionamientos y a auditar todas las operaciones que los clientes realizan.

Todo fichero de log supone un apunte de información por parte del programa que se esté ejecutando. Generalmente se pueden distinguir tres tipos de anotaciones:

1. Información de eventos, alertas y demás acciones que se lleven a cabo sobre o desde la instancia o bases de datos que la componen.
2. Estado general de los parámetros de la instancia o sus bases de datos, así como consejos de actuación en caso de que alguno sobrepase o no un determinado umbral de funcionamiento programado
3. Transacciones y operaciones que afecten a los datos o a la estructura de los mismos, es decir, sentencias DML y DDL respectivamente.

1.6.1. Ficheros Log en Oracle

Los ficheros logs más interesantes en Oracle son los siguientes:

Log de Alerta (Alert Log) Es un fichero de texto que registra errores y sucesos de la gestión de la base de datos cronológicamente.

Log de procesos en background Otro fichero de texto que registra los errores producidos por los procesos en background de Oracle (SMON, PMON, DBWR, etc.)

Ficheros LOG de usuarios (USER TRACE) A veces es necesario rastrear (trace) la actividad de determinados usuarios. Cuando estos reastreos se activan se crean ficheros de texto con la actividad generada por estos usuarios.

-
- **Actividad 1.9:** Consulta la ubicación de los archivos de LOG de tu instancia Oracle mediante las siguientes consultas:

```
select value from v$parameter where name = 'background_dump_dest';
select value from v$parameter where name = 'user_dump_dest';
```

Además de estos ficheros de texto, Oracle dispone de unos ficheros binarios llamados Redo Log que almacenan un histórico de todos los cambios efectuados en la base de datos. Si una BBDD se corrompe, el SGBD aplica todas las transacciones pendientes, usando la información existente en los redo logs. Estos redo logs también se pueden almacenar (archive redologs o archive logs) para implementar copias de seguridad en caliente y bases de datos en standby.

-
- **Actividad 1.10:** Activa el modo archive log en la instancia de tu BBDD Oracle mediante el siguiente procedimiento:

1. El archivo init\$ORACLE_SID.ora debe contener las siguientes líneas:

```
log_archive_start = true
log_archive_dest_1 = "location=/archivelog/bbdd REOPEN=5"
log_archive_format = arch_%t_%s.arc
```

2. A continuación, desde sqlplus, para la base de datos y arráncala en modo mount:

```
shutdown immediate;
startup nomount;
```

3. Cambia a modo archivelog, abre la base de datos e inicia el archivado:

```
alter database archivelog;
alter database open;
alter system archive log start;
```

1.6.2. Ficheros Log en DB2

La información del estado del servidor se localiza en el contexto de la instancia, y se ubica en el directorio guardado en el parámetro de la instancia llamado DIAGPATH:

```
db2 get dbm cfg|find DIAGPATH
```

El nombre del primero es db2diag.log, y el del segundo, nombre_instancia.nfy. Ambos ficheros son legibles por el administrador de base de datos. El db2diag.log constituye el punto de referencia de todo administrador para hacerse una idea de cómo está el sistema.

Cada BBDD tiene su juego independiente de ficheros de log binarios que almacenan los cambios en los datos. Estos pueden clasificarse a su vez en tres grupos: primarios, secundarios y archivados.

Los primarios y secundarios son ficheros de log *online*, esto es, deben encontrarse accesibles a la base de datos en todo momento, pues pueden guardar transacciones en curso. La diferencia entre primario y secundario es que los secundarios solo se usan si existe una transacción tan larga que los ficheros de log primarios no son capaces de contenerla. Esto es muy útil cuando la base de datos está configurada para tratar los ficheros de log de forma circular, lo cual significa que al terminar de escribir el último fichero *online* disponible, comenzará a sobreescribirse el primero, de modo que si hubiese una transacción muy larga que ocupase todos los ficheros de log, el gestor no tendría más remedio que arrojar un error para impedir la reutilización del fichero. Con la existencia de un número suficiente de ficheros secundarios, se evitaría ese error. Estos ficheros se pueden encontrar en el directorio que indique el parámetro de base de datos llamado LOGPATH:

```
db2 get db cfg for nombre_bd|find LOGPATH
```

Los *archivados* son ficheros de log que ya han sido usados y cerrados, y se copian con el objeto de formar parte de un backup. Configurando la base de datos para que genere ficheros de log archivados, y guardando estos en un lugar seguro, se posibilitan las recuperaciones de la base de datos hasta un punto determinado en el tiempo.

Los ficheros archivados se hallan donde apunte el parámetro LOGARCHMETH1:

```
db2 get db cfg for nombre_bd|find LOGARCHMETH1
```

1.6.3. Ficheros Log en MySQL

Los ficheros Log más interesante en MySQL son los siguientes:

Los registros de errores Muestran información sobre los sucesos anormales que se han producido a nivel de sistema en el servidor de base de datos, por ejemplo, que al arrancar el servidor no encuentre el fichero de configuración. La ubicación de este fichero se puede indicar mediante la opción `--log-file=nombre_fichero`. Si no se especifica un nombre, se escribe el fichero de errores en el directorio de datos con el nombre `nombre_servidor.err`. Además, en Windows todos estos errores y avisos se escriben en el registro de sucesos.

Los registros generales de consultas Almacenan todas las consultas solicitadas a la base de datos. Este modo por defecto no está activado puesto que activarlo repercutiría en el rendimiento del servidor de un modo muy negativo. Si se desea activar hay que añadir al fichero de configuración el parámetro `--log[=nombre_fichero]`.

Los registros de consultas 'lentas' Son aquellos que almacenan las consultas que pueden ser consideradas como pesadas en tiempo de ejecución. Este tipo de registro tampoco está activado y se puede configurar añadiendo los siguientes parámetros al fichero de configuración de MySQL:

```
--log-slow-queries[=file\_name] #nombre del fichero log  
long\query\time=segundos #segundos para considerarla sea lenta
```

Se puede examinar el registro de consultas lentas con la utilidad `mysqldumpslow`.

El Log binario Almacena cualquier modificación que se produzca en los datos, y sirve, por ejemplo, para replicar bases de datos o para restaurar una copia de seguridad del sistema en cualquier punto anterior. Para configurar el log binario, hay que incluir la opción `--log-bin[=nombre_fichero]`. MySQL generará ficheros de un tamaño igual al parámetro `max_binlog_size` y agregará una extensión numérica al nombre del registro binario para numerar los distintos ficheros binarios. Estos ficheros se pueden examinar con el comando `mysqlbinlog`.

◦ **Actividad 1.11:** Realiza los siguientes cambios en el fichero de configuración de MySQL para alterar la configuración de los logs:

1. Que el log de errores se guarde en `/home/mysql/errores_mysql.log`.
2. Que el log de consultas se guarde en `/home/mysql/querys.log`.
3. Que el log de consultas lentas se guarde en `/home/mysql/slow-querys.log`.
4. Las consultas lentas son aquellas que tardan más de 2 segundos.

Reinicia el servidor y genera consultas para comprobar que los cambios han surtido efecto. Puedes utilizar la consulta: `SELECT NOW(), SLEEP(3);` para generar una consulta lenta de más de 2 segundos.

1.7. Gestión del espacio de almacenamiento

Cada SGBD tiene su forma particular de gestionar físicamente la información de las tablas. Así, Oracle, DB2 y algunos motores de almacenamiento de MySQL como NDS organizan sus ficheros en espacios de tablas o *tablespaces*. Otros como SQL Server tienen grupos de ficheros o *filegroups*. Se llamen como se llamen, el objetivo es el mismo: agrupar ficheros de datos donde se almacena la información de una o varias tablas. Así, Oracle tiene en su instalación básica, entre otros, un tablespace llamado System donde se almacenan todas las tablas de gestión del propio SGBD y uno típico llamado USERS donde inicialmente se almacena la información de las tablas de los usuarios. El DBA puede variar la estructura del espacio de almacenamiento para adecuarlo a los requisitos del sistema.

Existen varios tipos de espacios de tablas:

1. Permanentes: Almacenan la información común que hay en las tablas.
2. De deshacer (UNDO): Guardan toda la información necesaria para deshacer transacciones al efectuar una operación *rollback* o recuperar una caída del sistema.
3. Temporales (TEMPORARY): Guardan los datos de intercambio en operaciones de ordenaciones, joins, agrupaciones, subconsultas, etc.

Cada espacio de tablas o grupo de ficheros suele estar formado por múltiples ficheros de datos o *datafiles*, donde cada uno puede estar ubicado en un dispositivo independiente, pudiendo, de esta manera, crear estructuras de datos tan grandes como sea necesario sin estar limitado por el tamaño máximo del sistema operativo. Es posible indicar si el fichero crecerá automáticamente hasta un límite establecido o se quedará con el tamaño asignado inicialmente.

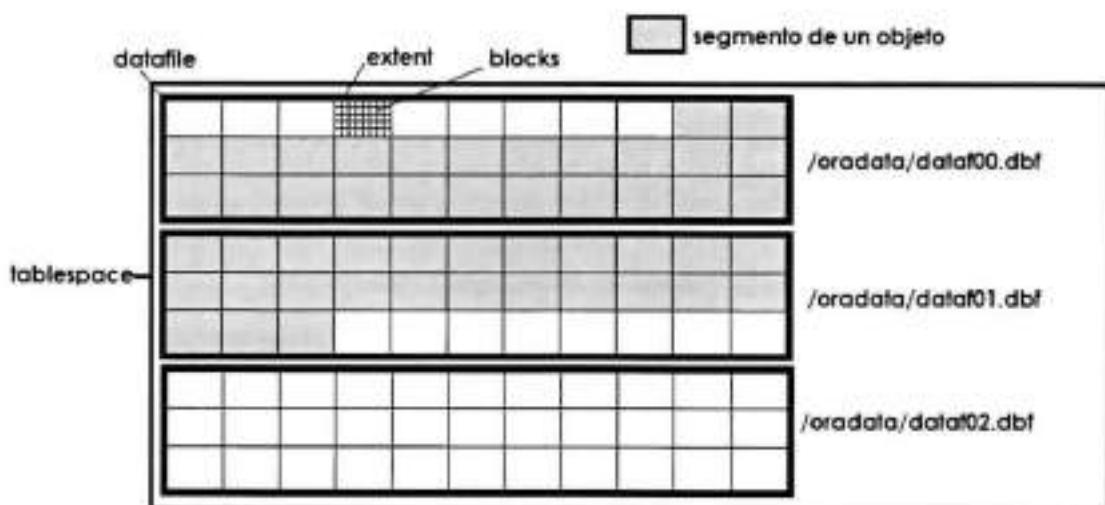


Figura 1.56: Organización del almacenamiento de un SGBD típico.

Al mismo tiempo, los espacios de tablas contienen segmentos. Cada segmento es la estructura donde se almacena un único objeto de la base de datos. Estos segmentos no tienen límite de tamaño, incluso pueden abarcar varios ficheros de datos. Cada segmento está dividido en extensiones o *extents*.

Las extensiones son trozos de segmento que se reservan en disco cuando el segmento se crea (initial) o cuando se agota el espacio en ese segmento y hay que ampliarlo (next). Cuando se crea el objeto, y por tanto su segmento, es posible especificar el tamaño de estos extents, tanto del *initial* como del *next*. Estas extensiones ocupan varios bloques de datos, cada uno de los cuales define la unidad mínima de almacenamiento. Esta unidad mínima de almacenamiento debe ser múltiplo de la unidad básica de lectura y escritura del disco y suele oscilar entre 4KB y 16KB.

Gracias a esta organización del espacio de almacenamiento se consiguen los siguientes objetivos:

1. Independencia del tamaño máximo de una tabla o estructura de datos de la capacidad del dispositivo.
2. Gracias a que los tablespaces forman una unidad lógica de almacenamiento, proporcionan independencia a cada una de las aplicaciones accediendo al SGBD.
3. Posibilidad de gestionar las cuotas de consumo de espacio de almacenamiento por parte de usuarios y aplicaciones.
4. Control de la disponibilidad de los datos. Se puede desactivar (tablespace offline) u activar (tablespace online) el acceso a determinados datos para poder realizar backups en caliente o conservar el sistema arrancado cuando ha ocurrido algún fallo en algún disco.
5. Posibilidad de encriptar los datos del tablespace con métodos estandar de encriptación como AES (Advanced Encryption Standard) o triple DES.

Toda la información sobre la formación y organización de los espacio de tablas están disponibles en el catálogo de metadatos, sin embargo, algunos gestores como Oracle permiten al DBA seleccionar que el SGBD maneje los espacios de tablas de forma local, es decir, que maneje los requerimientos de espacio para un *tablespace* por su cuenta eliminando la carga que supone para el diccionario de datos y mejorando de forma notable el rendimiento en el acceso a los datos. Sin embargo, esto implica que ciertas vistas del diccionario no se pueden usar, como la vista DBA_FREE_SPACE para comprobar el espacio no usado dentro del tablespace.

1.7.1. Sentencias SQL para la gestión de almacenamiento

A continuación, se muestran unas cuantas consultas muy útiles para manejar espacios de tablas en Oracle con las sentencias DDL CREATE, ALTER y DROP:

```

--Creación del tablespace Facturas con dos datafiles, uno autoextend y el otro no
--El tablespace debe estar manejado por el catálogo de metadatos.
CREATE TABLESPACE Facturas DATAFILE '/oradata/facturas01.dbf' SIZE 50M AUTOEXTEND OFF,
                           '/oradata/facturas02.dbf' SIZE 32M AUTOEXTEND ON NEXT 32M MAXSIZE 2048M;

--Creación de un TEMPORARY tablespace manejo de extensiones local
CREATE TEMPORARY TABLESPACE temporal1 TEMPFILE '/oradata/temporales/temp01.tmp'
SIZE 50M EXTENT MANAGEMENT LOCAL;

--Creación de un UNDO tablespace (si está creado reutilizar)
CREATE UNDO TABLESPACE deshacer01 DATAFILE '/oradata/undo/deshacer01.dbf'
SIZE 75M REUSE AUTOEXTEND ON;

--Creación de un usuario con default tablespace FACTURAS y quota de 100M en Facturas
CREATE USER francisco IDENTIFIED BY pacopwd01
DEFAULT TABLESPACE Facturas TEMPORARY TABLESPACE temporal1 QUOTA 100M ON Facturas;

--Creación de una tabla en el tablespace FACTURAS con initial y next extent
CREATE TABLE Liquidacion(
    CodigoLiquidacion INTEGER,
    DNI VARCHAR(10) REFERENCES Clientes(DNI),
    Cantidad NUMERIC(15,2),
    FECHA DATE
) TABLESPACE Facturas STORAGE (INITIAL 15K NEXT 5K MINEXTENTS 1 MAXEXTENTS UNLIMITED);

--Modificación del tablespace FACTURAS para agregar un datafile.
ALTER TABLESPACE Facturas ADD DATAFILE '/oradata/facturas03.dbf' SIZE 25M AUTOEXTEND OFF;

--Cambiar el tamaño a un datafile
ALTER DATABASE DATAFILE '/oradata/facturas03.dbf' RESIZE 50M;

--Tablesapce OFFLINE
ALTER TABLESPACE Facturas OFFLINE;

--Borrar tablespace Facturas
DROP TABLESPACE Facturas INCLUDING CONTENTS;

--Consultar espacio libre en los TABLESPACES
SELECT df.tablespace_name "Tablespace", df.bytes / (1024 * 1024) "Tamaño (MB)",
       SUM(fs.bytes) / (1024 * 1024) "Libre (MB)", Nvl(Round(SUM(fs.bytes) * 100/df.bytes),1) "% Libre",
       Round((df.bytes - SUM(fs.bytes)) * 100 / df.bytes) "% Utilizado"
  FROM dba_free_space fs,
       (SELECT tablespace_name,SUM(bytes) bytes FROM dba_data_files GROUP BY tablespace_name) df
 WHERE fs.tablespace_name (+) = df.tablespace_name
 GROUP BY df.tablespace_name,df.bytes
 UNION ALL
      SELECT df.tablespace_name tspace, fs.bytes/(1024*1024), SUM(df.bytes_free) / (1024 * 1024),
             Nvl(Round((SUM(fs.bytes)-df.bytes_used)*100/fs.bytes),1),
             Round((SUM(fs.bytes)-df.bytes_free)*100/fs.bytes)
        FROM dba_temp_files fs,
             (SELECT tablespace_name,bytes_free,bytes_used FROM v$temp_space_header
              GROUP BY tablespace_name,bytes_free,bytes_used) df
       WHERE fs.tablespace_name (+) = df.tablespace_name
      GROUP BY df.tablespace_name,fs.bytes,df.bytes_free,df.bytes_used ORDER BY 4 DESC;

```

1.8. Prácticas

Práctica 1.1: Instalación de MySQL en Windows

Conéctate a la página web <http://www.mysql.com/downloads/> y descárgate la última versión de MySQL Community Server para Windows e instálalo en tu ordenador.

Práctica 1.2: Instalación desde código fuente de mysql

En un ordenador con sistema operativo linux instala la versión 5 (cualquiera de ellas) del gestor de base de datos mysql, siguiendo los pasos:

1. Descarga el código fuente mysql*.tar.gz.
 2. Descomprime el código fuente.
 3. Prepara el entorno y compílalo con los parámetros adecuados, esto es, el directorio base en /usr/mysql5 y el directorio de datos en /usr/mysql5/data.
 4. Castellaniza los mensajes del servidor.
 5. Configura el servidor para que arranque automáticamente mysql 5 y se ejecute con el usuario mysql.
 6. Cambia los passwords de las cuentas anónimas y de root del servidor.
 7. Realiza pruebas de conectividad con el cliente mysql.
-

Práctica 1.3: Instalación de una segunda versión de mysql

Ahora instala en el ordenador de la práctica anterior con linux instala la versión 4 de MySQL Server (cualquiera de ellas), teniendo en cuenta que debes hacerla convivir con la versión instalada en la práctica anterior. Debes seguir los siguientes pasos:

1. Descarga el código fuente mysql*.tar.gz.
2. Descomprime el código fuente.
3. Prepara el entorno y compílalo con los parámetros adecuados, esto es, el directorio base en /usr/mysql4 y el directorio de datos en /usr/mysql4/data.
4. Para consolidar la compatibilidad en la ejecución de las dos versiones del gestor de base de datos a la vez, debes modificar varios parámetros, en concreto el número de

puerto al que se conecta el cliente (si has usado en mysql5 el 3306 por defecto, pon el 3307 en la versión 4), el pid_file y el socket que debe ser distinto que en la versión anterior.

5. Castellaniza los mensajes del servidor.
 6. Configura el servidor para que arranque automáticamente mysql 4 y se ejecute con el usuario mysql. A este le puedes llamar /etc/init.d/mysql4 y al enlace simbólico al directorio rc2.d también debes cambiarle el nombre.
 7. Cambia los passwords de las cuentas anónimas y de root del servidor.
 8. Realiza pruebas de conectividad con el cliente mysql, comprobando que puedes conectarte a ambas versiones.
-

Práctica 1.4: Instalación de Oracle en Windows

1. Conéctate a la página web <http://www.oracle.com> y descárgate la versión 11g de Oracle para Windows e instálalo en tu ordenador.
 2. Encuentra el Alert Log e inspecciónalo.
 3. Comprueba, mediante el visor de servicios de Windows, los servicios de Oracle que se han activado en tu sistema.
-

Práctica 1.5: Modificar sqlplus para tener histórico de comandos

Habráis comprobado que sqlplus para Linux no dispone de un histórico de comandos como el cliente de mysql. Para facilitar la vida al dba es posible *puentejar* la entrada de datos de sqlplus mediante la herramienta rlwrap. Instálala siguiendo los siguientes pasos:

1. Instala el comando rlwrap mediante el comando `sudo apt-get install rlwrap`.
 2. A continuación, modifica el fichero \$HOME/.bashrc del usuario oracle y añade la línea `alias sqlplus='rlwrap sqlplus'`.
 3. Conéctate con el usuario oracle a sqlplus y comprueba que si pulsas la tecla de cursor arriba tendrás disponible una lista de histórico de comandos.
-

Práctica 1.6: Instalación de DB2 express-C en Linux

Consulta la documentación de la página web de DB2 Express C para instalar este SGBD para alguna distribución de Linux.

Práctica 1.7: Gestión de almacenamiento

Ejecuta consultas SQL en Oracle para realizar las siguientes operaciones:

1. Crea un tablespace llamado PEDIDOS con tres datafiles, dos de ellos con autoextend OFF y uno de ellos que aumente automáticamente un 20 %. (PCTINCREASE). El tablespace debe estar manejado mediante el diccionario.
 2. Crea un nuevo tablespace llamado CONTABILIDAD con dos datafiles, uno con autoextend y otro sin autoextend manejado localmente por Oracle.
 3. Crea un tablespace temporal TEMP_PEDIDOS con un solo datafile de 50M manejado localmente por Oracle.
 4. Crea un usuario llamado Javier con quota ilimitada en el tablespace PEDIDOS cuyo tablespace temporal sea TEMP_PEDIDOS.
 5. Crea una tabla llamada PedidosPendientes con los campos que quieras, pero cuyo tablespace sea PEDIDOS y que su segmento tenga initial extent 20K y next 10K con un máximo de 100 extents.
 6. Crea un tablespace de UNDO con dos datafiles y los parámetros que deseas.
 7. Establece el estado del tablespace PEDIDOS OFFLINE.
 8. Borra el tablespace PEDIDOS y todo su contenido.
 9. Amplia en un datafile de 25M el tablespace de CONTABILIDAD.
 10. Cambia el tamaño del último datafile añadido al tablespace CONTABILIDAD estableciéndolo en 50M.
 11. Crea un procedimiento almacenado que muestre el estado de los tablespaces, sacando el espacio libre y usado tanto en MB como en porcentaje.
-

1.9. Amplía tus conocimientos

- Descárgate y prueba la versión de MySQL 7 para clusters.
- Intenta la configuración de una nueva instancia en Oracle de forma manual.
- Regístrate en foros oficiales, como los de MySQL u Oracle, <http://forums.oracle.com/>. Es una forma ideal de mantener tus conocimientos actualizados.
- Descárgate de <http://www.ibm.com/software/data/db2/express/> la última versión de DB2 express para linux, instálala y curiosea.
- Si tu instituto u organización tiene algún tipo de suscripción con Microsoft, prueba a instalar SQL Server.
- Prueba la instalación de otras alternativas a MySQL como PostgreSQL: <http://www.postgresql.org/download/>.
- Instala phpMyAdmin para tu servidor MySQL.
- Existen multitud de herramientas freeware que ayudan con la administración de los SGBD como TOAD, de Quest: <http://www.toadworld.com/Freeware/tabid/680/Default.aspx>.
- Hay multitud de recursos online en internet para mantenerte actualizado. Por ejemplo: <http://www.databasejournal.com/>. Una de las tareas más importantes de un DBA es estar al tanto de todas las novedades del mercado.
- Realiza siempre pruebas con varias configuraciones posibles cuando instales un SGBD. Te dará una visión más amplia de todas las opciones, y no te limitará a las instalaciones típicas.
- Si tienes la posibilidad, instala un par de servidores Oracle y prueba el Grid Control para manejarlos.
- Prueba el motor de almacenamiento NDS de MySQL. Experimenta con los tablespaces en MySQL.

ACCESO A LA INFORMACIÓN

Contenidos

- Creación y eliminación de vistas
- Creación, modificación y eliminación de usuarios
- Asignación y desasignación de derechos a usuarios
- Definición de roles. Asignación y desasignación de roles a usuarios
- Definición y utilización de perfiles
- Estructura del diccionario de datos

Objetivos

- Crear y borrar vistas personalizadas para cada tipo de usuario
- Proteger el acceso a la información de una base de datos a través de cuentas de usuario y roles
- Definir y eliminar cuentas de usuario asignando los privilegios adecuados sobre cada objeto
- Definir y eliminar roles de usuario asignando los privilegios adecuados sobre cada objeto.
- Definir y utilizar perfiles de usuario limitando a los usuarios el consumo de los recursos del sistema

La concurrencia en el acceso a la información por parte de múltiples usuarios motiva la necesidad de una protección frente a accesos indebidos y uso fraudulento de esa información compartida. Este capítulo introduce las técnicas más comunes para permitir y denegar el acceso a las partes más sensibles de una base de datos.

2.1. El acceso a la información

Cuando se administra la seguridad en el acceso a información de una base de datos, es común utilizar dos tipos de seguridad, la integrada con el sistema operativo y la proporcionada por el SGBD (nativa). En la seguridad integrada, se suele contar con los usuarios de un sistema de dominio o un servicio de directorio (LDAP) para proporcionar el acceso a determinados recursos del gestor de base de datos. En la seguridad nativa del SGBD, es el propio software servidor el que proporciona los mecanismos mediante los cuales se autoriza a un usuario a utilizar distintos elementos de bases de datos.

El alcance de este libro es tratar la seguridad nativa de un SGBD a través de SQL explicando el funcionamiento básico de la seguridad en MySQL, Oracle y DB2.

2.2. Las vistas

Una vista es una tabla sin contenido, totalmente virtual, que devuelve las filas resultado de ejecutar una consulta SQL. La diferencia con una consulta ejecutada directamente es que, mientras cada sentencia SQL enviada al SGBD tiene que pasar por un proceso de compilación, la vista es una consulta cuya definición ha sido almacenada previamente y que ya ha sido compilada, siendo por tanto el tiempo de ejecución bastante menor. También tiene una implicación importante en el hecho de que un usuario podría no tener acceso a la información de varias tablas y, sin embargo, sí tener acceso a la vista que consulta esas tablas, proporcionando de esta manera un acceso controlado solo a determinadas filas y columnas de esas tablas.



Figura 2.1: Vistas.

Por ejemplo, en una tabla de clientes, un usuario de una oficina de Madrid podría tener solo acceso a la información de los clientes de Madrid, y tan solo a ciertos campos. De esta manera, no tendría acceso a ningún campo de la tabla de clientes y, sin embargo, podría tener acceso a una vista que consulte aquellos clientes cuya provincia sea Madrid.

La sintaxis para crear una vista es la siguiente:

```
CREATE [OR REPLACE] VIEW [esquema.]nombre_vista [(lista_columnas)] AS
sentencia_select
```

La ejecución del CREATE VIEW provoca que se compile la sentencia select y que se almacene con el nombre nombre_vista. Los nombres de las columnas de la vista se pueden especificar mediante lista_columnas. Si se especifica la lista de columnas, cada columna tendrá el alias correspondiente, si no, se obtendrá el nombre devuelto por la consulta. Si la vista ya existe, se puede reemplazar con *OR REPLACE*.

Un ejemplo de CREATE VIEW es el siguiente:

```
CREATE VIEW nba.jugadoresMiami AS
SELECT Nombre, Posicion FROM nba.jugadores WHERE Nombre_equipo='HEAT';
SELECT * from nba.jugadoresMiami;
```

Además, se pueden crear vistas para que los usuarios no expertos puedan acceder de forma fácil a la información, proporcionándoles, a través de una vista, información obtenida a través de una sentencia SQL compleja:

```
CREATE VIEW VistaPedidos (CodigoPedido,Cliente,Total) AS
SELECT CodigoPedido, NombreCliente, SUM(Cantidad*PrecioUnidad)
FROM Clientes NATURAL JOIN Pedidos NATURAL JOIN DetallePedidos GROUP BY CodigoPedido;
```

Para eliminar una vista se hace uso del comando DROP VIEW:

```
DROP VIEW [esquema.]nombre_vista;
```

Hay pequeñas variaciones en la sintaxis de los comandos CREATE VIEW y DROP VIEW dependiendo del SGBD que se utilice. Además, se dispone también de un comando ALTER VIEW para hacer modificaciones a la definición de la vista. Para más información sobre estas variaciones, consultar los manuales de cada gestor.

¿Sabías que...? Una vista materializada es un tipo de vista que sí que contiene información. En todo momento el SGBD vuelca el resultado de la consulta en una *tabla caché* que existe de verdad (no es virtual) y que será actualizada de forma periódica con los resultados de la consulta SQL. Este tipo de tablas se utilizan en bases de datos multidimensionales y datawarehouse donde el acceso a tablas sencillas resulta muy costoso.

2.3. Los usuarios

Para crear cuentas de usuario que permitan a los usuarios acceder a ciertos objetos con un nivel determinado de privilegios hay que hacer uso del comando CREATE USER.

```
CREATE USER nombre_usuario IDENTIFIED BY 'password' [opciones];
```

Esta sencilla sentencia crea una cuenta usuario que permite la autentificación de un usuario en el SGBD a través de la password identificada mediante la opción IDENTIFIED BY. Esta sintaxis de create user es válida tanto para MySQL como para Oracle, aunque a Oracle se le pueden incluir multitud de opciones extras para dar características adicionales, como la asignación de cuota para añadir información a un tablespace o bloquear la cuenta temporalmente. En DB2 no se utiliza el comando CREATE USER puesto que la mayor parte de la gestión de usuarios se hace de forma integrada con el Sistema Operativo. A continuación se muestran ejemplos de creación de usuarios:

```
#creación de usuario en MySQL
CREATE USER paco IDENTIFIED BY 'o99238kjka';

--creación de usuario en Oracle
CREATE USER paco IDENTIFIED BY 'o99238kjka'
  DEFAULT TABLESPACE 'Nominas'
  QUOTA UNLIMITED ON 'Nominas'
  ACCOUNT LOCK;
```

- **Actividad 2.1:** Crea los usuarios Pedro y Javier en los SGBD mysql y Oracle con passwords apropiadas. En Oracle, asigna quota de 1MB en algún tablespace.

Para eliminar usuarios, se puede utilizar la sentencia DROP USER:

```
DROP USER nombre_usuario [CASCADE];
```

En Oracle, se puede incluir el token CASCADE para indicar que junto con el usuario se borren todos los objetos de su esquema. En MySQL, se pueden borrar a la vez varios usuarios, separando los nombres de los usuarios mediante comas.

Para modificar usuarios, Oracle utiliza el comando ALTER USER, que no está disponible en MySQL. Este comando ALTER USER usa las mismas opciones que CREATE USER.

```
--Modificación de usuario en Oracle
ALTER USER Paco IDENTIFIED BY 'nueva_pass'
  DEFAULT Tablespace 'Facturas';
```

En MySQL hay que modificar los usuarios actualizando sus datos en la propia tabla mysql.user del sistema. Por ejemplo, para cambiar el host desde el que el usuario Javier se puede conectar, habría que ejecutar la siguiente sentencia. Cuando en MySQL se modifica algún permiso modificando el contenido de las tablas del sistema, hay que ejecutar además el comando FLUSH PRIVILEGES para forzar al gestor a volver leer las tablas de permisos y que los cambios en los permisos sean efectivos desde ese momento.

```
#Modificación de usuario en mysql.
UPDATE mysql.user SET host='192.168.3.1' WHERE user='JAVIER';
FLUSH PRIVILEGES;
```

En MySQL también se puede renombrar un usuario conservando todos sus privilegios utilizando el comando RENAME USER y cambiar la password mediante el comando SET PASSWORD:

```
#Modificación de usuarios en MySQL. Cambiar nombre y password al usuario Paco
RENAME USER Paco@localhost TO PacoSanchez@localhost;
SET PASSWORD FOR Paco@localhost = PASSWORD('nueva_pass');
```

- **Actividad 2.2:** Ejecuta el siguiente comando para cambiar la password de un usuario en mysql:

```
UPDATE mysql.user SET Password = PASSWORD("nueva_pwd")
WHERE user='paco' AND host='localhost';
```

- **Actividad 2.3:** Cambia la password de los usuarios Pedro y Javier creados en Oracle y bloquea sus cuentas mediante la opción ACCOUNT UNLOCK.

2.4. Los privilegios

Un usuario puede obtener privilegios para manipular objetos de una base de datos con el comando GRANT. Asimismo, se le pueden denegar permisos con el comando REVOKE. Estos comandos varian en su sintaxis dependiendo del SGBD que se está usando puesto que el sistema de seguridad de cada uno es distinto. Además, influye también la forma en que el subsistema de permisos del SGBD se integra con el sistema operativo.

2.4.1. El sistema de privilegios de MySQL

La sintaxis del comando GRANT para MySQL es la siguiente:

```
GRANT tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
ON {nombre_tabla | * | *.* | base_datos.* | base_datos.nombre_tabla}
    TO usuario [IDENTIFIED BY [PASSWORD] 'password']
        [, usuario [IDENTIFIED BY [PASSWORD] 'password']] ...
    [WITH opcion [opcion] ...]

opcion =
    GRANT OPTION
    | MAX_QUERIES_PER_HOUR count
    | MAX_UPDATES_PER_HOUR count
    | MAX_CONNECTIONS_PER_HOUR count
    | MAX_USER_CONNECTIONS count
```

En MySQL se puede otorgar a un usuario permisos para hacer cualquier operación a nivel de host, de base de datos, de tabla o de columna. Así, es posible asignar, por ejemplo, permisos de SELECT sobre las columnas NombreCliente, Dirección y Telefono de la tabla clientes:

```
GRANT SELECT (NombreCliente, Telefono, Ciudad)
ON Clientes TO paco@localhost;
```

Con esta sentencia el usuario *paco@localhost* solo podrá seleccionar las columnas *NombreCliente*, *Telefono* y *Ciudad* de la tabla clientes, siéndole denegada una consulta del tipo *select * from clientes*.

tipo_privilegio es la clase de permiso que puede ser a otorgar, típicamente pueden ser *select*, *insert*, *update*, ... Pueden ser de los más variados. En el Cuadro 2.3 de la página 62 hay algunos ejemplos de los permisos disponibles en MySQL.

Estos tipos de privilegio se pueden aplicar a las siguientes expresiones:

Expresión	Se aplica el permiso a
nombre_tabla	La tabla nombre_tabla
*	Todas las tablas de la base de datos que se está usando
**	Todas las tablas de todas las bases de datos
base_datos.*	Todas las tablas de la base de datos db_name
base_datos.nombre_tabla	Solo la tabla nombre_tabla de la base de datos db_name

Cuadro 2.1: Tipos de objetos a los que se puede otorgar permisos.

Finalmente con 'TO usuario' se indica el usuario al que se quiere otorgar el permiso. Si el usuario user no existe, se crea, opcionalmente con la password indicada mediante la cláusula identified by.

Adicionalmente, se puede indicar ciertas opciones precedidas de la cláusula WITH:

Expresión	Función
GRANT OPTION	Permite conceder a otros usuarios los permisos que tiene el usuario, por tanto, el administrador debe ser muy cauto a la hora de conceder esta opción a los usuarios de la base de datos.
MAX_QUERIES_PER_HOUR count	Permite restringir el número de consultas por hora que puede realizar un usuario.
MAX_UPDATES_PER_HOUR count	Permite restringir el número de modificaciones por hora que puede realizar un usuario.
MAX_CONNECTIONS_PER_HOUR count	Permite restringir las conexiones (logins) por hora que realiza un usuario.
MAX_USER_CONNECTIONS count	Permite limitar el número de conexiones simultáneas que puede tener un usuario.

En cualquiera de las opciones MAX_ si a count se le da el valor 0, significa ilimitado.

Cuadro 2.2: Opciones adicionales a los permisos.

Privilegio	Significado
ALL [PRIVILEGES]	Da todos los permisos simples excepto GRANT OPTION
ALTER	Permite el uso de ALTER TABLE
ALTER ROUTINE	Modifica o borra rutinas almacenadas
CREATE	Permite el uso de CREATE TABLE
CREATE ROUTINE	Crea rutinas almacenadas
CREATE TEMPORARY TABLES	Permite el uso de CREATE TEMPORARY TABLE
CREATE USER	Permite el uso de CREATE USER, DROP USER, RENAME USER, y REVOKE.
CREATE VIEW	Permite el uso de CREATE VIEW
DELETE	Permite el uso de DELETE
DROP	Permite el uso de DROP TABLE
EXECUTE	Permite al usuario ejecutar rutinas almacenadas
FILE	Permite el uso de SELECT ... INTO OUTFILE y LOAD DATA INFILE
INDEX	Permite el uso de CREATE INDEX y DROP INDEX
INSERT	Permite el uso de INSERT
LOCK TABLES	Permite el uso de LOCK TABLES en tablas para las que tenga el permiso SELECT
PROCESS	Permite el uso de SHOW FULL PROCESSLIST
RELOAD	Permite el uso de FLUSH
REPLICATION CLIENT	Permite al usuario preguntar dónde están los servidores maestro o esclavo
REPLICATION SLAVE	Necesario para los esclavos de replicación
SELECT	Permite el uso de SELECT
SHOW DATABASES	SHOW DATABASES muestra todas las bases de datos
SHOW VIEW	Permite el uso de SHOW CREATE VIEW
SHUTDOWN	Permite el uso de mysqladmin shutdown
UPDATE	Permite el uso de UPDATE
USAGE	Sinónimo de 'no privileges', permite únicamente la conexión al gestor
GRANT OPTION	Permite dar permisos

Cuadro 2.3: Tipos de privilegios en mysql.

Algunos ejemplos de consultas para asignación de permisos en MySQL son los siguientes:

```
#Otorga permisos de select e insert a todas las tablas de nba
GRANT SELECT, INSERT on nba.* TO paco@localhost;

#Otorga todos los privilegios a la tabla Clientes de jardinería
GRANT ALL PRIVILEGES on jardineria.Clientes TO paco@localhost;

#Otorga permisos de select a todas las tablas de todas las bases de datos
#permitiendo al usuario ceder esos permisos a otros usuarios
GRANT SELECT on *.* to paco@localhost WITH GRANT OPTION;

#Otorga permisos de SELECT, INSERT, UPDATE y DELETE con un límite de 10
#consultas a la hora en la tabla jugadores de la nba
GRANT SELECT,INSERT,UPDATE,DELETE on nba.jugadores to paco@localhost
WITH MAX_QUERIES_PER_HOUR 10 MAX_UPDATES_PER_HOUR 10
```

La sentencia revoke deniega permisos a un usuario sobre un objeto. A continuación se describe la sintaxis:

```
REVOKE tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
ON {nombre_tabla | * | *.* | base_datos.* | base_datos.nombre_tabla}
FROM usuario [, usuario] ...
```

La sintaxis es muy parecida a la de la sentencia grant, a continuación se muestran unos cuantos ejemplos:

```
#quita el permiso de select en la tabla jardineria.Cliente
revoke select on jardineria.Clientes from paco@localhost;

#elimina el permiso ALL PRIVILEGES de todas las tablas
revoke all privileges on *.* from paco@localhost;

#quita los permisos de select e insert de todas las tablas de jardineria
revoke select,insert on jardineria.* from paco@localhost;
```

Otra forma de asignar y eliminar permisos en MySQL es utilizando las tablas del catálogo de metadatos, es decir, las tablas de la base de datos *mysql* creada en toda instalación del SGBD y que almacena toda la información sobre todos los objetos manejados por el gestor. En esta base de datos existen 5 tablas relacionadas con el sistema de permisos de MySQL, user, db, host, tables_priv y columns_priv. Estas tablas se pueden manipular manualmente con inserts y deletes para otorgar y denegar permisos a nivel de usuario, base de datos, equipo, tablas y columnas respectivamente.

A continuación se ilustra cómo MySQL procesa una consulta. Al recibir la instrucción SQL, se comprueba si el acceso a los diversos objetos están autorizados en cualquiera de estas tablas. Si ninguno de los niveles permite el acceso, la consulta es denegada por falta de permisos.

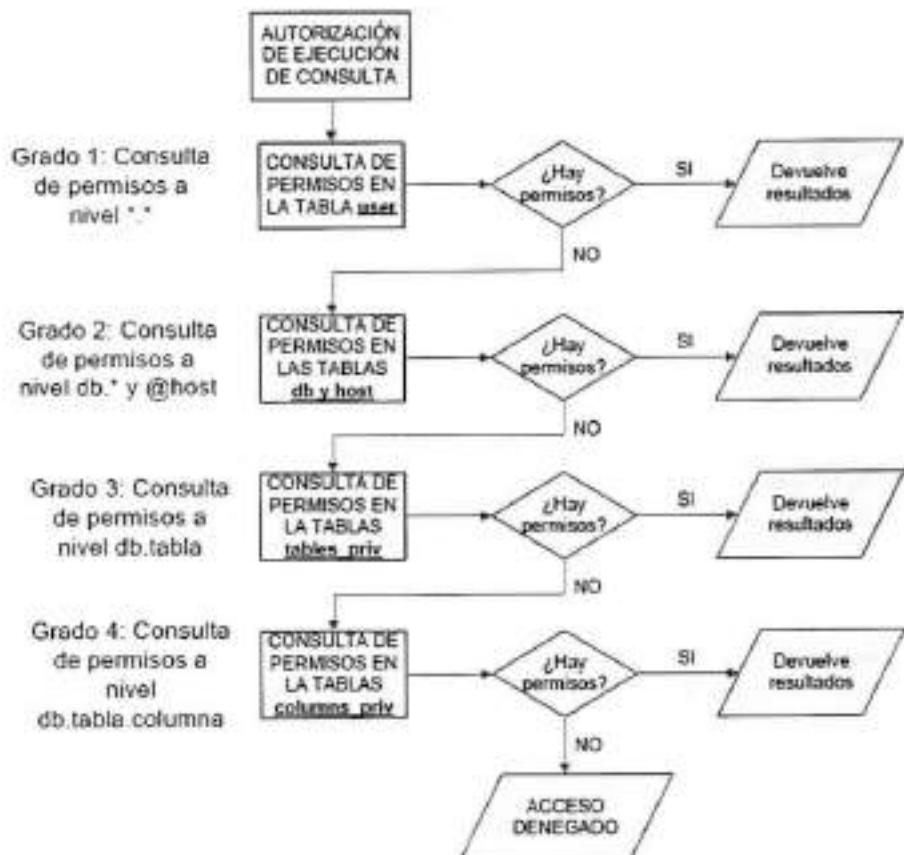


Figura 2.2: Estructura de permisos en mysql.

2.4.2. El sistema de privilegios de Oracle

El sistema de privilegios de Oracle clasifica los permisos en dos tipos, privilegios del sistema y privilegios de objetos. Además define los siguientes conceptos:

- *ROLES* Conjunto de privilegios que se pueden asignar a un determinado usuario. Un usuario puede pertenecer a múltiples roles.
- *PERFILES* Un perfil es un conjunto de restricciones sobre el uso de recursos. Cada usuario puede pertenecer a un único perfil.

De esta manera, Oracle permite asignar a un usuario privilegios del sistema, de objetos, roles y perfiles, todo ello a través de la sentencia GRANT:

```

GRANT privilegio [ON [esquema.]objeto] TO [usuario | rol | PUBLIC]
[WITH {GRANT | ADMIN} OPTION]

privilegio =
    tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
    | privilegio_sistema
    | rol

```

El privilegio a otorgar puede ser de varios tipos:

- Un privilegio sobre un objeto a un usuario o rol. Si el privilegio es INSERT o UPDATE se puede indicar sobre qué columnas se tiene el permiso. En este caso, se debe incluir la cláusula ON para indicar sobre qué objeto se va a aplicar el permiso.

```

-- permiso de select y update en dos columnas de la tabla Clientes
GRANT SELECT, UPDATE (NombreCliente, Telefono) ON Jardineria.Clientes To Javier;
-- todos los permisos sobre Pedidos al rol ADMINISTRACION
GRANT ALL PRIVILEGES ON Jardineria.Pedidos to ADMINISTRACION;

```

- Un privilegio de sistema sobre un usuario o rol.

```

--permiso para crear tablas
GRANT CREATE TABLE to Javier;
--permiso para crear tablas al rol ADMINISTRACION
GRANT CREATE TABLE to ADMINISTRACION;

```

- Un rol sobre un usuario u otro rol. El rol puede estar predefinido (rol de sistema) o puede ser definido por el usuario.

```

--asignar Rol de acceso y consumo a otro Rol
GRANT CONNECT, RESOURCE to ADMINISTRACION;
--asignar el rol ADMINISTRACION a Javier
GRANT ADMINISTRACION to Javier;

```

- Un privilegio de objeto o sistema a todo el mundo (PUBLIC).

```

--permiso de acceso y consumo de recurso a todos
GRANT CONNECT, RESOURCE to PUBLIC;
--permiso de update sobre las columnas Nombre y email de la tabla empleados
GRANT UPDATE (Nombre, email) ON Jardineria.Empleados to Public;

```

Privilegio	Significado
ALL [PRIVILEGES]	Da todos los permisos simples incluido GRANT OPTION
DELETE	Permite el uso de DELETE
INSERT	Permite el uso de INSERT
SELECT	Permite el uso de SELECT
UPDATE	Permite el uso de UPDATE
REFERENCES	Permite crear claves foráneas
EXECUTE	Permite ejecutar un procedimiento, función, paquete...
ALTER	Permite cambiar la definición de un objeto
INDEX	Permite crear índices de una tabla

Cuadro 2.4: Tipos de privilegios de objeto en Oracle.

Privilegio	Significado
ALTER DATABASE	Permite ejecutar el comando ALTER DATABASE
ALTER SYSTEM	Permite el uso de ALTER SYSTEM
AUDIT SYSTEM	Permite la auditoría mediante la sentencia AUDIT
CREATE [ANY] objeto	Permite el uso de CREATE para un tipo de objeto (TABLE, VIEW, PROCEDURE, USER, PROFILE, etc...)
ALTER [ANY] objeto	Permite el uso de ALTER para un tipo de objeto (TABLE, VIEW, PROCEDURE, USER, PROFILE, etc...)
SYSDBA	Permite ejecutar operaciones de parada y puesta en marcha de la BBDD
ANALYZE [ANY] objeto	Permite analizar un tipo de objeto

Cuadro 2.5: Tipos de privilegios de sistema en Oracle.

Las cláusulas WITH GRANT OPTION o WITH ADMIN OPTION transmiten la autorización de conceder ese privilegio de objeto o de sistema a otro usuario o rol. Para denegar privilegios sobre un objeto Oracle utiliza la sentencia REVOKE

```

REVOKE privilegio
[ON [esquema.]objeto]
FROM [usuario | rol | PUBLIC]

privilegio =
    tipo_privilegio [(columnas)] [, tipo_privilegio [(columnas)]] ...
    | privilegio_sistema
    | rol
  
```

Algunos ejemplos de esta sentencia son los siguientes:

```

--Denegar permiso de select en Clientes al rol ADMINISTRACION
REVOKE SELECT ON Jardineria.Clientes FROM ADMINISTRACION;
--Denegar rol CONNECT a todos
REVOKE CONNECT FROM PUBLIC;
--Denegar los roles ALMACEN Y CONTABILIDAD A Javier;
  
```

```

REVOKE ALMACEN,CONTABILIDAD FROM Javier;
--Denegar todos los privilegios a Javier en la tabla Pedidos
REVOKE ALL PRIVILEGES ON Pedidos FROM Javier;

```

Cuando se elimina un privilegio de un usuario, el SGBD elimina el permiso de ese usuario de forma inmediata a él y a todos los usuarios a los que se les haya concedido el privilegio con la opción WITH GRANT o WITH ADMIN.

Los roles

Para simplificar la tarea de organizar los permisos, Oracle permite la creación de Roles con la sentencia CREATE ROLE:

```

CREATE ROLE nombre_rol [opciones_identificación];
opciones_identificación:
  NOT IDENTIFIED
  | IDENTIFIED ( BY password | EXTERNALLY | GLOBALLY | USING [esquema].package)

```

Al crear un rol, se puede especificar el tipo de autentificación. Si se especifica NOT IDENTIFIED, el rol está autorizado por la base de datos y no es necesaria ninguna contraseña para activarlo, sin embargo si se especifica IDENTIFIED, el usuario debe ser autorizado antes de activar el rol. Dentro de IDENTIFIED existen varias posibilidades:

- BY password, donde el usuario debe indicar la contraseña antes de activar el papel.
- EXTERNALLY, para ser autorizado mediante un servicio externo (como el sistema operativo).
- GLOBALLY, que cede la autorización al servicio Oracle Security Service.
- Si se elige USING, se elige crear un rol de aplicación segura, es decir, un rol que solo puede ser activado por las aplicaciones que usen el paquete.

Cuando un usuario se conecta al SGBD, este otorga al usuario los permisos que se le hayan concedido explícitamente mediante la orden GRANT y los permisos que tengan los roles por defecto del usuario. Para asignar un rol por defecto a un usuario se utiliza la sentencia ALTER USER de la siguiente forma:

```

ALTER USER usuario DEFAULT ROLE [opciones_rol];
opciones_rol:
  rol [, rol] ...
  | ALL [EXCEPT rol]
  | NONE

```

Se puede anular el rol por defecto mediante la opción NONE, o habilitar todos los roles mediante ALL, o todos menos uno mediante ALL EXCEPT rol, o conceder varios separados por coma. Por ejemplo, para asignar los roles ADMINISTRACION y CONTABILIDAD al usuario Javier:

```
GRANT Contabilidad,Administracion to Javier;
ALTER USER Javier DEFAULT ROLE ADMINISTRACION,CONTABILIDAD;
```

Finalmente, se puede utilizar también la sentencia SET ROLE para dotar momentáneamente de ciertos privilegios a un usuario que tenga autorizado su uso. Por ejemplo, se puede crear el rol ALMACEN, con todos los privilegios sobre la tabla Productos y asignarselo al usuario Javier. Si el usuario Javier no activa el rol ALMACEN mediante SET ROLE y su contraseña, no podrá disfrutar de sus privilegios

```
CREATE ROLE ALMACEN IDENTIFIED BY AlmacenPwd;
GRANT ALL Privileges ON Jardineria.Productos TO ALMACEN;
GRANT ALMACEN to Javier;

--conexión con el usuario Javier
SQL> select count(*) from Jardineria.productos;
select count(*) from Jardineria.productos
*
ERROR en linea 1:
ORA-00942: la tabla o vista no existe

SQL> SET ROLE ALMACEN identified by AlmacenPwd;
Rol definido.

SQL> select count(*) from Jardineria.productos;
COUNT(*)
-----
276
```

Creación de perfiles

Los perfiles se usan para limitar la cantidad de recursos del sistema o de la base de datos disponibles para un usuario. Existe un perfil llamado DEFAULT que concede recursos ilimitados a todos los usuarios. Para evitar esto, se puede hacer uso de la sentencia CREATE PROFILE para crear nuevos perfiles.

```
CREATE PROFILE perfil LIMIT recursos ...
recursos =
    recurso { cantidad | UNLIMITED | DEFAULT }
    | password
```

```

recurso =
  SESSIONS_PER_USER
  | CPU_PER_SESSION
  | CPU_PER_CALL
  | CONNECT_TIME
  | IDLE_TIME
  | LOGICAL_READS_PER_SESSION
  | LOGICAL_READS_PER_CALL
  | COMPOSITE_LIMIT
  | PRIVATE_SGA
password =
  parametro { cantidad | UNLIMITED | DEFAULT }
  | PASSWORD_VERIFY_FUNCTION { Función | NULL | DEFAULT }
parametro =
  FAILED_LOGIN_ATTEMPTS
  | PASSWORD_LIFE_TIME
  | PASSWORD_REUSE_TIME
  | PASSWORD_REUSE_MAX
  | PASSWORD_LOCK_TIME
  | PASSWORD_GRACE_TIME

```

Los diferentes límites que se pueden utilizar están definidos en la siguiente tabla:

Limite	Función
SESSIONS_PER_USER	Número máximo de sesiones abiertas a la vez por un usuario
CPU_PER_SESSION	Tiempo límite para una sesión expresado en centésimas de segundos
CPU_PER_CALL	Tiempo límite de respuesta para una llamada (parse, fetch o ejecución) en centésimas de segundos
CONNECT_TIME	Tiempo límite total de una sesión, expresado en minutos
IDLE_TIME	Tiempo máximo de inactividad de una sesión expresados en minutos
LOGICAL_READS_PER_SESSION	Número máximo de bloques de datos leídos en una sesión
LOGICAL_READS_PER_CALL	Número máximo de bloques de datos leídos en una llamada
PRIVATE_SGA	Tamaño de memoria que una sesión puede reservar del espacio privado de memoria del SGA
COMPOSITE_LIMIT	Coste total en recursos de una sesión, expresado mediante el promedio de ciertos parámetros

Cuadro 2.6: Tipos de recursos.

Por ejemplo, se puede crear el perfil OPERADOR con los siguientes límites:

```
CREATE PROFILE OPERADOR LIMIT
  SESSIONS_PER_USER 5      -- 5 sesiones concurrentes
  CPU_PER_SESSION UNLIMITED -- Tiempo de CPU ilimitado
  IDLE_TIME 10             -- 10 minutos de inactividad
  CONNECT_TIME 120          -- 2 horas totales de conexión
;
```

También es posible alterar un perfil mediante la sentencia ALTER PROFILE:

```
-- modificar el perfil DEFAULT para que la password no caduque
alter profile DEFAULT LIMIT PASSWORD\_LIFE\TIME Unlimited;

-- modificar el perfil OPERADOR para que su límite de sesión sea de 1 hora
-- y un máximo de 10 sesiones concurrentes
alter profile OPERADOR LIMIT CONNECT_TIME 60 SESSIONS_PER_USER 10;
```

Un perfil se puede asignar en el momento de crear o modificar el usuario, por ejemplo:

```
-- creación de un usuario con el perfil OPERADOR
CREATE USER Francisco IDENTIFIED BY FrancisPwd
  QUOTA UNLIMITED ON Users
  PROFILE OPERADOR;
```

2.4.3. El sistema de privilegios de DB2

Desde el punto de vista de la seguridad, se puede decir que en DB2 existen tres grandes puntos de gestión de los privilegios de acceso:

1. Autenticación
2. Autorización
3. Privilegio

La autenticación busca responder a las siguientes preguntas: ¿quién puede acceder a la instancia o base de datos?, ¿dónde se verificará la password de usuario? En DB2 los usuarios de autenticación deben estar dados de alta en el sistema operativo, es decir, que para resolver la primera pregunta, bastaría con decir que solo los usuarios del sistema pueden acceder a la instancia o base de datos, pero jojo!, los usuarios pueden serlo del sistema local o de un sistema cliente remoto que esté intentando la conexión, esto sería así en función de cómo se defina la variada política de autenticación que dará respuesta a la segunda pregunta. La política de autenticación no solo permite o no la conexión desde clientes, sino que, entre otras cosas, indica si habrá encriptación o no del usuario y password, e incluso de los datos intercambiados durante la comunicación con el servidor.

La autorización hace referencia al nivel de permisos con que un usuario materializa el acceso a la instancia o base de datos. También determina los comandos que el usuario puede ejecutar, los datos que puede leer o modificar, o los tipos de objetos de base de datos que se le permite crear, modificar o borrar. Todas estas consideraciones están preprogramadas en siete grupos de autorizaciones que pueden ser concedidos, en función del caso, a un grupo de usuarios o a un usuario determinado.

- SYSADM: es el único al que se le permite modificar los parámetros de la instancia. Comparable a la autoridad del usuario *root* en Unix, puede ejecutar cualquier comando contra la instancia, contra todas las bases de datos de la instancia y contra todos los objetos de cualquier base de datos.
- SYSCTRL: puede hacer todas las actividades de administración (crear y borrar bases de datos y tablespaces) y mantenimiento (hacer backups, pasar estadísticas, actualizar parámetros de la base de datos), pero no podrá acceder a ningún dato de las bases de datos a no ser que le sean concedidos explícitamente los privilegios adecuados.
- SYSMAINT: solo puede ejecutar labores de mantenimiento (hacer backups, pasar estadísticas, actualizar parámetros de la base de datos).
- SYSMON: pueden obtener fotos estáticas de monitorización (en inglés *database system monitor snapshots*) de la instancia o sus bases de datos.
- DBADM: solo existe en el contexto de una base de datos concreta. Le está permitido crear y borrar tablas, ejecutar estadísticas, y conceder o quitar privilegios, pero en ningún caso podrá borrar la base de datos, ni crear tablespaces, ni hacer backups, o modificar parámetros de configuración.
- LOAD: permite ejecutar cargas de datos sobre las tablas y pasar estadísticas. Es un permiso a nivel de usuario.
- SECADM: es la autorización encargada de gestionar toda la implementación de las *LBAC* (del inglés *label-based access control=LBAC*), esto es el control de acceso basado en etiquetas.

Los privilegios propiamente dichos pueden ser a nivel de toda la base de datos o solo estar referidos a un objeto concreto. En esencia son prácticamente iguales que los permisos de Oracle, se conceden con el comando *GRANT* y se revocan con el *REVOKE*. Van desde el simple *CONNECT*, pasando por los DML (*SELECT, INSERT, DELETE, UPDATE*), hasta los típicos *ALTER, DROP...*

Uno de los permisos más característicos que existen en DB2 es el de *BIND*, que sirve para compilar los paquetes de código de programación estático que tradicionalmente componen este gestor.

Otro juego de privilegios muy peculiares en DB2 son los referentes al *LBAC*. Están disponibles desde la versión 9 y permiten restringir las lecturas y escrituras en una tabla por nivel de fila o columna! Su implementación si bien es cierto que tiene bastante complejidad, no es menos cierto que dotan al gestor de una potencia hasta ahora desconocida, por ejemplo, podría limitarse la consulta a la tabla de nóminas de una empresa, de modo que cada empleado solo pudiera consultar sus propios datos.

2.5. El catálogo de metadatos

El *catálogo de metadatos* o *diccionario de datos* es una de las partes más importantes de un SGBD. Proporciona información al DBA y al propio sistema de todas las estructuras de información almacenadas en una base de datos. La existencia del diccionario de datos está definida por la regla 4 de Codd, que expresa la necesidad de almacenar la descripción de la estructura de la base de datos de la misma forma que los datos normales almacenados en tablas. De esta manera, los usuarios podrán consultar el catálogo de metadatos para conocer, por ejemplo, sus privilegios, el estado de consumo de los recursos que tienen asignados o la estructura de las tablas y bases de datos a las que pueden acceder.

Cada SGBD organiza su propio catálogo con diferentes estructuras. Así por ejemplo, MySQL dispone de una base de datos especial *information_schema* donde se almacena la definición de las tablas, vistas, índices, estado del SGBD, etc. y otra llamada *mysql* donde almacena los usuarios y privilegios. Por su parte, Oracle almacena toda esta información a través de tablas pertenecientes al esquema del usuario SYS. DB2, en cambio, utiliza los esquemas SYSCAT, SYSSTAT y SYSIBMADM.

2.5.1. El catálogo de MySQL

Aparte de las tablas de la base de datos *mysql*, mostradas en la figura 2.2 de la página 64, se exponen a continuación algunas de las tablas más útiles del catálogo de metadatos de MySQL:

Tabla	Función
SCHEMATA	Bases de datos disponibles en el SGBD
TABLES	Tablas de todas las BBDD
COLUMNS	Columnas de todas las tablas
VIEWS	Vistas disponibles en todas las BBDD
SESSION_STATUS	Estado de la sesión actual
GLOBAL_STATUS	Estado del SGBD
SESSION_VARIABLES	Parámetros de configuración de la sesión
GLOBAL_VARIABLES	Parámetros de configuración del sistema
USER_PRIVILEGES	Privilegios de los usuario

Cuadro 2.7: Tablas de la BBDD *information_schema*.

Consultando estas tablas con sentencias *select* se puede obtener información muy útil:

```
#Consultar las tablas que contienen más de 2000 filas
SELECT table_name, table_schema
FROM information_schema.tables
WHERE table_rows>2000 AND
    table_schema='nba';
+-----+-----+
| table_name | table_schema |
+-----+-----+
| estadisticas | nba      |
| partidos     | nba      |
+-----+-----+

#Obtener los privilegios de los que dispone el usuario actual
#sobre los campos de tipo date de la BBDD jardineria

SELECT table_schema,table_name,column_name,privileges
FROM information_schema.columns
WHERE data_type='date' AND
    table_Schema='jardineria';

+-----+-----+-----+-----+
| table_schema | table_name | column_name | privileges   |
+-----+-----+-----+-----+
| jardineria  | Pagos      | FechaPago   | select,insert,update,references |
| jardineria  | Pedidos    | FechaPedido | select,insert,update,references |
| jardineria  | Pedidos    | FechaEsperada | select,insert,update,references |
| jardineria  | Pedidos    | FechaEntrega | select,insert,update,references |
+-----+-----+-----+-----+
```

Mediante el comando *show* de mysql, también es posible obtener información del catálogo de metadatos sin necesidad de realizar consultas tan complicadas. Por ejemplo, estas dos consultas para obtener el tiempo que lleva el SGBD activo (*UPTIME*) son equivalentes:

```
select * from session_status where variable_name='UPTIME';
show status like 'UPTIME';
```

2.5.2. El catálogo de Oracle

Oracle también utiliza vistas en su catálogo de metadatos. Para ello utiliza las vistas que comienzan por los prefijos user_*, all_ y dba_*. Las tablas que comienzan por user_* muestran información de los objetos que pertenecen al usuario. Por ejemplo, la vista user_tables muestra información de todas las tablas que pertenecen al usuario que inició la sesión. Si se ha iniciado sesión con el usuario jardineria y se consulta la tabla user_tables el resultado será el siguiente:

```
SQL> SELECT table_name FROM user_tables;

TABLE_NAME
-----
PAGOS
EMPLEADOS
GAMASPRODUCTOS
OFICINAS
CLIENTES
PEDIDOS
PRODUCTOS
DETALLEPEDIDOS

8 filas seleccionadas.
```

Las vistas con el prefijo all_* muestran la información de los objetos a los que se tiene acceso. Así, si se ha iniciado sesión con el usuario jardineria y se consulta la tabla all_tables se mostrarán, además de las tablas del esquema del usuario, aquellas a las que tenga algún tipo de conciencia.

```
SELECT owner,table_name FROM all_tables;
```

El administrador de la base de datos tiene acceso a las vistas del catálogo con prefijo dba_*. Estas vistas muestran todos los objetos de la base de datos.

Existen cerca de 1800 vistas distintas entre vistas user,all y dba. Para más información visitar la documentación online de Oracle: <http://www.oracle.com/pls/db111/homepage>

También utiliza las vistas v\$ que proporcionan estadísticas en tiempo real de la utilización del SGBD. Por ejemplo, se puede utilizar la vista v\$session para conocer las sesiones abiertas en el SGBD y terminarlas:

```
SELECT sid,serial#,status FROM v$session
  WHERE username = 'JUAN';
```

SID	SERIAL#	STATUS
7	14	ACTIVE
8	66	INACTIVE

2 rows selected.

```
ALTER SYSTEM KILL SESSION '7,14';
ALTER SYSTEM KILL SESSION '8,66';
```

2.5.3. El catálogo de DB2

El gestor de bases de datos DB2 crea y mantiene dos conjuntos de vistas de catálogo del sistema:

- Las vistas SYSCAT son vistas de catálogo de solo lectura que se encuentran en el esquema SYSCAT.
- Las vistas SYSSTAT son vistas de catálogo actualizables que se encuentran en el esquema SYSSTAT. Las vistas actualizables contienen información estadística utilizada por el optimizador.

Todas las vistas de catálogo del sistema se dan de alta durante la creación de la base de datos, y recogen toda la información sobre tablas (SYSCAT.TABLES), índices (SYSCAT.INDEXES), disparadores (SYSCAT.TRIGGERS)... y en general, sobre todos los objetos de la base de datos.

Para conocer de un vistazo rápido la totalidad de las vistas del catálogo y su uso, accédase al Centro de Información en línea de DB2, que puede encontrarse, por ejemplo, para la versión 9.7, en la url <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.sql.ref.doc/doc/r0011297.html>.

En DB2, además de las vistas de catálogo propiamente dichas, existen lo que se llaman vistas administrativas, que sirven para obtener de manera sencilla un sinfín de información utilísima en el día a día de la administración de las instancias y bases de datos: últimos backups realizados, ocupación de los tablespaces, sentencias que llevan más tiempo en ejecución, sentencias que se usan más frecuentemente, bloqueos, hitratio... Todas estas vistas administrativas pertenecen al esquema SYSIBMADM.

Por ejemplo, para obtener el porcentaje de utilización de los tablespaces, se podría hacer uso de la vista *SYSIBMADM.TBSP_UTILIZATION*:

```
select tbsp_id,
       substr(tbsp_name,1,20) as nombre_tablespace,
       tbsp_utilization_percent as porcentaje_utilizacion
  from SYSIBMADM.TBSP_UTILIZATION
 order by tbsp_utilization_percent
```

Para averiguar los últimos backups realizados sobre la base de datos y los logs necesarios para restaurarlos, la vista indicada sería *SYSIBMADM.DB_HISTORY*:

```
select backup_id, start_time, end_time, firstlog, lastlog
  from SYSIBMADM.DB_HISTORY
 order by start_time desc
```

Otra forma de acceder a información valiosa ya procesada en la base de datos, es mediante la utilización de las *funciones de tabla para snapshot* (*snapshot*=fotos estáticas sobre el estado de las bases de datos), las cuales son invocadas a través de una sentencia *select* desde su cláusula *from*, por ejemplo:

```
select snapshot_timestamp,
       substr(db_path,1,10) as directorio_de_bd,
       db_status
  from TABLE(SNAPSHOT_DATABASE('xxx_nombre_bd', -1)) as snap_bd
```

2.6. Prácticas

Práctica 2.1: MySQL: Recuperar la contraseña de root

Si has perdido la password de root y necesitas cambiarla realiza el siguiente procedimiento:

1. Para el servidor MySQL y arráncalo de nuevo manualmente con la opción `--skip-grant-tables` para eliminar la comprobación de usuarios.
2. Conéctate al servidor con el comando `'mysql -u root'` y cambia la password del root.
3. Para el servidor y arráncalo con el script `/etc/init.d/mysql`.
4. Comprueba que efectivamente ahora puedes entrar con la nueva password.

Práctica 2.2: MySQL: Crear usuarios y asignar permisos en local

Realiza las siguientes operaciones y almacena los comandos y los resultados en un fichero:

1. Crea un usuario llamado `paco@localhost` con la sintaxis `create user` con permisos de solo conexión y comprueba que se pueda conectar.
2. Crea un usuario llamado `juan@localhost` con la sintaxis `grant` con permisos de solo conexión y comprueba que se pueda conectar.
3. Otorga al usuario `paco@localhost` permisos de `select` en la tabla `jardineria.Clientes` y comprueba que se pueda consultar la tabla.
4. Otorga al usuario `juan@localhost` permisos de `select, insert y update` en las tablas de la base de datos `jardineria` con opción `GRANT`.
5. Conéctate con el usuario `juan` y otorga permisos a `paco` de selección en la tabla `jardineria.Empleados`.
6. Quitale ahora los permisos a `paco` de selección sobre la tabla `jardineria.Clientes`.
7. Conéctate con `root` y elimina todos los permisos que has concedido a `Paco` y `Juan`.
8. Otorga a `juan` los permisos de `SELECT` sobre las columnas `CódigoOficina` y `Ciudad` de la tabla `Oficinas` de la base de datos `jardineria`.
9. Conéctate con `juan` y ejecuta la query `'SELECT * from jardineria.Oficinas'` ¿Qué sucede?
10. Borra el usuario `paco@localhost`.

Práctica 2.3: MySQL: Crear usuarios y asignar permisos en remoto

Realiza las siguientes operaciones y almacena los comandos y los resultados en un fichero:

1. Crea un nuevo usuario llamado `usuario@direccion_ip` donde `direccion_ip` es una máquina de un compañero tuyo y `usuario` su nombre.
2. Otórgale permisos de selección en todas las tablas de la base de datos `jardineria`. Ten cuidado, es posible que tu servidor solo permita conexiones desde el ordenador local, para permitir conexiones remotas debes comentar la linea `bind-address` de tu fichero `my.cnf` que impide conexiones desde otros sitios que no sea el especificado (127.0.0.1). Asegúrate de reiniciar el servidor.
3. Pide a tu compañero que se conecte desde su máquina y que averigüe qué permisos le has otorgado. El a ti te pedirá lo mismo, es decir, que te conectes a su máquina, indica qué instrucción `sql` ejecutas para conocer los permisos que tienes.
4. Revócale los permisos concedidos al usuario `usuario@direccion_ip`.
5. Concédele ahora permisos de creación de tablas en una nueva base de datos que has creado.
6. Solicítale que se conecte y que pruebe a crear una tabla. ¿Puede consultar la información?
7. Borra ahora el usuario `usuario@direccion_ip`.
8. Conéctate a la bbdd `information_schema` con el usuario `root` y consulta las columnas que tiene la tabla `Oficinas` de la base de datos `jardineria`.
9. Con la bbdd `mysql` consulta qué privilegios tiene el usuario `juan@localhost` a nivel de servidor, a nivel de base de datos, a nivel de tablas y a nivel de columnas.

◦

Práctica 2.4: Oracle: Creación de usuarios, roles y perfiles

1. Crea dos roles, uno llamado `ADMINISTRACION` con todos los privilegios sobre la tabla `CLIENTES` y `PEDIDOS` del esquema `JARDINERIA` y otro llamado `CONTABILIDAD` con todos los privilegios sobre la tabla `PAGOS` del mismo esquema. El rol `CONTABILIDAD` estará identificado por contraseña.
2. A continuación, crea un usuario `Fernando` en Oracle con las siguientes opciones: `DEFAULT TABLESPACE Users`, `TEMPORARY TABLESPACE Temp`, `QUOTA 100 Megabytes` en `Users` y otorga el rol `CONNECT` y `RESOURCE`.

3. Otorga como Rol por defecto ADMINISTRACION al usuario Fernando.
4. Con el usuario Fernando, prueba a ejecutar una consulta sobre la tabla Clientes y otra sobre la tabla PAGOS.
5. A continuación, activa el Rol CONTABILIDAD y repite la consulta sobre la tabla PAGOS con el usuario Fernando.
6. Desactiva todos los roles del usuario Fernando, incluyendo el rol por defecto.
7. Consulta la tabla Clientes.

◦

Práctica 2.5: MySQL: Consulta del catálogo de metadatos

1. Con las tablas mysql.user y mysql.tables_priv crea una consulta para devolver los privilegios que tienen los usuarios sobre las tablas del SGBD.
2. Con las tablas de la base de datos information_schema muestra el tiempo en minutos que lleva arrancado el servidor.
3. Con la tabla views de la base de datos information_schema consulta la definición de las vistas creadas en el SGBD.
4. Consulta el número de filas que tiene cada tabla de la base de datos nba.
5. Muestra las columnas de la tabla Empleados de la base de datos jardineria.

◦

Práctica 2.6: Oracle: Consulta del catálogo de metadatos

1. Con la vista dba_tables consulta en qué esquema y tablespace está situada la tabla Empleados.
2. Con las tablas dba_data_files y dba tablespaces, consulta los tablespaces disponibles en el gestor y qué datafiles tiene cada tablespace mostrando el tamaño máximo del tablespace y el número de MB de cada datafile.
3. Consultar todas las vistas del propietario jardineria.
4. Utiliza la vista dba_users para consultar los usuarios a los que la password les ha expirado (EXPIRED).
5. Une el campo paddr con el campo addr de las vistas v\$session y v\$process para obtener los campos sid, serial, pid y program de las sesiones abiertas por el programa sqlplus.

◦

2.7. Amplía tus conocimientos

- Has visto cómo auditar consultas en MySQL con los parámetros log y log-slow-queries. Es una manera muy simple de auditar qué está sucediendo en el SGBD. Sin embargo, Oracle ofrece todo un mundo de posibilidades a la hora de auditar qué sucede en la base de datos. Ejecuta en Oracle el comando 'SHOW PARAMETER AUDIT' para consultar en qué fichero se guarda la auditoria.

Después, altera el parámetro audit_trail para iniciar la auditoria:

```
ALTER SYSTEM SET audit_trail=db SCOPE=SPFILE;
```

Ejecuta los siguientes comandos para activar la auditoria del usuario jardineria:

```
AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE,  
DELETE TABLE BY jardineria BY ACCESS;  
AUDIT EXECUTE PROCEDURE BY jardineria BY ACCESS;
```

A continuación, ejecuta algunas acciones con el usuario jardineria y después consulta la auditoria con sqlplus y la vista dba_audit_trail:

```
COLUMN username FORMAT A10  
COLUMN owner FORMAT A10  
COLUMN obj_name FORMAT A10  
COLUMN extended_timestamp FORMAT A35  
  
SELECT username, extended_timestamp, owner, obj_name, action_name  
FROM dba_audit_trail  
WHERE owner = 'jardineria'  
ORDER BY timestamp;
```

- Busca en algún foro de Internet un script para consultar el espacio libre de cada data file en Oracle.
- En Oracle es posible realizar consultas distribuidas entre dos tablas de diferentes bases de datos que pueden estar ubicados incluso en diferentes servidores. Investiga cómo realizar enlaces entre diferentes bases de datos Oracle con la orden CREATE DATABASE LINK.
- Estudia qué son los sinónimos en Oracle y cómo crearlos mediante la sentencia CREATE SYNONYM.

AUTOMATIZACIÓN DE TAREAS.

SCRIPTS DE ADMINISTRACIÓN

Contenidos

- » Herramientas para creación de guiones
- » Planificación de tareas de administración mediante guiones
- » Disparadores de sistema
- » Eventos de sistema
- » Excepciones servererror

Objetivos

- » Reconocer la importancia de automatizar tareas administrativas
- » Describir los distintos métodos de ejecución de guiones
- » Identificar las herramientas disponibles para redactar guiones
- » Saber usar guiones para automatizar tareas
- » Identificar los eventos susceptibles de activar disparadores
- » Definir disparadores
- » Utilizar estructuras de control de flujo
- » Saber adoptar medidas para mantener la integridad y consistencia de la información

Al finalizar este capítulo se conocerá la técnica de construcción de scripts, imprescindible para la simplificación y automatización de tareas administrativas y/o repetitivas.

3.1. Herramientas para creación de guiones

Los guiones (*scripts* en inglés) son programas más o menos complicados, que interactúan con la base de datos o alguno de sus resultados teniendo como objetivo hacer alguna modificación en ella o recoger alguna información útil para su administración. Existen muchas formas de construir guiones, no solo por su finalidad sino también por su contenido o el entorno en que se ejecutarán. Así pues, se podrán encontrar scripts escritos en C, o en PL/SQL, o en SQL/PL, o Perl, o pensados para el *awk*...

Sin duda en el mercado existen muchas herramientas para crear scripts, algunas basadas en consolas gráficas y otras no.

3.1.1. Herramientas gráficas

PLedit es un editor de PL/SQL de la marca Benthic Software y que junto con *Golden*, usado para la ejecución, constituyen una muy buena opción para el desarrollo de paquetes, procedimientos, funciones, triggers... y sentencias SQL en bases de datos Oracle, ya que es un producto libiano pero muy bien hecho, que además de ser muy funcional puede ser utilizado sin licencia definitiva durante un par de meses.

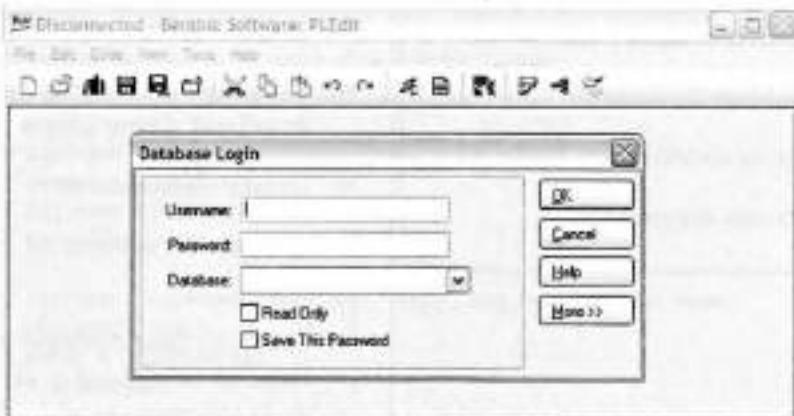


Figura 3.1: Conexión a la base de datos con el Pledit.

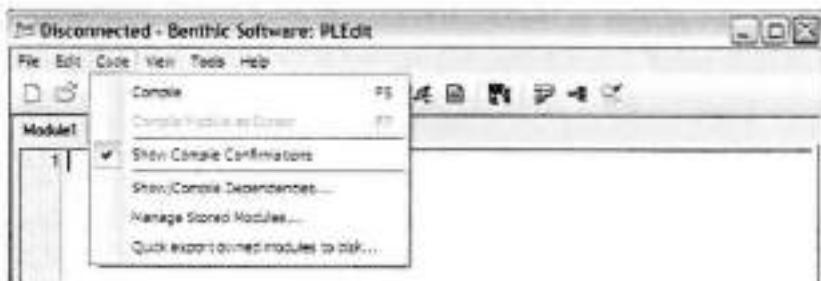


Figura 3.2: Compilación de código con el Pledit.

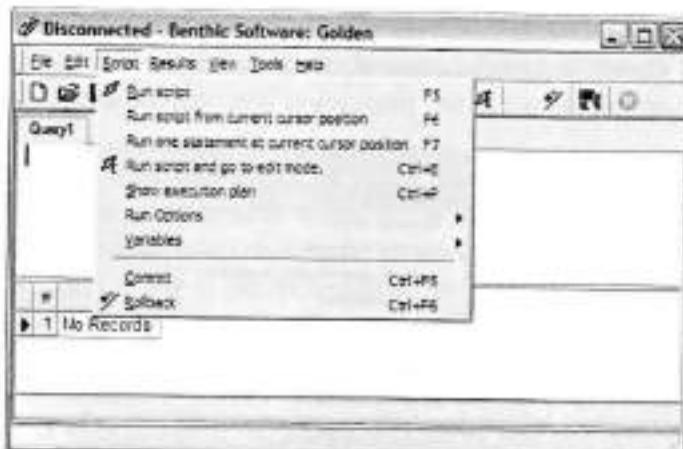


Figura 3.3: Manejo de scripts con Golden.

TOAD, de la firma QUEST SOFTWARE, por su facilidad de uso y potencia, suele ser una herramienta muy extendida en la administración de bases de datos Oracle. No obstante, en muchos entornos productivos su uso está terminantemente prohibido, no tanto por la cantidad de recursos que consume, sino por la mala utilización que se hace de la herramienta en general, sobre todo en el lanzamiento masivo, voluntario o involuntario, de consultas costosas. También hay disponible una versión gratuita.

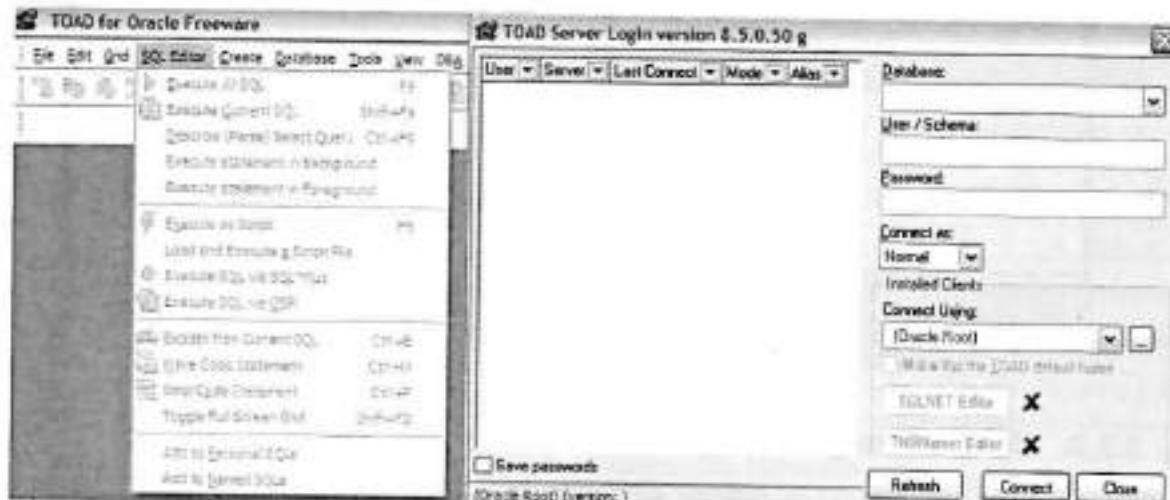


Figura 3.4: TOAD. Conexiones y opciones del SQL Editor.

El *Oracle Developer* es el producto que la factoría Oracle ofrece para el desarrollo de programas contra bases de datos Oracle. Es también muy versátil, a la par que caro.

Para DB2, el *Editor de mandatos* ofrece la posibilidad de ejecutar sentencias contra una base de datos desde un entorno gráfico, y sirve de puente para acceder a otras herramientas relacionadas con la programación, por ejemplo, el planificador de tareas. Pero el desarrollo propiamente dicho de scriptería habría que hacerlo desde el potente *IBM Data Studio* (evolucionado recientemente al *Optim Development Studio*), que está desarrollado

en eclipse. El *Editor de mandatos* se incluye en la instalación base del DB2, mientras que el *Data Studio* y el *Optim* se instalan aparte, pero ambos poseen versiones gratuitas, que si bien no están completas, sí permiten programar con amplia comodidad.

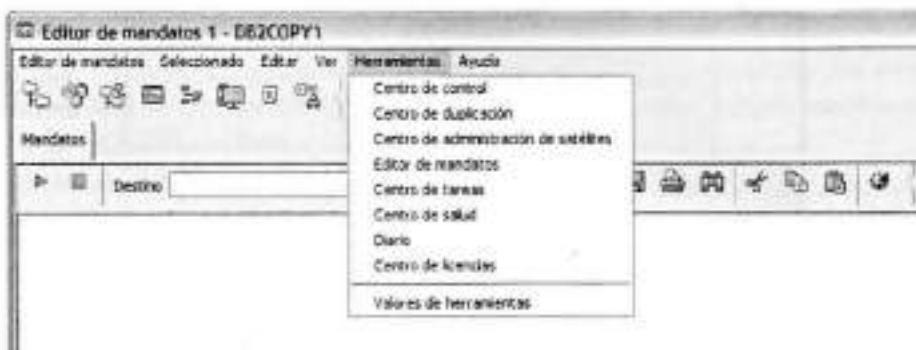


Figura 3.5: Herramientas accesibles a través del Editor de mandatos del DB2.

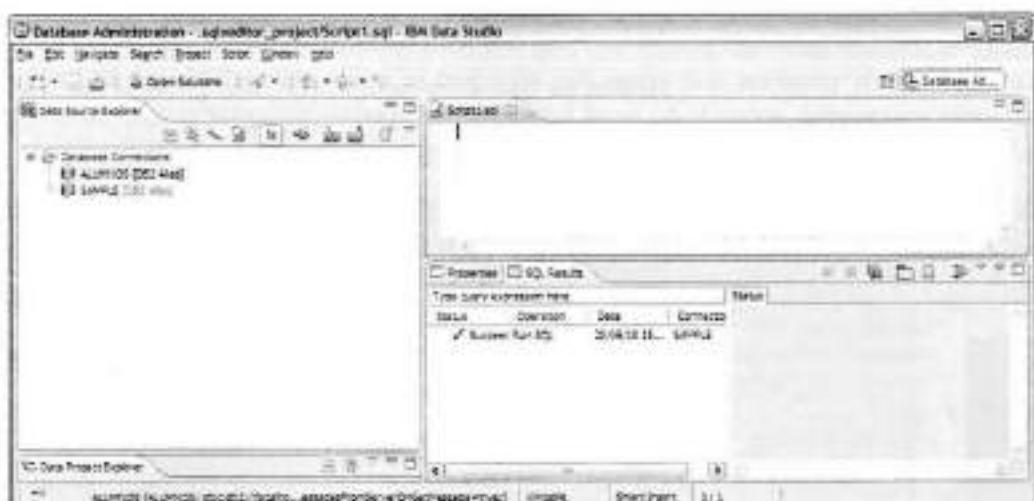


Figura 3.6: Ventana de edición de script en el IBM Data Studio.

3.1.2. Herramientas no gráficas

¿Qué pasa cuando no se dispone de herramienta gráfica alguna?, es decir, ¿qué pasa cuando el administrador se ve obligado a trabajar en un entorno que no le es familiar?: puede que sobre un ordenador ajeno que no tenga instaladas las herramientas que suele utilizar, o puede que la base de datos sobre la que tiene que actuar no permita conexiones remotas tales como las de las herramientas gráficas, o puede que el cliente para el que trabaje no haya pagado las licencias oportunas para poder trabajar con ellas, ¿qué pasa entonces cuando el administrador habituado a administrar a 'golpe de click de ratón' tiene que examinar, supervisar o incluso programar sin la ayuda de esas amigables e intuitivas herramientas gráficas? Pues el resultado es, ni más ni menos, que se convierte en un administrador de 'pantalla negra', es decir, un técnico todoterreno que lo mismo administra

desde el propio CPD donde se ubica la máquina que alberga la base de datos, sin tener ni siquiera conexión de red, o es capaz de hacer instalaciones de software sin disponer de un entorno gráfico como XWindows.

En esas circunstancias, probablemente el único modo posible para crear o modificar guiones sea el potente, pero complicado, editor *vi* de Unix. Merece la pena pues dedicarle una mención especial y emplear algo de tiempo en aprender a manejarlo. Para ejecutar *vi*, tan solo hay que invocarlo desde el sistema operativo:

```
$ vi nombre_fichero
```

Con este comando se abrirá el fichero que se desea editar o se creará uno nuevo si no existía previamente. A continuación, se ofrece una tabla con sus opciones de edición más utilizadas.

Opciones de comandos del vi	Propósito
Tecla Esc	Pasar de modo edición a modo comando
:q	Salir
:wq	Salir guardando los cambios
:q!	Salir sin guardar los cambios
:n	Ir al siguiente fichero editado
:número_de_línea	Posicionarse en un determinado número de línea
Tecla de mayúsculas + G	Ir al final del fichero
:set number	Numerar las líneas del fichero
Opciones de edición del vi	Propósito
j	Moverse una linea hacia abajo
Tecla Ctrl + d	Moverse una página hacia abajo
k	Moverse una linea hacia arriba
Tecla Ctrl + u	Moverse una página hacia arriba
l	Moverse un carácter a la derecha
h	Moverse un carácter a la izquierda
i	Cambiar a modo inserción
A	Cambiar a modo inserción pero desde el final de la línea actual
x	Borrar un carácter
dd	Borrar la linea actual completa
u	Deshacer la última actualización
.	Repetir la última edición efectuada
r	Sustituir un carácter por otro
cw	Cambiar una palabra entera por otra
/patrón	Buscar un texto concreto o un determinado patrón
n	Ir al siguiente texto encontrado según la búsqueda anterior

Cuadro 3.1: Opciones más usadas del editor *vi*.

Entre otras virtudes, se le puede achacar al *vi* el ser un estupendo editor para hacer sustituciones de texto siguiendo un patrón definido. Sirvan los siguientes ejemplos como muestra:

- `.3,5s/hola/adios/` Sustituye el primer *hola* encontrado en las líneas 3, 4 y 5, por la palabra *adios*.
- `.3,5s/hola/adios/g` Sustituye todos los *hola* encontrados en las líneas 3, 4 y 5, por la palabra *adios*.
- `.,5s/hola/adios/g` Sustituye todos los *hola* encontrados en las líneas comprendidas entre la actual y la número 5, por la palabra *adios*.
- `.,$s/hola/adios/g` Sustituye todos los *hola* encontrados en las líneas comprendidas entre la actual y la última, por la palabra *adios*.
- `:1,$s/*hola/adios/` Sustituye todos los *hola* hallados al comienzo de cada línea, por la palabra *adios*.
- `:1,$s/hola$/adios/` Sustituye todos los *hola* hallados al final de cada línea, por la palabra *adios*.

Con el *vi* se podrían, por ejemplo, crear programas en PL/SQL que luego se habrán de incorporar a la base de datos. En ausencia de herramientas gráficas, para comprobar el resultado de la compilación de esos programas PL/SQL se podrían usar las vistas *DBA_SOURCE*, donde se almacenará el código de los programas (ya sean paquetes, procedimientos, funciones o triggers), y *DBA_ERRORS*, donde se registrarán los errores de compilación ocurridos.

◦ **Actividad 3.1:** Desarrollar una consulta SQL que, basándose en las vistas *DBA_SOURCE* y *DBA_ERRORS*, obtenga los errores de compilación de un programa concreto, así como el número de línea del programa que provoca el error y su contenido.

3.1.3. Métodos de ejecución de guiones

Ahora bien, una vez escrito el script, ¿cómo puede ejecutarse? Para responder a esta pregunta habría que contestar primero a muchas otras: ¿su ejecución tiene que ser atendida o desatendida?, ¿online o en background?, si se está trabajando con una conexión abierta en remoto a través de la red, ¿es probable que se pueda perder la conexión en algún momento del proceso?, ¿se va a hacer en el momento actual o en un momento futuro determinado en el tiempo?, ¿debe activarse en función del resultado de algún otro programa?, ¿el resultado de su ejecución debe desencadenar alguna otra ejecución?, ¿debe imponérsele algún límite de tiempo en máquina?, ¿se tiene que ejecutar más de una vez o con alguna periodicidad concreta?, ¿cuánto se estima que pueda durar la ejecución del script?, ¿cuánto se prevee que pueda consumir de CPU?, ¿puede interferir a otros procesos arrancados en la máquina?, ¿la máquina está en producción?, ¿en preproducción?, ¿en desarrollo?...

Todas estas preguntas, y probablemente muchas otras, deberían ser contestadas antes de tomar la decisión definitiva de cómo ejecutar un script concreto, es decir, antes de utilizar alguno de los siguientes métodos:

comando at

`at` ejecuta, en el momento programado, un comando recogido del teclado o leído del fichero indicado con la opción `-f` de ejecución. Solo se ejecutará una vez, en el momento exacto que se haya programado. Por ejemplo, para lanzar el contenido del fichero `comandos_ejemplo.ksh` a las 9 de la noche del 17 de Febrero habría que escribir cualquiera de los siguientes comandos:

```
at -f comandos_ejemplo.ksh 9pm Feb 17
at -f comandos_ejemplo.ksh 21:00 Feb 17
at -f comandos_ejemplo.ksh 21:00 02/17/11
```

Figura 3.7: Programación con `at` de la ejecución de un fichero en un momento concreto.

Para lanzarlo dentro de un minuto:

```
at -f comandos_ejemplo.ksh now + 1 minutes
```

Figura 3.8: Programación con `at` de la ejecución de un fichero dentro de un minuto.

Para listar los trabajo programados:

```
at -l
2      Thu Feb 17 21:00:00 2011 a Pepito
1      Thu Feb 17 20:22:00 2011 a Pepito
```

Figura 3.9: Listado de las tareas pendientes de ejecución con `at`.

(Todas las posibles especificaciones de literales que se pueden usar para indicar el momento concreto pueden consultarse en el fichero: `/usr/share/doc/at/timespec`).

Los usuarios que pueden usar el `at` deben estar incluidos en el fichero `/etc/at.allow`, y en caso de que este no existiera, no estarlo en el `/etc/at.deny`. Si ninguno de los dos ficheros existiese, solo `root` podría usar el `at`.

crontab

Es la herramienta usada para manejar la tabla de programas del `cron`. El `cron` es una lista de comandos, programas o scripts dispuestos para ser ejecutados en un momento indicado. Cada ejecución se registra en una sola línea del `cron`, en la cual las cinco primeras posiciones especifican cuándo se ejecutará el comando, programa o script que figura a continuación. Por ejemplo:

```
30 * * * * /sbin/ping -c 1 193.100.0.1 >> /dev/null
```

Figura 3.10: Interpretación de los tiempos del cron.

La primera posición se refiere al minuto de ejecución, la segunda a la hora, la tercera al día, la cuarta al mes y la quinta al día de la semana, por lo que es fácil intuir que, a diferencia del at, el cron permite lanzar el programa más de una vez, por ejemplo, todos los días de la semana a una hora determinada, o como se refleja en el ejemplo anterior, en el minuto 30 de cada hora durante todos los días de la semana, es decir, que el ping se lanza cada media hora todos los días. Otros ejemplos de periodicidad en la ejecución:

* * * * * cada minuto, de cada hora, de cada día, de cada mes, es decir, se ejecuta cada minuto.

0 20 * 10 1-5 los días laborables de la semana (de Lunes a Viernes) a las 8pm en punto y solo el mes de Octubre.

0 0 1,15,28 * * de Lunes a Domingo a la medianoche (0am) los días 1, 15 y 28 de cada mes.

0,50 7,8 * * 1 todos los Lunes a las 7am, 7:50am, 8am y 8:50am.

También, a diferencia del at, existe un crontab para cada usuario y como el at, un usuario concreto puede utilizarlo siempre que esté incluido en el fichero /etc/cron.allow o en caso de que este fichero no exista, no lo esté en el /etc/cron.deny. Si no existiese ninguno de los dos, solo root tendría su propio cron.

Para ver cuál es el contenido actual del cron, basta con ejecutar el comando *crontab -l*. La edición del contenido del cron se hará con la ayuda del vi tras invocar el comando *crontab -e*.

nohup

nohup ejecuta un comando desvinculándolo de la sesión que lo lanzó, de modo que ante cualquier fallo de comunicación con la máquina, y por tanto, de pérdida de la sesión en curso, el comando sigue ejecutándose. También tiene la ventaja de que permite lanzar muchos comandos a la vez desde la misma sesión, sin necesidad de tener que esperar a que termine su ejecución, es decir, sirve para ejecutar en paralelo en vez de serializando.

El siguiente ejemplo muestra cómo se puede ejecutar en background (esto se consigue poniendo el último &) el script /home/pepito/ejemplo.ksh direccinando tanto su salida estándar como los posibles mensajes de error que pudiesen aparecer durante su ejecución (representados por el número 2) al fichero /home/pepito/ejemplo.log (representado por el &1).

```
nohup /home/pepito/ejemplo.ksh>/home/pepito/ejemplo.log 2>&1 &
```

Figura 3.11: Uso del *nohup*.

el comando db2

Cuando se trabaja con db2 los ficheros con comandos SQL o SQL/PL para DB2, pueden ejecutarse invocando la herramienta *db2*.

```
db2 -tvf fichero_con_sentencias
```

Figura 3.12: Ejecución de un script en DB2.

En la instrucción anterior, la *t* significa que en el fichero de comandos que se va a ejecutar, las sentencias SQL terminan en punto y coma (;), la *v* es de verbose (esto significa que se ofrecerán mensajes informando de los errores ocurridos y/o del estado del programa), y la *f* indica que el suministro de sentencias se hace mediante el fichero cuyo nombre aparece a continuación.

sqlplus

los ficheros con comandos SQL o PL/SQL de Oracle, pueden ejecutarse invocando la herramienta SQL*Plus, bien directamente desde la línea de comandos, o bien desde dentro de otro script. Basta con escribir una instrucción del estilo:

```
sqlplus usuario/password @fichero_con_sentencias.sql
```

Figura 3.13: Ejecución de un script en Oracle.

También mediante el uso de los signos <<, se puede montar un script directamente en línea de comandos.

```
sqlplus / as sysdba <<EOF
select * from v$database;
EOF
```

Figura 3.14: Script desde línea de comandos.

Obsérvese que, puesto que el \$ tiene en Unix un significado propio (para acceder al contenido de las variables), es necesario escaparlo mediante la barra invertida si se pretende escribirlo literalmente en línea de comando.

Desde dentro del propio SQL*Plus se pueden invocar comandos externos mediante el uso de la predirectiva `execute rc` o `host` o `!comando`

```
SQL> execute rc('date');
SQL> host date;
SQL> !date
```

Figura 3.15: Ejecución de comandos externos desde dentro del SQL*Plus.

```
SQL> execute rc('/home/pepito/ejemplo.ksh');
```

Figura 3.16: Ejecución de script desde dentro del SQL*Plus.

Es conveniente recordar que DB2, desde su versión 9.7, permite arrancar la instancia con la opción de compatibilidad con Oracle (`db2set DB2_COMPATIBILITY_VECTOR=ORA`), de modo que se ofrece la posibilidad de desarrollar directamente en PL/SQL, y en consecuencia, es posible utilizar también el programa `sqlplus`, es decir, que, en un porcentaje muy elevado, aprender a usar Oracle implica también aprender a usar DB2 (pero no al revés).

mysql

Cuando se trabaja con mysql es posible ejecutar los comandos de un fichero de texto desde la shell, esto es, en modo batch, redirigiendo al cliente mysql un fichero de entrada. Esto es útil para tareas administrativas donde se ejecutan varios ficheros de comandos, además de otras tareas de mantenimiento del servidor a través de un shell script:

```
#ejecucion en modo batch
-$ mysql -u root -pPassWdUsuario <script_coninstrucciones.sql

#ejecucion en modo batch almacenando resultados
-$ mysql -u root -pPassWdUsuario <script_coninstrucciones.sql >resultado
```

ControlM

ControlM es una de las herramientas más potentes de planificación que están actualmente en el mercado. No es específica de bases de datos, permite ejecutar cualquier scripts en el sistema operativo, así como enlazar o condicionar unas ejecuciones a otras.

Planificadores de tareas en base de datos

Existen planificadores gráficos, por ejemplo dentro del Enterprise Manager de Oracle, o el Optim Database Administrator del DB2, que lo que hacen es facilitar la gestión de los paquetes internos que llevan la cuenta de la ejecución de las tareas programadas de base de datos. En el caso de Oracle esta gestión se hace a través del paquete *dbms_scheduler* y sus procedimientos y funciones. Ejemplos de ellos serían los procedimientos *CREATE_SCHEDULE*, que es usado para crear un patrón de comportamiento, al que entre otras cosas, se le debe indicar la fecha de comienzo de la ejecución, el intervalo de repetición, la fecha de finalización...; o *CREATE_JOB*, que sirve para crear un trabajo completo, es decir, que además de los campos antes mencionados que se definen en el patrón, se le puede asociar directamente el nombre del programa que deberá ejecutarse.

3.1.4. Elección del Método de ejecución adecuado

Una vez visto lo anterior, la elección del método de ejecución se hará en función de las respuestas a las cuestiones planteadas originalmente:

- Si su ejecución puede ser desatendida, entonces es aconsejable usar o bien el *nohup* acabado en &, o bien cualquier planificador, desde los simples *at* y *crontab*, hasta el completo *ControlM*.
- Lo mismo sería aplicado en el caso de que la ejecución tenga que hacerse en background.
- En caso de que la ejecución sea online, pero existiese alguna probabilidad de que se pudiera perder la conexión en algún momento del proceso, lo más aconsejable es usar el *nohup*.
- Si no se va a ejecutar en el momento actual sino que se tiene que lanzar en un momento futuro determinado en el tiempo, habrá de usarse necesariamente un planificador.
- Si tiene que ser arrancado en función del resultado de algún otro programa, habrá de usarse el *ControlM* o algún otro planificador sofisticado, o ayudarse de algún programa de monitorización que esté alerta ante los procesos activos en máquina.
- Lo mismo podría aplicarse si el resultado de la ejecución del programa debe desencadenar alguna otra acción.
- Si se tiene que lanzar más de una vez y/o con alguna periodicidad concreta, el *at* no vale, tendría que usarse el *crontab* o los otros planificadores más complejos.
- Si lo que se busca es la minimización de los recursos de máquina consumidos (ya sea en cuanto a duración o cantidad de CPU precisada) porque no se esté en un entorno dedicado, o el entorno sea productivo y haya que asegurar la mínima incidencia de la ejecución del script hacia el resto de procesos que corren en máquina, es vital buscar una ventana de ejecución donde el conflicto de intereses entre procesos sea

asumible. Generalmente esto se consigue por la noche, momento en el cual se realizan, por ejemplo, los backups. También sería buena idea activar un programa espía que se ejecute en paralelo al script deseado y que monitorice constantemente los puntos delicados, pudiendo llegar a cancelar el proceso si fuera necesario. Además de esto, existen en el mercado herramientas basadas en perfiles de usuario donde se puede limitar el consumo de máquina que hace el programa ejecutado, en función de las características que se le hayan otorgado al usuario que lo lanzó, de modo que se le asigna una prioridad y privilegios de coste en tiempo, CPU...

3.2. Planificación de tareas de administración con scripts

El día a día de un administrador cualquiera, probablemente, comience con la supervisión del estado de todas las bases de datos que están bajo su control, la comprobación de la validez del backup hecho la noche anterior o de la finalización del paso de estadísticas. Tal vez incluso también tenga que extraer los datos representativos de las auditorías programadas y remitir el informe pertinente a su jefe.

Probablemente, esto tenga que hacerlo en su primera hora de trabajo porque posteriormente tendrá que dedicarse a resolver peticiones e incidencias. Quizá reciba el encargo de crear en todas las bases de datos bajo su tutela un usuario determinado porque la dirección del proyecto lo necesita para implantar una nueva solución. A lo mejor luego le piden que exporte los registros de una tabla y que en función del valor de uno de sus campos, cree para cada registro un directorio en sistema operativo cuyo nombre sea el contenido de otro campo...

Todos estos ejemplos son susceptibles de ser automatizados. Todo lo que sean tareas repetitivas, ya sea porque se necesitan hacer a menudo y de la misma manera, o porque en sí mismas presenten un patrón de iteración, es posible y preferible que estén automatizadas. Posible porque seguro que responden a un algoritmo lógico que puede programarse. Preferible porque todo lo que esté automatizado se ejecutará siempre de la misma manera, más rápido (aumentando considerablemente la efectividad en el trabajo) y por tanto a menos coste, y también porque evitará errores humanos, muy comunes cuando lo que se tiene que hacer es tan repetitivo que inevitablemente se convierte en algo aburrido.

Pero ¿cómo se automatizarían? La respuesta es sencilla: mediante scripts. Los script realmente son simples programas. Por su potencia, rapidez, efectividad y austerdad, en este capítulo se van a explicar las técnicas de implementación de scripts escritos en KShell (Korn Shell), que son sumamente portables dentro de los entornos Unix/Linux y por eso mismo susceptibles de ser reutilizados. También hay que mencionar otras alternativas, como los utilizados por los sistemas operativos basados en Microsoft Windows, desde los clásicos batch (.bat) hasta la apuesta de la compañía por Power Shell, una interfaz muy potente que trata de competir con las shell más renombradas como la mencionada Korn.

Korn Shell es un programa informático para Unix y Linux cuya función consiste en interpretar órdenes por líneas, y puede ser empleado como lenguaje de programación.

La combinación de KShell con PL/SQL es indicativa de éxito seguro (pocas cosas podrían encontrarse que no pudieran automatizarse con la ayuda de estos dos valiosos componentes).

Un script escrito en KShell es un simple fichero de texto que en la primera línea de su contenido presenta la predirectiva `#!/usr/bin/ksh`. El resto del fichero podrá contener comandos Unix de todo tipo, desde estructuras de control de flujo, pasando por sentencias escritas en perl o awk, hasta llamadas a otros programas como el `sqlplus` o la ventana de mandatos del DB2.

A continuación se expondrán algunos de esos comandos, los más comunes por su versatilidad y por las características del trabajo diario de todo administrador.

3.2.1. Estructuras de control de flujo

Son vitales para establecer los distintos caminos por los que podrá seguir la ejecución programada en el script. Al combinarlas con las condiciones impuestas para la toma de decisiones, se conseguirá desarrollar toda la lógica de negocio. Pero, ¿cómo se realizan en KShell las comparaciones de condiciones?:

Comparación	Sintaxis
Igual	=
Distinto	!=

Cuadro 3.2: Comparaciones alfanuméricas.

Comparación	Sintaxis
Igual	-eq
Distinto	-ne
Menor	-lt
Menor o igual	-le
Mayor	-gt
Mayor o igual	-ge

Cuadro 3.3: Comparaciones numéricas.

Y ¿cómo se enlazan esas comparaciones?:

Conjunción	Sintaxis entre corchetes	Sintaxis dentro de los corchetes
AND	&&	-a
OR		-o

Cuadro 3.4: Conjunciones.

Una vez conocida la técnica para relacionar condiciones, ahora si es posible aprender a manejar las típicas estructuras de control de flujo en un script escrito en KShell, para ello lo mejor es ilustrar cada caso con ejemplos:

if then/elif then/else/fi

Véase el siguiente ejemplo, en el que se ha utilizado el comando de Unix *echo* para escribir los mensajes explicativos.

```
if [ 4 -ge 6 ]; then
    echo "4 es mayor o igual que 6"
else
    echo "4 no es mayor o igual que 6"
fi
```

Es muy importante respetar los espacios en blanco entre los corchetes y las condiciones. Nótese que el ; sirve para ejecutar dos comandos en la misma línea, es decir que el ejemplo anterior realmente debería escribirse como:

```
if [ 4 -ge 6 ]
then
    echo "4 es mayor o igual que 6"
else
    echo "4 no es mayor o igual que 6"
fi
```

Otro ejemplo:

```
numero=4
if [ $numero -lt 6 ]; then
    echo "$numero es menor que 6"
fi
```

Nótese que dentro de un script, como desde la línea de comandos del Unix, al contenido de las variables se accede anteponiendo el carácter \$ al nombre de la variable.

Se puede otorgar valor a una variable asignándole el resultado de un comando, y ello gracias al uso de la tecla de acento inverso ` . Por ejemplo, para asignar a la variable llamada *so* el sistema operativo de la máquina donde se está trabajando, basta con ejecutar: *so='uname'*

```
#para averiguar qué sistema operativo tiene instalado la máquina:  
so='uname'  
if [ $so = "Linux" -o $so = "AIX" ]; then  
    echo "La máquina es Linux o AIX"  
else  
    echo "La máquina no es ni Linux ni AIX"  
fi
```

Del ejemplo anterior es fácil deducir que el carácter # precede a un comentario. Además, nótense que en las comparaciones entre cadenas son muy importantes los espacios en blanco antes y después del = o !=.

El ejemplo anterior es equivalente a (¡ojo! con los corchetes):

```
so='uname'  
if [ $so = "Linux" ] || [ $so = "AIX" ]; then  
    echo "La máquina es Linux o AIX"  
else  
    echo "La máquina no es ni Linux ni AIX"  
fi
```

Análogamente para el AND:

```
host='hostname'  
so='uname'  
if [ $host = "POKEMON" -a $so = "Linux" ]; then  
    echo "La máquina se llama POKEMON y es Linux"  
else  
    echo "La máquina no se llama POKEMON o no es Linux"  
fi
```

Que equivale a:

```
host='hostname'  
so='uname'  
if [ $host = "POKEMON" ] && [ $so = "Linux" ]; then  
    echo "La máquina se llama POKEMON y es Linux"  
else
```

```
    echo "La máquina no se llama POKEMON o no es Linux"
fi
```

Para combinar los AND y los OR, o bien se usa la cláusula *elif*, o necesariamente habrán de mezclarse las opciones *-a* y *-o*, con las **&&** y **||**) para agrupar condiciones:

```
host='hostname'
so='uname'
if [ $host = "POKEMON" -a $so = "Linux" ]; then
    echo "La máquina se llama POKEMON y es Linux"
elif [ $so = "AIX" ]; then
    echo "La máquina es AIX"
else
    echo "La máquina no es AIX y/o no se llama POKEMON y es Linux"
fi
```

```
host='hostname'
so='uname'
if [ $host = "POKEMON" -a $so = "Linux" ] || [ $so = "AIX" ]; then
    echo "La máquina es AIX o se llama POKEMON y es Linux"
else
    echo "La máquina no es AIX y/o no se llama POKEMON y es Linux"
fi
```

```
host='hostname'
so='uname'
if [ $host = "POKEMON" ] && [ $so = "Linux" -o $so = "AIX" ]; then
    echo "La máquina se llama POKEMON y es Linux o AIX"
else
    echo "La máquina no se llama POKEMON y/o no es ni Linux ni AIX"
fi
```

Para negar una condición basta con anteponer el carácter *!* a la condición, y entonces se considerará el contrario.

```
host='hostname'
so='uname'
if [ $host = "POKEMON" ] && ! [ $so = "Linux" -o $so = "AIX" ]; then
    echo "La máquina se llama POKEMON y no es ni Linux ni AIX"
else
    echo "La máquina no se llama POKEMON y/o es Linux o AIX"
fi
```

```

host='hostname'
so='uname'
if [ $host = "POKEMON" ] && [ $so != "Linux" ]; then
    echo "La máquina se llama POKEMON y no es Linux"
else
    echo "La máquina no se llama POKEMON y/o es Linux"
fi

```

- **Actividad 3.2:** En los ejemplos anteriores, ponerle manualmente valor a las variables *host* y *so* para obligar la salida de todos los mensajes.

Resulta muy útil saber que, *\$#* en las primeras líneas del script contiene el valor del número de parámetros de entrada que se le han suministrado al script en su ejecución. *\$1* almacenará el valor del primer parámetro pasado. *\$2* el del segundo... y así sucesivamente.

```

num=$#
if [ $# -ne 4 ]; then
    echo "Error, el número de parámetros pasados debería ser 4, pero es \"$num"
else
    if [ $2 != "n" ] && [ $2 != "y" ]; then
        echo "El segundo parámetro debe ser n o y. Vuelva a intentarlo"
    fi
fi

```

Otra técnica muy empleada es usar la opción *-f* para ver si un fichero existe.

```

if [ -f ejemplo.log ]; then
    echo "El fichero ejemplo.log existe"
fi

if [ ! -f ejemplo.log ]; then
    echo "El fichero ejemplo.log no existe"
fi

```

- **Actividad 3.3:** Con ayuda del comando *touch* (que sirve para crear ficheros, si es que estos no existen ya), crear el fichero *ejemplo.log* (*touch ejemplo.log*) y comprobar el funcionamiento de los dos ejemplos anteriores.

case/esac

Una manera muy útil de clasificar los posibles valores que puede tener una variable es usando el **case**, que puede combinarse también con otras estructuras de control de flujo. Es una forma fácil de ahorrar sentencias **if**.

```
so='uname'
case $so in
Linux) echo "1-La máquina es Linux"
        echo "Su nombre es `hostname`";;
AIX) echo "2-La máquina es AIX" ;;
Solaris) echo "3-La máquina es Solaris" ;;
*) echo "4-La máquina es $so" ;;
esac
```

```
aux='echo hola'
case $aux in
hola|adios) echo "Saludos";;
potaje) echo "Comida de puchero" ;;
zumo|cola) echo "Bebida"
        if [ $aux = "cola" ]; then
                echo "Mi favorita"
        fi;;
*) echo "Sin catalogar" ;;
esac
```

-
- **Actividad 3.4:** En los ejemplos anteriores, poner/cambiar manualmente el valor de las variables *so* y *aux* para comprobar todas las posibles opciones.
-

for do/done o for {}

A continuación se presenta un ejemplo en el que se hace uso de tres técnicas muy extendidas en Unix:

1. La salida de un comando se puede dirigir a un fichero usando el signo **>**, con lo que si el fichero no existe se creará, y si ya existe se sustituirá su contenido por la salida del comando. Usando **>>** lo que se consigue es añadir la salida del comando al final del fichero, respetando su contenido.
2. El comando **cat** es usado para mostrar el contenido de un fichero.
3. **/dev/null** es un dispositivo ficticio presente en la máquina muy útil para desechar cosas (como si fuera la papelera) o inicializar valores, archivos, ...

```
#la siguiente instrucción vacía el contenido del fichero listal.txt, o, si el
#fichero no existe, lo crea vacío
cat /dev/null>listal.txt
for i in 1 b 3
do
    echo $i >> listal.txt
done
cat listal.txt
```

La salida del ejemplo anterior es:

```
t
b
3
```

Es decir, que el bucle *for* puede activarse con una simple secuencia de valores (numéricos o alfanuméricos) enumerados en su cabecera. Es muy útil para llenar o, como se verá a continuación, recorrer ficheros.

```
for i in `cat listal.txt`
{
    echo $i
}
```

Cuya salida es:

```
1
b
3
```

Nótese que en el bucle *for*, las palabras clave *do/done* pueden ser sustituidas por llaves. ¿Pero cómo se comportaría el *for* si hubiera más de un componente por línea?

```
echo "4 casa 6">>>listal.txt
cat listal.txt
```

El contenido del fichero listal.txt es entonces:

```
1  
b  
3  
4 casa 6
```

```
for i in `cat lista1.txt`  
echo $i
```

```
1  
b  
3  
4  
casa  
6
```

Es decir, el *for* interpreta siempre los blancos como separador de valores, por lo que dentro del *for* no hay forma de distinguir qué valores estaban en la misma línea, y cuáles en líneas separadas. La solución a este posible problema pasa por utilizar el bucle *while* en vez del *for*.

while do/done

Usando el fichero *lista1.txt* anterior es sencillo ver cómo el *while* sí que distingue entre líneas diferentes, ya que cada línea se lee entera de una vez:

```
cat lista1.txt|while read i do  
echo $i  
done
```

De lo que se obtiene:

```
1  
b  
3  
4 casa 6
```

A continuación se expone un ejemplo más complejo usando el comando *expr*. El comando *expr* se usa para hacer operaciones aritméticas. Es muy importante poner espacios en blanco entre las variables, los valores y los operadores.

```
echo "hola adios">>lista2.txt
echo "zumo cola">>>lista2.txt
```

Luego, al hacer un `cat lista2.txt` se obtendrá:

```
hola adios
zumo cola
```

```
j=0
cat lista2.txt|while read i
do
    j='expr $j + 1'
    echo "El contenido de la linea $j es: $i"
done
```

Cuyo resultado es:

```
El contenido de la linea 1 es: hola adios
El contenido de la linea 2 es: zumo cola
```

También, al igual que en el caso del `if`, se puede preguntar por una condición que podrá ser simple o compuesta. Mientras la condición resulte ser verdadera, se ejecutará el contenido del bucle:

```
j=0
i=0
while [ $j -lt 10 -a $i -ne 4 ]
do
    j='expr $j + 1'
    if [ $j -le 2 -o $j -gt 7 ]; then
        i='expr $i + 1'
    fi
    echo $j $i
done
echo "Sale con j=$j e i=$i"
```

La salida por pantalla será:

```
1 1
2 2
3 2
4 2
5 2
6 2
7 2
8 3
9 4
Sale con j=9 e i=4
```

-
- **Actividad 3.5:** Variar los valores y operadores impuestos en las condiciones hasta comprobar que se entiende perfectamente el comportamiento del bucle *while*.

Para salir de una estructura de control de flujo, basta con escribir *break*:

```
for i in 1 2 3 4 5 6
do
    if [ $i -eq 4 ]; then
        break
    fi
    echo $i
done
```

De lo que se obtiene:

```
1
2
3
```

```
j=0
i=0
while [ $j -lt 10 -a $i -ne 4 ]
do
    j=`expr $j + 1`
    if [ $j -gt 7 ]; then
        break
    else
        i=`expr $i + 1`
    fi
    echo $j $i
done
echo "Sale con j=$j e i=$i"
```

Con resultado:

```
1 1
2 2
3 3
4 4
Sale con j=4 e i=4
```

3.2.2. Comandos Unix

Esta sección no pretende dar una explicación detallada de todos los comandos del Unix, sino más bien una orientación acerca del uso de aquellos comandos que por su naturaleza se usan más a menudo en las tareas administrativas de las bases de datos.

Como se verá, un mismo objetivo puede ser alcanzado de muchas maneras, en función del comando que se utilice, y todas estarán bien, pero no todas serán igual de eficientes. A la hora de elegir una forma u otra habrá que inclinarse por aquella solución más breve en cuanto a uso de comandos. Aquí no se hace tanto hincapié en ese hecho, ya que los ejemplos presentados pretenden ilustrar el funcionamiento de cada comando buscando puntos comunes entre todos ellos, para así asegurar mejor la comprensión del funcionamiento de cada comando.

Comandos de propósito general:

- **cat:** Muestra el contenido de un fichero. Es muy utilizado en KShell para proporcionar la entrada a otros comandos.
- **more:** También muestra el contenido de un fichero, pero, a diferencia del *cat* que ofrece el contenido de golpe, el comando *more* lo presenta a cachos, solo lo que quepa en pantalla cada vez, siendo la tecla de la barra espaciadora la encargada de pasar a la siguiente pantalla.

- **mkdir:** Crea un directorio.

Ejemplo para crear un directorio, incluyendo los directorios padre previos si estos no existiesen:

```
mkdir -p tmp/prueba/creacion/encadenada
```

- **touch:** Crea un fichero vacío:

```
touch tmp/para_borrar_0.txt
touch tmp/para_borrar_1.txt
touch tmp/prueba.txt
```

- **rm:** Borra ficheros:

```
rm tmp/para_borrar_0.txt
```

Admite caracteres comodín para determinar los ficheros a borrar:

```
rm tmp/para_b*.txt
```

- **ls:** Lista información sobre los ficheros dentro de un determinado directorio.

Para listar con notación larga (*l*) y ordenados por fecha de modificación (*t*) del más antiguo al más moderno (*r*), todos los ficheros con extensión *txt* hallados en el directorio *tmp* habría que usar:

```
ls -ltr tmp/*.txt
```

Para listar solo los nombres de los ficheros en columna:

```
ls -1 tmp/*.txt
```

Para listar el contenido de un directorio y de sus subdirectorios, y escribir la salida en un fichero:

```
ls -R tmp>>tmp/prueba.txt
```

- **chmod:** Otorga permisos sobre un fichero o directorio. Por ejemplo, para dar permisos de lectura, escritura y ejecución sobre un fichero para todos el mundo (el propietario del fichero, el grupo primario al que pertenece el propietario del fichero y el resto de usuarios de la máquina):

```
ls -l tmp/prueba.txt  
chmod 777 tmp/prueba.txt  
ls -l tmp/prueba.txt
```

Se puede usar con la opción *R* para hacer el cambio en todo la jerarquía de ficheros y directorios que cuelgan de un directorio concreto:

```
chmod -R 777 tmp/prueba.txt
```

- **chown:** Modifica el usuario propietario o grupo de un fichero o directorio. También se puede usar con la opción *R*. Sintaxis:

```
chown usuario:grupo fichero_o_directorio
```

- **su -:** Cambia de usuario de conexión. Con la opción *c* es posible usarlo en los scripts para ejecutar un comando en el entorno de otro usuario, pero ¡jojo!, si no se es un superusuario, pedirá la password. Sintaxis:

```
su - usuario -c "comando"
```

- **cd:** Permite moverse entre directorios. El punto simple representa el directorio actual. Los dos puntos hacen referencia al padre del directorio actual.

```
cd tmp/prueba/creacion/encadenada
```

- **pwd:** Informa del directorio actual.

- **id:** Informa del usuario propietario de la sesión de máquina actual.

- **mv:** Renombra ficheros o directorios, lo cual puede implicar cambio de ubicación.

Para cambiar el nombre de un directorio:

```
mv tmp/prueba/creacion tmp/prueba/renombrado
```

- **cp:** Copia ficheros o directorios.

```
cp tmp/prueba.txt /tmp/prueba/renombrado/prueba.txt
```

- **ps:** Muestra los procesos actuales en máquina.

```
ps -ef
```

- **kill:** Mata procesos en máquina. Para ello, antes que nada, se deberá averiguar primero el identificador de proceso que se quiere matar (segunda columna mostrada en la salida del comando *ps -ef*).

- **rcp o scp:** Copia desde o hacia un sistema remoto. Precisa del intercambio previo de claves públicas y privadas entre las máquinas y usuarios protagonistas. Sintaxis:

```
rcp objeto_de_la_copia usuario_remoto@host_remoto:destino_de_la_copia
```

- **rsh o ssh:** Ejecución en remoto de un comando, script... Al igual que el comando *rcp*, precisa del intercambio previo de claves públicas y privadas entre las máquinas y usuarios protagonistas. Sintaxis:

```
rsh host_remoto -n -l usuario_remoto "comando_a_ejecutar"
```

- **|:** No es un comando propiamente dicho. Sirve para concatenar comandos de modo que la salida del primero se considere la entrada al segundo.

- **tee:** En combinación con el **|**, lleva la salida del comando anterior a un fichero a la vez que la muestra por pantalla. Muy usado en la invocación del script. Sintaxis:

```
script_a_ejecutar|tee script_a_ejecutar.log
```

- **grep:** Filtra los registros de entrada según un patrón establecido.

Por ejemplo, para filtrar de entre todos los procesos arrancados en la máquina, solo los referidos al usuario *pepito*, basta con ejecutar:

```
ps -ef|grep pepito
```

Para filtrar un *ls -R* de modo que solo aparezcan los ficheros o directorios que empiecen por la letra *p*:

```
ls -R|grep ^p
```

Para seleccionar solo los que acaben en *.txt*:

```
ls -R|grep txt$
```

Aunque es cierto que estos dos últimos ejemplos duplican las funciones del *ls*, ya que bastaría con ejecutar *ls -R p** y *ls -R *txt* respectivamente, no es menos cierto que cuando el contenido del directorio es demasiado grande, los patrones de búsqueda del *ls* no pueden ser aplicados porque agotan la memoria destinada para ello, y en ese caso, el *grep* sería una buena alternativa.

Para filtrar todos los elementos de un *ls* que empiezan por *p* o *t*, o *0*, o *1*, o *2*, o *3*:

```
ls|grep "[p,t,0-3]
#o también
ls|grep -e ^p -e ^t -e ^0 -e ^1 -e ^2 -e ^3
```

Para obtener precisamente lo contrario (todos los elementos de un *ls* que no empiezan por *p* ni *t*, ni *0*, ni *1*, ni *2*, ni *3*):

```
ls|grep -v "[p,t,0-3]
```

Para averiguar si existe alguna base de datos Oracle arrancada en la máquina:

```
ps -ef|grep pmon
```

Para averiguar si existe alguna base de datos DB2 arrancada en la máquina:

```
ps -ef|grep db2sysc
```

Para filtrar sin tener en cuenta mayúsculas ni minúsculas:

```
ps -ef|grep -i DB2sysc
```

- **echo:** Muestra una línea de texto. Si el texto va sin comillas o entre dobles comillas es capaz de sustituir una variable referenciada con el \$ por su valor. Si va entre comillas simples escribe el texto literalmente.

Ejemplos:

```
saludo=hola
echo saludo
echo $saludo
echo "$saludo"
echo '$saludo'
```

Que arroja por pantalla:

```
saludo
hola
hola
$saludo
```

También permite montar una línea de texto ejecutando comandos con la tilde invertida:

```
echo "Los ficheros que acaban en txt son: `ls tmp/*.txt`"
```

- **xargs:** Usado junto con `|`, ejecuta el comando escrito a continuación pasándole por parámetro la salida del comando anterior (ignorando las líneas en blanco).
- `ls tmp/*.txt|xargs echo "Los ficheros que acaban en txt son: "`
- **sh:** Ejecuta el resultado del comando anterior. Es muy útil para ejecutar el resultado de sentencias montadas sobre la marcha, por ejemplo, para borrar los ficheros del ejemplo anterior:

```
cd tmp
ls *.txt|xargs echo "rm "|sh
cd ..
```

Pero habría sido más óptimo hacerlo simplemente con:

```
cd tmp
ls *.txt|xargs rm
cd ..
```

- **find:** Es un potentísimo buscador, que permite, entre otras muchas cosas, buscar incluso en el interior de los ficheros.

Para listar cualquier fichero con extensión `.txt` que se encuentre a partir del directorio actual (representado por el punto):

```
find . -name *.txt
```

Para listar cualquier fichero con extensión `.txt` que se encuentre a partir de un directorio llamado `prueba` y que cuelgue de cualquier directorio derivado del actual:

```
find */prueba -name "*.txt"
```

Para ejecutar la búsqueda del literal `hola` dentro de cada fichero encontrado:

```
find . -name "*.txt" -exec grep "hola" {} \;
```

En el listado anterior no aparece el directorio donde reside cada fichero encontrado, para remediarlo:

```
find .. -name "*.txt" -exec grep "hola" {} \; -ls
```

Para borrar todos los ficheros con extensión `.txt` hallados a partir del directorio actual:

```
find . -name "*.txt" -exec rm {} \;
```

Para borrar todos los ficheros con extensión *.txt* hallados a partir del directorio actual y que hace más de siete días que no se modifican:

```
find . -name "*.txt" -mtime +7 -exec rm {} \;
```

- **sort:** Ordena alfabéticamente las líneas de un fichero. Es muy socorrido usarlo para ordenar directamente la salida de otros comandos (mediante la combinación con *|*).

Por ejemplo, para ordenar un *ls* basta con ejecutar:

```
ls|sort
```

Para ordenar el contenido de un fichero eliminando las líneas duplicadas se puede usar la opción *-u*:

```
sort -u tmp/prueba.txt
```

- **wc:** Cuenta las líneas (*l*), palabras (*w*) y bytes (*b*) de un fichero.

Para contar las líneas:

```
wc -l tmp/prueba.txt
```

Para averiguar si un fichero tiene duplicados simplemente habrá que comprobar que *sort -u fichero|wc -l* es menor que *sort fichero|wc -l*:

```
cat /etc/group>>tmp/prueba.txt  
cat /etc/group>>tmp/prueba.txt  
echo "El número de registros del fichero es: `cat tmp/prueba.txt|wc -l`"  
echo "El número de registros del fichero sin duplicados es: `sort -u tmp/prueba.txt|wc -l`"
```

- **tail:** Presenta las últimas líneas de un fichero.

Por ejemplo, para mostrar las últimas 10 líneas:

```
tail -10 tmp/prueba.txt
```

Para mostrar, en tiempo real, la última línea generada en un fichero (de este comando se sale con tecla *Ctrl + c*):

```
tail -f tmp/prueba.txt
```

- **head:** Presenta las primeras líneas de un fichero.

Por ejemplo, para mostrar las primeras 10 líneas:

```
head -10 tmp/prueba.txt
```

-
- **Actividad 3.6:** Crear un fichero y rellenarlo con comandos Unix escritos en distintas líneas. Con la ayuda del *tail*, el *head* y el *sh*, idear un programa que vaya recorriendo el fichero y ejecutando cada línea por separado.
-

- **sleep:** Mantiene la ejecución en espera un tiempo determinado en segundos.
- **date:** Da la fecha del sistema. Muy útil para poner trazas en los programas.
- **mail:** Manda mensajes de correo electrónico. Para poder usarlo, debe estar configurado en la máquina el servicio de correo. Sintaxis:
`mail -s Asunto dirección_destino <fichero_donde_se_guarda_el_contenido_del_mensaje`
- **logrotate:** Sirve para controlar el almacenamiento de ficheros (tipicamente de log). Permite comprimirlos automáticamente, cambiarlos de ubicación, mandar mensajes... según el patrón definido previamente en un fichero de configuración.
- **diff:** Compara el contenido de dos ficheros línea a línea.

Comandos para tratar texto:

- **sed:** Modifica un texto según un patrón buscado. Precisa de un texto de entrada de dónde se extraerán las líneas a procesar, y arroja por el standard output (que, como se ha visto anteriormente, podrá dirigirse a un fichero usando los signos `>` o `>>`) las líneas ya procesadas con las sustituciones que correspondan. Las opciones de búsqueda de patrón son similares a las usadas en el editor *vi* para reemplazar texto.

Por ejemplo, para sustituir todos los *hola* encontrados en *fichero_entrada* por la palabra *adios*:

```
sed "s/hola/adios/g" fichero_entrada > fichero_salida
```

El comando *sed* puede ser utilizado con dobles comillas o con comillas simples, y al igual que el *echo*, esto servirá para indicar si se quiere o no que se interpreten las variables usadas en su interior.

Por ejemplo, si se definiera una variable `i=adios`, el comando anterior sería equivalente a:

```
sed "s/hola/$i/g" fichero_entrada
```

En cambio, si se utilizase el siguiente:

```
sed 's/hola/$i/g' fichero_entrada
```

se vería como los *hola* habrían sido sustituidos por *\$i* literalmente.

- **cut:** Borra o selecciona partes de cada línea de un fichero.

Por ejemplo, suponiendo que la salida del comando *date* fuese de la forma *mar 11 22:49:28 CET 2011*, la ejecución de

```
date|cut -f2 -d" "
```

arrojaría el valor del mes (*mar*), que corresponde al segundo campo (-f2) hallado en la linea considerando que el separador de campo impuesto es el espacio en blanco (-d""). El separador de campo solo puede ser un carácter. Si se necesitase que fuera más completo, habría que usar el *awk* en lugar del *cut*.

Para obtener todos los grupos dados de alta en la máquina:

```
cut -f1 -d"::" /etc/group
```

Para extraer el *sid* de las bases de datos Oracle arrancadas en la máquina:

```
ps -ef|grep pmon|cut -f3 -d"_"
```

- **awk (nawk...):** En sí mismo es un lenguaje de programación completo. Es capaz de trabajar con un texto de entrada, filtrándolo, haciendo sustituciones, o montando una lógica de programación tan compleja como se necesite. Para llevar a cabo esto, es imprescindible conocer la existencia de una serie de variables que poseen significado propio dentro del *awk*:

- FS=separador de campo a la entrada (por defecto un blanco).
- RS=separador de registro a la entrada (por defecto un retorno de carro).
- OFS=separador de campo a la salida (por defecto un blanco).
- ORS=separador de registro a la salida (por defecto un retorno de carro).
- \$NF=número de campos de un registro concreto. \$0 es el registro completo, \$1 el primer campo, \$2 el segundo..., hasta \$NF que representa el último.

En la sintaxis del *awk* se distinguen dos grupos diferentes, ambos delimitados por llaves y encerrados entre comillas simples. En el primero (que está precedido por la palabra clave *BEGIN*) se deben definir, si van a tener valores distintos de los de por defecto, las variables *FS*, *RS*, *OFS* y *ORS*. En el segundo grupo es donde se sitúa la lógica programada con la referencia a los campos encontrados según los separadores definidos. En este grupo resulta imprescindible el uso de la instrucción *print*, que sirve para escribir el texto que se ofrecerá a la salida del comando *awk*.

Ejemplos:

Para pasar la primera palabra de una línea a minúsculas:

```
echo "CASA PERRO COCHE" | awk '{print tolower($1)}'
```

cuyo resultado será *casa*.

Para extraer el *sid* de las bases de datos Oracle arrancadas en la máquina:

```
ps -ef | grep pmon | awk 'BEGIN {FS="pmon_"}{print $2}'
```

Para extraer los procesos *pmon* arrancados y los usuario que los lanzaron:

```
ps -ef | grep pmon | awk '{print $1, "$NF"}'
```

- **perl:** Al igual que el *awk*, en sí mismo también constituye un lenguaje completo de programación y puede ser utilizado para implementar dentro del script su propia parte de la lógica de negocio.

Ejemplo: para extraer la fecha de última modificación del fichero *tmp/prueba.txt* (la que refleja el *ls -l tmp/prueba.txt*) basta con ejecutar:

```
perl -e '$d=localtime ((stat(shift))[9]); printf "%04d-%02d-%02d %02d:%02d\n", $d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' tmp/prueba.txt
```

Comando para pedir ayuda:

- **man:** Ofrece ayuda del comando solicitado. Debe ser el primer punto de referencia para resolver cualquier duda encontrada, a la hora de trabajar con un comando de Unix o simplemente para ampliar el conocimiento sobre su uso, opciones...

Trucos de ejecución:

- Tal y como se explicó al tratar el *nohup*, al correr un comando o un script, puede ser muy útil dirigir el resultado de la ejecución a un fichero de salida y los posibles errores producidos a otro fichero. Lo primero se consigue mediante el uso del ya visto signo **>**. Lo segundo se indica definiendo dónde se quiere escribir la salida de los errores (salida 2).

Sintaxis:

```
comando_o_script>fichero_salida 2>fichero_errores
```

Si se prefiriese que los errores se volcasen al mismo fichero de salida, bastaría con referirse a él como la salida 1:

```
comando_o_script>fichero_salida 2>&1
```

- Como ya se adelantó en el capítulo de métodos de ejecución, es posible montar un script directamente en línea de comandos mediante el empleo de los signos **<<** seguidos de una palabra determinada que se considerará como clave. De esta forma, se interpretará que la escritura del comando no está finalizada hasta que no vuelva a escribirse la palabra clave definida a continuación de los signos **<<**. En el momento en que eso ocurra, el comando se dará por terminado y se ejecutará completo.

```
sqlplus / as sysdba <<EOF
select 'ORACLE_SID='||lower(name) from v$database;
EOF
```

Figura 3.17: Script desde linea de comandos.

Debe recordarse que, puesto que el \$ tiene en Unix un significado propio (para acceder al contenido de las variables), es necesario *escaparlo* mediante la barra invertida si se pretende escribirlo literalmente en línea de comando.

- Para salir del script KShell anticipadamente se pueden utilizar los comandos *exit* o *return*.

El estado resultante de toda ejecución se guarda en \$?, es decir, que si de la ejecución del script se saliera con un *return* 4, al preguntar acto seguido por el valor de \$?, este valdría 4, pero jojo!, \$? guarda el estado resultante de la última ejecución, esto significa que en el momento que se ejecutara cualquier otra cosa, por ejemplo un simple *ls*, el \$? pasaría a almacenar el estado resultante de la ejecución de ese *ls*. Tipicamente, cuando los comandos de Unix han ido "bien", esto es, han listado algún elemento en el caso del *ls* o han encontrado alguna coincidencia en el caso del *grep*, etc, el valor resultante guardado en \$? es 0.

3.3. Ejemplo de scripts de automatización

3.3.1. Ejemplo sencillo

El siguiente ejemplo ilustra cómo crear un script de automatización para el backup de todas las bases de datos de un servidor MySQL.

Los requisitos del script son los siguientes:

1. Debe realizar una copia de seguridad en caliente de todas las bbdd menos de *information_schema* con la utilidad *mysql_dump*.
2. Debe enviar las copias de seguridad de las bases de datos de menos de 10 Megabytes a un usuario de correo electrónico que se pasa como parámetro al script.
3. El script debe recibir los parámetros de conexión a la base de datos (usuario, password, host). Debe recibir además, el directorio destino donde almacenar los backups.
4. El resultado del backup deberá ser comprimido mediante la utilidad *gzip* y almacenar el fichero con el nombre *bbdd_fecha_host.gz*.

Solución

El uso del script será `copia_mysql.ksh usuario password host destino email`. El parámetro \$1, \$2 y \$3 servirá para conectarse a la base de datos con las utilidades *mysql* y *mysql_dump*; el parámetro \$4 indicará el directorio destino donde se almacenarán las copias (debe tener permisos) y el parámetro \$5 tendrá la cuenta de correo electrónico donde se enviarán las copias de seguridad. Mediante el condicional `if / $# -ne 5 /; then` se comprueba que se han pasado un número correcto de parámetros. Después, mediante el condicional `if / ! -d $destino /; then` se prueba la existencia del directorio. Si cualquiera de estas dos condiciones se cumple, el script finaliza retornando el valor 1. Si el número de parámetros es correcto y existe el directorio destino se invoca a la utilidad *mysql* en

modo batch para obtener la lista de bases de datos mediante `mysql -u \$usuario -h \$host -p\$password -Bse 'show databases'`. Esta lista de base de datos se recorre mediante un for, donde, por cada una de ellas se invoca al comando mysql_dump para efectuar el backup correspondiente (si la base de datos correspondiente no es information_schema). El dump generado por mysql_dump se envía a través de un pipe a gzip para que lo comprima. Después, se comprueba el tamaño de la base de datos mediante `ls -l \$fichero | awk 'print \$5'` y, si el tamaño es menor del establecido en la variable tam_max_email, se envía por correo como adjunto. Para enviar el correo como adjunto se utiliza uuencode, que codifica el fichero para ser enviado por correo electrónico mediante la utilidad mailx.

```

#!/usr/bin/ksh
#propósito: Genera un backup de todas las BBDD del servidor mysql, mandando cada copia de
#seguridad comprimida por correo electrónico.

#tamaño máximo para envío de emails en MB
tam_max_email=10

if [ $# -ne 5 ]; then
    echo "Error, el número de parámetros pasados debería ser 5, pero es \"$num"
    echo "Uso: copia_mysql.ksh usuario password host destino email"
    return 1
else
    #preparamos unas cuantas variables para proceder a realizar los backups
    usuario=$1
    password=$2
    host=$3
    destino=$4
    email=$5
    fecha=`date +"%d%b%Y"`
    if [ ! -d $destino ]; then
        echo "El directorio $destino no existe"
        return 1
    else
        #obtener la lista de bases de datos
        for bbdd in `mysql -u $usuario -h $host -p$password -Bse 'show databases'`
        {
            if [ $bbdd = "information_schema" ]; then
                echo "saltando catalogo..."
            else
                fichero="$destino/${bbdd}_${fecha}_${host}.gz"
                echo "Realizando backup de $bbdd en $fichero"
                mysqldump -u $usuario -h $host -p$password $bbdd | gzip -9 > $fichero

                #se envía por correo si el tamaño no supera el tamaño máximo definido
                tam_actual=`ls -l $fichero | awk 'print $5'`
                #tam_actual en megas
                tam_actual=`expr $tam_actual / 1024 / 1024`
                if [ $tam_actual -lt $tam_max_email ]; then
                    echo "enviando backup de $bbdd a $email"
                    uuencode $fichero $fichero | mail -s "Backup $fecha $bbdd" $email
                fi
            fi
        }
    fi
fi
return 0

```

Para invocar al script, primero, hay que darle permisos de ejecución, por ejemplo, con el comando `chmod 700 copia_mysql.ksh`, y a continuación:

```
$ sudo ./copia_mysql.ksh root root localhost /backups
root@alumno-desktop saltando catalogo...
Realizando backup de jardinería en /backups/jardineria_18042011_localhost.gz
enviando backup de jardinería a root@alumno-desktop
Realizando backup de mysql en /backups/mysql_18042011_localhost.gz
enviando backup de mysql a root@alumno-desktop
Realizando backup de nba en /backups/nba_18042011_localhost.gz
enviando backup de nba a root@alumno-desktop
Realizando backup de phpmyadmin en /backups/phpmyadmin_18042011_localhost.gz
enviando backup de phpmyadmin a root@alumno-desktop
Realizando backup de primeroasir en /backups/primeroasir_18042011_localhost.gz
enviando backup de primeroasir a root@alumno-desktop
```

3.3.2. Ejemplo avanzado

En un entorno Unix, lanzar en paralelo unas sentencias SQL a un conjunto de bases de datos (o instancias) suministradas en un fichero. Si el fichero estuviese vacío o no existiese, salir con error.

En función de la salida anterior, ejecutar un programa de monitorización, es decir, un script espía que controle la ejecución anterior y que decida, si han pasado más de 30 minutos y no han acabado, matar todos los procesos no finalizados que sigan en máquina.

Mandar un mensaje con el resultado final.

Premisas:

1. Los scripts se ejecutarán con un usuario que tenga privilegios tales que no precise suministrar la password al intentar autenticarse como los usuarios propietarios de las instancias.
2. Los usuarios propietarios de las instancias cargan las variables de su entorno Oracle directamente desde su `.profile`.

Solución

El ejemplo se resuelve con la implementación de tres script separados que se invocan entre ellos. La separación en programas diferentes normalmente surge a partir del análisis de las tareas que son independientes y/o reutilizables, o que pueden ser agrupadas por ser repetitivas, o simplemente, por limpieza y pulcritud en la programación.

A tal efecto se crea un script maestro (llamado `lanza_todo.ksh`) que invocará a los otros dos (`lanza_sentencia.ksh` y `lanza_espia.ksh`). En él se rellena el fichero con las bases de datos (`fichero_con_bd.txt`) donde se lanzarán las sentencias SQL deseadas. En este caso, las bases de datos escogidas son las relacionadas con las tres primeras instancias que aparecen en la salida del comando `ps -ef|grep pmon`.

Una vez creado el `fichero_con_bd.txt`, el `lanza_todo.ksh` invocará al script diseñado para ejecutar las sentencias SQL (`lanza_sentencia.ksh`) en las bases de datos suministradas

en el fichero creado anteriormente (el *fichero_con_bd.txt*). Se evaluará la salida resultado de la ejecución del *lanza_sentencia.ksh* para determinar si hubo algún error o no en su ejecución, y en caso afirmativo, se actuará en consecuencia ofreciendo el correspondiente mensaje de error. En caso de que no haya ningún error, se procederá a invocar el script de monitorización (*lanza_espia.ksh*). Al igual que antes, se evaluará su salida y si todo ha terminado sin necesidad de matar ningún proceso, se examinarán los ficheros de log de las conexiones a cada instancia (llamados */tmp/sentencia.ksh<sid>*), para extraer el resultado de las sentencias SQL lanzadas en cada base de datos.

Se considerará que las sentencias SQL han sido resueltas con éxito, si en cada fichero de log aparece al menos una línea prefijada con la palabra *FILTRO* (como ya se verá, es un literal impuesto en todas las sentencias SQL programadas en el script *lanza_sentencia.ksh*), lo cual indicará que la conexión a la base de datos ha sido posible y que al menos una sentencia SQL ha arrojado algún valor.

Con todos los resultados de las sentencias SQL recopilados o todos los errores obtenidos, se monta un mensaje que será enviado por correo electrónico a la dirección *root@alumno-desktop*.

El script *lanza_sentencia.ksh* comprueba la existencia y el contenido del primer fichero pasado por parámetro que se supone contiene las instancias en las que se ejecutarán las sentencias SQL objeto de este ejemplo (el *fichero_con_bd.txt* del *lanza_todo.ksh*). Este script admite también otros dos parámetros: el nombre del fichero (referenciado por la variable *fichaux*), que servirá para guardar el script con las sentencias SQL que se lanzarán en las bases de datos y el directorio (variable *dirtrab*) donde se dejarán tanto el *fichaux*, como el log de su ejecución en cada base de datos. De la llamada que se hace dentro del *lanza_todo.ksh* al *lanza_sentencia.ksh* es fácil comprobar que *fichaux=sentencia.ksh* y *dirtrab=/tmp/*. Puesto que el nombre de los ficheros de log resulta de concatenar el *sid* de la instancia con el *fichaux* dentro del *dirtrab*, se puede concluir que los ficheros de log son los ya citados */tmp/sentencia.ksh<sid>*.

El *fichaux* se rellena con el código correspondiente de un script KShell, que abre una sesión de *sqlplus* y ejecuta tres consultas. A la salida de cada una de las sentencias *select* se le añade el prefijo *FILTRO* con dos propósitos bien distintos: por un lado servirá para determinar con facilidad si la conexión ha tenido éxito y se ha llegado a resolver alguna sentencia, y por otro será muy útil para filtrar las respuestas a las consultas, ya que serán respuestas solo las líneas que a su comienzo tengan la palabra *FILTRO*.

Merece la pena meditar acerca del hecho de que el script con las sentencias SQL (*fichaux* o *sentencia.ksh*) se lanza para cada instancia con *nohup* y en *background*, permitiendo de esta manera ejecutarlo para todas las instancias en paralelo.

El script *lanza_espia.ksh* acepta como parámetro de entrada (almacenado en la variable *script_a_monitorizar*) el nombre del programa cuyo tiempo de ejecución debe monitorizarse. En este ejemplo, los scripts que pretenden monitorizarse son los *fichaux*, es decir, el *sentencia.ksh* que aparece en *lanza_todo.ksh*. Si llevan más de 30 minutos en máquina, el *lanza_espia.ksh* los mata y sale con un valor distinto de 0.

```

#!/usr/bin/ksh
#lanza_todo.ksh | tee lanza_todo.log 2>&1
#Propósito: invoca el lanzamiento de las sentencias, la monitorización y manda
#un mensaje de correo informando del resultado

#Inicialización del fichero que servirá para mandar los mensajes
cat /dev/null>mensaje.txt
#Como ejemplo, se seleccionan las tres primeras bases de datos Oracle que se
#encuentren levantadas en la máquina, es decir, que tengan arrancado su
#proceso ora_pmon_sid
#Obtención del sid:
ps -ef|grep ora_pmon_|grep -v grep|awk '{BEGIN {FS="_"}}{print $NF}'|head -3>fichero_con_bd.txt
lanza_sentencia.ksh fichero_con_bd.txt sentencia.ksh /tmp/ |tee lanza_sentencia.log 2>&1
res=$?
if [ $res -eq 0 ]; then
    #Se lanzaron correctamente los scripts de sentencias
    lanza_espia.ksh sentencia.ksh|tee lanza_espia.log 2>&1
    res=$?
    if [ $res -eq 0 ]; then
        #Todo ha terminado sin necesidad de matar ningún proceso
        #Se recogen los resultados de las sentencias de los Ficheros
        #/tmp/sentencia.ksh$! para mandar un mensaje con todos ellos
        #Al mismo tiempo se analiza si el resultado obtenido de cada instancia
        #es correcto, es decir, si las sentencias se resolvieron con éxito
        for i in `cat fichero_con_bd.txt`
        do
            echo "">>mensaje.txt
            echo "INSTANCIA: \"$i\">>mensaje.txt
            #La conexión a la base de datos tuvo éxito si se contestó a las
            #sentencias, y esto es así siempre que la primera palabra del resultado
            #de las sentencias sea FILTRO
            grep '^FILTRO' /tmp/sentencia.ksh$!>/dev/null
            res=$?
            if [ $res -eq 0 ]; then
                #Se encontró la palabra FILTRO al comienzo de alguna línea dentro del
                #fichero de resultados de la instancia $!
                #Se recogen solo los resultados de las sentencias, los cuales están
                #aparecidos de la palabra FILTRO
                #Se quita la palabra FILTRO porque no aporta nada en el mensaje final
                grep '^FILTRO' /tmp/sentencia.ksh$!|sed "s/^FILTRO://">>>mensaje.txt
            else
                echo "Sentencias sin resultado.¡Problemas de conexión!">>mensaje.txt
                echo "Revisar el fichero de log /tmp/sentencia.ksh$!">>mensaje.txt
            fi
        done
    else
        #Ha habido que matar procesos
        echo "Error, ha habido que matar procesos. Revisar los ficheros de log." >>mensaje.txt
        echo "lanza_espia.log y \"`ls /tmp/sentencia.ksh*`\">>mensaje.txt
    fi
else
    echo "Error, no se han podido lanzar las sentencias.">>mensaje.txt
    echo "Revisar el fichero de log lanza_sentencia.log">>mensaje.txt
fi
cat mensaje.txt
mail -s Resultado root@alumno-desktop < mensaje.txt

```

Figura 3.18: lanza_todo.ksh.

```

#!/usr/bin/ksh
#lanza_sentencia.ksh fichbd fichaux dirtrabajo > lanza_sentencia.log 2>&1
#Propósito: lanza en paralelo a un conjunto de bases de datos suministradas en
#un fichero, unas sentencias sql. Si el fichero estuviese vacío o no existiera,
#sale con error.

num=$#
if [ $# -ne 3 ]; then
    echo "Error, el numero de parametros pasados debería ser 3, pero es \"$num"
    echo "Uso: lanza_sentencia.ksh fichbd fichaux dirtrabajo > lanza_sentencia.log 2>&1"
    return 1
else
    fichbd=$1
    fichaux=$2
    dirtrab=$3
    if [ ! -f $fichbd ]; then
        echo "Error, el fichero $fichbd no existe."
        return 2
    elif [ `cat $fichbd|wc -l` -eq 0 ]; then
        echo "Error, el fichero $fichbd esta vacío."
        return 3
    fi
    #Se podrían validar todas las condiciones, por ejemplo, que el
    #fichero_auxiliar no existiera ya en el directorio_trabajo, que el
    #directorio_trabajo acabara en /, que tuviese permisos de ejecución para
    #todo el mundo, que tuviese espacio libre suficiente...
fi

#Con la ayuda de un script auxiliar, se monta en $directorio_trabajo
#la ejecución de las sentencias
echo "#!/usr/bin/ksh">>$dirtrab$fichaux
echo "sqlplus / as sysdba <<EOF">>>$dirtrab$fichaux
echo "select 'FILTRO:BD='||name from v$database;">>$dirtrab$fichaux
echo "select 'FILTRO:ESTADO='||status from v$instance;">>$dirtrab$fichaux
echo "select 'FILTRO:VERSION='||banner from v$version;">>$dirtrab$fichaux
echo "EOF">>$dirtrab$fichaux

#el prefijo FILTRO servirá para formatear la salida dejando solo las líneas de interés

#se le dan permisos de ejecución para todo el mundo puesto que lo ejecutara
#el propietario de cada instancia
chmod 777 $dirtrab$fichaux

for i in `ps -ef|grep ora_pmon_|grep -v grep|awk '{print $1 ":"$NF}'`
{
    usu=`echo $i|cut -f1 -d":"`
    sid=`echo $i|cut -f2 -d": "|cut -f3 -d"_"`"
    echo $usu"----"$sid

    nohup su - $usu -c "$dirtrab$fichaux >$dirtrab$fichaux$sid 2>&1" &
}

return 0

```

Figura 3.19: lanza_sentencia.ksh.

```

#!/usr/bin/ksh
#lanza_espia.ksh script_a_monitorizar> lanza_espia.log 2>&1
#Propósito: es un programa espía que monitoriza la ejecución de los
#script_a_monitorizar y, si llevan más de 30 minutos en máquina, los mata y
#sale del script con un valor distinto de 0.

script_a_monitorizar=$1

#cada minuto comprueba si siguen los script en máquina
#lleva un contador de modo que cuando se alcance la media hora,
#mata los procesos que quedan todavía
contador=0
ps -ef|grep $script_a_monitorizar|grep -v grep>/dev/null
res=$?
while [ $res -eq 0 -a $contador -lt 30 ]
do
    echo "Se espera un minuto"
    sleep 60
    ps -ef|grep $script_a_monitorizar|grep -v grep>/dev/null
    res=$?
    contador=`expr $contador + 1`
done
if [ $contador -ge 30 ]; then
    echo "Hay que matar procesos"
    #se buscan los procesos que queden en máquina, se extrae su pid y se matan
    ps -ef|grep $script_a_monitorizar|grep -v grep|for i in `awk '{print $2}'`"
    do
        echo "Proceso matado: $i"
        kill $i
    done
    return 1
fi

return 0

```

Figura 3.20: lanza_espia.ksh.

Una vez creados en máquina los tres scripts que resuelven el ejemplo, deben otorgárseles permisos de ejecución como mínimo para el usuario propietario:

```

#permisos de rwx para el usuario propietario
#permisos de r-- para su grupo y el resto
chmod 744 lanza_*,ksh

```

o **Actividad 3.7:** Modificar el ejemplo para que tome las sentencias SQL a ejecutar de un fichero externo. Cambiar las sentencias SQL propuestas por la de lanzar estadísticas de la base de datos completa.

o **Actividad 3.8:** Modificar el ejemplo para que coja por parámetro el tiempo para tomar la decisión de matar los procesos.

3.4. Disparadores de sistema

En una base de datos es imprescindible mantener la integridad de la información almacenada para que toda ella sea consistente en un momento concreto en el tiempo. Asegurar esa integridad es posible y para ello puede recurrirse a varios tipos de programaciones.

1. La propia aplicación que ataca la base de datos implementando la lógica de negocio determinada, es decir, que dentro del aplicativo, toda acción de crear, borrar o modificar cualquier registro de la base de datos debe necesariamente ir precedida de la comprobación de que el dato tratado es compatible con el resto de los datos guardados en la base de datos en ese instante de tiempo concreto.
2. Las restricciones o *CONSTRAINTS* tipo *CHECK*, o restricciones de valor único, claves primarias y foráneas...
3. Los disparadores o *TRIGGERS* que pueden llegar a ser programas complejísimos donde se tengan en cuenta infinidad de circunstancias de la lógica de negocio.

Los disparadores se ejecutan en función de determinadas acciones llevadas a cabo sobre tablas o vistas, o también, solo en el caso de Oracle, sobre la base de datos en general, a través de las ocurrencia de eventos acaecidos sobre la misma.

Dejar en manos de la aplicación la responsabilidad de garantizar la coherencia de los datos almacenados en la base de datos es muy arriesgado, básicamente porque podrían darse errores o faltas de programación que corromperían la información. Por ejemplo, ¿cómo se comportaría la aplicación ante un fallo en la comunicación con la base de datos? Para aceptar esa responsabilidad la aplicación debería estar muy bien hecha, cuidando todos los detalles de la lógica de negocio, teniendo en cuenta todos los supuestos de mal funcionamiento, y sobre todo, desarrollando una gestión de excepciones que le permitieran asegurar que todos los errores en la manipulación de los datos son adecuadamente interceptados y tratados.

En cambio, garantizar la coherencia de los datos solo mediante restricciones y disparadores, si bien sería sinónimo de calidad, también implicaría lentitud en las operaciones de inserción, borrado o modificación de los registros, ya que probablemente cualquier paso desencadenaría una cadena de infinidad de comprobaciones previas y a posteriori.

Por ello, como suele pasar siempre, hay que llegar a un término medio entre calidad y viabilidad. Conviene usarlo todo, pero sin abusar de nada, y por supuesto, supeditar las decisiones al resultado de unas buenas pruebas de funcionamiento y carga donde se mida no solo la fiabilidad del dato, sino también los recursos consumidos para conseguirlo.

Puesto que en el primer libro de esta serie (el de *Gestión de Bases de Datos*) ya se trataron con suficiente profundidad la definición de restricciones, así como los disparadores asociados a tablas o vistas (por las acciones de inserción, actualización o borrado), en esta ocasión se expondrán con más detalle los distintos tipos de eventos que pueden activar un disparador dentro de la base de datos.

- **Actividad 3.9:** Aprende todo lo que se puede saber sobre triggers en el DB2 consultando la siguiente página web del Centro de Información de DB2 9.7:
http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/con_ibm_db2_luw_admin_dbobj.doc/doc/c0004113.html.

3.5. Eventos de sistema

Efectivamente, los disparadores son activados como consecuencia de la ejecución de un determinado evento, pero además de esto, pueden ser programados para que arranquen en tres momentos o formas distintas:

1. Antes del evento: el código del disparador se lanza antes del evento.
2. Despues del evento: el código está programado para que corra inmediatamente después del evento.
3. En vez del evento: el disparador se ejecuta en sustitución del evento.

¿Sabías que ... ? En Oracle el término evento también sirve para identificar una condición que escribirá una traza en los ficheros de traza de usuario. Es una forma muy útil de seguirle la pista a determinados errores. En DB2 la palabra evento está también asociada a los monitores de eventos, imprescindibles para la optimización de la base de datos.

Los eventos susceptibles de activar disparadores son muy variados y aunque dependen del gestor que se esté tratando, en general se pueden dividir en tres grandes grupos: DML, DDL y de base de datos. Los dos últimos (DDL y de base de datos) son también llamados disparadores de sistema.

¿Sabías que ... ? En Oracle, los disparadores de sistema pueden ser deshabilitados incluyendo el parámetro oculto `_system_trig_enabled=false` en el fichero de parámetros de la instancia (ya sea el `pfile` o el `initsid.ora`).

- **DML:** trigger data manipulation language (disparadores de lenguaje de manipulación de datos). Son los típicos que se definen en función de las opciones sidu (select, insert, delete y update) sobre tablas o columnas de tablas.
- **DDL:** trigger data definition language (disparador de lenguaje de definición de datos). El disparador entraría en juego cada vez que se realizará la operación correspondiente de lenguaje de definición de datos dentro de un esquema concreto, si este se ha especificado, o si no, respecto de la base de datos en general.

- De base de datos: trigger de base de datos.

Por lo novedoso, aquí se tratarán con más detenimiento los de bases de datos.

En Oracle, algunos de los eventos de base de datos que pueden ser capturados son:

- Logon: se activa en el inicio de sesión.
- Logoff: se activa en el finalización de sesión (desconexión). Debe tenerse en cuenta que no se activará el disparador *logoff*, por no haber desconexión literal, si la sesión es matada bruscamente, o la instancia es finalizada con *shutdown abort* o *shutdown immediate*.
- Startup: se activa en el inicio de la instancia.
- Shutdown: se activa en el parada de la instancia.
- Servererror: se activa ante cualquier error ocurrido en la base de datos que no sea el ORA-01034 (*ORACLE not available*), ORA-01403 (*no data found*), ORA-01422 (*exact fetch returns more than requested number of rows*), ORA-01423 (*error encountered while checking for extra rows in exact fetch*) y ORA-04030 (*out of process memory when trying to allocate string bytes*).

Ejemplos:

```
create table tabla_auditoria_conexion (quien varchar2(30), cuando date, que varchar2(6));

create or replace trigger trg_logon_db
  after logon on database
begin
  insert into tabla_auditoria_conexion (quien, cuando, que)
  values (user, sysdate, 'logon');
end;
/
```

Figura 3.21: Trigger logon.

```
create trigger trg_logoff_db
before logoff on database
begin
  insert into tabla_auditoria_conexion (quien, cuando, que)
  values (sys_context('userenv','session_user'), sysdate, 'logoff');
end;
/
```

Figura 3.22: Trigger logoff.

- **Actividad 3.10:** Provocar la activación de los disparadores anteriores. Comprobar que se han producido las inserciones correspondientes en la tabla *tabla_auditoria_conexiones* definida.

3.6. Excepciones servererror

Ya se ha dicho que una de las claves para mantener la integridad de los datos es ser muy cuidadoso en la programación e intercesión de las excepciones, no solo a nivel del aplicativo, sino también en el desarrollo de los disparadores.

Como ya se vio en el libro de *Gestión de Bases de Datos*, el mecanismo conocido como excepción es vital para la implementación de la gestión de errores. Como complemento a la información que en ese libro se ofrecía, aquí se expondrá, mediante un ejemplo, cómo se pueden registrar los errores a partir del evento de trigger *servererror*):

```
create table tabla_errores_capturados (
    fecha           date,
    nombre_usuario  varchar2( 30),
    terminal        varchar2(50),
    num_err         varchar2(10),
    mensaje         varchar2(4000)
);
create or replace trigger captura_errores
    after servererror on database
begin
    FOR n IN 1..ora_server_error_depth LOOP
        insert into tabla_errores_capturados values (sysdate, ora_login_user,
            ora_client_ip_address, ora_server_error(n), ora_server_error_msg(n));
    END LOOP;
end;
/
```

Figura 3.23: Trigger servererror.

Donde *ora_server_error_depth*, *ora_login_user*, *ora_client_ip_address*, *ora_server_error*, *ora_server_error_msg* son funciones propias del gestor desarrolladas para ser usadas en este tipo de disparadores.

- **Actividad 3.11:** Provocar la activación del trigger anterior (por ejemplo mediante la consulta a una tabla que no exista). Comprobar que se ha producido la inserción correspondiente en la tabla definida a tal efecto.

3.7. Prácticas

Práctica 3.1: Análisis de tablas

Se desea realizar periódicamente el análisis de las tablas de las bases de datos de un servidor MySQL para comprobar si existen problemas.

1. Crear un script que recorra todas las tablas de todas las bases de datos ejecutando para cada una de ellas el comando `analyze table esquema.tabla`.
2. El script deberá enviar por correo electrónico los problemas detectados.
3. El script se deberá ejecutar todas las noches a la 1 de la madrugada.

◦

Práctica 3.2: Parada de mantenimiento

Se va a proceder a hacer un mantenimiento de una máquina Unix, para lo cual es preciso recorrer todas las instancias de base de datos (Oracle y MySQL) levantadas en la máquina y pararlas.

Las premisas son:

- Al autenticarse con el usuario propietario de la instancia ya se tienen las variables del entorno del gestor cargadas directamente en su `.profile`.
- El script lo ejecutará root.

Se pide:

1. Crear un script llamado `parar.ksh` que pare todas las bases de datos.
2. Crear un script llamado `arrancar.ksh` que arranque de nuevo las bases de datos.

◦

Práctica 3.3: Políticas de backup

Con objeto de implementar la política de backup de una base de datos Oracle en un entorno Unix, se debe realizar y dejar planificado el script de backup de dicha base de datos, el cual se lanzará todas las noches a las tres de la madrugada.

Además, debe conocerse que la política de backup adoptada indica que el backup debe ser completo y directamente a disco, pero solo han de guardarse en la máquina los siete últimos realizados.

Codificar en ksh el script de backup.

◦

Práctica 3.4: Inventario de Bases de Datos

Por encontrarse sin garantía, se precisa cambiar un conjunto de máquinas Unix por máquinas AIX, para lo cual es necesario inventariar todas las bases de datos (Oracle y MySQL) levantadas en dichas máquinas Unix.

Resulta también de vital importancia conocer el estado en el que se encuentran (si están abiertas o solo montadas..., y si registran algún tipo de error en sus ficheros de alertas).

Los supuestos con los que habrá de abordarse el problema son:

- Existe conexión rsh entre máquinas y la conexión a las máquinas se puede hacer con root (o cualquier otro usuario con privilegios suficientes como para autenticarse como el propietario de cada instancia).
- El usuario propietario de cada instancia tiene en su .profile cargadas las variables del gestor sobre el que reside.

Se pide: crear un script llamado inventario.ksh que realice el inventario solicitado de todas las bases de datos alojadas en los servidores. Para ello el script tomará como entrada un fichero con una lista de los servidores que se quieren inventariar. En la salida del script aparecerá la siguiente información:

Nombre de máquina	Base de Datos	Tipo (Oracle /MySQL)	Estado
asterix	Contabilidad	Oracle	Abierta
.....			

Práctica 3.5: Migración de SGBD

En un entorno Unix, para ahorrar en el coste de la licencia del gestor, se ha realizado la migración de una base de datos Oracle a DB2 Express-C. Para dar por buena la migración hay que verificar que todos los objetos de usuario de la base de datos Oracle, están creados en las bases de datos destino del gestor DB2 Express-C, para lo cual se ha pensado en construir un script que exporte a fichero de texto los nombres y tipos de los objetos de ambos gestores, y luego compare los ficheros.

Construir un script llamado resultado_migracion.ksh que muestre por pantalla las discordancias de la siguiente manera:

Usuario	Tipo de Objeto	Nombre de Objeto
jardineria	Tabla	Clientes
.....		

3.8. Amplía tus conocimientos

1. Crea una cuenta en Metalink: <https://support.oracle.com/CSP/ui/flash.html>.
2. Explora todas las posibilidades de Metalink como página de soporte para BBDD Oracle.
3. Para realizar scripts en windows hay múltiples alternativas. Consulta la página <http://technet.microsoft.com/es-es/scriptcenter/dd742419.aspx> para aprender más acerca de Power Shell y la página <http://msdn.microsoft.com/en-us/library/ms950396.aspx> para ver otras alternativas en la ejecución de scripts en sistemas de tipo Microsoft Windows.
4. Consulta el Centro de Información del DB2:
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>
5. Consigue el libro "Learning the Korn Shell" de la editorial O'Reilly.
6. Busca en Internet ejemplos de los comandos estudiados y otras funcionalidades.
7. Busca, además, las diferencias entre los distintos tipos de scripts en Unix (ksh, bash, csh)...
8. Descarga las cheatsheets (chuletas) de shell script para Linux de
<http://www.cheat-sheets.org/#ShellScript>



OPTIMIZACIÓN DEL RENDIMIENTO: MONITORIZACIÓN Y OPTIMIZACIÓN

Contenidos

- » Herramientas de monitorización disponibles en un SGBD
- » Elementos y parámetros susceptibles de ser monitorizados
- » Optimización
- » Herramientas y sentencias para la gestión de índices
- » Herramientas para la creación de alertas de rendimiento

Objetivos

- » Identificar las ventajas e inconvenientes de la creación de índices
- » Optimizar la estructura de la base de datos así como los recursos usados por el sistema gestor
- » Evaluar el rendimiento de las consultas
- » Programar alertas de rendimiento

En este capítulo se explicarán las técnicas que permiten sacar el máximo provecho a una base de datos.

4.1. Introducción

Debido a la complejidad de este tema, así como a la profundidad con la que quiere tratarse, se hace inviable, por la extensión que ello supondría, elaborarlo para los tres gestores que son protagonistas a lo largo de todo este libro. Por ello se ha optado por desarrollarlo solo desde la perspectiva de Oracle.

4.2. Herramientas de monitorización de un SGBD

Monitorizar es observar algo, no obstante, se debe tener muy en cuenta que observar consume recursos, es decir, si el objetivo de la monitorización es observar el rendimiento, no sería lógico sobrecargar la base de datos con consultas de monitorización demasiado pesadas, ya que esto incrementaría el uso de recursos y podría convertirse en parte del problema que se quiere investigar. Por ello, para evitar que la monitorización llegue a ser nociva, debe meditarse muy bien con qué se monitoriza, durante cuánto tiempo se hace y cada cuánto tiempo se lanza dicha monitorización.

Muchas empresas se han dedicado a desarrollar multitud de herramientas para la monitorización de bases de datos. Se optará por el uso de unos productos u otros en función de los objetivos que se persigan, y teniendo muy presente que la herramienta más conveniente siempre será la menos intrusiva, aunque esto está supeditado en gran medida a lo que se pretenda hacer a posteriori con los datos obtenidos de la monitorización.

Por regla general, todo lo que sea de tipo gráfico consumirá más recursos y por tanto será más costoso, tanto para el sistema donde se ejecute como para la base de datos que se esté analizando. En cambio, el acceso a tablas y vistas del gestor a través de consultas realizadas desde, por ejemplo, el SQL*Plus, será mucho más granulado, puntual y básico, por lo que resulta bastante menos intrusivo, aunque todo depende de qué y cómo se esté consultando en cada momento.

Además de las herramientas específicas de base de datos, también son muy útiles comandos propios del sistema operativo donde se esté ejecutando el gestor, por ejemplo, los comandos usados para comprobar los procesos de base de datos levantados en la máquina, su consumo de recursos, la identificación del pid de las sesiones locales...

A continuación se detallan las herramientas más usadas para llevar a cabo la monitorización.

Para el caso de Oracle, la mayoría de ellas se nutren de los datos recogidos por el OWI (del inglés *Oracle Wait Interface*, interfaz en espera de Oracle), que es el modo en que se le indica a Oracle que se ponga en modo depuración, es decir, que cada vez que se haga algo quede grabado en algún sitio, por ejemplo, si se va a realizar una lectura, antes de hacerla la sesión avisa de que va a tener lugar y así, mientras se está realizando, se puede ir grabando el estado de la sesión. Siempre se puede ver ese estado consultando las vistas *v\$session* y *v\$session_wait*, pero ocurre que muchas veces la acciones que se quieren registrar duran tan poco, que o se graban por medio de una traza o no hay manera de verlas con detalle.

4.2.1. Consolas gráficas: Enterprise Manager, Grid Control, Toad...

Son herramientas muy vistosas e intuitivas que posibilitan la administración de las bases de datos a golpe de *clic* de ratón. Permiten manejar, según el caso, una o varias bases de datos desde la misma consola.

Todas parten de la misma premisa: no es necesario saberse ningún comando de memoria, con lo que se facilita y simplifica muchísimo la administración de instancias y bases de datos, con el riesgo que ello conlleva, ya que lo puede usar cualquier persona, aunque no sepa exactamente qué está haciendo en cada *clic*.

El gran inconveniente que tienen es la cantidad de recursos que consumen.

A lo largo de esta sección, aparecerán numerosas imágenes que ilustrarán sobradamente el uso del *Enterprise Manager* en las labores de monitorización.

4.2.2. Vistas dinámicas

Son las vistas del propio catálogo de Oracle. En ellas se almacena la totalidad de la información que posee el gestor.

Un ejemplo muy común que ilustra su uso en monitorización es la búsqueda de bloqueos. Dadas dos sesiones, en la sesión 1 se lanza una consulta sobre la tabla *emp* agregando la cláusula *for update* para bloquear uno o varios registros. En la sesión 2 se intenta hacer la actualización de algún registro de la tabla *emp* de los consultados en la sesión 1.

```
SQL> set lines 120 pages 5000 long 20000000
SQL> select *
from emp for update;
   ID NOM      ID_DEP      SAL
-----  -----
    4 caro        1       50
    5 otro 2      1       50
    1 john        1       40
    2 jose        2       40
    3 pepe        2       40
```

Figura 4.1: Sesión 1.

¿Sabías que ...? La forma de evitar que en SQL*Plus los registros resultantes de las consultas salgan partidos en dos líneas separadas es mediante el comando *set lines*, que sirve para definir cuántos caracteres saldrán por línea.

```
SQL> update emp  
2 set nom='JOHN'  
3 where id=1;
```

Figura 4.2: Sesión 2.

En este escenario, la sesión 2 se queda *colgada* esperando a que la sesión 1 termine su transacción, y hasta que no la finalice mediante *commit* o *rollback*, la sesión 2 no podrá continuar con su ejecución.

El bloqueo producido puede ser evaluado con una simple consulta de las vistas del catálogo *v\$session* y *v\$lock*.

```
SQL> set lines 140 pages 5000 long 2000000  
SQL> col bloqueador for a25  
SQL> col bloqueado for a25  
SQL> select (select username from v$session where sid=a.sid) bloqueador,  
2 a.sid,  
3 'BLOQUEAA',  
4 (select username from v$session where sid=b.sid) bloqueado,  
5 b.sid  
6 from v$lock a, v$lock b  
where a.block = 1  
and b.request > 0  
and a.id1 = b.id1  
and a.id2 = b.id2;  
  
BLOQUEADOR          SID 'BLOQUEAA'  BLOQUEADO          SID  
-----  
TJ                  137  BLOQUEA A  TJ                  134
```

¿Sabías que ...? Para forzar en SQL*Plus a que el número de caracteres destinados a representar una columna sea el adecuado basta con usar la directiva *col nombre_columna for xxxx*, donde *xxxx* sería el número de caracteres definidos para la columna *nombre_columna*.

También podrá verse desde la pestaña *Rendimiento* de la consola gráfica del *Enterprise Manager* en el apartado *Bloqueos de instancia*.

Sesiones de Bloqueo (bloq) > Sesiones Bloqueantes

Conectado como SYSTEM

Página Generada: 04-nov-2011 12:49:56

Ver Recalcar | Actualizar

Analiza Tabla | Elimina Tabla

Seleccionar Básico de Sesión: Bloqueadas Sólo

Nombre de Sesión de la Sesión Valor Hash de Clave de SID, Espacio de Espera P1 P2 P3 Sesiones en Espera

Sesiones Bloqueantes

TJ 158 158 SQL*Plus:oci:thin: 11116999761 0 0 0

TJ 6 6 20200000007 Application sess TJ - con lock contention 1419060046 303531509 0

Enlaces de Control Adicionales

Las sesiones principales y los datos de SQL principal de ASH aparecen en la página Actividad de Sesión Superior.

- Actividad de Sesión Superior
- Documentación Principales
- SQL Duplicado
- Análisis de Bloqueos
- Descripción de Historial
- Asignación de Recursos
- Migración Automática de Trabajo Bases de Datos
- Descripción de Historial

Figura 4.3: Bloqueos.

4.2.3. dbms_monitor

Es un paquete instalado dentro del gestor. Forma parte de las API que este incorpora, y es muy útil para conocer de forma sencilla lo que está haciendo una determinada sesión. Por ejemplo, para obtener una traza donde quede registrado qué hace un usuario concreto, basta con ejecutar los siguientes pasos:

1. Obtención del *sid* y el *serial#* del usuario para identificar su sesión. Para ello, se le pide al usuario (en este caso llamado *TJ*) que se conecte, y una vez lo ha hecho, se ejecuta la siguiente consulta:

```
SQL> set lines 120 pages 5000 long 20000000
SQL> select sid, serial#, program
  2  from v$session
  3 where username='TJ';

      SID    SERIAL# PROGRAM
-----+
      158        6 sqlplus.exe
```

2. Activación de la traza mediante el paquete dbms_monitor:

```
SQL> exec dbms_monitor.session_trace_enable(158,6,TRUE,TRUE);
Procedimiento PL/SQL terminado correctamente.
```

3. Ejecución, por parte del usuario, de la consulta o procedimiento que desea monitorizarse:

```
SQL> select * from emp;
  ID NOM          ID_DEP      SAL
-----  -----
  4 caro           1       50
  5 otro 2         1       50
  1 john           1       40
  2 jose           2       40
  3 pepe          2       40
```

4. Desactivar la monitorización para finalizar la creación del fichero de trazas:

```
SQL> exec dbms_monitor.session_trace_disable(158,6);
Procedimiento PL/SQL terminado correctamente.
```

5. Examinar la traza. Se buscará el fichero de traza que tenga la fecha de la monitorización realizada en el directorio al que apunte el parámetro `user_dump_dest` de la instancia (para verlo basta con ejecutar: `show parameter user_dump_dest`) si es versión 10g o, si es versión 11g, usando el ADRCI que se explicará más adelante. En los sistemas Unix el nombre del fichero contiene el número de proceso correspondiente a la sesión que se está monitorizando. En Windows hace referencia al número de hilo.

Existe un truco para encontrar la traza más fácilmente que consiste en ponerle un identificador como sufijo, esto hay que hacerlo modificando un parámetro de la sesión antes de la generación de la propia traza:

```
SQL> alter session set tracefile_identifier='sufijo_deseado';
Sesión modificada.
```

De modo que la traza podría ser un fichero llamado por ejemplo `test2_ora_5416_sufijo_deseado.trc`.

Una traza tiene el siguiente aspecto:

```
Dump file c:\oracle\product\10.2.0\admin\test2\udump\test2_ora_5040.trc
Mon Jun 8 11:22:54 2011
ORACLE 10.2.0.4.0 - Production version=0
svrsql=14 svrstr=3
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Windows XP Version 5.1 Service Pack 3
CPU : 2 - type 646, 2 Physical Cores
Process Affinity : 00000000
Memory (Avail/Total): Ph:2132M/3573M, PgF:2887M/3486M, VA:1344M/2047M
Instance name: test2

Redo thread mounted by this instance: 1
```

```

Oracle process number: 13
Windows thread id: 5040, image: ORACLE.EXE (SHAD)

*** 2011-01-08 11:22:54.567
*** ACTION NAME:( ) 2011-01-08 11:22:54.578
*** MODULE NAME:(SQL+Net) 2011-01-08 11:22:54.578
*** SERVICE NAME:(SYSAUDERS) 2011-01-08 11:22:54.578
*** SESSION ID:(158,6) 2011-01-08 11:22:54.578

PARSING IN CURSOR #1 id=158 dep=6 uid=38 oct=3 lid=38 tim=163764698236 h=285061697 ad="2aaebec"
select count(*) from emp
END OF STAT
PARSE #1:c#0,e#17,p#0,c#0,r#0,m#1,i#0,d#0,sg#1,tim=163764698266
EXEC #1:c#0,e#72,p#0,c#0,m#1,i#0,d#0,sg#1,tim=1637646982721
WAIT #1: nam='SQL+Net message to client' ela= 5 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=1637646982737
FETCH #1:c#0,e#17,p#0,c#0,r#1,m#0,d#0,sg#1,tim=1637646982757
WAIT #1: nam='SQL+Net message from client' ela= 238 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=16376469828076
FETCH #1:c#0,e#17,p#0,c#0,r#1,m#0,d#0,sg#1,tim=16376469828174
WAIT #1: nam='SQL+Net message to client' ela= 3 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=16376469834801
WAIT #1: nam='SQL+Net message from client' ela= 5688 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=16376469844085
*****  

PARSING IN CURSOR #3 id=152 dep=6 uid=38 oct=47 lid=38 tim=163764545704 h=1523988111 ad="2bae304"
BEGIN DBMS_OUTPUT.GET_LINESIZE(lines); lines;
END OF STAT
PARSE #3:c#0,e#17,p#0,c#0,r#0,d#0,sg#1,tim=163764545705
WAIT #3: nam='SQL+Net message to client' ela= 5 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=16376454572593
EXEC #3:c#0,e#4564,p#0,c#0,r#0,d#0,sg#1,tim=16376454573362
WAIT #3: nam='SQL+Net message from client' ela= 207964 driver id=1111838976 #bytes=1 p3=0 obj#=-1 tim=163767300133
STAT #3 id=1 c#1 pid=0 p#0 os#=0 osn='OSR AGGREGATE (crx=1257 prw# 0 wtw# time=31835 us)'
STAT #3 id=1 c#1 pid=280000 p#0 os#=0 osn='TABLE ACCESS FULL TAB1 (crx=1257 prw# 0 wtw# time=80067 us)'
XCTEND rlibk#, rd_only=1

```

Para traducir esa información a texto más legible se puede emplear la utilidad *tkprof* pasándole como parámetro el nombre de la traza que desea leerse y el nombre del fichero donde esta quedará *traducida*. El resultado podría ser algo como:

```

*****
count = number of times OCI procedure was executed
cpu   = cpu time in seconds executing
elapsed = elapsed time in seconds executing
disk  = number of physical reads of buffers from disk
query = number of buffers gotten for consistent read
current = number of buffers gotten in current mode (usually for update)
rows  = number of rows processed by the fetch or execute call
*****

select count(*)
from
tabl

call      count      cpu      elapsed      disk      query      current      rows
-----  -----  -----  -----  -----  -----  -----  -----
Parse        1      0.00      0.00          0          0          0          0
Execute      1      0.00      0.00          0          0          0          0
Fetch        2      0.04      0.03          0       1257          0          1
-----
total       4      0.04      0.03          0       1257          0          1

```

```

Misses in library cache during parse: 1
Optimizer mode: ALL_ROWS
Parsing user id: 38

Rows      Row Source Operation
-----
1   SORT AGGREGATE (cr=1257 pr=0 pw=0 time=31835 us)
200000  TABLE ACCESS FULL TAB1 (cr=1257 pr=0 pw=0 time=800067 us)

Elapsed times include waiting on following events:
Event waited on          Times     Max. Wait  Total Waited
-----                Waited
SQL*Net message to client        2          0.00      0.00
SQL*Net message from client        2          0.00      0.00
*****
```

4.2.4. La traza 10046: *alter session, oradebug, sql_trace*

La traza 10046 cuenta lo que se está ejecutando en cada momento en la sesión, es decir, que no se intuye lo que pueda estar pasando sino que se ve exactamente lo que está sucediendo. Se puede saber si se están haciendo más o menos lecturas, o si se está consumiendo tiempo en eventos de espera que no aparecen en las vistas *v\$session* o *v\$session_wait*, o incluso se puede conocer el plan de ejecución elegido por el optimizador. En general, servirá para responder a la pregunta: ¿qué y cómo ejecuta la sesión las sentencias?

Esta traza se puede activar en la sesión en curso a través de la ejecución de la sentencia de modificación de parámetros de la sesión (*alter session*). También puede activarse automáticamente incluyendo esa sentencia dentro de la ejecución del cuerpo de un disparador *on logon*.

```

SQL> alter session set events='10046 trace name context forever, level 12';
Sesión modificada.
SQL> select count(*) from tab1;
COUNT(*)
-----
200000
SQL> alter session set events='10046 trace name context off';
Sesión modificada.
```

Esto generará una traza del mismo estilo que la explicada en la sección anterior.

Otra forma de interceptar el evento 10046 es mediante la utilidad *oradebug* que además sirve también para realizar volcados de memoria. Esta herramienta es muy intrusiva por lo que debe utilizarse con mucho cuidado, solo después de haber valorado adecuadamente las ventajas que se obtendrían con su ejecución frente a los inconvenientes que ella provocará.

```
SQL> oradebug setmypid
Sentencia procesada.
SQL> oradebug event 10046 trace name context forever, level 12;
Sentencia procesada.
```

Y para desactivarlo:

```
SQL> oradebug event 10046 trace name context off;
Sentencia procesada.
```

Definiendo en la base de datos el parámetro *sql_trace*, también se conseguiría la generación automática de los ficheros de trazas.

```
SQL> alter system set sql_trace=TRUE;
Sistema modificado.
```

4.2.5. Informes AWR

Los informes estáticos obtenidos a través de la herramienta *AWR* (del inglés *Automatic Workload Repository*, repositorio automático de carga de trabajo) ofrecen toda la información necesaria para apreciar dónde existe un mayor consumo de recursos dentro de la base de datos. La capacidad de generar informes *AWR* está contenida en el paquete de diagnóstico, el cual se licencia aparte, pese a estar incluido en la instalación normal del gestor, aunque, eso sí, no se encuentra activado por defecto. La activación o no del paquete de diagnóstico se logra en función del parámetro *statistics_level*, el cual puede tener tres valores distintos:

- *BASIC*: funcionalidad desactivada.
- *TYPICAL*: activa el paquete de diagnóstico, los *Advisor* (del inglés, *aconsejadores*) de ciertas memorias, y también recoge estadísticas del optimizador, de la distribución del tiempo de la base de datos...
- *ALL*: además de lo mencionado en *TYPICAL*, se añaden otras estadísticas, entre ellas las de los planes de ejecución y algunas de sistema operativo.

Los informes del *AWR* son fotos o instantáneas (en inglés *snapshots*) sobre el estado de la base de datos en un intervalo de tiempo concreto (tiempo de muestra). Los datos presentes en ellos deberán ser analizados sin perder de vista el contexto del tiempo de muestreo y la cantidad de muestras obtenidas, ya que en función de esto los datos serán

más o menos relevantes. Por defecto las fotos se toman cada hora, lo cual puede ser poco representativo. Lo ideal es generar fotos cada 5 o 10 minutos, durante media hora o una hora como máximo.

Para modificar ese tiempo de muestra hay que hacer uso del procedimiento *modify_snapshot_settings* hallado dentro del paquete *dbms_workload_repository*. Por ejemplo, para obtener *snapshots* cada 45 minutos (es decir, que contendrán información de 45 minutos, o lo que es lo mismo, que los totales sobre los consumos de recursos estarán referidos a 45 minutos) bastaría con ejecutar:

```
SQL> exec dbms_workload_repository.modify_snapshot_settings(interval=>45);
PL/SQL procedure successfully completed.
```

La base de datos almacena la información de los *snapshots* en el tablespace *sysaux* durante un tiempo medido por el parámetro *retention*. Este parámetro puede ser modificado también con el procedimiento antes nombrado, lo cual es muy recomendable para evitar el colapso por falta de espacio del tablespace *sysaux*.

Los informes se generan invocando el script *awrpt.sql*, que se encuentra ubicado en el directorio *\$ORACLE_HOME/rdbms/admin*, y se visualizan con la ayuda de un servidor web porque su formato está dado en html.

Los informes AWR son la evolución de los antiguos informes del paquete *statspack*. Comienzan con una cabecera que identifica el informe y expone las condiciones en las que se tomó la muestra:

DB Name	DB ID	Instance	Instnum	Release	RAC	Host
	SnapId	Snap Time	Sessions	Cursors/Sessions		
test2	2193 test2		1	10.2.0.4.0	NO	cse
Begin Snap:	47406	17-Feb-11 07:00:22	284		6.8	
End Snap:	47408	17-Feb-11 08:00:37	340		7.7	
Closed:		60.36 (mins)				
DB Time:		852.63 (mins)				

Figura 4.4: Cabecera del informe AWR.

A continuación ofrece un resumen de los puntos principales:

- Cache Sizes.
- Load Profile.
- Instance Efficiency Percentages.
- Shared Pool Statistics.
- Top 5 Timed Events.

Y después muestra el grueso del informe:



Figura 4.5: Menú del informe AWR.

Los informes AWR son tremadamente útiles para encontrar la causa de algún comportamiento anómalo en la base de datos, o simplemente, para ver cómo se está comportando esta y buscar dónde puede mejorar. En cualquier caso, lo primero que habría que analizar es la información de la cabecera, puesto que es donde se indica el rango horario de la muestra, el consumo de tiempo de base de datos (*DB Time*), el número de sesiones ocurridas, el número de *snapshots* tomados y el tiempo total del informe (*Elapsed*). Nótese que, en la figura 4.4, el *DB time* es mayor que el *Elapsed*, esto es normal en entornos que cuentan con muchas CPUs.

Lo segundo que habría que observar es el *Load Profile* (del inglés, perfil de carga) que es donde se refleja el comportamiento que tuvo la instancia durante el tiempo de muestra:

	Per Second	Per Transaction
Logical reads:	3,150,882.13	139,863.24
Logical writes:	181,118.81	12,421.77
Block changes:	11,785.96	764.73
Physical reads:	19,303.60	1,254.84
Physical writes:	1,296.94	90.79
User calls:	12,082.39	785.29
Parses:	251.74	16.36
Hard parses:	1.57	0.10
Sorts:	3,598.19	231.26
Lagons:	2.66	0.17
Executes:	11,315.80	748.47
Transactions:	15.39	

Figura 4.6: Perfil de carga del informe AWR.

Sin tener referencias de lo que estaba atacando la base de datos en el momento de la obtención de este informe, es un poco más complicado sacar alguna conclusión, no obstante, ya se podría plantear algún argumento de análisis. Por ejemplo, comparando los cambios en los bloques (*Block changes*) con las lecturas lógicas (*Logical reads*), se ve que el 6 % de

los bloques que se leen, son modificados. O también que existe un bajo porcentaje de *Hard Parking*, por lo que se puede descartar que el consumo de CPU se vea penalizado por el *parsing* (concepto que se explicará más adelante).

A la par que el *Load Profile* también se podrían evaluar los eventos de espera, es decir, dónde se produce más consumo de tiempo:

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		23,907		41.8	
Db file sequential read	9,115,694	22,198	2	38.8	User I/O
Db file scattered read	4,254,794	7,970	2	13.9	User I/O
control file sequential read	2,585,278	1,454	1	2.5	System I/O
PX Deq Credit send bld	31,010	1,011	33	1.8	Other

Figura 4.7: Cinco máximos eventos de espera del informe AWR.

Un error muy común es pensar en que si hay mucha espera es porque algo dentro de la base de datos está funcionando mal. Lo que aparece realmente son los *cinco máximos eventos de espera* del informe, y puesto que, la base de datos SIEMPRE tiene eventos de espera, incluso cuando no tiene actividad, este apartado nunca estará vacío.

Por lo general, una base de datos 'saludable' debe tener como sus tres primeros eventos de espera máximos el *CPU time*, el *db file sequential read* y el *db file scattered read*.

Para saber qué es lo que provoca los eventos de espera, se debe consultar el apartado de las estadísticas SQL del informe AWR:

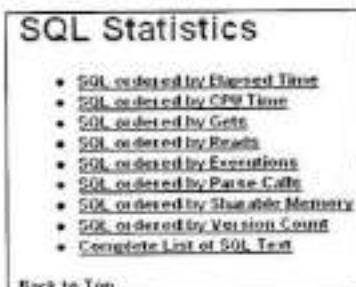


Figura 4.8: Estadísticas SQL del informe AWR.

Así, si lo que se necesita es ver qué sentencia consume más *Gets* o qué sentencia tiene más *Reads*, basta con buscar qué sentencia está causando el mayor consumo y focalizar en ella el estudio.

Además de todos estos datos, los informes AWR proporcionan información detallada sobre los ficheros, la instancia ..., y hasta sobre los parámetros.

4.2.6. OVO, NetCool, Tivoli, Patrol...

Para el caso de monitorización de espacios, chequeo de listeners o disponibilidad de instancias se pueden usar herramientas como OVO, NetCool..., que precisan de la instalación

de un software ligero en la base de datos que hace de agente para recopilar la información sobre los aspectos que desean monitorizarse dentro de la misma.

Particularmente, OVO tiene predefinidas un conjunto de consultas que se lanzan directamente al sistema operativo o a la base de datos, y que le permiten conocer su estado. Además, tiene prevista la programación de límites máximos y umbrales que podrían hacer saltar alarmas, y en consecuencia ejecutar determinadas acciones, por ejemplo, enviar correos electrónicos.

4.3. Elementos y parámetros susceptibles de ser monitorizados

Aunque, por lo general, cuando se habla de monitorización a secas, implícitamente se está hablando de rendimiento, es decir, su meta es la optimización, cuando el jefe solicita que se monitorice la base de datos automáticamente, habría que preguntarle: ¿cuál es el objetivo o propósito de la monitorización: espacio, errores...?

Algunas de los elementos que pueden evaluarse son, entre otros: errores (contenido de fichero de alertas y de trazas), rendimiento, consumo de recursos del sistema operativo, listener, bloqueos, disponibilidad y espacio de ocupación de:

- Tablespaces.
- Disco destino de los ficheros de log archivados.
- Disco destino de los backups.
- Disco destino de las trazas y el fichero de alertas. En Oracle 10g es donde apuntan los parámetros *background_dump_dest*, *user_dump_dest* y *core_dump_dest*. Desde la versión 11g, existe una herramienta llamada ADRCI (del inglés *Automatic Diagnostic Repository Call Interface*, interfaz de llamada del repositorio de diagnóstico automático) que informa directamente sobre incidencias y errores en la instancia o base de datos, y paralelamente guarda la ubicación de los elementos de diagnóstico (como pueden ser las trazas o el fichero de alertas) en la vista *v\$diag_info*.

INST_ID	NAME	VALUE
1	Diag Enabled	TRUE
1	ADR Base	c:\oracle\product\11.0.1
1	ADR Home	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1
1	Diag Trace	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\trace
1	Diag Alert	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\alert
1	Diag Incident	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\incident
1	Diag Cdump	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\cdump
1	Health Monitor	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\hm
1	Default Trace File	c:\oracle\product\11.0.1\diag\rdbs\111r1\t11r1\trace\t11r1_ora_7088.trc
1	Active Problem Count	0
1	Active Incident Count	0

Algunos de los parámetros de Oracle susceptibles de ser monitorizados pueden ser:

- Los de memoria para ver cómo se está haciendo el redimensionamiento automático con ASMM (se estudiará más adelante): `__db_cache_size`, `__java_pool_size`, `__large_pool_size`, `__shared_io_pool_size`, `__shared_pool_size`, `__streams_pool_size`.
- Los de UNDO: `undo_management`, `undo_retention`, `undo_tablespace`.

Para saber qué monitorizar es importante tener un método, que bien podría ser el siguiente:

1. Definir objetivo.
2. Elegir herramienta de monitorización.
3. Si es posible, dividir en gránulos el objetivo.
4. Elegir cuál de los gránulos se debe observar según el objetivo.
5. Verificar si existe mayor granularidad que cumpla el objetivo.
6. Repetir desde el paso 3 hasta que se cumpla el total del objetivo.

También es importante meditar sobre cuándo monitorizar, ya que, como ya se ha dicho, monitorizar siempre tiene un coste, no solo a nivel de base de datos por las posibles consultas que puedan hacerse a vistas del catálogo, sino también a nivel de sistema operativo, y todo, absolutamente todo, afectará al rendimiento global de la base de datos.

A continuación se detallan dos casos prácticos.

4.3.1. Caso 1

Se desean monitorizar las cinco consultas más costosas que genera un usuario llamado *TJ*.

Como se ha visto, la resolución del problema planteado pasa por la aplicación del método explicado:

- Definir objetivo: las cinco consultas más costosas de *TJ*.
- Elegir herramienta de monitorización: las gráficas o vistas dinámicas a través del SQL*Plus.
- ¿Qué hay que monitorizar?: sesiones.
- Dentro de las sesiones, ¿qué hay que monitorizar?: las sesiones que sean del usuario *TJ*, es decir, que cumplan la condición `username='TJ'`.
- Dentro de las sesiones de *TJ*, ¿qué hay que monitorizar?: las consultas que más CPU consumen.

- Dentro de las consultas que más CPU consumen de todas las sesiones de *TJ*, ¿qué hay que monitorizar?: las cinco que más consumen.
- ¿Hay que monitorizar algo más?: no, así que manos a la obra.

Desde SQL*Plus se podría resolver el problema usando las vistas *v\$session* para extraer la información de las sesiones, y *v\$sqlstat*, para filtrar por las estadísticas de las consultas.

```
set lines 125 pages 50000 long 200000000
with
sel as
(select s.sid, s.program, a.sql_id,
       s.status, s.event, s.seconds_in_wait,
       a.buffer_gets, a.version_count, a.executions,
       a.cpu_time, a.disk_reads, a.sql_fulltext
  from v$session s, v$sqlstats a, v$process p
 where s.sql_id= a.sql_id
   and s.paddr = p.addr
   and s.sql_hash_value <> 0
   and s.username = 'TJ'
  order by &1)
select *
  from sel
 where rownum <= 5
 /
```

Usando *Enterprise Manager*, basta con dirigirse a la pestaña de *Rendimiento* desde la página de inicio, y una vez allí, localizar *Enlaces de control adicionales*, que permitirá acceder a *Sesiones de búsqueda*, donde se podrán filtrar las sesiones, por ejemplo, por el nombre de usuario (*TJ* en este caso).

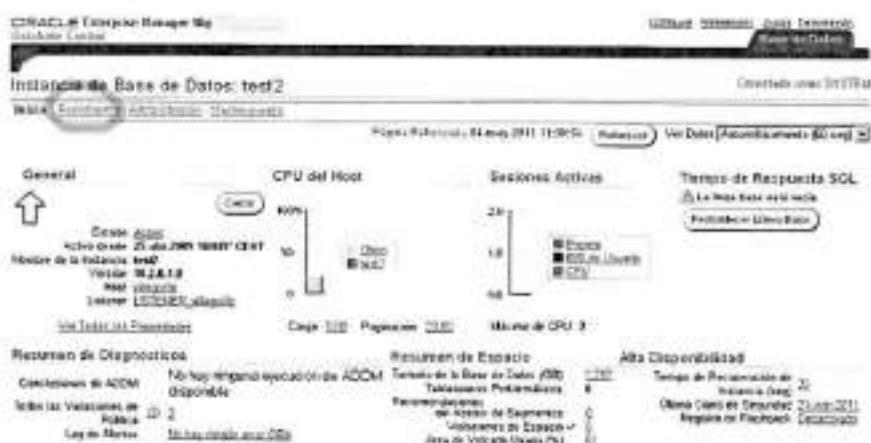


Figura 4.9: Página de inicio del Enterprise Manager.

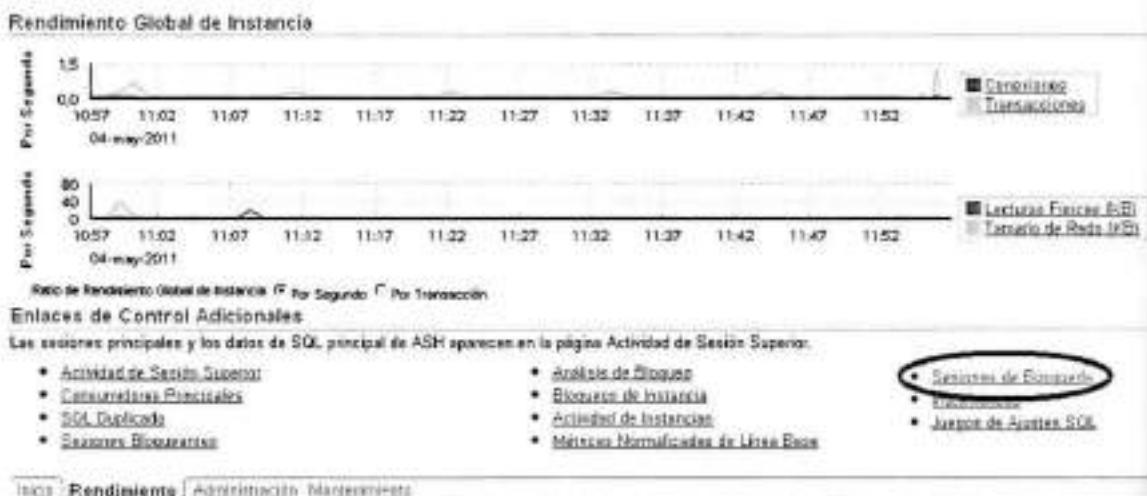


Figura 4.10: Pestaña de Rendimiento.



Figura 4.11: Sesiones de búsqueda.

En la ventana de *Sesiones de búsqueda* existe un cuadro de texto donde se puede establecer el límite de CPU por el que quieren filtrarse las sesiones. Una vez aplicado el filtro de usuario y CPU, solo resta pinchar sobre las cinco primeras sesiones que aparezcan

y examinar toda la información que de ellas se tiene, incluyendo la consulta que se esté ejecutando.

Figura 4.12: Detalle de la información de una sesión.

4.3.2. Caso 2

Se produce el siguiente problema: no es posible la conexión a la base de datos por parte de un usuario concreto.

Aplicando el método explicado anteriormente surgen las siguientes preguntas:

- Definir objetivo: averiguar por qué no se puede llevar a cabo la conexión.
- Elegir herramienta de monitorización: SQL*Plus y comandos de sistema operativo.
- ¿Qué hay que monitorizar?: la viabilidad de las nuevas conexiones.
- ¿Qué habría que monitorizar/comprobar para asegurar la viabilidad de las nuevas conexiones?:
 - Si la base de datos está abierta.
 - ¿Se está intentando la conexión desde una máquina remota o en local a través del protocolo Oracle Net (es decir, con la @)? en caso afirmativo, es necesario comprobar que el listener esté levantado.
 - Posibles errores resultantes del comando de conexión, por ejemplo, que el nombre de usuario o la contraseña empleada no fueran correctos.
 - Mensajes aparecidos en el fichero de alertas.
- ¿Hay que monitorizar algo más?: no, así que manos a la obra.

Para ver si la base de datos está levantada, primero habría que asegurarse de que los procesos adecuados están en máquina (en Unix/Linux se haría con el comando `ps -ef|grep pmon`), y luego, mediante una sesión de SQL*Plus, se intentaría abrir una conexión a la base de datos (a ser posible como '`/ as sysdba`'). Con una simple consulta al campo `status` de la vista `v$instance`, se sabría si la base de datos está abierta (*OPEN*) o solo montada (*MOUNT*), en cuyo caso sería normal que no aceptara conexiones.

Si la base de datos estuviera abierta y la conexión se estuviera haciendo a través de Oracle Net, el siguiente paso sería comprobar que el listener está definido y funcionando, para ello bastaría con emplear el comando `lsnrctl status nombre_del_listener` de Oracle directamente desde una sesión de sistema operativo.

Los mensajes de error de los comandos de Oracle suelen ser de la forma ORA-xxxxx, y normalmente van acompañados de un texto que explica u orienta sobre la causa del error. Para detallar esa información se puede usar, desde una sesión de sistema operativo, el comando `oerr ora xxxxx`. Si se desea ampliar la información acerca del error, o incluso consultar casos de error aparecidos en otras instalaciones similares, es muy útil el acceso a *Metalink*.

El consejo del buen administrador...

Metalink, recientemente integrada en una página de soporte global de todos los productos marca Oracle, se puede encontrar en la web <https://support.oracle.com/CSP/ui/flash.html> y es de uso obligado para todo administrador de Oracle que se precie. El único inconveniente que tiene es que precisa de una cuenta de usuario que necesariamente debe estar ligada a un contrato de soporte de Oracle, y lamentablemente, esta condición no está al alcance de todo el mundo...

4.4. Optimización en Oracle

La palabra más común e importante que se emplea en toda optimización es *DEPENDE*. Dar información sobre el comportamiento de la base de datos para lo que sea dependerá de muchos factores, por lo que si alguien decide decir que para un parámetro concreto el valor X es el adecuado, puede que esté mintiendo. Por ello, en este capítulo no se encontrarán afirmaciones cerradas sobre lo que se debe o no hacer, sino recomendaciones probablemente útiles para que en última instancia, la opción de tomar o no una decisión se haga después de un análisis real de la situación.

¿Sabías que ...? Tom Kyte, que es de los profesionales de Oracle con más renombre a nivel mundial, dice acerca de la optimización: 'no existe una bala de plata para hacer optimizaciones, si existiera ya estaría implementada dentro del gestor y no se podría cambiar'.

En muchas ocasiones optimizar una base de datos simplemente implica dejar de hacer las cosas mal. Otras veces, cuando todo se ejecuta como se debe, optimizar se traduce en hacer que todo siga bien, pero consumiendo el mínimo número de recursos del sistema.

El consejo del buen administrador...

Si alguien en algún libro o documento muestra una solución, antes de aplicarla a una base de datos en producción, hay que probar y volver a probar, y seguir probando hasta no estar completamente convencidos de la utilidad de la solución.

Conviene tener muy claro que existen fundamentalmente dos tipos de bases de datos según el modo en el que se explota la información que ellas almacenan. Pertener a un grupo u otro condicionará en gran medida los hitos de optimización que podrán perseguirse, así como los valores que deberán manejarse para determinados parámetros. Los dos tipos son:

- **OLTP:** del inglés *OnLine Transaction Processing* (procesamiento de transacciones online), son bases de datos orientadas a la transacción, es decir, a manejar grandes cantidades de transacciones por segundo, por lo que su configuración estará enfocada a mejorar la velocidad de una transacción. Las transacciones se controlan con un doble objetivo, por un lado, se trabaja para que se realicen con éxito y por otro, para asegurar que en caso de fallo se puedan recuperar. Un ejemplo claro de una base de datos transaccional es una base de datos que soporta las compras hechas en un supermercado. No hay más que fijarse en toda la información contenida en el ticket de compra, para hacerse una idea de la cantidad de datos que manejará cada transacción hecha en dicha base de datos.
- **DSS y DW:** del inglés *Decision Support System* (sistema de soporte a la decisión) y *Data Warehouse* (almacén de datos), respectivamente, son bases de datos con muchos terabytes o incluso petabytes de información, ya que por lo general, se destinan a agrupar mucha información del negocio (también históricos) necesaria para la toma de decisiones. De cara al rendimiento, estas bases de datos plantean dos momentos conflictivos: la carga de los datos y las consultas hechas para su extracción (con agrupamientos, ordenaciones, sumatorios, conteos, medias...) Un ejemplo pueden ser los DW de marketing, con toda la información de las ventas, y las consultas enfocadas a los totales, diarios, semanales, mensuales y anuales de las ventas de los distintos productos.

Una posible clasificación de las situaciones a que se enfrenta la labor de optimización podría ser:

- Situación proactiva: resultante de una monitorización programada en la que se buscan posibles problemas de rendimiento, o también, resultante de un análisis del rendimiento tras un cambio en el aplicativo.
- Situación reactiva (llamada coloquialmente *de apaga fuegos*): se tiene que resolver un problema real y concreto de rendimiento. Esto es muy común, por ejemplo, después

de la subida de versión del aplicativo o de la base de datos, o debido a un aumento en la carga o explotación de la información que contiene la base de datos.

Para enfrentarse a cualquier problema de rendimiento, resulta de mucha ayuda tener siempre presente que los sistemas gestores de bases de datos como Oracle, DB2 o MySQL son *solo* programas. Normalmente un gran programa en C que funciona de la misma manera que cualquier programa en C: reserva memoria, guarda información en ficheros y tiene hasta trazas para saber qué se está haciendo en cada instante de la ejecución. Visto desde esa perspectiva, el gestor no parecerá un monstruo de siete cabezas con un complejidad infinita, sino que se verá como muchos, muchísimos, programitas coordinados para un objetivo común: gestionar información.

Puesto que cada gestor de bases de datos tiene su propia arquitectura, la optimización de ciertos puntos dependerá del propio gestor que se esté usando, pero lo que es común para todos ellos es *el método de optimización*. Es importante recalcar que, al igual que en la monitorización, toda optimización comienza con el diseño del método de trabajo, para evitar perder el tiempo y dejarse cosas por el camino.

Por ejemplo, en el caso de que se esté focalizando la optimización en el rendimiento de las sesiones, un método sencillo podría ser el siguiente:

1. Describir levemente la arquitectura sobre la que se está trabajando: el sistema operativo, la aplicación y por último la base de datos.
2. Buscar o establecer una situación inicial que servirá de referencia: será la línea base.
3. Encontrar cuál o cuáles son las sesiones/procesos que consumen más recursos.
4. Definir objetivo.
5. Observar qué hacen y qué consumen.
6. Proponer una acción.
7. Repetir desde el paso 4 hasta que se cumpla el objetivo.

Dentro del punto 1 es de vital importancia conocer cómo ataca la aplicación a la base de datos, ya que de ello dependerá el modo en que ha de buscarse la información referida a las sesiones.

En el caso de Oracle, la aplicación puede estar utilizando un pool de conexiones (servidores compartidos o *shared servers*) o conexiones directas a la base de datos (dedicadas). A su vez, la base de datos puede estar funcionando también internamente de dos formas distintas: con *dispatchers* o con conexiones, y sesiones directas.

Esto quiere decir que, por un lado, la aplicación puede estar ejecutando una tarea en una o más sesiones, y por otro, que la base de datos provee a las sesiones que se conectan con uno o más procesos compartidos.

Un similitud de esto podría ser el servicio de camareros de un restaurante (sería la aplicación), y por otro, el servicio de cocineros (la base de datos). Si existiese un conjunto de camareros encargados de servir todas las mesas, se podría decir que hay un pool de

camareros; si hubiese un camarero dedicado solo a una mesa y siempre a la misma mesa, sería un camarero dedicado. Si la comida de una mesa concreta la puede preparar cualquier cocinero, se hablaría de *dispatchers*, en cambio, si la prepara solo una persona y siempre la misma persona, sería un cocinero dedicado.

Para trabajar los puntos 3 y 5 del método podrían utilizarse cualquiera de las herramientas de monitorización ya explicadas en la primera sección de este capítulo. Un buen punto de partida serían los informes de eventos de espera de la base de datos (dentro de los AWR), porque es donde se detallan las acciones que en esta se están realizando.

A lo largo de esta sección se irán proporcionando técnicas para afrontar el resto de puntos de este método o de cualquier otro, y no solo dentro del contexto de las sesiones. Para ello, se abordará el rendimiento desde cinco perspectivas diferentes, que sumados al apartado específico de índices que se expondrá en la sección siguiente, darán una completísima visión sobre qué implica optimizar una base de datos Oracle.

- **Instancia:** procesos y procesos en background (log writer, process monitor, system monitor, data block writer, checkpoint, reco), sga (redo buffer, buffer cache - default buffer cache, pools 2k 4k 8k 16k 32k-, shared pool, large pool, stream pool, java pool), pga y memory target.
- **Recursos:** memoria, procesador=cpu, i/o, red.
- **Estadísticas.**
- **Particionamiento y paralelización.**
- **Consultas:** plan de ejecución, coste, tuning, joins y hints.

No obstante, el rendimiento debe verse de manera global, ya que por ejemplo en un momento alto de transacciones se generará necesariamente mucha información de Redo, que a su vez implicará una alta generación de Undo, que a su vez terminará en elevada generación de Archivelogs, que a su vez derivará en muchas actualizaciones en el Fichero de Control.

Como se aprecia, todo forma parte de una compleja maquinaria. Aumentar o disminuir el valor de un parámetro concreto puede provocar consecuencias buenas o malas, probablemente buenas y malas al mismo tiempo, porque puede estar perjudicando algún punto de la base de datos en beneficio de otro. Por tanto, en un proceso de optimización, el dba será como un malabarista pendiente de cientos de bolas, y es ahí, donde seguir un método es importante para no dejarse ninguna bola por el camino.

Por otro lado, y aunque parezca mentira, merece la pena recalcar que para optimizar una base de datos Oracle solo se necesita saber observar y realizar operaciones de aritmética sencilla.

4.4.1. Instancia

La instancia se compone de memoria y de una serie de procesos. A diferencia de otros gestores como el DB2 o Informix, que pueden tener más de una base de datos por instancia,

en Oracle una instancia tiene una sola base de datos. E incluso, en modo RAC (del inglés *Real Application Cluster*), que es la opción de alta disponibilidad más utilizada en Oracle, una base de datos tiene más de una instancia, normalmente en máquinas distintas.

Tener una base de datos con buen rendimiento implica tener todos los procesos de la instancia ejecutándose sin problemas y un tamaño de memoria óptimo para el máximo nivel de carga que se pueda alcanzar, es decir, tener todos los componentes de la instancia trabajando adecuadamente.

A continuación se analizarán los componentes uno a uno.

Procesos y procesos en background

Los procesos que componen la instancia se llaman *Server Processes* y los hay de dos tipos: sesiones, y procesos en background. Como ya se ha explicado, los *Server Processes* pueden ser conexiones dedicadas o conexiones compartidas (*shared servers*). En los *shared servers* existe la conexión pero no existe la sesión, lo cual tiene sus ventajas (requieren menos memoria) y también tiene sus inconvenientes: al tener un pool de sesiones que da servicio a todas las conexiones en lugar de una sesión por cada conexión, se pueden producir muchos eventos de espera. Si se trata de una aplicación con mucha concurrencia, cuantas más conexiones existan, más esperas se observarán, y peor será el rendimiento de la aplicación.

Para buscar posibles problemas de contención en las conexiones, se pueden consultar las vistas *v\$dispatcher* y *v\$dispatcher_rate*. Para analizar si hay o no *shared servers* y qué está pasando con ellos, pueden hacerse dos cosas: evaluar el parámetro *shared_servers* y consultar la vista *v\$queue*.

Los procesos en background se levantan en el inicio de la instancia y realizan tareas de mantenimiento y control de la base de datos. El dba no tiene poder para obligar a estos procesos a que realicen una u otra tarea de forma directa, pero sí podrá influir en ellos indirectamente.

Unos procesos son comunes a todas las bases de datos, y otros solo están presentes en función de las opciones que se tengan configuradas. Aquí solo se tratarán los seis procesos comunes: LGWR, DBWR, PMON, SMON, CKPT y RECO.

1. **Log Writer: LGWR.** Este proceso se encarga de escribir en disco la información que está en la *Redo Buffer* (que se estudiará más adelante). Es muy importante porque mientras él trabaja el resto de *Server Processes* espera, prueba de ello es el evento de espera *log file sync* que aparece muy comúnmente en las bases de datos OLTP. Cuando un *Server Process* ejecuta un *commit*, el LGWR se dispara para guardar la información de la transacción validada, mientras aquel espera hasta que esta información sea grabada en disco y quede de forma permanente. Si en vez de un *commit* hubiera 10000 sesiones realizando *commit*, la contención por esperas sería casi inevitable. Cuando los entornos son DSS, al no haber carga de operaciones de modificación de datos, sino solo de tipo consultivo, la importancia del LGWR disminuye considerablemente.

El LGWR se ejecuta como consecuencia de alguna de las siguientes acciones:

- Han pasado 3 segundos desde la última activación del LGWR
 - El *Redo Buffer* se ha llenado la tercera parte de su espacio
 - Un *Server Process* ha ejecutado un *commit* o un *rollback*
 - Se lo ha pedido el DBWR
2. **DataBase WRiter: DBWR.** Su trabajo consiste en escribir los bloques *sucios* a disco. Los bloques *sucios* son bloques de datos que han sido modificados estando en memoria (en la *Buffer Cache* que se explicará más adelante). Con la inicialización del parámetro *db_writer_processes* se pueden configurar el número de DBWR que se levantarán. En la versión 11gR2 puede haber hasta 35 DBWRs. El número adecuado de DBWR estará en función de la cantidad que aparezca bajo el evento de espera *free buffer wait* (espera por buffer libre).
3. **Process Monitor: PMON.** Es el encargado de deshacer las transacciones que dejan las conexiones perdidas. Por ejemplo, si una sesión se pierde y estaba bloqueando una tabla, el proceso que se encarga de liberar esa tabla para que la puedan usar el resto de procesos es el PMON. Además, se encarga de registrar el servicio (parámetro *service_name*) en el *listener*. Sin este proceso la base de datos deja de funcionar, es por ello que suele ser el proceso más usado para realizar las búsquedas de instancias levantadas en la máquina (*ps -ef|grep pmon*).
4. **System Monitor: SMON.** Entre otras cosas se encarga de limpiar el tablespace temporal. Además también deshace las transacciones que son grandes, pero hay que decir que no es prudente abusar de esta labor porque no es muy conveniente de cara al rendimiento.
5. **Checkpoint: CKPT.** El CKPT cuida de que las cabeceras de los ficheros de datos y el fichero de control queden sincronizadas en cada maniobra de *checkpoint*, la cual se lleva a cabo con la actuación del CKPT, el LGWR y todos los DBWR que existan (los DBWn). Un *checkpoint* es la escritura en disco de bloques modificados en memoria. Sin entrar en mucho detalle, basta saber que hay varios tipos de checkpoints: *thread*, *database*, *segment*, *datafile*, *tablespace* e *incremental*. Dependiendo de la operativa que se esté llevando a cabo, se ejecutan unos u otros. Es uno de los procesos más importantes de la base de datos, siendo su objetivo principal el reducir el tiempo de recuperación de datos en caso de caída de la base de datos.
6. **Reco: RECO.** Realiza las recuperaciones de transacciones en BBDD distribuidas.

Como ya se ha dicho, el dba no puede cambiar el comportamiento de los procesos en background directamente, pero la configuración de los parámetros de la instancia hace que estos procesos se comporten de una forma u otra. Un ejemplo claro es el caso del parámetro *fast_start_mttr_target*, que especifica el número en segundos que la base de datos debería tardar en realizar una recuperación después de una caída abrupta de la instancia, es decir, establece cuánta información tendrá que rehacerse en disco aplicando ficheros de log, esto es, cuántos datos *sucios* pueden quedar pendientes en cada momento. Por tanto, este

parámetro afecta al comportamiento de los DBWR, que son los que realizan el *vaciado* de los bloques *sucios*, y los responsables del evento de espera *free buffer waits*, ya que cuanto más rápido trabajan, más rápido dejan libres los buffers que guardan los cambios para ser usados de nuevo, pero eso sí, con el consecuente aumento de lecturas y escrituras a disco (el I/O).

Además de los procesos en background, existen otros tipos de procesos que dependen de estos, son los procesos esclavos. Los hay de muchos tipos: de ejecución de sentencias en paralelo (procesos Pnnn), de escrituras a disco (procesos Innn), de colas avanzadas (*Advance Queueing* en inglés, Qnnn), de trabajos (*jobs* en inglés, Jnnn). Son también procesos en background, pero dependen completamente de otros procesos en background.

SGA

Es la memoria que comparten todos los usuarios. Está dividida en varias partes (también llamados segmentos o cachés) y, aunque en las versiones actuales Oracle maneja el tamaño de estos segmentos de manera automática (utilidad ASMM, *Automatic Shared Memory Management*, direccionamiento de memoria compartida automático), en ocasiones, como ocurre con todos los programas, este redimensionamiento automático tiene problemas y es el dba el responsable de redefinir el tamaño de esos segmentos para que sea el apropiado según la carga de trabajo que requiera la base de datos.

Las diferentes cachés pueden verse ejecutando la siguiente consulta a la vista *v\$sgainfo*:

NAME	BYTES	RES
Fixed SGA Size	1336932	No
Redo Buffers	6103040	No
Buffer Cache Size	146800640	Yes
Shared Pool Size	88080384	Yes
Large Pool Size	4194304	Yes
Java Pool Size	4194304	Yes
Streams Pool Size	0	Yes
Shared IO Pool Size	0	Yes
Granule Size	4194304	No
Maximum SGA Size	418484224	No
Startup overhead in Shared Pool	41943040	No
Free SGA Memory Available	167772160	
12 rows selected.		

Y también, para detallar más el consumo:

```
SQL> compute sum of bytes on pool
SQL> break on pool skip 1
SQL> select *
  2  from v$sgastat
```

```
3 order by 1,2;

POOL      NAME          BYTES
-----
java pool  free memory    4194304
*****
sum           4194304

large pool   PX msg pool    491520
               free memory    3702784
*****
sum           4194304

shared pool  1:kngisga      16052
               ADR_CONTROL     1056
+ ++
               CCUR           3551456
+++
               PCUR           2286216
...
               Xslongops       14240
               xsoqmehift     4196
               xsoqojhifl     3300
               xsoqophift     4196
               xsoqsehifl     2404
               xssinfo         5560
*****
sum           88085076

               buffer_cache    146800640
               fixed_sga        1336932
               log_buffer       6103040
*****
sum           154240612
852 rows selected.
```

De estos segmentos, aquí solo se tratarán los seis más representativos:

- 1. Redo Buffers:** Este espacio en memoria se dimensiona con el parámetro *log_buffer*.

NAME	TYPE	VALUE
log_buffer	integer	5906432

Controla el tamaño en memoria donde se almacena la información de redo, y será por tanto quien determine una de las causas por las que trabaja el proceso LGWR, que es el encargado de mantener la consistencia en los datos.

Todos los comandos que se ejecutan en una transacción se almacenan en el buffer de redo, pero no los datos. Básicamente contiene información sobre índices y undo, de modo que se asegure el poder deshacer o rehacer la transacción. Al no guardar el dato en sí mismo, el espacio consumido es menor y se hace más manejable.

Un buen tamaño de buffer de redo como punto de partida podría ser el tamaño de un gránulo. El gránulo es el tamaño mínimo en que Oracle parte la memoria. Cuando hay una SGA menor de 512Mb el granulo es de 4Mb, cuando es mayor de 512Mb el granulo es de 16Mb. Sirvan como referencia casos reales de bases de datos con altísima transaccionalidad con tamaño de buffer de no más de 21Mb.

No obstante, vincular el tamaño del buffer de redo al tamaño de la SGA o de la base de datos es un error, ya que aquél no depende de estos, sino del grado de transaccionalidad que soporta la base de datos. Por ejemplo, si hubiese pocos datos pero muchas actualizaciones, es decir, que la carga es alta, un tamaño alto de buffer de redo sería apropiado, en cambio, si hubiese muchas consultas y pocos cambios, aunque la SGA fuese enorme (24Gb, por ejemplo), se podría perfectamente tener un *log_buffer* de 14Mb. Si fuese más grande lo único que se conseguiría es desperdiciar espacio en memoria.

2. **Buffer Cache Size:** En esta porción de memoria se almacenan los datos de todo aquello que se ejecuta en la aplicación. Se divide a su vez en tres cachés básicas: *Default*, *Keep* y *Recycle*.

Oracle divide las tablas en pequeños trozos que se llaman bloques (de 8Kb por defecto). La *Default Buffer Cache* contiene los bloques de los datos de las consultas ejecutadas. Sirva el siguiente ejemplo para ilustrar su funcionamiento: un usuario A en una sesión 1 consulta el nombre del cliente Z, la primera vez que se ejecuta, lo que hace el gestor es copiar en memoria esta información. Cuando el usuario B en una sesión 2 quiere consultar la información del mismo cliente, el gestor no tiene que ir hasta el disco para obtener esta información, puesto que ya la tiene en memoria, con lo que se puede ahorrar hasta un 95% de tiempo en la sesión 2, ya que el tiempo de lectura en memoria es muy inferior al tiempo de lectura en disco.

Ahora bien, puede ser que no solo sean los datos del cliente Z los que estén en memoria, ya que en los bloques que contienen sus datos se almacenan muchas filas de una misma tabla, por lo que puede que el usuario C, que busca el cliente X, encuentre en los bloques que el usuario A subió a memoria, la información del cliente X.

Es decir, que el tamaño de la *Buffer Cache* influirá directamente en la cantidad de I/O (lecturas y escrituras a disco) que realice la base de datos (generalmente, a más *Buffer Cache*, menos I/O).

Para evitar que una sola sesión agote la *Buffer Cache*, Oracle impone internamente mediante parámetros ocultos un límite máximo de consumo por sesión.

La estructura de la *Buffer Cache* es sencilla, consta de dos listas doblemente enlazadas para determinar dónde y cómo están los bloques, y de unas estructuras llamadas *buckets* donde se almacenan los bloques con los datos propiamente dichos. Una de

esas listas es una LRU (del inglés *Last Recently Used*, usado más recientemente), con una zona llamada *caliente* y otra *fria*. Si la lista está llena y es necesario subir a memoria más bloques, se quitarán los bloques *frios* de memoria para poner unos nuevos. A medida que estos bloques son más usados, puede ir a los sectores *calientes* de la lista, esto hace que sea más difícil que no estén en la lista, y aumenta la probabilidad de que otras sesiones encuentren estos bloques en memoria y no sea necesario ir a disco a buscarlos.

Pero además de consultar, también se puede querer cambiar o insertar datos. Si el bloque en disco se cambia mientras que está presente en memoria, el gestor marca este bloque como *sucio*. Un bloque sucio pasa a una lista que se llama LRUW. De nuevo una lista LRU que le indica al proceso DBWR qué bloques tiene que *limpiar*. El funcionamiento de esta caché se puede analizar en la vista *v\$db_cache_advice*. Además, existe una vieja fórmula para determinar la eficiencia en el trabajo de la *Buffer Cache* que concluye con la definición del *Hit Ratio*, BCHR:

$$BCHR = \frac{1 - (Physical\ Reads - (Physical\ Reads\ Direct + Physical\ Reads\ Direct(lob)))}{(Db\ Block\ Gets + Consistent\ Gets - (Physical\ Reads\ Direct + Physical\ Reads\ Direct(lob)))}$$

Donde:

- **Physical reads:** son las lecturas físicas, lo cual implica que el bloque de información buscado tiene que ser leído desde el disco. Las lecturas físicas directas: (*PhysicalReadsDirect* y *PhysicalReadsDirect(lob)*)¹ son las lecturas que no buscan en la caché quedando excluidas del cálculo de la hit ratio.
- **Gets:** son las lecturas lógicas, lo que significa que el bloque buscado ya está en memoria. La diferencia entre *DbBlockGets* y *ConsistentGets* radica en que el primero es una lectura que necesita la última versión del bloque y al segundo le vale la versión que pueda darle una lectura consistente.

Por tanto, el BCHR representa el porcentaje de información que se lee desde la *Buffer Cache*, comparado con la que se tiene que buscar directamente a disco. Y teniendo en cuenta que leer información de disco es más costoso que leerla directamente de la memoria, se ve fácilmente que un porcentaje alto en esta fórmula es buen indicativo, ya que implicaría que no se necesita ir mucho a disco para realizar las operaciones.

Los valores que aparecen en la fórmula se extraen de la vista *v\$sysstat*. Como regla general, en entornos OLTP el resultado de la fórmula debe arrojar un valor por encima del 90%, mientras que en entornos DSS se considerará bueno si es más del 70%.

Existe un error muy extendido respecto de la comprensión del funcionamiento del BCHR y es pensar que si por ejemplo la base de datos objeto de estudio almacena

¹Las lecturas físicas directas lob son aquellas producidas por la lectura de objetos almacenados en tipos LOB (Large Object)

información equivalente a 1G en el disco, y se cuenta con una *Buffer Cache* de también 1G, no existirían lecturas físicas porque todos los bloques de datos estarían en memoria, pero esto es falso si se realizan cambios en esos bloques, ya que entonces se crearian segmentos de UNDO y estos necesariamente van a disco, y como es de disco de donde se leen, las lecturas físicas estarían inevitablemente garantizadas.

Si se conociera de antemano que unos datos son muy usados, sería posible obligar al gestor a mantenerlos en memoria. Esto se hace mediante el uso de la *Keep Buffer Cache*, cuyo tamaño se define con el parámetro *db_keep_cache_size*.

Si, por el contrario, lo que se pretende es deshacerse de esos bloques cuanto antes porque se sabe que su uso no es habitual, se podría hacer uso de la *Recycle Buffer Cache*, la cual se controla con el parámetro *db_recycle_cache_size*.

Ambas cachés se manejan a nivel de segmento, es decir, al crear la tabla o índice es donde debe indicarse el deseo de almacenarlos en la caché correspondiente.

```
CREATE TABLE tabla_ejemplo
  (ID      NUMBER,
   INFO    VARCHAR2(10))
TABLESPACE users
STORAGE(BUFFER_POOL KEEP);
```

Fuera de la *Buffer Cache* también se pueden encontrar porciones de memoria donde se hayan almacenados bloques de datos, estos son las pools con los distintos tamaños de bloques permitidos (2K, 4K, 8K, 16K y 32K), que darán servicio a los tablespace que tengan definidos esos tamaños de bloque. Aunque lo recomendable es que los tablespace usen el mismo tamaño de bloque que el de la base de datos, cuando en un entorno OLTP está previsto que un tablespace transportable (que puede llevarse a otra base de datos como quien se lleva un CD a otro ordenador) vaya a formar parte de un DW o entorno DSS, lo normal es que ese tablespace tenga un tamaño de bloque de 16K o 32K, que es lo que se usa en esos entornos, pese a que en OLTP las bases de datos se crean con tamaño de bloque mucho más pequeño.

NAME	TYPE	VALUE
db_16k_cache_size	big integer	0
db_2k_cache_size	big integer	0
db_32k_cache_size	big integer	0
db_4k_cache_size	big integer	0
db_8k_cache_size	big integer	0

3. **Shared Pool Size:** En los últimos años, el uso de la *Shared Pool Size* se ha incrementado considerablemente. Ha pasado de almacenar 500 estructuras de información en

la versión 10gR2 a guardar 800 en la 11gR2. Esta caché es la encargada de contener parte del diccionario de datos, así como cierta información de Undo, pero su función principal sigue siendo la misma de siempre: albergar la *Library Cache*.

La *Library Cache* guarda todas las sentencias, tanto SQL como PL/SQL, que se ejecutan dentro de la base de datos (solo las sentencias, no los datos, que ya se ha visto que se almacenan en la *Buffer Cache*). Estas sentencias se dejan en memoria durante algún tiempo por si volvieran a ser ejecutadas.

Uno de los principales problemas de la *Shared Pool* es que puede llegar a fragmentarse, y como resultado, arrojar el famoso error ORA-04031. Este error no quiere decir que no haya memoria libre en la *Shared Pool*, lo que significa es que no se encontró una porción de memoria contigua (lo que se llama un *chunk*) del tamaño necesario para ubicar la sentencia que se desea ejecutar. Esto ocurre principalmente por dos motivos: aplicaciones que fragmentan mucho la *Shared Pool* o ejecución de consultas demasiado grandes. El que la consulta sea demasiado grande no es porque su resultado ocupe mucho, sino porque el propio tamaño de la consulta es excesivamente largo. Suele ser muy común, por ejemplo, en consultas que se generan de forma dinámica con la mala práctica de concatenar valores en la cláusula *where*. Oracle implementa una solución para otorgar un espacio reservado a las sentencias que tienen un tamaño grande: el espacio reservado de la *Shared Pool* (*Shared Pool Reserved Size*).

SQL> show parameter shared_pool_reserved_size		
NAME	TYPE	VALUE
shared_pool_reserved_size	big integer	6501171

Para ver en detalle el uso de esta parte de memoria, se debe consultar la vista *v\$shared_pool_reserved*.

SQL> select free_space,used_space 2 from v\$shared_pool_reserved 3 / FREE_SPACE USED_SPACE	
4683536	0

Nótese que 6501171 es distinto de 4683536+0, esto es debido a que Oracle no reserva de inmediato ni el espacio de la *shared_pool_reserved_size*, ni el de ninguna otra de las pool de memoria. Los valores impuestos son simplemente límites máximos o mínimos, pero esto no quiere decir que realmente se usen desde el momento mismo del inicio de la instancia. Para ver con más detalle la estructura en memoria de la *shared_pool_reserved*, se pueden consultar las vistas *x\$ksmspr*, *x\$ksppi* y *x\$ksppcv*,

aunque no es aconsejable hacerlo en entornos de producción porque provocan bloques que pueden hacer incluso caer la base de datos.

```
SQL> select ksmchcom, sum(ksmchsiz)
  2  from x$ksmspr
  3* group by rollup(ksmchcom)

KSMCHCOM      SUM(KSMCHSIZ)
-----
free memory          4683536
reserved stope       1056
                      4684592

SQL> select a.ksppinm "Parameter", b.ksppstvl "Value", a.ksppdesc "Description"
  2  from x$ksppi a, x$ksppcv b
  3  where a.indx = b.indx
  4    and substr(ksppinm,1,1)='_'
  5* and a.ksppinm like '%reserved_min%'
  5* order by a.ksppinm

Parameter          Value   Description
-----
_shared_pool_reserved_min_alloc 4400   minimum allocation size in bytes for reserved area
of shared pool
```

No solo es importante el tamaño de lo que se ejecuta, también es importantísimo el cómo se libera de la memoria. No liberar los cursos abiertos provoca que la *Shared Pool* crezca de forma desproporcionada. Una simple conexión a la base de datos, implica abrir un montón de cursos. Esto se puede ver en la vista *v\$open_cursor*:

```
SQL> select sid, count(8) from v$open_cursor group by sid;
      SID COUNT(8)
-----
        1      8
       13      6
       31      2
       14      1
       24      4
       27      3
       15     23
       16      1
```

El máximo número de cursos que pueden abrirse en una base de datos está establecido por el parámetro *open_cursors*.

```
SQL> show parameter cursor
cursor_space_for_time           boolean    FALSE
```

open_cursors	integer	300
session_cached_cursors	integer	50

Oracle maneja los cursores con listas que tienen, de nuevo, el algoritmo LRU, por lo que si es necesario introducir en la *Shared Pool* un cursor y no hay más espacio libre, la forma de conseguir hueco es eliminando el cursor que lleva más tiempo sin ser usado. Cuando el parámetro *cursor_space_for_time* está en *false*, se da permiso al gestor para que quite el cursor que lleva más tiempo sin ser usado aún cuando este esté abierto. En cambio, si está con valor *true*, solo lo podría quitar si estuviese cerrado, por lo que es fácil deducir que, en este caso, es necesario dedicar más memoria a la caché *Shared Pool*, aunque bien es cierto, que probablemente ahorrará en consumo de CPU. Si en los informes *AWR* apareciese un porcentaje de fallos en la *Library Cache*, el valor del parámetro debería ponerse a *false*, ya que, si fuese *true*, al dejar los cursores más tiempo en memoria sin poder liberarlos, aumentaría ese porcentaje de fallos.

El parámetro *session_cache_cursors* fue creado con Oracle Forms (herramienta para crear aplicaciones basadas en formularios que se nutren de una base de datos Oracle), y hace referencia a otra lista adicional de cursores paralela a la lista de *open_cursors*. Es útil cuando una aplicación utiliza una y otra vez los mismo cursores. En el caso de los *Forms*, cuando se navega entre las diferentes pantallas, los cursores ejecutados en formularios previos permanecen abiertos, y así al regresar a ellos durante la navegación por la aplicación, no es necesario volver a abrirlos, lo cual implica, como siempre, menos consumo de CPU, pero más consumo de memoria.

Otro parámetro relevante en el uso de la *Shared Pool* es el *cursor_sharing*, que también se tratará en la sección de Recursos. Allí se verá que puede tener alguno de los siguientes valores: *EXACT*, *FORCE* o *SIMILAR*; y que estos dos últimos posibilitan que el gestor cambie automáticamente los literales de las sentencias ejecutadas por variables *bind*, lo cual implica un aumento en el uso de la memoria en la cache de la *Shared Pool* debido a la generación de cursores hijo que esa transformación provoca.

Utilizando la API *dbms_shared_pool* se puede hacer que cursores, bloques PL/SQL anónimos y objetos, queden cacheados en la *Shared Pool*.

4. **Large Pool Size:** Esta sección de memoria se emplea en caso de hacer uso de RMAN, ejecutar sentencias en paralelo, o utilizar servidores compartidos (*shared servers*).

En los scripts de backups con RMAN es necesario crear canales de comunicación con la base de datos (*allocate channel*):

```
Run
{
  allocate channel c1 type 'sbt' params...
  ...
}
```

```
release channel c1
}
```

Cada vez que se crea un canal se abre una conexión a la base de datos, pero la información no fluye como las sesiones normales, parte de esos datos se almacenan en la *Large Pool*.

La paralelización se estudiará más adelante, pero sirva como adelanto el decir que los mensajes que se transmiten entre los procesos en paralelo necesitan un lugar en memoria, este lugar es la *Large Pool*, por lo que de nuevo, el tamaño de esta estructura dependerá también del uso o no de consultas en paralelo.

Como se ha mencionado antes, los procesos pueden funcionar de forma dedicada o compartida, dependiendo el tamaño de la memoria, del uso o no de esta configuración. Un proceso en memoria de un proceso dedicado tiene más o menos la siguiente estructura:

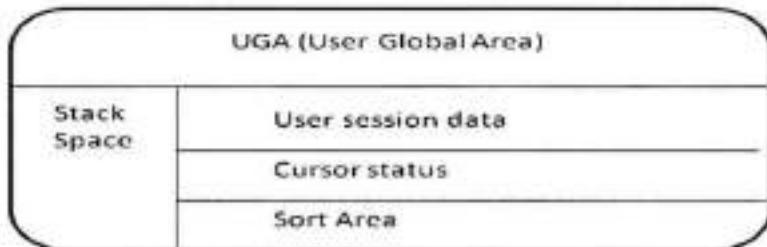


Figura 4.13: Distribución de memoria de un proceso dedicado.

Cuando los procesos no se conectan de forma dedicada sino compartida usando *shared servers*, esta estructura cambia y lo que se observa de UGA (del inglés *User Global Area*, área global de usuario, es decir, la memoria que necesita una sesión) no se almacena en cada uno de los pedazos de memoria privada de los procesos, sino que se guarda en la *Large Pool*.

La UGA posee parte de los resultados de las sentencias que se ejecutan, y aunque los bloques estarán en la *Buffer Cache*, en ocasiones ocurre que para proteger la *Buffer Cache*, el gestor realiza casi toda la operativa de las sentencias en la UGA, lo que puede llegar a provocar que en algunos informes AWR, dentro de la sección de *Actividad de la Instancia*, la cantidad de UGA sea incluso mayor que la cantidad de *consistent gets*.

Por tanto, para que haya un buen rendimiento de la base de datos, es necesario que la *Large Pool* tenga un tamaño suficientemente grande para proveer a todas las conexiones de su información. Este tamaño de UGA que es necesario tener en cuenta en la *Large Pool* puede averiguarse consultando la vista *v\$sysstat* filtrada por la cadena *UGA*:

```
SQL> select name, value from v$sysstat where upper(name) like '%UGA%';
NAME          VALUE
-----
session uga memory      21482049736
session uga memory max   99997356
```

5. **Java Pool Size:** Oracle puede almacenar y ejecutar código Java desde su propio motor, y para ello una de las cachés que tiene es la Java Pool.
6. **Streams Pool Size:** Es el segmento de memoria utilizado para replicar información hacia bases de datos en standby.

Es buena idea tener activado el dimensionamiento automático de la memoria, esto es, el ASMM, pero también es bueno controlarlo un poco para evitar errores. Por ejemplo, ¿qué pasaría si la *Shared Pool* creciera desproporcionadamente por culpa de la excesiva generación de cursor hijo? Pues podría estar adueñándose de espacio en memoria que en circunstancias normales correspondería a la *Buffer Cache*, provocando con ello que aumentara el I/O.

Eso no ocurriría si se blindara de alguna manera el tamaño de la *Buffer Cache*, asegurándole un mínimo de espacio, lo cual se conseguiría asignándole un valor concreto al parámetro de esa caché (*db_buffer_cache*) en el arranque de la instancia (en el pfile o en el spfile). Y así con el resto de segmentos de memoria. Siempre es recomendable tener unos mínimos configurados en las caches para poder tener un poco de control en el rendimiento de la base de datos.

El parámetro *sga_target* será el que defina cuánto espacio podrá manejar el gestor para redimensionar en memoria. Existe otro parámetro, la *sga_max_size*, que indica el tamaño máximo de memoria que podrá reservar la instancia dentro del sistema operativo. Si *sga_target* fuese mayor que *sga_max_size*, la *sga_max_size* se ampliaría automáticamente para tomar el valor de la *sga_target*. Si el valor definido para las cachés fuese mayor que el de la *sga_max_size*, aparecería un error al iniciar la instancia y se abortaría el proceso.

PGA

La PGA o *Program Global Area* (área global de programa) es esa pequeña porción de memoria dedicada a cada uno de los procesos que atacan a la base de datos. El tamaño de esta está en función del tipo de conexión que se realiza, ya que, como se ha dicho anteriormente, cuando los procesos no se conectan de forma dedicada sino compartida, la memoria de las sesiones (UGA) no se almacena en la PGA, sino que se guarda en la *Large Pool* dentro de la SGA, por lo que cuando los procesos son dedicados, el consumo de memoria en la PGA es mayor.

La PGA contiene información sobre los cursorios de la sesión, su estado y otros datos de las propias sentencias ejecutadas. Además en ese pedazo de memoria se realizan operaciones

de ordenación (*sort* en inglés), operaciones de *hash* y creación de índices *bitmaps*.

El tamaño de la PGA se controla con el parámetro *pga_aggregate_target*, aunque a diferencia de lo que ocurría con la SGA, no se reserva todo el espacio en memoria. Para conocer qué tamaño total se usa de PGA, se puede recurrir a la vista *v\$process*.

```
SQL> select sum(pga_used_mem)
  2  from v$process;
SUM(PGA_USED_MEM)
-----
28117932
```

Una PGA muy pequeña no soportará ciertas operaciones, por lo que serán necesarias escrituras en el tablespace temporal, lo cual ralentizará las ejecuciones. Un tamaño inicial para configurar el valor de la PGA podría ser, dada una cantidad de memoria prevista para la base de datos:

- en entornos OLTP, usar: 80 % para la SGA (*sga_max_size*) y 20 % para la PGA (*pga_aggregate_target*).
- en entornos OLAP o DSS (por la cantidad de operaciones de ordenación, creación de índices, o consultas con *order by* o *distinct*), usar: 50 % para la SGA y 50 % para la PGA.

Estas son solo recomendaciones iniciales, una vez está la base de datos en producción, se puede consultar la vista *v\$pga_target_advice* para controlar el uso de la PGA.

Un indicativo de que es necesario aumentar el tamaño de la PGA es el número de *multi-pass* que se han producido. Cuando el gestor calcula el tamaño de memoria que necesita una operación, y luego ese tamaño no es suficiente, se ve obligado a hacer lo que se llama un *one-pass*, es decir, un redimensionamiento de memoria; cuando tiene que realizar varios redimensionamientos para la misma operación, se dice que ha hecho un *multi-pass*. Este tipo de comportamiento degrada mucho el rendimiento de las operaciones que lo provocan, por lo que deberían evitarse en la medida de lo posible. En la vista *v\$sql_workarea_histogram*, se pueden evaluar el número de *one-pass* y *multi-pass* ocurridos.

```
SQL> desc v$sql_workarea_histogram
Name          Null?    Type
-----        -----
LOW_OPTIMAL_SIZE      NUMBER
HIGH_OPTIMAL_SIZE     NUMBER
OPTIMAL_EXECUTIONS    NUMBER
ONEPASS_EXECUTIONS    NUMBER
MULTIPASSES_EXECUTIONS NUMBER
TOTAL_EXECUTIONS      NUMBER
```

También, en los informes de AWR se puede analizar la presencia del *passing*:

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	1,465,779	1,465,779	0	0
64K	128K	346	346	0	0
128K	256K	254	254	0	0
256K	512K	1,500	1,500	0	0
512K	1024K	57,198	57,198	0	0
1M	2M	5,435	5,435	28	0
2M	4M	177	70	106	1
4M	8M	63	12	51	0
8M	16M	5	0	5	0
16M	32M	4	0	4	0
32M	64M	3	0	1	2
64M	128M	2	0	4	4
128M	256M	1	0	1	0
256M	512M	1	0	0	1

Figura 4.14: Extracto del informe AWR: one-pass y multi-pass.

En este informe se aprecia que se está realizando *multi-pass*, algo que no conviene, pero también es cierto que un tamaño excesivo en la *pga_aggregate_target* puede llevar a malos comportamientos del optimizador, como por ejemplo, que determine que todas los joins vayan por *hash*. Lo mejor es no tener *passing*, pero si existe, se debe comprobar que para la mayoría de las operaciones realizadas, la PGA tiene un tamaño adecuado.

Memory target

Ya se ha visto que en la versión 10g de Oracle, aparece el ASMM, pues bien, en la versión 11g, Oracle va más allá y mezcla la SGA con la PGA introduciendo los siguientes parámetros:

NAME	TYPE	VALUE
memory_max_target	big integer	400M
memory_target	big integer	400M

Es decir, que se define una memoria total para la base de datos, y esta la reparte entre las distintas cachés. De nuevo pueden surgir una infinidad de problemas: malas consultas, aplicaciones que maltratan a la *Shared Pool*, y demás factores que pueden hacer que el rendimiento de la base de datos decaiga, por lo que lo recomendable es establecer unos mínimos para determinadas cachés con el objetivo de que el rendimiento no decaiga en momento críticos. Los redimensionamientos automáticos de las memorias se pueden ver en la vista *v\$memory_resize_ops*:

```
SQL> select component,initial_size,target_size,status
  2  from v$memory_resize_ops
  3 /
COMPONENT          INITIAL_SIZE TARGET_SIZE STATUS

```

shared pool	0	88080384	COMPLETE
PGA Target	0	167772160	COMPLETE
java pool	0	4194304	COMPLETE
streams pool	0	0	COMPLETE
large pool	0	4194304	COMPLETE
RECYCLE buffer cache	0	0	COMPLETE
SGA Target	0	251658240	COMPLETE
DEFAULT buffer cache	146800640	159383552	COMPLETE
DEFAULT buffer cache	146800640	146800640	COMPLETE
ASM Buffer Cache	0	0	COMPLETE
DEFAULT buffer cache	0	146800640	COMPLETE
DEFAULT 2K buffer cache	0	0	COMPLETE
DEFAULT 4K buffer cache	0	0	COMPLETE
DEFAULT 8K buffer cache	0	0	COMPLETE
DEFAULT 16K buffer cache	0	0	COMPLETE
DEFAULT 32K buffer cache	0	0	COMPLETE
KEEP buffer cache	0	0	COMPLETE
PGA Target	167772160	155189248	COMPLETE
SGA Target	251658240	264241152	COMPLETE

Del resultado de la consulta anterior es fácil ver que, efectivamente, se han realizado cambios en los tamaños de las cachés, ya que en muchas, el valor del campo *target_size* ha variado respecto del inicial.

4.4.2. Recursos

Se entiende por recurso alguno de los siguientes elementos: memoria, procesador (*CPU*), dispositivos de entrada/salida (*I/O*) y red (en inglés *network*).

La forma en que una sentencia es ejecutada determina drásticamente el mayor o menor consumo de estos recursos, pero es el sistema operativo, y no la base de datos, quien al final se encarga de gestionar dichos recursos, por lo que es muy importante conocer los comandos adecuados de sistema operativo que darán información sobre su estado.

Por ejemplo, dada una sesión concreta (con *sid* conocido), a través de las vistas *v\$session* y *v\$process* se puede saber el identificador de proceso (*PID*) de sistema operativo que da servicio a la sesión de base de datos, y con ese dato es fácil interrogar al sistema operativo para averiguar qué está haciendo esa sesión y cuántos recursos está consumiendo (basta con invocar los comandos *truss*, *strace*, *tusc*).

```
SQL> select p.spid
   from v$session s, v$process p
  where s.paddr=p.addr
    and s.sid = 81
```

Como ya se explicó, dentro de la base de datos Oracle, los informes estáticos obtenidos a través de la herramienta *AWR* ofrecen toda la información necesaria para apreciar donde existe un mayor consumo de un recurso en concreto.

Pero, ¿cómo se debe estudiar cada recurso?

Memoria

Obviamente, el consumo de memoria depende de lo que se esté ejecutando en cada momento, pero aparte de esto, está también determinado por la configuración que tenga prevista la base de datos para soportar las sesiones.

Ya se sabe que si las sesiones son en modo compartido, es decir, con *shared server*, la memoria usada se obtendrá de la *Large Pool* que reside dentro de la *SGA*, por lo que el máximo consumo de memoria estará limitado por el parámetro *sga_max_size*, que es el que define el tamaño máximo de memoria que podrá reservar la instancia dentro del sistema operativo.

En cambio, si las sesiones tienen procesos dedicados, el consumo total de memoria será como máximo lo que necesita cada sesión (definida por la *UGA*) por todas las sesiones en curso que tenga la base de datos, las cuales no podrán superar el valor asignado al parámetro *sessions*.

Para ver la memoria que consume un proceso concreto dentro de la base de datos, debe recurrirse a los campos *pga_used_mem*, *pga_alloc_mem*, *pga_freeable_mem* y *pga_max_mem* de la vista *v\$process*. Para ver lo que consume respecto del sistema operativo, se pueden usar los comandos *sar*, *vmstat*, *glance*, *top* o *topas*.

CPU

Podría ocurrir que en la ejecución de un programa no se accediera al disco o no se hiciera uso de la red, pero nunca dejaría de hacerse uso de la CPU. Todo pasa por el procesador, por lo que suele ser siempre el más perjudicado ante cualquier ejecución.

Según los informes *AWR*, un procesador puede estar ocupado haciendo una de las siguientes tres cosas:

- *Recursive CPU usage*: uso de CPU para consultas recursivas. Cada ejecución lleva implícitas una serie de consultas internas que Oracle debe realizar al diccionario de datos antes de determinar el plan de ejecución, además de otra cantidad de transformaciones y operaciones internas para saber, por ejemplo, dónde ubicar la sentencia dentro de la *Shared Pool*. Todo esto consume CPU y está registrado en *recursive CPU usage*. Un ejemplo muy gráfico para cuantificar este coste de CPU es la diferencia de consumo aparecido en las estadísticas de una *select* hecha sobre una tabla directamente o hecha sobre la tabla referenciada a través de un sinónimo público. En este último caso, puesto que el gestor se ve obligado a consultar el diccionario de datos para averiguar a qué objeto apunta el sinónimo, el número aparecido en la línea de las llamadas recursivas (*recursive calls*) de las estadísticas de la *select* (obtenidas al usar el comando *set autot on stat* previo a la *select*) es mayor, es decir, hay más llamadas recursivas.
- *Parse time CPU*: tiempo de CPU dedicado al *parsing*.
- *Other CPU*: equivale a la resta del total de consumo de CPU menos la suma de las dos anteriores.

¿Qué es el *parsing*? Lo primero que hace Oracle cuando se ejecuta una sentencia es comprobar que sintácticamente esta es correcta. Luego la almacena en memoria asociándola con un identificador especialmente creado para ella: el *sql_id*. La almacena en memoria con el objetivo de compartirla en el caso de que otro usuario o la misma sesión quiera ejecutarla de nuevo.

Para pasar las comprobaciones y demás chequeos, la primera vez que se ejecuta la base de datos realiza lo que se llama un *hard parsing*. Cuando ya se encuentra en memoria, los chequeos que realiza son mucho menores que la primera vez, y a esto se le llama *soft parsing*.

Uno de los mayores problemas del consumo de CPU viene dado por el uso de sentencias que no se pueden compartir, como por ejemplo las dos siguientes:

```
select nom from emp where id=1234;
select nom from emp where id=4321;
```

Al haber ejecutado las consultas usando valores literales en la cláusula *where*, ambas sentencias precisan de un *hard parsing*. Para evitar esto, se pueden ejecutar las consultas usando variables *bind*, es decir, en vez de usar valores literales, usar variables que tomarán los valores por parámetro dentro de la aplicación. Con esta sencilla medida, el optimizador no necesitará hacer *hard parsing* sino que hará *soft parsing*, con lo que se ahorrará consumo de CPU, eso sí, en detrimento de un mayor consumo de la caché de memoria *Shared Pool*, que será donde tendrán que almacenarse los cursores hijo que la sentencia con las variables *bind* genere.

Para facilitar esto, existe un parámetro (el *cursor_sharing*) que permite indicarle a Oracle que sustituya automáticamente los valores literales por variables *bind*, lo malo es que el valor por defecto de ese parámetro (*EXACT*) inhibe esta capacidad, por lo que es sumamente recomendable cambiar ese parámetro explicitamente para asignarle alguno de los otros dos valores admitidos (*FORCE* o *SIMILAR*).

I/O

Evitar que el acceso de lectura o escritura de los datos almacenados en disco suponga un cuello de botella en el rendimiento de una base de datos es prioritario en todo proceso de optimización, ya que, entre otras cosas, mucho consumo de I/O aumentará inevitablemente el consumo de CPU.

En una base de datos Oracle puede haber cinco tipos distintos de ficheros guardados en disco:

- de redo (*redofiles*).
- temporales (*tempfiles*).
- de datos (*datafiles*): del tablespace SYSTEM, de los tablespaces de UNDO, o del resto de tablespaces.

- de control (*controlfiles*).
- de log archivados (*archivelog*).

Dado el siguiente extracto sobre los eventos de espera de un informe AWR:

Event	Waits	Time(s)	Avg Wait(ms)	% Total	Call Time	Wait Class
log file sync	434,204	5,708	13		25.8	Commit
log file parallel write	276,295	2,083	8		22.5	System I/O
db file sequential read	99,530	1,657	5		20.5	User I/O
CPU time		1,038			18.3	
db file scattered read	98,503	1,001	3		6.8	User I/O

Figura 4.15: Extracto del informe AWR: eventos de espera.

¿qué se puede decir que está pasando? que el evento *log file sync* es el que más tiempo de espera acumula. Lo primero que habría que revisar es cómo está configurado todo lo que tenga que ver con los ficheros de redo. Se puede empezar por examinar en qué tipo de discos se están almacenando. La información de redo es sin duda alguna la más importante dentro de Oracle, por lo que se recomienda colocar los ficheros de redo en los discos más rápidos que se tengan. Actualmente, los discos más rápidos del mercado son los de estado sólido (SSD=Solid-State Drive) y las pruebas han demostrado que son una buena alternativa para albergar los ficheros de redo.

Los ficheros temporales se utilizan cuando, por ejemplo, se crean tablas temporales (*create temporary table*) o cuando la *PGA* no tiene tamaño suficiente para soportar la operación de la sesión en memoria, y necesita hacer lo que se llama *spilling*, es decir, reubica la operación en curso, generalmente una ordenación (*sort*) en el tablespace temporal. Un ejemplo simple de ejecución de *sort* es la creación de un índice. Al crear un índice en una tabla es necesario hacer una ordenación de los valores de la tabla. Si es demasiado grande y no se puede realizar en memoria, es necesario crear segmentos temporales para llevarlo a cabo, y estos se crean en el tablespace temporal.

En los sistemas Unix y Linux, el fichero temporal es un archivo que originalmente solo consume 128K, y a medida que lo va necesitando va aumentando y también disminuyendo de tamaño. En ocasiones, esto puede confundir bastante ya que en sistema operativo figurará que el fichero es del tamaño diseñado en la sentencia de creación del tablespace temporal, mientras que el verdadero espacio ocupado en disco es variable según demanda.

```
SQL> select file_name from dba_temp_files;
FILE_NAME
-----
/opt/oracle/datos/t11r2/temp01.dbf
```

```
ls -l /opt/oracle/datos/t11r2/temp01.dbf
-rw-r----- 1 oracle dba 268443648 2011-03-10 11:00 /opt/oracle/datos/t11r2/temp01.dbf
```

```
du -ks /opt/oracle/datos/t11r2/temp01.dbf
1284  /opt/oracle/datos/t11r2/temp01.dbf
```

Del *ls -l* se deduce que el fichero es de 268443648 bytes, mientras que del *du -ks* resulta que el archivo solo ocupa 1284Kb.

El tablespace SYSTEM contiene toda la información de los metadatos de la base de datos, pero los usuarios directamente no pueden hacer modificaciones sobre él. No obstante, es importante saber que si se tiene activada la auditoría se debe monitorizar periódicamente el crecimiento de este tablespace, ya que la información sobre esta se almacena en dos tablas dentro de dicho tablespace.

El tablespace de UNDO apareció en la versión 9i de Oracle con la novedosa característica del *Automatic UNDO Management* (direcciónamiento automático del UNDO) y es de vital importancia en el funcionamiento normal de la base de datos porque en él se guardan los valores antiguos de los registros modificados. Su tamaño estará en función del espacio que necesiten las transacciones en curso, más el espacio destinado a guardar los valores antiguos de tantas transacciones pasadas como se quieran recuperar con la opción del *Flashback Query* y *Flashback Table* (flashback se traduce como ir al pasado). Por ejemplo, para extraer el contenido de una tabla tal y como estaba 24 horas antes, habría que ejecutar: *select * from emp as of timestamp sysdate - 1*.

Para llevar una tabla al pasado habría que activar previamente la opción *row movement* (movimiento de fila):

```
SQL> alter table tj.tab1 enable row movement;
Tabla modificada.
SQL> flashback table tj.tab1 to timestamp sysdate - 1;
Flashback terminado.
SQL> alter table tj.tab1 disable row movement;
Tabla modificada.
```

Con objeto de evitar que aparezca el error *ORA-01555* (que significa que una consulta no puede obtener los bloques de datos adecuados porque han sido cambiados y tampoco existen ya los originales en los bloques de UNDO), el parámetro *undo_retention* debe tener un valor adecuado. Lo ideal es poner como mínimo el valor de la duración de la consulta más larga ejecutada. Para averiguarlo basta con lanzar la siguiente *select*:

```
SQL> select max(maxquerylen) from dba_hist_undostat;
MAX(MAXQUERYLEN)
-----
120
```

El resto de tablespaces tienen la información de usuario o aplicación; tablas, índices...

Antiguamente existía una máxima de optimización que decía que los índices debían estar en discos distintos de aquellos que albergaban los datos, para evitar contención de lectura/escritura en los discos, es decir, para evitar que el I/O se convirtiera en un cuello de botella. Actualmente, esa recomendación está obsoleta y la razón es bien sencilla: la tecnología de las nuevas cabinas de disco hacen muy difícil (cuando no burocráticamente imposible) hacer esa separación. Básicamente el motivo es porque el dispositivo de almacenamiento que se pone a disposición de la base de datos, o de cualquier otro aplicativo, no es un disco físico como tal, sino un conjunto de pedacitos de disco (una LUN).

A la evolución de las cabinas de disco hay que sumarle la evolución en los gestores de base de datos que cada vez con más fuerza están tendiendo a manejar por sí mismos el almacenamiento que se les proporciona directamente de las cabinas de disco, sin pasar por la manipulación que tendría que hacer el sistema operativo si se usara el modelo tradicional de ficheros como dispositivos en modo crudo (*raw device*) o de ficheros en sistema de ficheros (*filesystems*). En Oracle este nuevo concepto de gestión propia del almacenamiento surgió en la versión 10g con el nombre de ASM (del inglés *Automatic Storage Management*, gestión automática del almacenamiento). En la versión 11g se amplia el concepto con el ACFS (del inglés *ASM Cluster File Systems*).

Red

La fibra y las redes actuales proporcionan una estabilidad sorprendente en las comunicaciones de hoy en día, sin embargo, desde el punto de vista del dba, es posible mejorarlas dimensionando adecuadamente dentro del *listener* el tamaño del *SDU* (del inglés *Session Data Unit*, unidad de datos de sesión), para optimizar la tasa de transferencia de datos enviados a través de la red.

Por ejemplo, un programa cliente se conecta a través de SQL*Net, es decir, a través del listener a una base de datos situada en una máquina remota. Si el cliente hiciera muchas llamadas con pocos datos, sería conveniente disminuir el *SDU* para que el tamaño del paquete de llamada fuese más pequeño y la comunicación fuera más rápida. Pero si fuesen pocas consultas con muchos datos, lo ideal sería aumentar el *SDU* para que los datos se pasasen más ágilmente.

Un ejemplo muy gráfico sobre esto se consigue tan solo con modificar el parámetro `arraysize` para una sesión de SQL*Plus en la que se hace una simple consulta a una tabla ejemplo. El parámetro `arraysize` actúa en el SQL*Plus sobre la cantidad de información enviada y recibida, tal como el `SDU` lo hace respecto del listener.

```
SQL> show arraysize
arraysize 15
SQL> set autot on stat
SQL> select *
  2  from tab_we;
  ...
1979,50903 aaaaaaa10489zzzzz
9621,74166 aaaaaaa10490zzzzz
```

```

9      2471,65852 aaaaaaa10491zzzzzz
10     1430,0229 aaaaaaa10492zzzzzz
11     8230,46474 aaaaaaa10493zzzzzz
12     10000 filas seleccionadas.

13
14
15 Estadísticas
16 -----
17     0 recursive calls
18     0 db block gets
19     726 consistent gets
20     0 physical reads
21     0 redo size
22     486322 bytes sent via SQL*Net to client
23     7722 bytes received via SQL*Net from client
24     668 SQL*Net roundtrips to/from client
25     0 sorts (memory)
26     0 sorts (disk)
27     10000 rows processed

28
29 SQL> set arraysize 5000
30 SQL> show array
31 arraysize 5000
32 SQL> select *
33   2 from tab_we;
34 ...
35 2471,65852 aaaaaaa10491zzzzzz
36 1430,0229 aaaaaaa10492zzzzzz
37 8230,46474 aaaaaaa10493zzzzzz
38 10000 filas seleccionadas.

39
40 Estadísticas
41 -----
42     277 recursive calls
43     0 db block gets
44     99 consistent gets
45     71 physical reads
46     0 redo size
47     419812 bytes sent via SQL*Net to client
48     407 bytes received via SQL*Net from client
49     3 SQL*Net roundtrips to/from client
50     5 sorts (memory)
51     0 sorts (disk)
52     10000 rows processed

```

De las líneas 2, 22 y 23 se ve que para un *arraysize* de 15 se envían 486322 bytes al cliente, y se reciben 7722, mientras que si el *arraysize* es de 5000 (línea 29), se envían 419812, pero solo se reciben 407 (líneas 47 y 48).

A continuación se muestra un extracto del fichero *listener.ora* donde se puede ver la definición del SDU:

```

SID_LIST_LISTENER=
  (
    SID_LIST=
      (
        SID_DESC=
          (
            SDO=4096)
          (SID_NAME=test2)
          (ORACLE_HOME = C:\oracle\product\10.2.0\db_1)
        )
      )
  )

```

4.4.3. Estadísticas

El componente del gestor que lleva el peso de la decisión sobre qué camino hay que seguir para resolver una sentencia es el CBO (*Cost Based Optimizer*, optimizador basado en coste). Para poder trabajar debe contar con información estadística que le ayude a trazar el plan de ejecución adecuado. Si no se tiene una buena información sobre los objetos, es muy improbable que el gestor genere un buen plan de ejecución.

¿Sabías que . . . ? El libro más popular sobre Oracle es acerca del CBO (del inglés *Cost-Based Optimizer*, optimizador basado en costes) y lo ha escrito Jonathan Lewis (un supergurú). En él se describen muchos de los comportamientos en que se basa Oracle para poder obtener el plan de ejecución con el menor coste posible. El libro se titula *Cost-Based Oracle Fundamentals*.

Un plan de ejecución no es más que la forma en que Oracle determina que debe ir a buscar los datos resultado de una consulta, o una sentencia DML. El dato se encuentra almacenado en una tabla o segmento hallado en un tablespace concreto. A su vez, el segmento está dividido en extensiones y las extensiones en bloques. Para extraer un dato específico el optimizador debe decidir qué bloques son necesarios para devolver el resultado deseado, por lo que si cuenta con índices, tablas y columnas, lo ideal es buscar el dato revisando la menor cantidad posible de bloques. Esto es un plan de ejecución: ¿dónde hay que ir para obtener el resultado de la consulta?; que es lo mismo que decir; ¿qué plan se debe seguir para obtener los datos?

A menudo surge un debate nada baladí acerca de quién es el responsable de determinar qué estadísticas son las más adecuadas para una base de datos concreta: ¿el desarrollador o el dba? El desarrollador explota la información de la base de datos, pero no tiene por qué conocer los entresijos de la misma (¿o sí?), mientras que el administrador se encarga de mantener la base de datos en un estado óptimo que permita su explotación adecuadamente, pero no tiene por qué saber cómo se está accediendo a esos datos (¿o sí?). Para no entrar en polémicas se puede recurrir a la figura del analista, que es el que diseña la base de datos y se supone conoce la distribución de sus datos físicamente. Además, debe saber cómo funciona la aplicación y cómo se va a comportar en tiempo real.

Sea quien sea, lo que está claro es que tener unas estadísticas adecuadas y actualiza-

das es algo imprescindible. Se puede elegir recolectar datos estadísticos a cuatro niveles distintos, recogiéndose en cada nivel un tipo de información u otra:

- **Sistema:** rendimiento y uso de I/O, rendimiento y uso de CPU.
- **Diccionario.**
- **Tabla:** número de filas, número de bloques, la media de la longitud de la columna.
- **Índice:** altura del índice, número de hojas, clustering factor, número de claves distintas.
- **Columnas/histogramas:** número de valores distintos, cantidad de nulos, extensión de estadísticas, histograma.

El gestor tiene en cuenta cómo es el entorno donde se encuentra para elaborar el cálculo de lo que le supone llevar a cabo las operaciones de ejecución de las sentencia, es decir, que la misma sentencia ejecutada en dos bases de datos idénticas, pero ubicadas en entornos diferentes, tendrá probablemente distinto plan de ejecución. Esto es muy común que ocurra cuando se suben a la máquina de producción versiones de programas desarrollados en un entorno de desarrollo. Típicamente las máquinas de producción son más potentes que las de desarrollo, por lo que puede ocurrir, por ejemplo, que en el entorno de desarrollo una consulta tenga menor coste yendo por índice, y en cambio, en producción, la misma consulta resulte más óptima trayéndose todos los bloques de los datos de la tabla directamente desde el disco.

Pues bien, para conocer cómo es el entorno, se usan las estadísticas de sistema, las cuales se obtienen usando el procedimiento *dbms_stats.gather_system_stats*.

Las otras estadísticas que tienen que ver más con la administración de base de datos que con las aplicaciones que usan la base de datos, son las estadísticas del diccionario. Normalmente, ya sea por desconocimiento o por mantener la estabilidad del gestor, no se usan, pero si en la base de datos que se esté trabajando se realizan muchas operaciones de DDL (esquemas que crecen y desaparecen, sinónimos públicos nuevos, creación y borrado de índices a diario, ampliación, creación y borrado de ficheros de datos o tablespaces), será altamente recomendable refrescar periódicamente las estadísticas guardadas sobre el diccionario. Para hacerlo, basta con invocar el procedimiento *dbms_stats.gather_dictionary_stats*.

También hay un procedimiento destinado a tomar estadísticas solo de un esquema concreto, por ejemplo el *TJ*:

```
SQL> exec dbms_stats.gather_schema_stats('TJ');
Procedimiento PL/SQL terminado correctamente.
```

¿Sabías que ...? Se pueden extraer estadísticas del diccionario obteniendo las estadísticas del esquema *SYS*, porque este es el dueño del diccionario, pero, según la documentación de Oracle, el procedimiento *dbms_stats.gather_dictionary_stats* realiza más tareas a parte de solo la obtención de las estadísticas de los objetos del usuario *SYS*.

Dentro del paquete `dbms_stats` existe además un procedimiento (el `dbms_stats.gather_database_stats`) para tomar las estadísticas de la base de datos completa, pero estas no incluirán al diccionario.

Las estadísticas de los segmentos, ya sean tablas o índices, se consiguen invocando a `dbms_stats.gather_table_stats` y `dbms_stats.gather_index_stats`, respectivamente. Las estadísticas de las columnas están implícitas en las de las tablas.

Debe tenerse muy en cuenta que cuando el tamaño de las tablas o índices es muy grande, no es conveniente tener que recorrerse todo el segmento para extraer sus datos estadísticos, ya que esa operación podría llevar incluso días. Es por ello que resulta de mucha utilidad indicar qué porcentaje del segmento se va a visitar, y luego, el gestor extrapolará los datos al resto del segmento. Esto se hace definiendo un valor para el parámetro `ESTIMATE_PERCENT` de los procedimientos del paquete `dbms_stats`. Con valores bajos es posible que no se consiga una imagen real del segmento. Un símil de esto pueden ser las encuestas que se realizan sobre la intención de voto antes de unas elecciones, cuanta más gente es preguntada, más posibilidades hay de que el resultado de la encuesta coincida con el resultado de la votación, eso sí, siempre que todos los encuestados digan la verdad y además después vayan a votar. Pues aquí es lo mismo, con un porcentaje mayor de muestra más se acercarán los datos recolectados en las estadísticas a la información real de la descripción de la tabla o índice, pero más tardarán en elaborarse dichas estadísticas.

Otro parámetro importantísimo hayado en el procedimiento `dbms_stats.gather_table_stats` es el `METHOD_OPT`, ya que con él se definen qué columnas van a tener histogramas. Pero, ¿qué son los histogramas?

Estadísticamente hablando, un histograma es la representación de la distribución de los datos de una variable con relación a su frecuencia. Por ejemplo, imagínese que en una tabla llamada `tab1` existe una columna numérica (`col1`) con diez posibles valores distintos (del 1 al 10) almacenados en cinco grupos (llamados `buckets`, jojo!, no confundir con los `buckets` de la `Buffer Cache`), por lo que cada `bucket` guarda dos valores: 1-2, 3-4, 5-6, 7-8, 9-10. Imagínese que la tabla tiene mil registros, los cuales cumplen la siguiente distribución de valores:

Valor	Nº de registros con ese valor
1	10
2	10
3	10
4	10
5	10
6	10
7	800
8	40
9	50
10	50

Cuadro 4.1: Distribución de valores en la columna.

O gráficamente por buckets:

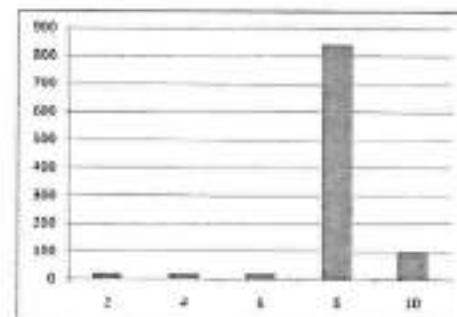


Figura 4.16: Histograma.

El optimizador ve algo parecido a esto, y gracias a ello, es capaz de obtener la selecti-

vidad y la cardinalidad (se estudiarán más adelante), decisivas para saber si tiene que usar o no un determinado índice.

Imagínese ahora que se estuviera realizando una consulta sobre esa columna y no se tuviera su distribución de valores. Imagínese que solo se conociera que la tabla tiene mil registros y que la columna tiene diez valores diferentes (esto sería equivalente a tener un índice sobre la columna, pero no su histograma). Sin saber su distribución exacta, el gestor parte del supuesto de que la columna posee una distribución uniforme, así que ante la sentencia:

```
select * from tab1 where col1=7;
```

Oracle interpretaría que debe extraer 100 registros (selectividad por cardinalidad), cuando la realidad es que son 800. Es más, al tener un índice sobre esa columna, el gestor decidiría usarlo (haría *index range scan*, escaneo por rango de índice), con lo que tendría que leerse el 80 % del índice más el 80 % de la tabla (para obtener el resto de las columnas de la tabla, ya que se seleccionan todas). En cambio, si tuviera histogramas, habría decidido hacer directamente un *full scan* (lectura completa de la tabla), con lo que se habría ahorrado un 60 % del esfuerzo anterior.

Es conveniente conocer que los histogramas en Oracle cuentan con una limitación: el máximo número de *buckets* que se pueden crear es de 255. Esto puede provocar que los histogramas sean uniformes respecto de los 255 *buckets*, o todo lo contrario, e incluso puede llegar a ocurrir que se obtenga un histograma sesgado.

Para obtener histogramas sobre todas las columnas de una tabla basta con ejecutar:

```
SQL> begin
 2  dbms_stats.gather_table_stats(tabname=>'TAB1',
 3                                ownname=>'TJ',
 4                                method_opt=>'FOR ALL COLUMNS SIZE AUTO');
 5  end;
 6  /
Procedimiento PL/SQL terminado correctamente.
```

Con *SIZE AUTO* se está dejando al gestor decidir si crea o no histogramas sobre esa columna, basándose en la distribución de datos y en la carga de trabajo (la cual la obtiene de la tabla interna *col_usage\$*).

Otros posibles valores para *SIZE* son:

- *REPEAT*: crea histogramas de las columnas que ya tienen histogramas.
- *SKEWONLY*: decide si obtiene o no histogramas en función de la definición de la tabla.
- Número de buckets que se crearán (máximo 255).

Para ver la distribución de las columnas que tienen histogramas, se puede consultar la vista *dba_histograms*:

```
SQL> desc dba_histograms
Nombre          Nulo?  Tipo
-----
OWNER           VARCHAR2(30)
TABLE_NAME      VARCHAR2(30)
COLUMN_NAME     VARCHAR2(4000)
ENDPOINT_NUMBER NUMBER
ENDPOINT_VALUE  NUMBER
ENDPOINT_ACTUAL_VALUE  VARCHAR2(1000)
```

Donde *ENDPOINT_VALUE* contiene el valor máximo del *bucket*, y *ENDPOINT_NUMBER* la cantidad de valores hasta ese *bucket*.

4.4.4. Particionamiento y paralelización

Cuando se trabaja con VLDB (*Very Large DataBases*, bases de datos muy grandes), ya sean DW o OLTPs muy grandes, es normal encontrar tablas particionadas, ya que se hacen indispensables para un óptimo rendimiento.

Tampoco es menos común invocar las consultas usando paralelización, es decir, ejecutarlas utilizando más cantidad de CPU, de modo que varias conexiones se reparten la consulta y obtienen su resultado en paralelo, un cachito cada una, para así acabar cuanto antes.

Particionamiento

Lo mejor es explicar esta técnica con un ejemplo. Se tiene una tabla con el histórico de pedidos de una empresa. La tabla guarda hasta un año los pedidos en su OLTP, porque ese es el tiempo en que Hacienda les puede pedir datos sobre las transacciones bancarias hechas sobre sus pedidos (al hablar de histórico necesariamente se está involucrando un campo de tipo fecha). La tabla tiene 120 millones de filas, y ocupa unos 25Gb, por lo que parece obvio que realizar una consulta sobre ella sería costosísimo, y hacer sumatorios o cuentas resultaría inviable. La solución: particionar la tabla, esto es, dividir la tabla en tablas muchísimo más pequeñas para que las operaciones realizadas sobre ellas se hagan en un tiempo razonable. El campo y el tamaño de las particiones se convierten entonces en claves importantes para llegar a conseguir objetivos.

El método para particionar una tabla es muy simple, basta con responder a un par de preguntas: ¿qué columna siempre aparece en las consultas a esa tabla? Si la respuesta fuese ninguna, se formularía la siguiente pregunta: ¿qué columna es la que más aparece en las consultas a esa tabla? Esa será la columna clave para hacer la división en tablas más pequeñas. En el ejemplo anterior podría ser el mes.

Una vez seleccionada la columna, hay que elegir, en función de los valores que almacene y de su variedad, qué tipo de particionamiento se le aplicará. La distribución puede hacerse de varias formas distintas:

- Por rango: mediante un intervalo de valores, por ejemplo entre fechas.
- Por lista: imponiendo valores concretos, por ejemplo los doce meses del año.
- Por hashing: aleatoriamente. Tiene como objetivo realizar una distribución equitativa de los registros sobre las diferentes particiones. Es útil para particionar tablas donde no hay unos criterios de particionado claros.
- Por intervalo: en lugar de indicar los rangos de valores que van a determinar cómo se realiza el particionado, el sistema automáticamente crea las particiones cuando se inserta un nuevo registro en la tabla.
- Composite: permite mezclar los tres métodos de particionado anteriores, formando así particionados compuestos, es decir, en un primer particionado se puede emplear un tipo determinado, y luego para cada partición, realizar un segundo nivel de particionado utilizando otro método.
- System: se define la tabla particionada indicando las particiones deseadas, pero no se indica una clave de particionamiento, ya que es la propia sentencia *insert* la encargada de indicar en qué partición se insertan los datos.

En el ejemplo propuesto originalmente, se podría particionar por lista usando los meses del año, quedando entonces la tabla original dividida en doce tablas con 10 millones de registros cada una, aproximadamente.

El algoritmo de hashing da muy buenos resultados al aplicarlo sobre columnas numéricas con mucha variedad en sus valores, así se logra mayor uniformidad en la distribución y con ello mayor efectividad en las búsquedas. Además, para conseguir el mismo objetivo, lo óptimo es imponer un número de particiones que sea potencia de 2. Esto lo ilustra muy bien el siguiente ejemplo en el que se crean dos tablas con 100000 registros cada una. A la primera se le hacen cuatro particiones, mientras que a la segunda se le crean cinco.

```
SQL> create table tab_par_4
  2  (id number,
  3   info varchar2(100),
  4   constraint tab_par_4_pk primary key (id))
  5   partition by hash (id)
  6* partitions 4
Table created.

SQL> create table tab_par_5
  2  (id number,
  3   info varchar2(100),
  4   constraint tab_par_5_pk primary key (id))
  5   partition by hash (id)
  6* partitions 5
Table created.
```

Figura 4.17: Creación de las tablas particionadas.

```

SQL> insert into tab_par_4
  2 select rownum, 'aaaaa'||rownum
  3 from dual
  4 connect by rownum <= 100000;
100000 rows created.
SQL> commit;
Commit complete.
SQL> insert into tab_par_5
  2 select rownum, 'aaaaaa'||rownum
  3 from dual
  4 connect by rownum <= 100000;
100000 rows created.
SQL> commit;
Commit complete.

```

Figura 4.18: Inserción de registros en las tablas particionadas.

Después de pasar las estadísticas, se obtienen las siguientes distribuciones de datos en las tablas particionadas:

```

SQL> select TABLE_NAME, PARTITION_NAME, NUM_ROWS, BLOCKS
  2 from dba_tab_partitions
  3* where table_name in ('TAB_PAR_4', 'TAB_PAR_5')
  4 order by 1,2;
TABLE_NAME PARTITION_NAME    NUM_ROWS     BLOCKS
-----  -----
TAB_PAR_4  SYS_P21          24945        73
SYS_P22          24956        73
SYS_P23          25209        73
SYS_P24          24890        73
TAB_PAR_5  SYS_P30          12603        43
SYS_P31          24956        73
SYS_P32          25209        73
SYS_P33          24890        73
SYS_P34          12342        43

```

Figura 4.19: Distribución de los datos en las tablas particionadas.

La distribución es completamente uniforme en la *tab_par_4*, mientras que en la *tab_par_5* se observa que la primera y última partición (las *SYS_P30*, *SYS_P34*) contienen menos registros que la partición central (*SYS_P32*).

Paralelización

Otra técnica muy empleada es la paralelización, que si bien puede llegar a ser muy efectiva, no es menos cierto que penaliza el consumo de CPU y memoria de la caché *Large*.

Pool, y en ocasiones, incluso puede hacer aumentar el tiempo de respuesta, ya que no siempre el hardware es suficientemente adecuado para paralelizar todo lo que se le pide, por ello, como siempre, para utilizarlo convenientemente es necesario un mínimo de sentido común.

Una consulta en paralelo consume, en un momento dado, más CPU que una consulta normal. Esto es debido a que se lanzan varias conexiones, cada una ejecutando una parte del trabajo. No es lógico, por tanto, si la máquina donde reside la base de datos tiene solo una CPU exigirle una paralelización de grado 8...

A continuación, se muestra el plan de ejecución (obtenido con el comando *set autotrace* que se explicará más adelante en el apartado de *Plan de ejecución* dentro de *Consultas*) de una consulta paralelizada en grado 8 hecha en una tabla de 4 particiones sobre una máquina con 2 CPUs.

```

18:16:34 SQL> set autot on stat exp
18:16:44 SQL> select /*+ parallel(8) */ count(8)
18:16:48   2 from tab_par_4;
COUNT(8)

100000
Elapsed: 00:00:04.50
Execution Plan
-----
Plan hash value: 1052512711

| Id | Operation          | Name      | Rows|Cost(%CPU)|Time     | TQ |INOUT|PQ Distrib|
| 0 | SELECT STATEMENT   |           | 1| 10 (10)|00:00:01|    |   |
| 1 |  SORT AGGREGATE    |           | 1|       |          |    |   |
| 2 |   PX COORDINATOR   |           |       |       |          |    |   |
| 3 |     PX SEND QC (RANDOM) | :TQ10000 | 1|       |          | (Q1,00|P->S| QC (RAND) |
| 4 |     SORT AGGREGATE  |           | 1|       |          | (Q1,00|PCMP |
| 5 |     PX BLOCK ITERATOR |          | 100K| 10 (10)|00:00:01|(Q1,00|PCMC |
| 6 |       INDEX FAST FULL SCAN|TAB_PAR_4_PK|100K| 10 (10)|00:00:01|(Q1,00|PCMP |

Note
-----
- automatic DOP: computed degree of parallelism is 8
Statistics
-----
 109 recursive calls
 248 db block gets
1601 consistent gets
 244 physical reads
 852 redo size
418 bytes sent via SQL*Net to client
415 bytes received via SQL*Net from client
  2 SQL*Net roundtrips to/from client
  1 sorts (memory)
  0 sorts (disk)
  1 rows processed

```

El plan de ejecución de la misma consulta hecha sin paralelización:

```

SQL> set autot on stat exp
SQL> select count(8)
  2 from tab_par_4;
COUNT(8)

100000

Execution Plan

Plan hash value: 1926177822

| Id | Operation           | Name      | Rows | Cost (%CPU)| Time     |
| 0 | SELECT STATEMENT   |          | 1    | 72  (9) | 00:00:01 |
| 1 |  SORT AGGREGATE    |          | 1    | 72  (9) |          |
| 2 |   INDEX FAST FULL SCAN| TAB_PAR_4_PK| 100K | 72  (9) | 00:00:01 |

Statistics
-----
 1 recursive calls
 0 db block gets
250 consistent gets
 0 physical reads
 0 redo size
418 bytes sent via SQL*Net to client
415 bytes received via SQL*Net from client
 2 SQL*Net roundtrips to/from client
 0 sorts (memory)
 0 sorts (disk)
 1 rows processed

```

Aunque todavía no se sepa interpretar el plan de ejecución, es fácil comparar los dos bloques titulados *Statistics*. Se ve claramente que, en el primero de ellos, los números aparecidos (las operaciones que se estiman se tienen que realizar para resolver la consulta) son bastante más grandes que en el segundo. Puesto que el primer bloque hace referencia al coste que tendría la consulta ejecutada con la paralelización de grado 8 y el segundo es el coste de la consulta sin paralelización, huelga decir por tanto que, para este caso planteado, no conviene usar paralelización. Si en lugar de 2 CPUs la máquina tuviese 16, los resultados habrían sido bien diferentes.

Lo que hace el gestor para ejecutar la sentencia en paralelo es crear unos procesos esclavos que se encargarán de parte de los datos, es decir, que la consulta se subdivide en pequeñas partes. Estos procesos esclavos procesan su parte de información, y cuando cada uno de ellos termina, el *Server Process* realiza el filtrado o la operación que necesite para ofrecer la respuesta completa.

Las vistas en las que se puede observar y controlar el comportamiento de las sentencias en paralelo son: *v\$px_session*, *v\$px_sesstat*, *v\$px_process*, *v\$px_process_sysstat*, *v\$px_buffer_advice*, *v\$px_instance_group*, *v\$px_slave*, *v\$pxq_sesstat*, *v\$pxq_sysstat* y *v\$pxq_tqstat*.

4.4.5. Consultas

Las consultas son básicamente la forma en la que se explota una base de datos. Por lo general, todos los clientes quieren tener un tiempo de respuesta pequeño, y aquí es muy importante entender que **no todo lo que cuesta menos es más rápido**.

Cuando el gestor se enfrenta a una consulta o cualquier otra sentencia SQL, ya sea DDL o DML, lo primero que hace es trazar su **plan de ejecución** para averiguar su **coste**. Una vez se tiene el coste, el dba podrá decidir si es necesario o no **afinar** la sentencia (hacerle un *tuning*) buscando mejorar su rendimiento. Para lograrlo, será imprescindible detenerse a examinar los **índices** que tienen o podrán tener las tablas que intervienen en la sentencia ejecutada, así como cobrarán un papel muy relevante los **joins** (uniones entre tablas) que aparezcan en la consulta y los posibles **hints** (predirectivas de ejecución) que puedan ser definidos.

Plan de ejecución

Como ya se ha visto, el plan de ejecución es simplemente la descripción de lo que el gestor tiene que hacer para poder ejecutar la sentencia, es decir, el cómo debe ir a buscar los resultados de la consulta o de los DMLs lanzados, esto es, las diferentes rutas de acceso que puede elegir el optimizador para desarrollar la sentencia.

El plan de ejecución de una sentencia concreta se puede obtener sin necesidad de que esta se ejecute completamente o simplemente tras su ejecución. Para ello, se puede usar el paquete *dbms_xplan* o el comando del SQL*Plus *set autotrace*.

Para poder invocar el *set autotrace* es necesario ejecutar el programa *\$ORACLE_HOME/rdbms/admin/utlxplan.sql*, y para ello es imprescindible tener los privilegios del role *PLUTRACE* que es creado con el programa *\$ORACLE_HOME/sqlplus/admin/plustrce.sql* por el usuario SYS.

La salida del *autotrace* ofrece tres bloques de información:

- Resultado de la ejecución.
- Plan de ejecución.
- Estadísticas de la ejecución.

La instrucción *set autotrace on*, o su abreviatura *set autot on*, muestra los tres bloques, en cambio, *set autotrace traceonly* (o *set autot trace*), se salta el primero, es decir, que la sentencia no se llega a ejecutar del todo. Con *set autotrace traceonly explain* (o *set autot trace exp*) solo se mostrará el segundo bloque y con *set autotrace traceonly statistics* solo se verán las estadísticas. La sintaxis completa es:

```
SET AUTOT[RACE] {OFF | ON | TRACE[ONLY]} [EXP[LAIN]] [STATISTICS]
```

El paquete *dbms_xplan* es el API que proporciona Oracle para extraer esa información desde cualquier entorno, no solo desde SQL*Plus. Se puede utilizar también de varias maneras, por ejemplo, con la opción *EXPLAIN PLAN FOR* se le puede indicar una consulta y, sin necesidad de que esta se ejecute, el paquete ofrecerá el 'posible' plan de ejecución con solo invocar la siguiente *select*:

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Si se sabe que la sentencia objeto de estudio se está ejecutando en el momento actual, o que todavía está en memoria, se puede incluso obtener su plan de ejecución con:

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_CURSOR(<sql_id>));
```

Donde el *sql_id* corresponde a una columna de la vista *v\$session* o *v\$sql*.

Si lo que se desea es conocer el plan de ejecución de una consulta que ya ha sido ejecutada pero que todavía se conserva en el AWR, se puede recurrir a él con la *select*:

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY_AWR(<sql_id>));
```

Además, *dbms_zplan* ofrece muchos formatos y también permite ver cualquier plan de ejecución diseñado para una consulta concreta (una misma consulta puede tener varios planes de ejecución).

A continuación, se muestra el plan de ejecución de una consulta hecha sobre una única tabla que no tiene índices:

```
SQL> set autot on exp stat
SQL> select *
  2. from tab_pp
  3. where id=10000;
ID INFO
-----
10000 aa10000zz
Plan de Ejecución
-----
Plan hash value: 3398924514
-----| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)| Time     |
-----| 0 | SELECT STATEMENT |          | 1    | 14   | 105  (3)| 00:00:02 |
* 1 |  TABLE ACCESS FULL| TAB_PP   | 1    | 14   | 105  (3)| 00:00:02 |
-----| Predicate Information (identified by operation id): |
-----| 1 - filter("ID"=10000)
Estadísticas
-----
0 recursive calls
0 db block gets
317 consistent gets
0 physical reads
0 redo size
487 bytes sent via SQL*Net to client
396 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
1 rows processed
```

La forma de leer el plan de ejecución es de dentro hacia afuera y de abajo hacia arriba. En este caso el gestor tiene que realizar dos operaciones:

1. la lectura completa (*full scan*) de la tabla *tab_pp*.
2. la *select* de los datos.

Además, indica cuántas filas estima el gestor que se devolverán en cada una de las operaciones, dato que se extrae de las estadísticas que tiene la tabla. Teniendo en cuenta la cantidad de filas que se devolverán, y puesto que por las estadísticas se conoce el tamaño medio de cada fila, puede concluir con la cantidad de bytes que va a recuperar por operación, y hacer una estimación del tiempo que tardará en realizarla, así como del **coste** de CPU que tendrá la operación.

Consistent gets (CG) indica (a groso modo) la cantidad de bloques que se tienen que leer para resolver la consulta. Casi toda la optimización de las bases de datos consiste en bajar la cantidad de CG procurando no generar un crecimiento desproporcionado de otros recursos. Cuando una sentencia tiene muchos CG, puede que tenga que hacer mucho I/O, y esto es directamente proporcional al consumo de CPU, por lo que lo primero que se debe hacer es buscar dentro de las estadísticas de consumo de SQL (en los informes AWR), las sentencias que tengan mayor consumo de CG y mejorárlas. Así que se puede decir que no importa tanto el número de lecturas físicas, como intentar conseguir el menor CG posible, el cual depende de la concurrencia, de la cantidad de sentencias DML, y por supuesto de lo óptimo que sean las consultas.

Coste

Muchas personas relacionan directamente el coste de la consulta con el tiempo consumido, y esto no es así, aunque en ocasiones sí que se cumpla que a menor coste, menor tiempo. El optimizador siempre escogerá el camino que le suponga menor coste. El coste se calcula a partir de fórmulas relacionadas con lo que le puede llevar al gestor realizar una operación y en ellas puede influir: la memoria (PGA), las lecturas de bloques sencillos (*single block reads*), las lecturas de bloques múltiples, el consumo de CPU y muchos factores más dependiendo de lo que se ejecute, y lo que se tenga que hacer para que se ejecute (crear objetos sobre la marcha, por ejemplo). Por tanto, para poder estimar lo que le cuesta al gestor realizar una operación, este tiene que conocer el entorno en el que se encuentra: tiene que saber cómo es su CPU.

La información que tiene sobre su entorno se puede ver en la vista *aux_stats\$*, pero no será una información válida o completa hasta que no se le haya proporcionado al gestor la oportunidad de registrar las características de su máquina. Esto se hace dejando activas las estadísticas de sistema durante un tiempo en el que se realicen suficientes operaciones. Como ya se sabe, para activar las estadísticas de sistema hay que ejecutar:

```
SQL> exec dbms_stats.gather_system_stats('START');
```

Para desactivarlas:

```
SQL> exec dbms_stats.gather_system_stats('STOP');
```

Una vez esté adecuadamente rellena, la vista *aux_stats\$* contendrá los valores para llenar las variables de la fórmula que Oracle ofrece para el cálculo de un *tablescan* (esta es una operación donde se lee toda la tabla):

$$Cost = \frac{NSRds \cdot sreadtim + NMRds \cdot mreadtim + \frac{NCPUCycles}{cpuspeed}}{sreadtim}$$

Donde:

NSRds: Número de lecturas de bloques solitarios

NMRds: Número de lecturas de bloques de forma múltiple

NCPUCycle: Número de ciclos de CPU

cpuspeed: Número de ciclos de CPU por segundo

sreadtim: Tiempo medio de lectura en bloques solitarios

mreadtim: Tiempo medio de lectura en bloques múltiples

El coste también está relacionado con otros dos conceptos básicos de la teoría de conjuntos: la selectividad y la cardinalidad.

Selectividad

La selectividad representa una fracción de filas de una fuente de filas.

Imagínese una tabla con los pacientes de un hospital, que posee una columna con el número de habitación que ocupan. Imaginense que el total de pacientes es 100 y que están distribuidos en 10 habitaciones. Con estas condiciones, ¿cuál sería la probabilidad de que un paciente ocupase la habitación 1? (10 habitaciones / 100 pacientes) * 100 = 10 %, es decir, una selectividad del 0,1.

Oracle, en ausencia de histogramas, opera de modo parecido, y usa la selectividad para averiguar el número de filas que van a cumplir una determinada condición impuesta.

Cardinalidad

La cardinalidad no es más que el número de filas de una fuente de filas (del inglés *row source*), ya sea la fuente de filas o datos una tabla, un índice o el resultado de un *join*.

Así que para la sentencia:

```
select * from pacientes where habitacion=1;
```

se obtiene el plan de ejecución:

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT		10	370	3 (0) 00:00:01	
* 1 TABLE ACCESS FULL PACIENTES		10	370	3 (0) 00:00:01	

donde se ve que el optimizador supone que obtendrá 10 filas (selectividad por cardinalidad), ya que interpreta que hay una distribución uniforme de los valores en la columna por la que se filtra el resultado.

El gestor deduce que tiene que hacer un escaneo completo de la tabla (*table access full* o *table scan*) y que el coste será 3. La siguiente salida muestra la información que arroja la traza 10053 (que es la traza específica para obtener los datos concretos y completos de cardinalidad, coste...):

```
SINGLE TABLE ACCESS PATH
Single Table Cardinality Estimation for PACIENTES[PACIENTES]
Table: PACIENTES Alias: PACIENTES
Card: Original: 100.000000 Rounded: 100 Computed: 100.00 Non Adjusted: 100.00
Access Path: TableScan
Cost: 3.00 Resp: 3.00 Degree: 0
Cost_io: 3.00 Cost_cpu: 54607
Resp_io: 3.00 Resp_cpu: 54607
Best:: AccessPath: TableScan
Cost: 3.00 Degree: 1 Resp: 3.00 Card: 100.00 Bytes: 0
```

Tuning de consultas

La forma clásica de hacer tuning de una consulta es aplicando un método basado en la observación:

1. Entender sin mucha profundidad lo que se está ejecutando.
2. Obtener el plan de ejecución.
3. Buscar posibles comportamientos no deseados.
4. Corregir (cambiar ruta de acceso, hints, estadísticas).
5. Volver al paso 2 hasta encontrar que la consulta tiene el resultado deseado.

Desde la versión 10g de Oracle, todo es más sencillo, sobre todo cuando se tienen muchas consultas que mejorar. Se cuenta con el paquete *dbms_sqltune*, que además, desde la versión 11g no solo sirve para afinar consultas en solitario, sino que se constituye como una completa herramienta (llamada *SQL Tuning Advisor*, consejero de afinamiento de consultas) para asegurar estabilidad en los planes de ejecución.

Para trabajar con el paquete *dbms_sqltune* lo que hay que hacer es crear una tarea de optimización, ejecutarla y observar el resultado (habiéndolo obtenido previamente el identificador de la consulta sobre la que se quiere trabajar).

Por ejemplo, dada la consulta *select count(8) from tab_pp where id=100000*:

1. Se obtiene el identificador (*sql_id*) de la consulta a mejorar:

```

SQL> select sql_id, sql_fulltext
  2  from V$SQL
  3  where sql_text like '%tab_pp%'
  4 /
SQL_ID      SQL_FULLTEXT
----- -----
49nlu3bfkF75p select count(8) from tab_pp where id=100000
br0z824fu6zyx select sql_id, sql_fulltext
                     from V$SQL
                   where sql_text like '%tab_pp%'

9a5pv9f76rgtn insert into tab_pp_sin_index values (1000000,'b')
1us8u7wrp3j80 select sql_id, sql_fulltext
                     from V$SQL
                   where sql_text like '%tab_pp%'
```

2. Se crea una tarea de optimización (con nombre *t1*):

```

SQL> declare
  2  t varchar2(100);
  3  begin
  4  t := dbms_sqltuné.create_tuning_task(sql_id => '49nlu3bfkF75p',
                                         scope=>'COMPREHENSIVE',
                                         time_limit=> 200,
                                         task_name=>'t1');
  5 end;
  6 /
Procedimiento PL/SQL terminado correctamente.
```

3. Se ejecuta la tarea de optimización *t1*:

```

SQL> exec dbms_sqltuné.execute_tuning_task('t1');
Procedimiento PL/SQL terminado correctamente.
```

4. Se observa la recomendación que ofrece la tarea de optimización *t1*:

```

SQL> select dbms_sqltuné.report_tuning_task('t1') sal from dual;
SAL
-----
GENERAL INFORMATION SECTION
-----
Tuning Task Name          : t1
Tuning Task Owner         : SYS
Scope                      : COMPREHENSIVE
Time Limit(seconds)       : 200
Completion Status         : COMPLETED
Started at                 : 03/21/2011 23:33:24
Completed at               : 03/21/2011 23:33:25
Number of Index Findings : 1
```

```

-----
Schema Name: TJ
SQL ID    : 49nlu3bfkf75p
SQL Text   : select count(8) from tab_pp where id=100000

FINDINGS SECTION (1 finding)

1- Index Finding (see explain plans section below)

El plan de ejecución de esta sentencia se puede mejorar mediante la creación
de uno o más índices.

Recommendation (estimated benefit: 99,04%)

- Puede ejecutar el Asesor de Acceso para mejorar el diseño del esquema
físico o crear el índice recomendado.
create index TJ.IDX$$_02840001 on TJ.TAB_PP("ID");

Rationale

La creación de índices recomendados mejora significativamente el plan de
ejecución de esta sentencia. Sin embargo, puede ser preferible ejecutar el
"Asesor de Acceso" mediante una carga de trabajo SQL representativa en
contraposición a una única sentencia. Esto permitirá obtener
recomendaciones de índice globales que tienen en cuenta la sobrecarga de
mantenimiento de índice y el consumo de espacio adicional.

EXPLAIN PLANS SECTION

1- Original

Plan hash value: 929901270

| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU) | Time      |
| 0 | SELECT STATEMENT   |           | 1    | 5     | 105 (3) | 00:00:02 |
| 1 |  SORT AGGREGATE    |           | 1    | 5     | 1          |           |
|* 2 |   TABLE ACCESS FULL | TAB_PP   | 1    | 5     | 105 (3) | 00:00:02 |

Predicate Information (identified by operation id):
2 - filter("ID"=100000)

2- Using New Indices

Plan hash value: 3689481318

| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU) | Time      |
| 0 | SELECT STATEMENT   |           | 1    | 5     | 1 (0) | 00:00:01 |
| 1 |  SORT AGGREGATE    |           | 1    | 5     | 1          |           |
|* 2 |   INDEX RANGE SCAN | IDX$$_02840001 | 1    | 5     | 1 (0) | 00:00:01 |

Predicate Information (identified by operation id):
2 - access("ID"=100000)

```

Se ve que, con la creación del índice recomendado, se pasa de tener un plan de ejecución con coste 105 a tenerlo con coste 1, pero eso sí, tal y como se indica, si

se crea el índice, hay que tener en cuenta también la sobrecarga de operaciones por el mantenimiento del nuevo índice y el consumo de espacio adicional que conlleva almacenar ese índice en disco.

Joins (composiciones)

Como regla general mientras más tablas existan dentro de la cláusula *from* de una consulta, más complicado será para el gestor hacerse una idea de las combinaciones y accesos que debe afrontar para evaluar la consulta.

Oracle solo une dos tablas cada vez, es más, ni siquiera une las tablas, sino los resultados (*los resultsets*). Por ejemplo, dada la siguiente consulta:

```
select a.col1, a.col2, ... b.col1, b.col2...
  from tab1 a, tab2 b, tab3 c, tab4 d, tab5 e, tab6 f, tab7 g
 where a.col1=b.col1 and a.col1=c.col1...
```

Oracle realizará la composición entre el resultado de la búsqueda que hace dentro de *tab1*, para unirla con el resultado de *tab2* y así sucesivamente.

El optimizador no realiza operaciones con tres o más tablas a la vez, por lo que la clave para obtener un mejor comportamiento de la consulta está en el orden en el cual se colocan las composiciones. Es lógico pensar que el gestor irá evaluando las composiciones poco a poco según el orden de los predicados, hasta dar con la solución más óptima, no obstante, sigue siendo muy grande para cualquier motor tener que evaluar $7!$ (combinatoria de $7=5040$ combinaciones posibles) ordenaciones distintas para elegir el *join* más óptimo. Por ello es conveniente no usar más de cinco tablas por *from*, máxime cuando ninguno de sus campos van a estar presentes en las columnas resultado.

Para resolver las sentencias, el gestor puede usar cuatro tipos de *join*:

1. **Cartesian join** (producto cartesiano): se utiliza cuando no queda más remedio, ya que su resultado es la multiplicación de las filas de las fuentes de datos (tablas) que entran en juego en la sentencia. Por ejemplo:

```
SQL> set autotrace exp
SQL> select * from tab1, tab2;
Plan de Ejecución
-----
Plan hash value: 4050065471

| Id | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time      |
| 0 | SELECT STATEMENT   |       | 3400 | 37400 |    72  (0)| 00:00:01 |
| 1 |  MERGE JOIN CARTESIAN |       | 3400 | 37400 |    72  (0)| 00:00:01 |
| 2 |   TABLE ACCESS FULL | TAB2 |    50 |   300 |     3  (0)| 00:00:01 |
| 3 |   BUFFER SORT        |       |    68 |   340 |    69  (0)| 00:00:01 |
| 4 |   TABLE ACCESS FULL | TAB1 |    68 |   340 |     1  (0)| 00:00:01 |
```

2. **Nested loop join** (composición de bucle anidado): es el escogido cuando se están manejando fuentes de datos pequeñas. El concepto *pequeño o grande* se mide respecto del tamaño de la PGA, que es donde deben alojarse las operaciones de ordenación y demás, que necesitará la resolución de la sentencia. En el siguiente ejemplo se crean dos tablas, una con mil registros (*tab1*) y la otra con diez (*tab2*), y se define una *foreign key* en la *tab1* hacia la *tab2*:

```

SQL> create table tab2
  2  (id number,
  3   info varchar2(10),
  4   constraint pk_tab2 primary key (id));
Table created.

SQL> create table tab1
  1  (id number,
  2   info varchar2(10),
  3   fk_tab2 number,
  4   constraint pk_tab1 primary key (id),
  5   constraint fk_tab1_tab2 foreign key (fk_tab2) references tab2(id));
Table created.

SQL> create index ind_fk_t1_t2 on tab1(fk_tab2);
SQL> insert into tab2
  2  select rownum, 'aaaa' || rownum
  3  from dual
  4  connect by rownum <=10;
10 rows created.
SQL> commit;
Commit complete.

SQL> insert into tab1
  2  select rownum, 'aaa'||rownum, round(dbms_random.value(1,10))
  3  from dual
  4  connect by rownum <= 1000;
1000 rows created.
SQL> commit;
Commit complete.

```

a continuación, se pregunta por el plan de ejecución que se obtendría de una sentencia que relaciona las dos tablas:

```

SQL> explain plan for
  2  select t1.id, t1.info
  3  from tab1 t1
  4  inner join tab2 t2
  5  on t1.fk_tab2=t2.id
  6  where t2.id=1;
Explained.

```

siendo el resultado:

```
SQL> select * from table(dbms_xplan.display);
PLAN_TABLE_OUTPUT
Plan hash value: 3601615389

| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU) | Time      |
| 0 | SELECT STATEMENT   |           | 57   | 969  |    3  (0) | 00:03:08 |
| 1 | NESTED LOOPS       |           | 57   | 969  |    3  (0) | 00:03:08 |
|* 2 |  INDEX UNIQUE SCAN | PK_TAB2   | 1    | 3    |    0  (0) | 00:00:01 |
| 3 | TABLE ACCESS BY INDEX ROWID | TAB1   | 57   | 798  |    3  (0) | 00:03:08 |
|* 4 |  INDEX RANGE SCAN  | IND_FK_T1_T2 | 57   | 1    |    1  (0) | 00:01:03 |

Predicate Information (identified by operation id):
2 - access("T2","ID"=1)
4 - access("T1","FK_TAB2"=1)
```

Donde se ve que lo primero que haría el gestor sería ir a buscar la información que se encuentra en el índice *IND_FK_T1_T2*, después extraería los datos de la tabla *tab1*, y realizaría un *nested loop join* con el índice *PK_TAB2*. ¿Por qué no iría luego a la tabla *tab2* para coger la información de sus filas? sencillamente porque no se necesita la información de la fila completa. El optimizador es lo suficientemente inteligente para que en estos casos, cuando sabe que los datos se encuentran en el índice, no realice más que el acceso al índice únicamente.

- 3. Hash join (composición hash):** cuando las fuentes de datos son muy grandes, el optimizador puede optar por crear una tabla *hash* en memoria de una fuente de datos menor para luego, contrastarla con la fuente de datos original. Una tabla *hash* no es más que una estructura de datos que mapea unas claves (las columnas del *join*) con unos valores. Al tratarse de una estructura en memoria, Oracle evalúa la cantidad de memoria de que dispone (mediante el parámetro *pga_aggregate_target*), y en función de eso, determinará si le es posible o no crear tablas *hash*. Si al final decide usar *hash join*, este podrá ser de tres tipos distintos: *optimal*, *one-pass* o *multi-pass*. Como ya se vio, el *one-pass* y el *multi-pass* suponen redimensionamiento de memoria, obligando a hacer lo que se denomina *hash spilling*, que no es más que el uso de segmentos temporales (dentro del *TEMPORARY TABLESPACE*) para poder realizar las operaciones. Esto por supuesto no es bueno para el rendimiento de las consultas.
- 4. Sort merge join (composición combinación de ordenaciones):** implica realizar una ordenación en cada una de las fuentes de filas presentes en la consulta. Puede llegar a ser muy costoso o todo lo contrario, si se cumple que las fuentes de filas ya están ordenadas o no hay que hacer más operaciones de ordenación adicionales. El *sort merge join* primero ordena las fuentes de filas y luego las combina, esto se ve claramente en el siguiente ejemplo:

Id Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0 SELECT STATEMENT		100	2400	8 (13)	00:08:19
1 MERGE JOIN		100	2400	8 (13)	00:08:19
* 2 TABLE ACCESS BY INDEX ROWID TAB2	PK_TAB2	10	10	2 (0)	00:02:05
3 INDEX FULL SCAN		1000	14000	1 (0)	00:01:03
* 4 SORT JOIN		1000	14000	6 (17)	00:06:15
5 TABLE ACCESS FULL	TAB1	1000	14000	5 (0)	00:05:12

Se aprecia cómo primero hace un *sort* de *tab1*, pero no de *tab2* porque accede a ella por el índice de la clave primaria (recuérdese que los índices están SIEMPRE ordenados). Después realiza la combinación de ambas fuentes de datos (*MERGE JOIN*).

Oracle recomienda usar el *nested loop join* con fuentes de datos pequeñas, el *hash join* con fuentes de datos grandes, y cuando los predicados de las consultas no son de igualdad y los datos están ya ordenados, lo ideal es usar el *sort merge join*. No obstante, la elección de un tipo u otro de *join*, dependerá también de la situación concreta que se esté tratando, así como de los valores que tengan los parámetros de base de datos. El gestor intentará elegir la operación que le lleve menos coste, pero es posible que la información que tenga sea incorrecta, o que la consulta esté tan mal creada que no pueda conseguir un buen plan de ejecución.

Hints

Un *hint* es una indicación que se le da al optimizador para que realice cierta operación durante la ejecución de una consulta.

El ejemplo dado más comúnmente para ilustrar su utilidad es el caso de tener muchas tablas en la cláusula *from* de la sentencia. Ya se explicó anteriormente que Oracle no realiza operaciones con más de dos tablas, así que ante un escenario de n tablas, tendría que hacer la friolera de $n!$ combinaciones entre tablas para averiguar cuál de todas esas combinaciones es la más óptima. Por medio de los *hints*, sería posible indicarle al gestor el camino que se sabe más óptimo para resolver esas combinaciones.

También se puede presentar el caso de que el optimizador, por contar con malas estadísticas, o con información incompleta, o por tener que hacer demasiados cálculos, o simplemente porque no acierta, se empeñe en realizar un *nested loop join*, cuando con un *hash join* obtendría mejores resultados. Con el uso del *hint* apropiado se podría obligar al gestor a escoger un tipo de *join* u otro.

Los *hints* pueden ser agrupados en diferentes categorías:

- Métodos y objetivos de la optimización.
- Habilitar características del optimizador.

- Rutas de acceso.
- Orden de *joins*.
- Operaciones de *joins*.
- Actualización de aplicaciones online.
- Ejecuciones en paralelo.
- Transformación de consultas.
- Otros.

Los más usados son los de rutas de acceso, orden y operaciones de *joins*, y los de ejecuciones en paralelo. Los de rutas de acceso son los que sirven para especificar qué objeto se debe usar para la implementación del plan de ejecución. Por ejemplo, imagínese que sobre una tabla (*t1*) existiesen tres índices y para la resolución de una consulta concreta el optimizador no estuviese eligiendo el índice apropiado. Pues bien, para especificarle qué índice debe elegir (en este caso el llamado *indice2*), bastaría con ejecutar la sentencia de la siguiente manera:

```
select /*+ INDEX(t1 indice2) */ t1.col1
from t1
where ...
```

Otro ejemplo claro es cuando se quiere paralelizar una sentencia y se sabe exactamente la cantidad de paralelismo que debe utilizarse. Imagínese que se cuenta con una máquina que tiene 16 CPU y se desea ejecutar con paralelismo de 8 una consulta sobre la tabla *t1*, que ha sido creada con el valor por defecto en su especificación del *DEGREE*. La sentencia que debería utilizarse tendría la siguiente forma:

```
select /*+ parallel(8) */
t1.col1 from t1
where ...
```

Cuando se tiene mucho conocimiento del sistema en el que se trabaja (la versión, nivel de parches, estadísticas, estructura de los datos...), el uso de *hints* ofrece la gran ventaja de poder controlar los planes de ejecución de las sentencias, pero implicitamente arrastra también varios problemas, ya que el gestor puede informar de lo que hacen los *hints*, pero estos pueden cambiar el comportamiento según la versión que se use, incluso después de la simple aplicación de un parche o tras una migración (si el nombre de los objetos cambia, por ejemplo), por lo que usar *hints* puede disminuir considerablemente la capacidad de adaptación de la aplicación a los cambios de su entorno.

- **Actividad 4.1:** Con todo lo expuesto en esta sección, repasar el método propuesto al comienzo de la misma y enumerar las técnicas aprendidas para afrontarlo.

4.5. Herramientas y sentencias para la gestión de índices

El acceso al dato es fundamental en el rendimiento de cualquier gestor de base de datos. Al manejar tanta información es necesario ayudar al gestor con objeto de evitar que tenga que recorrerse toda una tabla para ir en busca de un dato.

Volviendo al ejemplo de la tabla *tab_pp* que no tenía índices:

```
SQL> desc tj.tab_pp
Nombre          Null  Tipo
-----          ----  -----
ID              NUMBER
INFO            VARCHAR2(10)
```

La tabla contiene 100000 registros. Y de nuevo el plan de ejecución de la consulta hecha sobre ella para obtener el registro que cumple que *id=10000* es:

```
SQL> set autot on exp stat
SQL> select *
  2  From tab_pp
  3 where id=10000;
   ID INFO
   10000 aa10000zz
Plan de Ejecución
-----
Plan hash value: 3398924514
| Id | Operation          | Name      | Rows  | Bytes | Cost (CPU) | Time     |
|  0 | SELECT STATEMENT   |           | 1    | 14   | 105 (3) | 00:00:02 |
|* 1 | TABLE ACCESS FULL | TAB_PP   | 1    | 14   | 105 (3) | 00:00:02 |
Predicate Information (identified by operation id):
   1 - filter("ID"=10000)
Estadísticas
-----
   0 recursive calls
   0 db block gets
 317 consistent gets
   0 physical reads
   0 redo size
 487 bytes sent via SQL*Net to client
 396 bytes received via SQL*Net from client
   2 SQL*Net roundtrips to/from client
```

```
0 sorts (memory)
0 sorts (disk)
1 rows processed
```

Como ya se dijo, el *table access full* implica que el gestor ha leído todos los bloques de la tabla, se los ha llevado hasta memoria y después los ha filtrado para solo mostrar la fila que se necesita. Concretamente, ha consumido 317 *consistent gets*. Esto es tremadamente ineficiente sobre todo si se tienen muchas tablas, muchos registros y mucha información.

Desde hace mucho tiempo, en informática, existen muy buenos métodos de búsqueda, entre ellos el algoritmo de *B-Tree* o el *BITMAP*. Los sistemas gestores de bases de datos se ahorran enormes cantidades de búsquedas aplicando estos algoritmos sobre las tablas para generar segmentos físicos de mucho menor tamaño. Se llaman índices y trabajan de la misma forma que las páginas amarillas o cualquier índice de un libro, o un simple enrutador, esto es, indicando la dirección exacta donde se puede encontrar un dato concreto.

Siguiendo con el caso de la tabla *tab_pp*, ¿cuánto ahorra en la consulta anterior la creación de un índice?

```
SQL> create index idx_tab_pp on tab_pp(id);
Índice creado.
```

El plan de ejecución con el índice creado es:

```
SQL> set autot on exp stat
SQL> select *
  2  from tab_pp
  3  where id=10000;
ID INFO
-----
10000 aa10000zz
Plan de Ejecución
-----
Plan hash value: 2237551901
-----
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
| 0  | SELECT STATEMENT   |           | 1    | 14   |    2  (0) | 00:00:01 |
| 1  |  TABLE ACCESS BY INDEX ROWID| TAB_PP   | 1    | 14   |    2  (0) | 00:00:01 |
|* 2  |  INDEX RANGE SCAN   | IDX_TAB_PP| 1    | 1    |    1  (0) | 00:00:01 |
-----
Predicate Information (identified by operation id):
-----
 2 - access("ID"=10000)
Estadísticas
-----
 1 recursive calls
 0 db block gets
 4 consistent gets
 1 physical reads
 0 redo size
 487 bytes sent via SQL*Net to client
```

```

396 bytes received via SQL*Net from client
 2 SQL*Net roundtrips to/from client
 0 sorts (memory)
 0 sorts (disk)
 1 rows processed

```

Con el índice creado, solo ha consumido 4 *consistent gets*, frente a los 317 anteriores. Esto es básicamente porque el índice indica directamente en qué bloque está el dato buscado.

Pero también hay que tener en cuenta que cada vez que se crea un índice se están haciendo más pesadas las sentencias DML sobre la tabla. Una forma sencilla de probar esto es abriendo una sesión, insertando una fila sobre una tabla con índice y sobre otra sin índice copia de la *tab_pp* (*tab_pp_sin_index*), y consultando las estadísticas de operaciones de redo de la sesión (a través de las vistas *v\$mystat* y *v\$statname*).

Respecto de la tabla sin índice:

```

SQL> select st.name, ms.value
 2 from v$mystat ms, v$statname st
 3 where st.name like '%redo%'
 4 and ms.statistic# = st.statistic#
 5 and ms.value <> 0;
ninguna fila seleccionada
SQL> insert into tab_pp_sin_index values (1000000,'b');
1 fila creada.
SQL> commit;
Confirmación terminada.
SQL> select st.name, ms.value
 2 from v$mystat ms, v$statname st
 3 where st.name like '%redo%'
 4 and ms.statistic# = st.statistic#
 5 and ms.value <> 0;
NAME                                VALUE
-----                                 -
redo synch writes                      1
redo entries                           1
redo size                             468

```

Y la información de redo generada al insertar una fila en la tabla *tab_pp* (que ya tenía el índice creado):

```

SQL> select st.name, ms.value
 2 from v$mystat ms, v$statname st
 3 where st.name like '%redo%'
 4 and ms.statistic# = st.statistic#
 5 and ms.value <> 0;
ninguna fila seleccionada
SQL> insert into tab_pp values (1000000,'b');
1 fila creada.
SQL> commit;
Confirmación terminada.

```

```

SQL> r
 1 select st.name, ms.value
 2 from v$sysstat ms, v$statname st
 3 where st.name like '%redo%'
 4 and ms.statistic# = st.statistic#
 5 and ms.value > 0
NAME                      VALUE
-----                    -----
redo synch writes          1
redo entries                1
redo size                  680

```

Al tener el índice creado, se han generado 212 bytes más de carga de redo. Si esto se extrae al caso real de una tabla que tenga 20 columnas y 3 índices, y que trabaje en una base de datos OLTP, sería bueno plantearse este sobreesfuerzo para determinar si merece realmente la pena o no crear un índice. Como ya se ha comentado, la información de redo es la información más crítica de la base de datos. El comportamiento del LGWR hace que la base de datos vaya mejor o peor. Cuantos más índices haya, más información de redo se generará y más carga de trabajo recaerá sobre el LGWR.

Una vez más merece la pena reflexionar sobre el hecho de que en una base de datos todo está conectado. El menor detalle que se cambie con el propósito de mejorar alguna cosa, puede hacer empeorar otra.

Una herramienta muy potente que ayuda a decidir sobre la conveniencia, o no, de la creación de un determinado índice para mejorar algún aspecto concreto de una sentencia, es el paquete *dbms_sqltune* cuya utilización ya se vio en el tuning de consultas. Véase aquí un nuevo ejemplo al respecto:

```

SQL> create table tab_aa
 2  (id number,
 3   info varchar2(30));
Tabla creada.
SQL> insert into tab_aa
 2  select rownum, dbms_random.string('a',30)
 3  from dual
 4  connect by rownum <= 10000
10000 filas creadas.
SQL> commit;
Confirmación terminada.

SQL> exec dbms_stats.gather_table_stats('TJ','TAB_AA')
Procedimiento PL/SQL terminado correctamente.

SQL> declare
 2   consulta VARCHAR2(100) := select id from tab_aa where id=2345;
 3   t VARCHAR2(30);
 4 begin
 5   t := dbms_sqltune.create_tuning_task(sql_text => consulta,
 6                                         scope=>'COMPREHENSIVE', time_limit=>600, task_name=>'tarea1');
 7 end;
 8 /
Procedimiento PL/SQL terminado correctamente.

SQL> exec dbms_sqltune.execute_tuning_task('tarea1')

```

Procedimiento PL/SQL terminado correctamente.

```
SQL> select dbms_sqltune.report_tuning_task('tarea2') sal from dual;
SAL
```

GENERAL INFORMATION SECTION

Tuning Task Name	:	tarea1
Tuning Task Owner	:	TJ
Scope	:	COMPREHENSIVE
Time Limit(seconds)	:	600
Completion Status	:	COMPLETED
Started at	:	05/04/2011 18:54:32
Completed at	:	05/04/2011 18:54:32
Number of Index Findings	:	1

Schema Name:	TJ
SQL ID	: dr5pk0h8a7526
SQL Text	: select id from tab_aa where id=2345

FINDINGS SECTION (1 Finding)

1- Index Finding (see explain plans section below)

El plan de ejecución de esta sentencia se puede mejorar mediante la creación de uno o más Indices.

Recommendation (estimated benefit: 94,51%)

- Puede ejecutar el Asesor de Acceso para mejorar el diseño del esquema físico o crear el índice recomendado.

```
create index TJ.IDX$$_03590001 on TJ.TAB_AA("ID");
```

Rationale

La creación de Indices recomendados mejora significativamente el plan de ejecución de esta sentencia. Sin embargo, puede ser preferible ejecutar el "Asesor de Acceso" mediante una carga de trabajo SQL representativa en contraposición a una única sentencia. Esto permitirá obtener recomendaciones de índice globales que tienen en cuenta la sobrecarga de mantenimiento de índice y el consumo de espacio adicional.

EXPLAIN PLANS SECTION

1- Original

Plan hash value: 1855471094

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	4	18 (0)	00:00:01
* 1	TABLE ACCESS FULL	TAB_AA	1	4	18 (0)	00:00:01

Predicate Information (identified by operation id):

1 - filter("ID"=2345)

2- Using New Indices

Plan hash value: 1595896910

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	4	1 (0)	00:00:01
* 1	INDEX RANGE SCAN	IDX\$\$_03590001	1	4	1 (0)	00:00:01

```
Predicate Information (identified by operation id):
  1 - access("ID"=2345)
```

La efectividad de un índice depende de tres factores:

1. La cardinalidad de la/s columna/s que lo componen, esto es, la cantidad de valores distintos en relación con la cantidad de valores que tiene la tabla. Si el índice es una clave primaria, su cardinalidad será máxima.
2. El modo de acceso al índice. Los índices siempre se crean sobre columnas que son usadas en las sentencias, si no, no tiene sentido crearlos y pueden incluso ir en detrimento del rendimiento.
3. El *clustering factor* (CF). Los índices están ordenados por la/s columna/s. El clustering factor es un número que representa la dispersión de los datos en la tabla.

Ya se sabe que los datos están en bloques, ahora bien, un bloque de un índice puede contener uno o más valores de la/s columna/s, es decir, en un solo bloque puede haber muchas llaves (se llama llave a la/s columna/s que define/n el índice). Esta llave está asociada a un ROWID, que es un valor que identifica únicamente al registro y además contiene la información para localizar el bloque físicamente dentro del disco, ya que proporciona el número del fichero, el número de la extensión y el identificador del bloque.

Ahora bien, el gestor tiene que determinar si debe usar el índice directamente, o si tiene primero que ir al índice y después a la tabla (con lo que generará más *consistent gets*), o directamente a la tabla... Y ¿esto por qué? La contestación es simple: dependerá del CF.

Cuando el gestor crea un índice, lo hace ordenándolo físicamente en disco por el campo o los campos que lo componen. Imaginese, por ejemplo, que en la tabla *tab_pp* el valor 1 y 2 de la columna *id* estuviesen en el bloque 1, entonces, por ahora, se podría decir que el CF vale solo 1. Si el valor 3 de *id* estuviese en un bloque diferente al bloque 1, entonces el CF se incrementaría en una unidad, y así se haría cada vez que entrara en juego un bloque diferente. Es decir, cuanto más desordenada aparezca la tabla físicamente en el disco, más alto será el valor que tenga el CF, pudiendo llegar su valor incluso a igualar el número de registros de la tabla. Cuando el valor del CF es muy bajo, se puede decir que se acerca a la cantidad de bloques con datos que tiene la tabla. Lo ideal es que ocurra precisamente eso, que el CF sea lo más pequeño posible, siendo entonces comparable a la cantidad de bloques de la tabla.

Es muy importante también darse cuenta de que con las inserciones, borrados y actualizaciones de los registros en la tabla, los índices pueden llegar a fragmentarse, quedando espacios vacíos en disco que hacen crecer el tamaño del índice desproporcionadamente. Esto puede verse consultando las vistas *dba_extents* y *dba_segments*.

Un mantenimiento muy recomendable ante esta situación consiste en la recreación del índice, consiguiendo con ello sanearlo y ahorrar el correspondiente número de bloques en disco, con lo que el *INDEX FAST FULL SCAN* será menos costoso porque tendrá que leer un menor número de bloques. Sirva el siguiente ejemplo como ilustración de este hecho.

```

SQL> delete tab_aa where id<= 10000;
5000 filas suprimidas.
SQL> commit;
Confirmación terminada.

SQL> insert into tab_aa
  2  select dbms_random.value(10000,20000), dbms_random.string('a',30)
  3  from dual
  4* connect by rownum<=10000
10000 filas creadas.
SQL> commit;
Confirmación terminada.

SQL> select segment_name, bytes/1024/1024
  2  from dba_segments
  3  where owner='TJ'
  4* and segment_name like '%AA%'
SEGMENT_NAME      BYTES/1024/1024
----- -----
TAB_AA                  2
IDX_TAB_AA              2

SQL> ALTER INDEX IDX_TAB_AA REBUILD PARALLEL 2 NOLOGGING;
Índice modificado.

SQL> ALTER INDEX IDX_TAB_AA NOPARALLEL;
Índice modificado.

SQL> select segment_name, bytes/1024/1024
  2  from dba_segments
  3  where owner='TJ'
  4* and segment_name like '%AA%'
SEGMENT_NAME      BYTES/1024/1024
----- -----
TAB_AA                  2
IDX_TAB_AA              ,0625

```

4.6. Herramientas para la creación de alertas de rendimiento

Pueden ser también consideradas herramientas de monitorización, ya que lo que hacen es arrojar información ante la ocurrencia de un evento (igual que las trazas).

En Oracle la principal herramienta es el paquete *dbms_server_alert*. Forma parte de la API de Oracle. Permite configurar valores máximos y mínimos para determinados eventos, de modo que hagan saltar alertas que quedarán registradas en las vistas del sistema (*dba_outstanding_alerts*, *dba_alert_history*) y podrán consultarse, bien mediante sentencias SQL, o a través de la interfaz gráfica de *Enterprise Manager*, desde su página de inicio.

The screenshot shows the Oracle Database Control interface under the 'Alertas' section. It includes a header with categories (Toda, Crítica, Advertencia), a table for 'Alertas Relacionadas' with columns like Gravedad, Nombre del Destino, Categoría, Nombre, Mensaje, Muestra Dispuesta, Última Vizual, and Última Hora, and a summary section for 'Actividad del Trabajo' with counts for Ejecuciones Planificadas, Ejecuciones en Ejecución, Ejecuciones Suspensas, and Ejecuciones Problemáticas.

Figura 4.20: Visualización de alertas.

Por ejemplo, si se deseara configurar una alerta de ocupación del tablespace `users` que hiciera saltar un warning cuando su ocupación fuese del 75 % y una alerta crítica cuando esté en el 90 %, bastaría con ejecutar el procedimiento `set_threshold`.

```
SQL> BEGIN
 2   dbms_server_alert.set_threshold(dbms_server_alert.tablespace_pct_full,
 3                                     dbms_server_alert.operator_ge, 75,
 4                                     dbms_server_alert.operator_ge, 90,
 5                                     1, 1, 'test2',
 6                                     dbms_server_alert.object_type_tablespace, 'USERS');
 7 END;
 8 /
```

Procedimiento PL/SQL terminado correctamente.

También se podría hacer desde la consola gráfica, accediendo al menú de *Enlaces relacionados*, entrando en *Gestionar métricas*, y una vez allí pulsando en la opción de *Editar umbrales*.

The screenshot shows the 'Metrics Management' section of the Oracle Database Control interface. It includes a table for 'Enlaces Relacionados' with metrics like 'Uso de Archivo Usado (%)', 'Error del Log de Alertas de Bloqueo del Proceso de Archivado', 'Estado de Error del Log de Alertas de Bloques del Proceso de Archivado', 'Usuario de Auditoria', 'Tiempo Medio de Lectura de Archivo (centisegundos)', and 'Tiempo Medio de Escritura de Archivo (centisegundos)'. There is also a 'Especificación de Varios Umbrales' button.

Figura 4.21: Gestionar métricas.

¹ Instancia de Base de Datos: test2 - Gestionar Métricas - Editar Umbrales

Editar Umbrales

Puede definir un uso de umbrales crítico y de advertencia para cada una de las siguientes métricas. Cuando se alcance un umbral, se generará una alerta y, si se especifica, se ejecutará la acción de respuesta. La acción de respuesta puede ser cualquier comando o archivo de comandos con una ruta de acceso totalmente cualificado con la que se puede acceder al agente de gestión.

CONSEJO: Algunas métricas no pertenecen en modo por defecto de umbrales para todos los objetos controlados. Haga clic en "Especificar Varios Umbrales" para definir los umbrales para los objetos específicos.

Enlace Relacionado: [Resumen del Diagnóstico](#)

Selección de Métrica	Operador de Comparación	Unidad de Advertencia	Basural Crítico	Acción de Respuesta
<input checked="" type="radio"/> Área de Archivado Usada (%)	>	80		
<input type="radio"/> Error del Log de Alertas de Bloqueo del Proceso de Archivado	Crítico		ORA-00600	
<input type="radio"/> Estado de Error del Log de Alertas de Bloques del Proceso de Archivado	>	0		
<input type="radio"/> Usuario de Auditoria	=	SYS		
<input type="radio"/> Tiempo Medio de Lectura de Archivo (centisegundos)	>			
<input type="radio"/> Tiempo Medio de Escritura de Archivo (centisegundos)	>			

Figura 4.22: Editar umbrales.

4.7. Prácticas

Práctica 4.1: Obtención y búsqueda de trazas

Dado el nombre de un usuario concreto, rastrear la ejecución de una de sus sesiones con el API de Oracle *dbms_monitor*. Realizar también el proceso completo de monitorización desde *Enterprise Manager*.

Práctica 4.2: Directorio de trazas

Obtener el directorio donde se almacenan las trazas en un versión 11g. ¿En qué directorio Oracle almacena las trazas de rastreo de la sesión en una versión 10g?

Práctica 4.3: Creación de consultas de monitorización

Crear una consulta que detecte las sesiones activas (que no sean procesos en background) y que muestre el texto del SQL que se está ejecutando. Crear también una consulta que averigüe la cantidad de consumo de PGA de la sesión de un usuario concreto. Usando la vista *v\$session*, crear una consulta que detecte los bloqueos entre sesiones.

Práctica 4.4: Consumo de CPU

Buscar, con ayuda del *Enterprise Manager*, la consulta que consume más CPU.

Práctica 4.5: Planes de ejecución

Con la ayuda de los informes AWR, obtener todos los planes de ejecución que se han ejecutado con *dbms_xplan*.

Práctica 4.6: Información de REDO

Obtener la cantidad de información de Redo que se ha generado a lo largo de un día en una base de datos que tiene el archive log activado (pista: *v\$archived_log*). ¿Qué vista ofrece el tamaño óptimo de los ficheros de redo en función de la actividad de la base de datos?

Práctica 4.7: Eventos de espera

Dado los siguientes eventos de espera:

Event	Wait(s)	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
log file sync	560,073	47,007	84	43.2	Commit
CPU time		14,526		13.4	
db file sequential read	1,656,031	11,080	7	10.2	User I/O
enq: TX - allocate ITL entry	2,156	4,886	2,313	4.8	Configuration
db file scattered read	838,695	3,747	4	3.4	User I/O

Figura 4.23: Eventos de espera.

¿Qué se puede deducir?

Práctica 4.8: PGA Aggr Target Histogram

¿Qué sugiere el siguiente *PGA Aggr Target Histogram*?

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	1,465,779	1,465,779	0	0
64K	128K	346	346	0	0
128K	256K	254	254	0	0
256K	512K	1,500	1,500	0	0
512K	1024K	57,198	57,198	0	0
1M	2M	5,435	5,409	26	0
2M	4M	177	70	106	1
4M	8M	63	12	51	0
8M	16M	5	0	5	0
16M	32M	4	0	4	0
32M	64M	3	0	1	2
64M	128M	8	0	4	4
128M	256M	1	0	1	0
256M	512M	1	0	0	1

Figura 4.24: One-pass, multi-pass.

Práctica 4.9: Optimización desde Enterprise Manager

A la vista de la siguiente gráfica en la ventana de rendimiento:

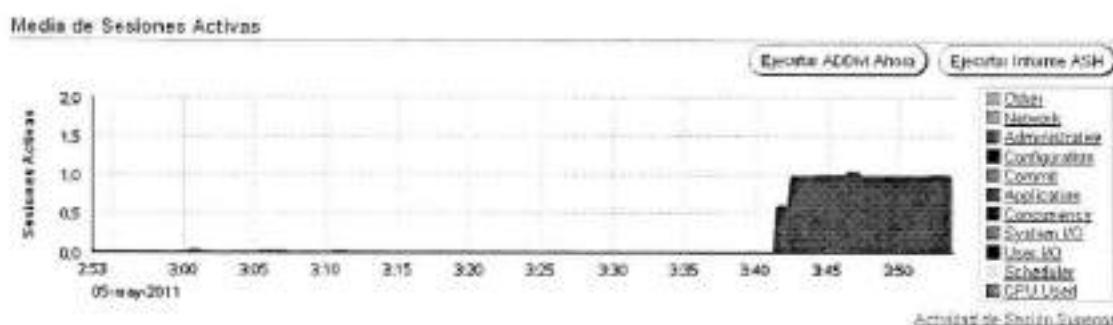


Figura 4.25: Sesiones activas.

siendo la gráfica de las esperas de aplicación de la forma:

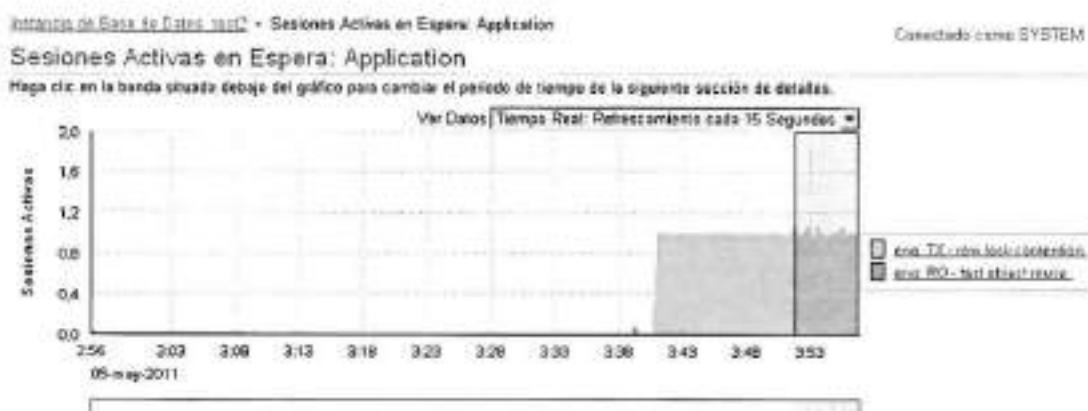


Figura 4.26: Esperas de aplicación.

¿Qué se puede deducir?

Práctica 4.10: Ejecución repetida

A continuación se presentan los planes de ejecución y sus correspondientes estadísticas de la ejecución de una misma sentencia dos veces seguidas. Los planes de ejecución se han obtenido activando el *autoTrace* con la opción de *traceonly* desde una sesión SQL*Plus.

```
SQL> select * from tab1_mr;
10000 filas seleccionadas.
Transcurrido: 00:00:00.56
Plan de Ejecución

Plan hash value: 554326416

| Id | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time   |
| 0 | SELECT STATEMENT  |          | 10000 | 146K  |    12  (0) | 00:00:01 |
| 1 |  TABLE ACCESS FULL| TAB1_MR | 10000 | 146K  |    12  (0) | 00:00:01 |

Estadísticas
-----
      0 recursive calls           819692 bytes sent via SQL*Net to client
      0 db block gets             55385 bytes received via SQL*Net from client
      5014 consistent gets        5001 SQL*Net roundtrips to/from client
      0 physical reads            0 sorts (memory)
      0 redo size                 0 sorts (disk)
  10000 rows processed

SQL> select * from tab1_mr
10000 filas seleccionadas.
Transcurrido: 00:00:00.15
Plan de Ejecución

Plan hash value: 554326416

| Id | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time   |
| 0 | SELECT STATEMENT  |          | 10000 | 146K  |    12  (0) | 00:00:01 |
| 1 |  TABLE ACCESS FULL| TAB1_MR | 10000 | 146K  |    12  (0) | 00:00:01 |

Estadísticas
-----
      0 recursive calls           189944 bytes sent via SQL*Net to client
      0 db block gets             407 bytes received via SQL*Net from client
      40 consistent gets          3 SQL*Net roundtrips to/from client
      0 physical reads            0 sorts (memory)
      0 redo size                 0 sorts (disk)
  10000 rows processed
```

¿Qué diferencia existe entre las dos ejecuciones? ¿Por qué la primera tarda casi cuatro veces más que la segunda? Hacer una prueba similar.

Práctica 4.11: Evento de espera *log file sync*

Dada las siguientes estructuras de programas PL/SQL:

```
FOR i IN 1..1000 LOOP
    INSERT INTO ...
    COMMIT;
END LOOP;
```

y

```
FOR i IN 1..1000 LOOP
    INSERT INTO ...
END LOOP;
COMMIT;
```

Razonar y comprobar cuál de ellos causará menos eventos de espera *log file sync* registrados en los informes AWR.

Práctica 4.12: Creación de índices

Crea índices para mejorar el tiempo de respuesta de la base de datos jardinería. Comprueba con el plan de ejecución que se usan estos índices para la ejecución de la consulta:

1. Un índice para buscar clientes por la región en la que se encuentran.
 2. Un índice para buscar los productos por proveedor.
 3. Un índice para localizar oficinas por código postal.
 4. Un índice para buscar empleados por email.
 5. Un índice para localizar productos por precio.
-

4.8. Amplía tus conocimientos

1. En MySQL el tuning es bastante más sencillo aunque más limitado. Busca en la página web de documentación cómo realizar optimización en este SGBD. Consulta cómo analizar tablas (ANALYZE TABLE), optimizarlas (OPTIMIZE TABLE) y mostrar el plan de ejecución (EXPLAIN).
2. Consulta para qué sirven las variables table_cache y key_buffer en MySQL y qué tienen que ver con el rendimiento del SGBD.
3. El parámetro sort_buffer_size reserva una cantidad de memoria para las ordenaciones. Consulta más sobre este parámetro y haz pruebas de rendimiento con consultas y ordenaciones.
4. Puedes comprobar el hit ratio en MySQL con el comando SHOW STATUS LIKE 'Key %' y usando la fórmula $100 - ((\text{Key_reads}/\text{Key_read_request}) * 100)$.
5. Consulta la página de documentación de Oracle <http://tahiti.oracle.com>.
6. Aprende más sobre la lectura de las trazas leyendo los siguientes documentos de Metalink: *Doc ID 41634.1: TKProf Basic Overview* y *Doc ID 39817.1: Interpreting Raw SQL_TRACE and DBMS_SUPPORT.START_TRACE output*.
7. Amplía conocimientos sobre el Java en las bases de datos, consultando el manual: *Oracle Database Java Developer's guide*.
8. Amplía conocimientos sobre el segmento de memoria de la SGA Stream Pool, utilizado en Dataguard, consultando *Oracle Streams Replication Administration's Guide*.
9. Aprende lo que son las lecturas y escrituras consistentes consultando el capítulo 7 del libro de Tom Kyte *Expert Oracle Database Architecture 9i & 10g*, y también en la url del blog de Jonathan Lewis <http://jonathanlewis.wordpress.com/2009/06/12/consistent-gets-2/>.
10. Documéntate acerca de las *Very Large Database* en <http://www.vldb.org/>.
11. Profundiza en los algoritmos de *B-Tree* y *BITMAP* para la creación de índices.
12. Aprende técnicas de rendimiento sobre DB2 usando las herramientas: Event monitor, System monitor, Data Studio Administration Console, DB2 Performance Expert, Optim Query Tuner.
13. Consulta el Centro de Información de DB2 para resolver dudas y ampliar conocimientos:
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.idm.tools.doc/doc/c0055013.html>.

14. Busca información sobre la activación de los monitores en DB2, pues son imprescindibles para disponer de toda la información relativa a la optimización (extraída mediante snapshots).
15. Ejecuta la generación de snapshots (fotos sobre el estado de las bases de datos) en DB2 (*db2 get snapshot for db on nombre_base_de_datos*) y estudia el contenido de su salida.
16. Practica la monitorización en DB2 con las vistas administrativas de SYSIBMADM a nivel de:
 - Instancia: *SNAPDBM, SNAPFCM, SNAPFCM_PART, SNAPSITCHES ...*
 - Base de datos: *SNAPDB, SNAPDB_MEMORY_POOL, SNAPHADR, SNAPUTIL, SNAPUTIL_PROGRESS, SNAPDETAILSTORAGE_PATHS*
 - Aplicación: *SNAPAPPL, SNAPAPPL_INFO, SNAPLOCKWAIT, SNAPSTMT, SNAPAGENT, SNAPSUBSECTION, SNAPAGENT_MEMORY_POOL*
 - Bloqueos: *SNAPLOCK*
 - SQL dinámico: *SNAPDYN_SQL*
 - Buffer Pools: *SNAPBP, SNAPBP_SWITCH*
 - Tablespaces: *SNAPTBSP, SNAPTBSP_PART, SNAPTBSP QUIESCER, SNAPCONTAINER, SNAPTBSP RANGE*
 - Tabla: *SNAPTAB, SNAPTAB_REORG*
 - Parámetros de configuración: *DB_PARTITION, DBCFG, DBMCFG, REG_VARIABLES*
 - Variables de entorno: *ENV_INST_INFO, ENV_PROD_INFO, ENV_SYS_INFO*
17. Busca en el Centro de Información del DB2 las siguientes vistas administrativas de SYSIBMADM con datos calculados: *APPLICATIONS, APPL_PERFORMANCE, BP_HITRATIO, BP_READ_IO, BP_WRITE_IO, LOCKS_HELD, LOCKWAIT, LOG_UTILIZATION, LONG_RUNNING_SQL, QUERY_PREP_COST*.
18. Busca en el Centro de Información del DB2 la explicación de las tablas particionadas y los cubos (tablas multidimensionales), que son técnicas muy aplicadas en bases de datos con tablas muy grandes.
19. Apúntate a la comunidad de desarrolladores de DB2, entra en su página web y bájate un montón de información, productos y trucos (<http://www.ibm.com/developerworks/ssa>).

BBDD DISTRIBUIDAS Y REPLICADAS

Contenidos

- » BBDD y SGBD distribuidos
- » Componentes de una BBDD distribuida
- » Técnicas de fragmentación
- » Consultas distribuidas
- » Transacciones distribuidas
- » Optimización de consultas sobre bases de datos distribuidas
- » Replicación
- » Ejemplo de replicación en MySQL

Objetivos

- » Reconocer la utilidad de las bases de datos distribuidas
- » Identificar los componentes de las bases de datos replicadas y distribuidas
- » Conocer cómo realizar consultas y transacciones distribuidas
- » Describir las distintas políticas de fragmentación de la información
- » Crear e implantar bases de datos distribuidas
- » Configurar un nodo maestro y otro esclavo para replicar bases de datos

En este capítulo se explican las diferencias entre bases de datos distribuidas y replicadas y se realizan operaciones básicas para que el alumno aprecie la complejidad de este tipo de sistemas y al mismo tiempo su sencilla implantación, gracias a las propias herramientas proporcionadas por los SGBD

5.1. BBDD y SGBD distribuidos

Cuando una base de datos está controlada por más de un servidor se dice que está distribuida. Estos servidores están interconectados mediante una red de telecomunicaciones y los SGBD proporcionan mecanismos para poder consultar la información independientemente del servidor y la ubicación de los datos. Así por ejemplo, una consulta con una join entre la tabla de pedidos y la de clientes puede estar accediendo a dos servidores con dos instancias de bases de datos distintas. Incluso, si las tablas están particionadas es posible que la misma consulta esté accediendo a más de dos servidores. Esto proporciona una especie de balanceo de carga para que la información pueda ser accedida lo más rápidamente posible. De esta manera se consiguen los siguientes beneficios:

1. Más rapidez en el acceso a los datos y capacidad de almacenamiento. Al aumentar el número de ordenadores donde se depositan los datos, aumenta la capacidad de computación, y por tanto, la capacidad de búsqueda de la información.
2. Acceso más flexible. El número de usuarios conectados que puede mantener el sistema se incrementa al haber más capacidad de cómputo y, gracias a la red de la BBDD distribuida, los usuarios pueden acceder desde más ubicaciones.
3. Escalabilidad. Distribuyendo una base de datos en distintas ubicaciones se permite incrementar de forma muy sencilla los recursos en cualquier momento para acomodar el sistema a las demandas de los usuarios y las aplicaciones.
4. Fiabilidad y cierta capacidad de tolerancia a fallos. Realizando una buena política de fragmentación de la base de datos es posible conseguir que aunque algún nodo de la base de datos distribuida sufra algún percance, el resto del sistema siga funcionando con normalidad.

Por otro lado, no todo son beneficios, no hay que perder de vista que estos sistemas son muy costosos de implementar no solo por la duplicidad de recursos en términos de red y servidores, también por la dificultad de la administración de los datos ubicados en distintas localizaciones y porque se introduce un elemento caótico como la dependencia de las redes de comunicaciones para el correcto funcionamiento del intercambio de los datos. Además, la atomicidad de las operaciones es más complicada de conseguir por lo que la gestión transaccional tiene dificultades añadidas como se comentará en la sección 5.3.2.

5.1.1. Componentes de una BBDD distribuida

Para poder crear una base de datos distribuida hay que tener los siguientes componentes.

1. Sistemas gestores con capacidades distribuidas o un SGBD distribuido en sí mismo.
2. Un conjunto de bases de datos locales a cada uno de los servidores que albergarán la BBDD.

3. Una red de comunicaciones, generalmente basada en el protocolo TCP-IP para poder aprovechar las infraestructuras existentes.
4. Enlaces entre las BBDD locales. Los SGBD deben ser capaces de establecer enlaces entre las diversas bases de datos locales, de tal manera que cualquier consulta pueda ser redirigida al servidor donde está la BBDD local.
5. Un diccionario de datos global que indique en qué ubicación está cada uno de los datos que componen la BBDD distribuida.

BASE DE DATOS DISTRIBUIDA

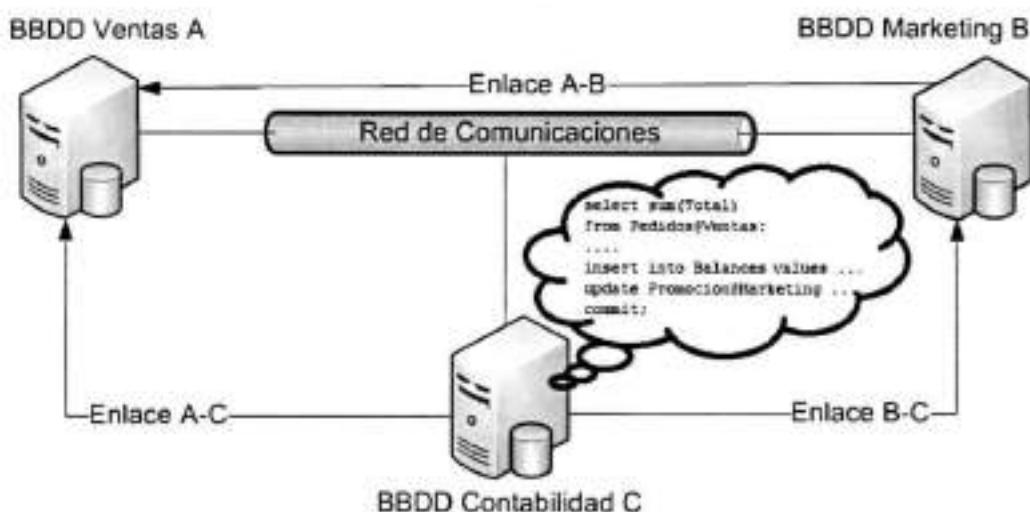


Figura 5.1: Componentes de una base de datos distribuida.

Dependiendo del tipo de componentes elegidos en la formación de una base de datos distribuida se pueden tener:

1. BBDD distribuidas homogéneas: En estas el SGBD utilizado es el mismo para todas las BBDD locales. Generalmente, el SGBD proporciona la tecnología para el enlace de las distintas BBDD locales.
2. BBDD distribuidas heterogéneas: Las bases de datos están distribuidas independientemente del SGBD utilizado como local. En este caso, las aplicaciones que consultan la base de datos distribuida juegan un papel fundamental como mediador.

5.1.2. Otras clasificaciones

Una alternativa a la clasificación anterior podría hacerse atendiendo a los recursos que realmente se encuentran distribuidos, de esta forma, sería muy oportuna la distinción entre bases de datos distribuidas e instancia distribuida.

Bases de datos distribuidas

De todos es conocida la capacidad que existe dentro de una base de datos de hacer referencia a objetos de otra base de datos. En el caso de Oracle esto se consigue fácilmente con la definición de los llamados *dblinks* (*database links* o enlaces de base de datos), que son básicamente puentes de comunicación hacia bases de datos remotas. En MySQL es posible acceder a tablas remotas utilizando el motor de almacenamiento *FEDERATED*. En DB2, en cambio, aunque el objetivo es el mismo, el concepto es ligeramente diferente.

En una instancia DB2, a parte de las bases de datos que existan en local, también se pueden dar de alta bases de datos remotas, sobre las que se podrán hacer conexiones y referencias dentro de las consultas. El comando de catalogación es la forma en la que se le presenta a la instancia DB2 la ubicación de una base de datos remota, indicándole la ip de la máquina remota, el puerto donde escucha su instancia, el nombre de la base de datos y el alias por el que se la referenciará (nótese que son los mismos campos que se usan en la definición de un *dblink* de Oracle). Una vez que la base de datos remota ha sido catalogada, ya puede ser usada en cualquier momento.

Además en DB2, con la instalación del software *DB2 connect* también sería posible interrelacionar bases de datos de entornos Unix/Linux/Windows con bases de datos del *mundo Host*, todo ello también a través de la catalogación.

Por tanto, gracias a los *database links* y a la catalogación, es posible crear complejas estructuras de bases de datos. Estructuras que forman auténticas arquitecturas de bases de datos distribuidas, donde la limitación en la ubicación del dato viene impuesta por la red.

A parte de todo esto, en DB2, y solo en DB2, existe una solución de base de datos que en sí misma es ya una base de datos distribuida. Es una característica añadida a la edición *Enterprise Edition* del DB2, y es conocida con las siglas DPF (del inglés *Database Partitioning Feature*, característica de particionamiento de base de datos).

La utilidad DPF hace posible que una base de datos esté distribuida en tantos nodos como sea preciso para garantizar una velocidad de procesamiento óptima. Con DPF, el tamaño de una base de datos solo está limitado por el número de máquinas de que se disponga (máxima escalabilidad). Está especialmente indicado para grandes DataWarehouse, o bases de datos OLTP que necesiten un rendimiento muy bueno. Permite que múltiples máquinas SMP estén clusterizadas juntas bajo una única base de datos.

Gracias a un potentísimo algoritmo de hashing, la tecnología DPF es capaz de dividir una tabla en tablas más pequeñas separadas en distintos nodos. A su vez, dentro de cada nodo, también se pueden aplicar las técnicas tradicionales de particionamiento de tablas, con lo que se puede conseguir que el acceso al dato sea inmediato.

Instancias distribuidas

Cuando lo que se encuentra distribuido no son los datos sino la memoria, es decir, las distintas cachés que almacenan los bloques leídos, los bloques sucios..., entonces aparece el concepto de instancia distribuida, ya que es en el contexto de la instancia donde se halla la reserva de la memoria en sistema operativo.

En este escenario los datos son siempre compartidos, ubicados generalmente en un único sitio accesible a todos los servidores que forman parte de la arquitectura distribuida, aunque bien es cierto que dentro de cada base de datos también puede hacerse uso de los datos distribuidos explicados en el apartado anterior.

Este tipo de arquitectura es muy apreciada y utilizada como solución de alta disponibilidad.

En Oracle esta alternativa la ofrece el producto llamado RAC (*Real Application Cluster*) donde se precisan al menos dos servidores con almacenamiento compartido (podría montarse también en un solo servidor, pero no con intereses productivos). El almacenamiento compartido albergaría la base de datos propiamente dicha, y en cada uno de los servidores se levantaría una instancia de base de datos. La grandeza del RAC reside en el potentísimo mecanismo que el gestor tiene implementado para la compartición de memoria (la *caché fusión*). Además, en RAC, la configuración del listener posibilita repartir entre los servidores automáticamente las peticiones enviadas a la base de datos en función de la carga de trabajo que tenga cada instancia, y también es capaz de bascular el servicio de un servidor a otro en caso de caída de alguna máquina.

El enfoque de MySQL depende de la utilización del motor de almacenamiento *NDB Cluster* y de la instalación del SGBD configurado para el funcionamiento en Cluster. Se necesitan mínimo dos servidores donde cada instancia ataca a una base de datos ubicada en un almacenamiento compartido.

En DB2 se pueden encontrar dos productos bien distintos para satisfacer esta característica de alta disponibilidad y reparto de carga:

- XKote: pese a no ser un producto IBM, es la opción de alta disponibilidad más antigua diseñada para las bases de datos DB2 en entornos Unix/Linux/Windows. Es competencia directa del RAC de Oracle.
- PureScale: es la alternativa de alta disponibilidad que la factoría IBM ha diseñado para hacer frente a una caída de la máquina que soporta la instancia. El escenario mínimo planteado estaría compuesto por dos máquinas con almacenamiento compartido que tuvieran creadas sendas instancias DB2, de modo que desde ambas se accediera a la/s misma/s base/s de datos. Además, un tercer nodo albergaría la herramienta clave que haría de intermediario para la gestión de la memoria compartida entre las instancias. Con esto se aseguraría balanceo y reconexión instantánea de las sesiones en curso ante cualquier contratiempo, así como pérdida nula o mínima de datos en vuelo.

5.2. Técnicas de fragmentación

El contenido de una tabla puede separarse en varios fragmentos que contendrán suficiente información como para poder reconstruir la tabla original en caso de que sea necesario. Cada fragmento se encontrará en un nodo diferente, pero esta distribución a través de distintos nodos no duplica los registros, solo existirá una copia de cada elemento, por lo

que, al no haber replicación, disminuye el coste de almacenamiento en detrimento de la disponibilidad y fiabilidad de los datos en caso de caída de algún nodo.

La fragmentación puede ser horizontal o vertical y se puede aplicar sucesiva y alternativamente sobre la misma tabla:

Horizontal: los fragmentos son subconjuntos de una tabla y se definen a través de una operación de selección. La tabla original se reconstruye en base a una operación de unión de los fragmentos componentes.

Vertical: los fragmentos son subconjuntos de los atributos con sus valores. Para poder recomponer la tabla original, cada fragmento debe incluir la clave primaria de la tabla.

Mixta: se mezclan las dos técnicas anteriores.

En el capítulo anterior ya se explicó lo que era una tabla particionada, en esencia parece que es lo mismo que la fragmentación horizontal, pero hay una diferencia bastante notable: cada fragmento de una tabla fragmentada se halla en un nodo diferente.

Un ejemplo claro de aplicación de las técnicas de fragmentación se puede ver en las bases de datos DB2 que usan la utilidad DPF nombrada en la sección 5.1.2 en la página 208.

Para que una fragmentación sea correcta, esta debe cumplir con las siguientes reglas:

- Debe ser completa, es decir, cada elemento de la tabla debe estar en algún fragmento.
- Debe ser reconstruible, lo cual implica que debe ser posible conseguir una tabla completa a partir de la combinación de todos los fragmentos.
- Los fragmentos deben ser disjuntos, esto significa que si la fragmentación es horizontal, un elemento que esté en un fragmento concreto no puede estar en ningún otro fragmento más que en aquel. En el caso de fragmentación vertical, puesto que es necesario que se repitan las claves primarias, esta condición solo se tiene que cumplir para el conjunto de atributos que no son clave primaria.

5.3. Consultas distribuidas

Algunos SGBD manejan el concepto de sistemas de bases de datos distribuidas de forma que cualquier aplicación que haga consultas al gestor pueda acceder a su base de datos de forma local y además, acceder a datos de bases de datos remotas.

Oracle basa su concepto de bases de datos distribuidas en un objeto llamado *DATABASE LINK* (Enlace a base de datos). Los DATABASE LINKs son una conexión unidireccional entre una base de datos (cliente) y otra (servidor). Oracle puede soportar bases de datos homogéneas y heterogéneas. Para soportar sistemas heterogéneos es necesario que Oracle pueda acceder a los datos como si estuviese accediendo a otra base de datos Oracle, por lo

que cuenta con agentes llamados *Transparent Gateways* (Pasarelas transparentes). Estos agentes son interfaces que proveen todo lo necesario para poder recuperar, insertar, actualizar o borrar información de forma *transparente* para las aplicaciones que se ejecutan desde Oracle.

5.3.1. DB Links

Un database link es una conexión entre bases de datos. Es unidireccional por lo que se crea desde una base de datos que quiere referenciar a otra. Para que un usuario pueda crear estos DB Links son necesarios los siguientes privilegios:

- CREATE DATABASE LINK (En local).
- CREATE PUBLIC DATABASE LINK (En local).
- CREATE SESSION (En remoto).

Es posible consultar los DB Links en las vistas USER_DB_LINKS, ALL_DB_LINKS y DBA_DB_LINKS dependiendo del usuario con el que se esté conectado al SGBD.

Existen también los enlaces compartidos o *SHARED DB LINKS*. La diferencia es que múltiples procesos clientes pueden usar el enlace compartido simultáneamente. Para crear un DB link es necesario que exista la conexión entre las bases de datos a través de Oracle Net Services, por lo que se debe crear una entrada en el tnsnames.ora de la base de datos a la que se quiere referenciar. Dentro de la definición se puede elegir si la conexión es dedicada o compartida, y no es lo mismo definir que la conexión que va a usar el DB link sea compartida a crear un DB link compartido. A continuación, se muestra un ejemplo de creación de un DB link para conectar una BBDD en versión 10gR2 con una BBDD 11gR2. La entrada en el fichero tnsnames.ora es:

```
t11r2_u =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.129)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SID = t11r2)
  )
)
```

t11r2_u en este caso, es el nombre de la entrada para la conexión remota al servidor con dirección IP *192.168.1.129*.

Una vez agregada la entrada al fichero se puede utilizar la herramienta tnsping para comprobar que hay conectividad:

```
C:\Documents and Settings\usuario>tnsping t11r2_u

TNS Ping Utility for 32-bit Windows: Version 10.2.0.4.0 - Production on 05-MAY-2011 23:23:47

Copyright (c) 1997, 2007, Oracle. All rights reserved.

Archivos de parámetros utilizados:
C:\oracle\product\10.2.0\db_1\network\admin\sqlnet.ora

Adaptador TNSNAMES utilizado para resolver el alias
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.129)
(PORT = 1521)) (CONNECT_DATA = (SERVER = DEDICATED) (SID = t11r2)))
Realizado correctamente (40 msec)
```

Una vez creada la entrada del fichero tnsnames y probada, se pueden crear DB links públicos o privados, y es posible, además, usar un usuario específico para la conexión en la base de datos a referenciar. Para crear un DB Link desde el usuario local al usuario remoto, se ejecutaría el siguiente comando:

```
SQL> CREATE DATABASE LINK tj_11r2
  2  CONNECT TO usuario IDENTIFIED BY password
  3  USING 't11r2_u';

Enlace con la base de datos creado.
```

Al ejecutar el comando anterior, se crea un objeto llamado tj_11r2 que usará la entrada tnsnames *t11r2_u* para conectarse a la BBDD remota. Después, para probar la conexión, se utiliza el nombre del enlace (tj_11r2) para ejecutar una consulta:

```
SQL> select * from dual@tj_11r2;
D
-
X
```

También se pueden crear DB Links públicos que pueden ser usados por cualquier usuario, y además, sean usados por usuarios que también existan en la BBDD que se referencia:

```
SQL> CREATE PUBLIC DATABASE LINK tj_11r2 2  USING 't11r2_u';

Enlace con la base de datos creado.
```

5.3.2. Ejecución de consultas distribuidas

Las consultas distribuidas son consultas que recuperan información de 2 o más nodos en un entorno de bases de datos distribuido. No es lo mismo que hablar de consultas remotas, las consultas remotas son consultas que se realizan a una o más tablas de una base de datos remota. Es importante diferenciar entre estos dos tipos de consultas las distribuidas y remotas porque internamente son diferentes.

Un ejemplo de una consulta remota es el siguiente:

```
SQL> select count(8)
  2   from pacientes@t11r2;
-----  
COUNT(8)  
-----  
100
```

T11R2 es un DB LINK público creado a una BBDD 11gR2 en Linux y la base de datos cliente está en 10gR2 en Windows. La ejecución de la consulta anterior prueba que existen conexión entre versiones diferentes. Esto se logra a través de Oracle Net Services.

En una consulta distribuida puede darse el caso de que existan relaciones fragmentadas. Esto se ilustra en el ejemplo siguiente:

Primero se muestra la tabla PACIENTES, fragmentada en dos bases de datos en distintos servidores y con la siguiente descripción:

Nombre	NULL?	Tipo
ID	NOT NULL	NUMBER
NOMBRE		VARCHAR2(100)
HABITACION		NUMBER

Nombre	NULL?	Tipo
ID	NOT NULL	NUMBER
NOMBRE		VARCHAR2(100)
FECHA_ENTRADA		DATE

Ahora, se consulta un paciente con ID 1, su habitación y fecha de ingreso:

```

SQL> select l.id, l.habitacion, r.fecha_entrada
  2    from pacientes l,
  3         pacientes@t11r2 r
  4   where l.id=r.id
  5     and l.id=1
  6 /

ID HABITACION FECHA_EN
-----
1          10 26/04/11

```

Esta consulta extrae información de dos bases de datos ubicadas en servidores distintos, por tanto, es distribuida. Además, desde la base de datos servidor se puede ver lo siguiente en la vista V\$SESSION:

```

SQL> select sid, serial#, username, program, event, machine , server
  2    from v$session
  3   where username='TJ';

SID SERIAL# USERNAME PROGRAM      EVENT           MACHINE      SERVER
-----  -----  -----  -----  -----
29      48    TJ      ORACLE.EXE  SQL*Net message from client  VILLEGUILLO  DEDICATED

```

Esto demuestra que cada vez que se realice una consulta a una base de datos remota, se abre una sesión con server process dedicado si se define en el tnsnames.ora que sea dedicada. Dejar las conexiones abiertas puede llegar a ser un problema: si se va a crear un proceso que lanza cientos de consultas y que va a muchos nodos en un entorno de bases de datos distribuido, y que además este proceso lo van a ejecutar muchas sesiones, en diferentes nodos, es seguro que se generará una explosión de conexiones abiertas, incrementando de forma radical el consumo de recursos.

Para evitar esta explosión de conexiones abiertas, se puede ejecutar el cierre de los DB Links ejecutando:

```

SQL> alter session close database link t11r2;
Session modificada.

--Esto cierra la sesión en la base de datos servidor:
SQL> select sid, serial#, username, program, event, machine, server
  2    from v$session
  3   where username='TJ'

no rows selected

```

5.4. Transacciones distribuidas

Una transacción distribuida es una transacción que contiene una o más sentencias y que se realiza en 2 o más nodos dentro de un entorno de base de datos distribuido.

También existen las transacciones remotas, y al igual que las consultas remotas, son transacciones con una o más sentencias que se realizan en una base de datos remota.

Los elementos que pueden intervenir a tener en cuenta en una transacción distribuida se pueden ver en la siguiente figura:

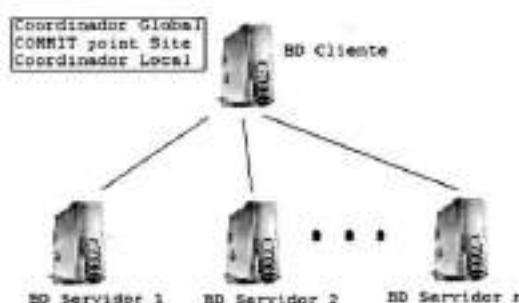


Figura 5.2: Transacción distribuida.

En esta figura se observan los siguientes elementos:

BD Cliente Es un nodo que actúa como cliente donde se referencia la transacción distribuida. Esta BD tiene los DB links que conectan con el resto.

BD Servidor Son las bases de datos que contienen información que va a ser modificada por la transacción distribuida.

Coordinador Local Es un nodo que debe referenciar datos sobre otros nodos para completar su parte de la transacción distribuida.

Coordinador Global El nodo donde se origina la transacción distribuida.

Commit Point Site Es el nodo que inicia las operaciones de COMMIT o ROLLBACK con las instrucciones dadas por el coordinador global. Este punto se define con el parámetro COMMIT_POINT_STRENGTH, en un entorno homogéneo todas las BBDD tienen un parámetro como este. La base de datos con el valor mayor en este parámetro se convierte en COMMIT POINT SITE.

5.4.1. TWO-PHASE COMMIT

Un SGBD debe garantizar la atomicidad en la información de sus bases de datos y en los entornos con bases de datos distribuidas también. Para que esto se produzca, se usa el mecanismo de TWO-PHASE COMMIT o proceso de COMMIT en entornos distribuidos. Su ejecución es un poco más complicada. Se divide en 3 partes:

Fase de preparación Es la primera fase y consiste en que el coordinador global informa a todos los nodos que van que se va a realizar COMMIT o ROLLBACK. Esto se traduce en que cada nodo se *prepara* para una finalización de la transacción distribuida. Para ello realiza 2 tareas:

1. Almacena la información de Redologs.
2. Genera un bloqueo distribuido sobre la tabla para prevenir lecturas.

Cuando se realizan estas 2 operaciones el nodo dice que está *comprometido* para la finalización de la transacción distribuida.

Fase de commit Después de que todos los nodos se comprometen a realizar la transacción, se realizan los siguientes pasos:

1. El coordinador global avisa al COMMIT POINT SITE de que se va a realizar el COMMIT.
2. El COMMIT POINT SITE realiza el COMMIT.
3. El COMMIT POINT SITE indica al coordinador global que hizo el COMMIT.
4. Los coordinadores globales y locales envian mensajes a todos los nodos para hacer COMMIT de la transacción.
5. Cada nodo realiza su parte local del COMMIT de la transacción distribuida y quitan el bloqueo distribuido generado.
6. Todos los nodos notifican a al coordinador global que han realizado el COMMIT.

Fase de olvido Cuando todos notifican al COMMIT POINT SITE que se ha realizado el COMMIT, se realizan las siguientes tareas:

1. El COMMIT POINT SITE borra la información acerca del estado de la transacción distribuida.
2. El COMMIT POINT SITE informa al coordinador global que ha borrado ese estado.
3. El coordinador global borra la información de la transacción distribuida.

5.4.2. Ejemplo de transacción distribuida

Con la tabla pacientes ubicada de forma fragmentada en dos bases de datos, se realizan las siguientes operaciones:

```
SQL> select *
  2  from pacientes;

    ID NOMBRE          HABITACION
-----+
      1 Paciente1        10
```

```

SQL> select *
2*   from pacientes@t11r2

      ID NOMBRE          FECHA_EN
-----+
      1 Paciente1        26/04/11

SQL> update pacientes
2  set nombre='P1'
3  where id=1;

1 fila actualizada.

SQL> update pacientes@t11r2
2  set nombre='P1'
3* where id=1

1 fila actualizada.

SQL> update pacientes
2  set habitacion=10
3  where id=2;

1 fila actualizada.

```

Antes de realizar commit, se puede observar qué está sucediendo, las transacciones creadas y los bloqueos generados en la BD servidor que en este caso es t11r2:

```

SQL> select saddr, sid, serial#, username, program, machine
2  from v$session
3  where username = 'TJ';

SADDR     SID SERIAL# USERNAME      PROGRAM                MACHINE
-----+-----+-----+-----+
37EF0738    28      46 TJ          sqlplus@caar (TNS V1-V3) caar
37EFACCC    29      72 TJ          ORACLE.EXE           VILLEGUILLO

SQL> select SES_ADDR,START_TIME,STATUS,XID
2  from v$transaction;

SES_ADDR START_TIME      STATUS      XID
-----+-----+-----+-----+
37EFACCC 05/06/11 16:03:43 ACTIVE      0600000062010000

SQL> select *
2  from v$lock
3  where sid=29;

```

ADDR	KADDR	SID	TY	ID1	ID2	LMODE	REQUEST	CTIME	BLOCK
37ACAS24	37ACA550	29	AE	100	0	4	0	731	0
00579434	00579464	29	TM	13704	0	3	0	593	0
369C8094	369C80D4	29	TX	393216	354	6	0	593	0

Se observa que la transacción existe, que la transacción es generada por la sesión de la conexión de la BD cliente y que tiene un bloqueo exclusivo sobre fila (LMODE=3 rowX) sobre la tabla PACIENTES:

```
SQL> select object_id
  2  from dba_objects
  3  where object_name='PACIENTES';

OBJECT_ID
-----
13704
```

A continuación, también se muestran sesiones, transacciones y bloqueos en la BD Cliente:

```
SQL> select saddr, sid, serial#, username, program
  2  from v$session
  3* where username='TJ'

SADDR          SID      SERIAL# USERNAME           PROGRAM
-----        -----      -----   -----
30F42BFC       146        1773  TJ                sqlplus.exe

SQL> select SES_ADDR,START_TIME,STATUS,XID
  2* From v$transaction

SES_ADDR START_TIME      STATUS          XID
-----  05/06/11 16:02:46    ACTIVE          08002E00F8180000

SQL> select *
  2  from v$lock
  3  where sid=146;

ADDR          KADDR      SID TY     ID1      ID2      LMODE REQUEST      CTIME      BLOCK
-----        -----      -----  ---  -----  -----  -----  -----  -----  -----
2F5C67E4  2F5C67FC  146  TM    45926      0      3      0      1629      0
2F616204  2F616228  146  TX    524334    6392      6      0      1749      0
```

También se puede observar la información de los DB links abiertos en la sesión con la vista V\$DBLINK. Esta consulta se ejecuta en la propia sesión que tiene la transacción:

```

SQL> desc v$dblink
Nombre                                         ?Nula?   Tipo
-----
DB_LINK                                         VARCHAR2(128)
OWNER_ID                                         NUMBER
LOGGED_ON                                         VARCHAR2(3)
HETEROGENEOUS                                     VARCHAR2(3)
PROTOCOL                                         VARCHAR2(6)
OPEN_CURSORS                                      NUMBER
IN_TRANSACTION                                    VARCHAR2(3)
UPDATE_SENT                                         VARCHAR2(3)
COMMIT_POINT_STRENGTH                           NUMBER

SQL> select *
  2  from v$dblink;

DB_LINK      OWNER_ID LOG HET PROTOC OPEN_CURSORS IN_ UPD COMMIT_POINT_STRENGTH
-----      -----
T11R2        38 YES NO UNKN          0 YES YES           1

```

Una vez realizado el COMMIT, desaparecen las transacciones y los bloqueos. Y la vista de estado de DB links abiertos en la sesión muestra:

```

SQL> select *
  2  from v$dblink;

DB_LINK      OWNER_ID LOG HET PROTOC OPEN_CURSORS IN_ UPD COMMIT_POINT_STRENGTH
-----      -----
T11R2        38 YES NO UNKN          0 NO NO           1

```

Cuando una transacción falla en alguno de los puntos la transacción se convierte en *transacción dudosa*. Puede pasar cuando algún nodo se cae, la conexión de red en la comunicación de algún nodo falla o cualquier error de software no controlado. El proceso RECO es el encargado de resolver las transacciones dudosas automáticamente y hasta que RECO no pueda resolver la transacción dudosa, el dato queda bloqueado para lectura o escritura.

5.5. Optimización de consultas sobre bases de datos distribuidas

Además de pensar en la propia optimización de la consulta en sí misma, cuando se realiza una consulta distribuida, hay que tener en cuenta los siguientes parámetros:

OPEN_LINKS Configura la cantidad de máxima de conexiones abiertas concurrentemente para bases de datos remotas por cada sesión.

COMMIT_POINT_STRENGTH Especifica el valor de commit point site en una transacción. El nodo con el valor más alto se transforma en el commit point site.

Lo primero a tener en cuenta de optimización de consultas distribuidas son las estadísticas, tanto de las estadísticas de sistemas, como las de todos los segmentos que puedan intervenir en la consulta. Se ha de recordar que es transmisión de información por red, por lo que es importante que sea la mínima cantidad. Por lo que de nuevo hay que pensar en *Consistent Gets*. También hay que pensar en buscar el tamaño óptimo de SDU para el Listener, muy grande cada paquete podría penalizar si tenemos consultas con poca información. Y muy pequeño pasaría lo contrario. En las estadísticas de ejecución se pueden buscar tamaños adecuados con relación a las consultas ejecutadas.

Para optimizar consultas distribuidas en Oracle se suelen utilizar dos técnicas. Usar consultas derivadas para disminuir la cantidad de información que se quiere recuperar a través de la red y la utilización de los hints *NO_MERGE* y *DRIVING_SITE*.

5.5.1. Optimización mediante consultas derivadas

Las consultas derivadas o *IN LINE VIEWS* se convierten en una estructura en memoria. Utilizando estas construcciones, se puede disminuir la cantidad de información que se desea recuperar. Por ejemplo, la consulta:

```
select l.id, l.habitacion, r.fecha_entrada
  from pacientes l,
       pacientes@t11r2 r
 where l.id=r.id
   and l.id=1
```

se puede optimizar utilizando una tabla derivada de la siguiente manera:

```
select l.id, l.habitacion, r.fecha_entrada
  from pacientes l,
       (select id, fecha_entreda
        from pacientes@t11r2
       where id=1) r
 where l.id=r.id
   and l.id=1
```

De esta manera, se recupera tan solo la información concerniente a un único paciente (el paciente con id=1), en lugar de todos los pacientes de la base de datos remota.

5.5.2. Optimización mediante hints

En entornos distribuidos es más común que el optimizador pueda no ser del todo bueno con los planes de ejecución, así que el uso de Hints (y su constante control) puede ser útil en

mucho casos. Con los Hints se pueden controlar cómo realizar una consulta. Por ejemplo, NO_MERGE evita los posibles no usos de consultas derivadas porque el optimizador puede determinar cambiar una tabla derivada por una join. Con DRIVING_SITE se define en qué nodo se va a desarrollar la consulta, de tal manera que la BD donde se mezcle la información sea la que va a recibir menor cantidad de información a través de la red. Ejemplos de los cambios que producen estos Hints:

Ejecución sin hint:

```
SQL> select l.id, l.habitacion, r.fecha_entrada
  2   from pacientes l,
  3        (select id, fecha_entrada
  4         from pacientes@t11r2
  5        where id=1) r
  6  where l.id=r.id
  7* and l.id=1

          ID HABITACION FECHA_EN
----- -----
      1          10 26/04/11
```

Plan de ejecución (sin hint):

Plan de Ejecución								
Plan hash value: 3825983327								
#	Operation	Name	Rows	Bytes	Cost (CPU)	Time	Inst	[IN-OUT]
1	0 SELECT STATEMENT		1	38	2	(0) 00:00:01		
1	1 TABLE ACCESS BY INDEX ROWID PACIENTES		1	16	1	(0) 00:00:01		
2	2 NESTED LOOPS		1	38	2	(0) 00:00:01		
3	3 READTEXT	PACIENTES	1	22	1	(0) 00:00:01	T11R2	R+%
4	4 INDEX RANGE SCAN	IDX_PK_PACIENTES	1	1	0	(0) 00:00:01		

Ejecución con el hint NO_MERGE:

```
SQL> select /*+ NO_MERGE(r) */ l.id, l.habitacion, r.fecha_entrada
  2   from pacientes l,
  3        (select id, fecha_entrada
  4         from pacientes@t11r2 r
  5        where l.id=r.id
  6        and l.id=1;

          ID HABITACION FECHA_EN
----- -----
      1          10 26/04/11
```

Plan de ejecución (con hint NO_MERGE):

Plan de Ejecución

Plan hash value: 3635963379

Id Operation	Name	Rows	Bytes	Cost (CPU) Time	Inst IN-OUT
0 SELECT STATEMENT		1	38	2 (0) 00:00:00	1
1 TABLE ACCESS BY INDEX ROWID PACIENTES		1	18	1 (0) 00:00:00	1
2 NESTED LOOPS		1	38	2 (0) 00:00:00	1
3 REMOTE	PACIENTES	1	22	1 (0) 00:00:00 T1IR2 R->S	
+ 4 INDEX RANGE SCAN	IXI_PK_PACIENTES	1	0	0 (0) 00:00:00	1

Ejecución con el hint DRIVING_SITE:

```
SQL> select /*+ DRIVING_SITE(r) */
  2      l.id, l.habitacion, r.fecha_entrada
  3      from pacientes l,
  4          pacientes@t11r2 r
  5      where l.id=r.id
  6      and l.id=1;
```

ID HABITACION FECHA_EN
1 10 26/04/11

Plan de ejecución (con el hint DRIVING_SITE):

Plan de Ejecución

Plan hash value: 1266383717

Id Operation	Name	Rows	Bytes	Cost (CPU) Time	Inst IN-OUT
0 SELECT STATEMENT REMOTE		1	38	2 (0) 00:02:05	1
1 NESTED LOOPS		1	38	2 (0) 00:02:05	1
2 TABLE ACCESS BY INDEX ROWID PACIENTES		1	12	1 (0) 00:01:03 T1IR2	
+ 3 INDEX RANGE SCAN	IXI_PK_PACIENTES	1	0	0 (0) 00:00:01 T1IR2	
4 REMOTE	PACIENTES	1	28	1 (0) 00:01:03 R->S	

5.6. Replicación

La replicación es un proceso distinto de la distribución de una BBDD. Esta técnica está pensada para incrementar el rendimiento de los accesos y consultas a una BBDD mediante la introducción de redundancia en los datos. La diferencia es que, mientras en una BBDD distribuidas no hay redundancia en los datos almacenados (aunque si en las estructuras), en las BBDD replicadas se copia tanto la estructura como la información almacenada en las estructuras.

Para replicar una base de datos se necesita un nodo *maestro*, canalizador de todas las modificaciones sobre los datos y unos nodos *esclavos* que recibirán las actualizaciones del maestro. El nodo maestro, por tanto, es el que almacena todos los cambios sobre los datos. Estos cambios se almacenan en los ficheros de logs del SGBD para posteriormente poder ser aplicados por los nodos esclavos. Por tanto, los nodos esclavos importarán cada cierto tiempo esos ficheros de logs para aplicar los cambios y así mantener una copia actualizada para los usuarios conectados a los esclavos.

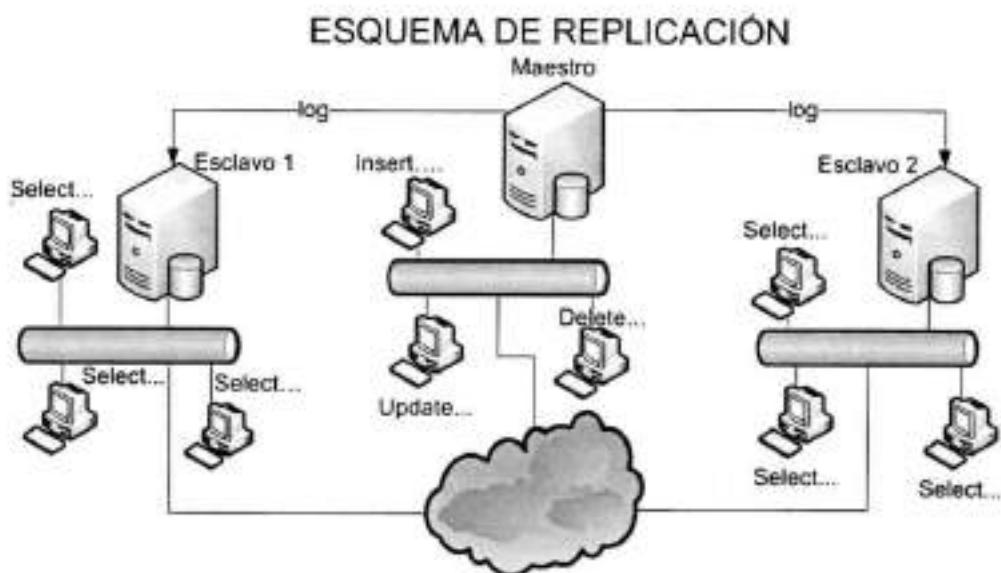


Figura 5.3: Esquema de BBDD replicadas.

Hay que tener en cuenta cuando se trabaja con BBDD replicadas todas las actualizaciones deben realizarse en el nodo maestro dejando los nodos esclavos solo para consultas. Se pueden distinguir dos tipos de replicación:

Síncrona Los cambios se propagan cada vez que se realiza una operación. Se mantiene una conexión abierta entre el nodo maestro y los nodos esclavos para el intercambio constante de cambios en el nodo maestro.

Asíncrona La más común. Los cambios se propagan cada cierto tiempo.

Estos esquemas de replicación son muy útiles en las siguientes circunstancias:

1. Cuando la base de datos es sometida a una gran carga de consultas. En este caso los servidores esclavos distribuyen y balancean esta carga de operaciones.
2. Cuando se necesita tolerancia a fallos. Si el servidor maestro falla, uno de los nodos esclavos puede hacerse cargo y promocionar a maestro.
3. Realizar operaciones de copias de seguridad que podrían afectar al rendimiento del nodo maestro. De esta manera, se puede hacer la copia de seguridad en un nodo esclavo sin necesidad de saturar al nodo maestro.
4. Alta demanda de conexiones de usuarios. En muchas ocasiones las comunicaciones se convierten en el cuello de botella del sistema. Un esquema replicado puede aliviar esta sobrecarga de la red por exceso de conexiones de los clientes.

Para replicar una BBDD hay que seguir los siguientes pasos:

1. Seleccionar un servidor como maestro y el resto como esclavos.
2. Realizar una copia de seguridad de la base de datos del ordenador maestro y volcarla en los servidores esclavos. Es muy importante que no se arranque la base de datos hasta que el proceso de configuración de la replicación haya terminado, puesto que el estado inicial de las bases de datos debe ser idéntico para lograr una sincronización perfecta.
3. Activar el log binario en el servidor maestro para que todos los cambios queden almacenados en los logs. Es esencial si se quieren propagar los cambios en los datos a los esclavos.
4. Configurar un usuario de replicación en el nodo maestro. Este usuario deberá tener los permisos necesarios para que los esclavos puedan conectarse para obtener las copias de los logs con las actualizaciones de los datos.
5. Configurar los nodos esclavos y maestro con un identificador para que cada nodo sepa cuál es su función dentro del esquema de replicación. Por ejemplo, el maestro puede tener un identificador=1 y los esclavos = 2, 3 ...
6. Ejecutar los comandos de inicio de la replicación en los nodos esclavos.

5.7. Ejemplo de replicación. MySQL

Aquí se ilustrará este proceso de replicación de una BBDD MySQL. Se parte de dos servidores MySQL, ambos con la misma base de datos *jardinería* en el mismo estado, justo después de haber hecho una copia de seguridad en el nodo maestro y haberla restaurado en el esclavo. Hay que tener en cuenta que la replicación que se va a utilizar es la replicación asíncrona. El ratio de actualización suele ser de 1MB por segundo dependiendo de la

capacidad del servidor y de las comunicaciones. Si se desea utilizar replicación sincrona habría que utilizar la solución MySQL Cluster. Los parámetros a utilizar son los siguientes:

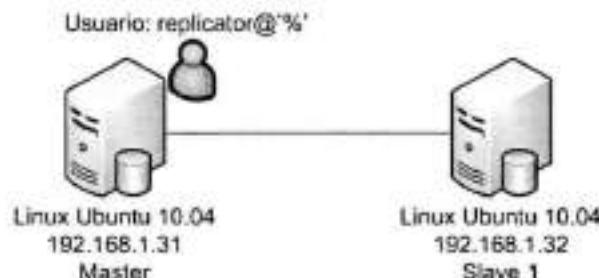


Figura 5.4: Configuración de maestro y esclavo.

A continuación, crear el usuario replicador en el maestro y darle permisos de replication slave

```
maestro
CREATE USER 'replicator'@'%' IDENTIFIED BY 'replslave';
GRANT REPLICATION SLAVE ON *.* to 'replicator'@'%';
```

Ahora hay que asegurarse de que en el esclavo podemos conectarnos con el usuario replicador al maestro:

```
esclavo
mysql -u replicator -h 192.168.1.31 -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 43
...
mysql> exit
```

Editar el fichero my.cnf del nodo maestro para activar el log binario y darle una identificación al maestro con el comando `sudo vi /etc/mysql/my.cnf`:

```
maestro
[mysqld]
log_bin=mysql-bin.log
server-id=1    #Es el maestro
```

Reiniciar de nuevo el maestro para que tome los cambios en el fichero de configuración:

```
maestro
sudo /etc/init.d/mysql restart
```

Una vez configurado el maestro, hay que comprobar el estado del maestro y anotar los parámetros *File* y *Position* que serán necesarios para arrancar la replicación en el esclavo:

```
maestro
mysql> show master status;
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000001 | 269788 |          |          |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

A continuación configurar el nodo maestro cambiando los siguientes parámetros de configuración con el comando `sudo vi /etc/mysql/my.cnf`:

```
esclavo
[mysqld]
server-id=2    #Es el esclavo con id=2, Si hubiera más esclavos, habría que
#nombrarlos con identificadores distintos.
```

Reiniciar de nuevo el esclavo para que tome los cambios en el fichero de configuración:

```
esclavo
sudo /etc/init.d/mysql restart
```

Ahora que están configurados tanto maestro como esclavo, hay que iniciar la replicación, para ello, ejecutamos en el esclavo los siguientes comandos:

```
esclavo
mysql> change master to
-> MASTER_HOST='192.168.1.31',
-> MASTER_USER='replicator',
-> MASTER_PASSWORD='replslave',
-> MASTER_LOG_FILE='mysql-bin.000001',
-> MASTER_LOG_POS=269788;
Query OK, 0 rows affected (0.05 sec)

mysql> start slave;
Query OK, 0 rows affected (0.00 sec)
```

Ya está arrancada la réplica. Ahora solo hay que ejecutar unas cuantas operaciones DML de prueba en el maestro para ver si propaga correctamente a los servidores esclavos:

```
maestro
mysql> update Empleados set Nombre='John',Apellido1='Keiko' where CodigoEmpleado=31;
```

Inmediatamente después es posible ver los cambios en el nodo esclavo:

```
mysql> select Nombre,Apellido from Empleados where CódigoEmpleado=31;
+-----+-----+
| Nombre | Apellido |
+-----+-----+
| John   | Keiko    |
+-----+-----+
```

Para comprobar el estado de la replicación se ejecuta el siguiente comando en el maestro:

```
mysql> show processlist\G;
***** 1. row *****
Id: 37
User: replicator
Host: slave.local:34289
db: NULL
Command: Binlog Dump
Time: 512
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
```

Se puede apreciar que el proceso con pid 37 es el encargado de enviar los log binarios al servidor esclavo. Si se ejecuta el mismo comando en el esclavo:

```
mysql> show processlist\G;
***** 1. row *****
Id: 38
User: system user
Host:
db: NULL
Command: Connect
Time: 753
State: Waiting for master to send event
Info: NULL
***** 2. row *****
Id: 39
User: system user
Host:
db: NULL
Command: Connect
Time: 438
State: Has read all relay log; waiting for the slave I/O thread to update it
Info: NULL
```

En este caso muestra el estado del nodo esclavo esperando la siguiente actualización del maestro (pid 38) y la última actualización que llevó acabo (pid 39).

5.7.1. Replicación en DB2 y Oracle

En DB2 la replicación se puede conseguir de dos maneras:

- Con la configuración HADR (del inglés *High Availability Disaster Recovery*, alta disponibilidad en recuperación ante desastres): está compuesta por dos bases de datos, una primaria y otra secundaria (o en *standby*), cada una con su instancia y su almacenamiento, de modo que ante la caída del primario, el secundario toma el control y se convierte en primario absorbiendo toda la carga de trabajo.

Es muy sencillo de montar, simplemente hay que inicializar los parámetros de configuración donde se indica la ip y el puerto de las instancias protagonistas, y restaurar un backup.

En combinación con la opción ACR (del inglés *Automatic Client Reroute*, enrutamiento automático de clientes), esta solución es capaz de redireccionar automáticamente las peticiones hechas de un nodo hacia el otro, ante una caída del primario.

Debido a que se fundamenta en el intercambio de la información apuntada en los ficheros de log online, no soporta las operaciones hechas sin logging (las que no pasan por los ficheros de log), ya que no se replicarían al otro nodo, dejando por tanto la base de datos secundaria desincronizada, incoherente y corrupta, respecto del nodo primario.

- Con la herramienta QReplication (del inglés *Query Replication*, replicación de consultas): consiste en la replicación de datos desde una base de datos maestra hacia una o varias bases de datos esclavo. Los datos que se transferirán están escogidos de entre todos los disponibles en la base de datos maestra, de modo que a las esclavas solo llegará el subconjunto programado a tal efecto. Es decir, a diferencia del HADR, donde la base de datos secundaria es idéntica a la primaria, con QReplication no es así. La transferencia de datos solo afecta a los programados previamente, estos pueden ser todas las tablas de la base de datos o no, o incluso las tablas completas o no. Esta característica lo convierte en un producto muy flexible.

En Oracle, la replicación se realiza a través de la configuración *Data Guard*. Consiste en montar una base de datos activa y múltiples pasivas o *stand-by*. Al igual que en MySQL y en DB2, todas las operaciones de log generadas se replican de la activa a las *stand-by*.

5.8. Prácticas

Práctica 5.1: Fragmentación horizontal de tablas

Una empresa de telefonía está implementando un sistema de recepción de incidencias. Estas incidencias se recogerán desde un único número de asistencia telefónica y se almacenarán en una base de datos distribuida. Esta tendrá 3 nodos Oracle. Se va a fragmentar una tabla llamada incidencias de forma horizontal, de tal manera que las provincias cuyo nombre comience por las letras de la A a la C estén en el nodo 0, de la D a la M en el nodo 1 y de la O a la Z en el nodo 2. De esta manera estarán balanceadas en número de registros por el número de población. Se pide:

1. Crear la estructura de las tablas en los tres nodos. La tabla incidencia tendrá los campos IdIncidencia, Fecha, Código de Cliente, Texto de la incidencia, Provincia y Estado (abierta / cerrada).
2. Crear un enlace de base de datos entre cada uno de los servidores.
3. Ejecutar una consulta distribuida que devuelva todos los registros de la tabla incidencia.
4. Crear un procedimiento en el nodo 1 que inserte una incidencia en el nodo que le corresponda según la letra de la provincia.
5. Crear un procedimiento que borre una incidencia del nodo adecuado basándose en el campo de provincia.
6. Crea un procedimiento que actualice el estado de la incidencia realizando la operación en el nodo que tenga el registro.

A continuación se desea fragmentar la tabla a través del campo IdIncidencia y el operador Mod (Resto de la división entera). La operación IdIncidencia Mod 3 indicará en qué nodo debe estar almacenado el registro (0,1,2). Para asignar el IdIncidencia sin duplicados se va a crear una secuencia (objeto sequence) para que antes de insertar un registro se consulte y se inserte en el nodo adecuado. Se pide:

1. Crea la secuencia con el comando create sequence en el nodo 0.
2. Crea una función llamada SiguienteIncidencia que devuelva el siguiente número de secuencia.
3. Ejecuta la consulta select siguienteIncidencia()@nodo0 desde los otros dos nodos.

Práctica 5.2: Fragmentación vertical

Una aplicación comercial de marketing tiene una tabla de clientes muy grande con los siguientes campos:

- Campos principales: Código de Cliente, Nombre, Apellido1, Apellido2, email, identificador de facebook, dirección, código postal y país.
- Campos secundarios: pelicula favorita 1, pelicula favorita 2, hobbie 1, hobbie 2, otros intereses y comentarios.

Se pretende fragmentar la tabla utilizando una base de datos distribuida de dos nodos, para lo cual se pide:

1. Realizar la fragmentación de la tabla separando los campos principales y creándolos en un nodo y los campos secundarios (gustos y aficiones) creándolos en otro nodo.
2. Crear un procedimiento que muestre por pantalla la información de un cliente determinado. El procedimiento recibirá como parámetro el código de cliente y consultará los dos nodos para mostrar toda la información.
3. Crear un procedimiento que elimine un cliente de la base de datos.

○

Práctica 5.3: Consultas distribuidas

Sigue los siguientes pasos para ejecutar consultas distribuidas en Oracle:

1. Genera una copia de la máquina virtual con tu servidor Oracle, cámbiale la dirección IP y asegúrate de que tienen conectividad entre ambas.
2. Localiza el fichero tnsnames de tu servidor oracle original y agrega una entrada para que se pueda crear un enlace a la máquina virtual copiada.
3. Prueba con la utilidad tnsping que es posible conectar de la máquina virtual original a la copiada.
4. Crea un usuario llamado User1 en la instancia oracle de la máquina virtual original con los permisos CREATE DATABASE LINK y CREATE PUBLIC DATABASE LINK y el rol RESOURCE.
5. Crea un usuario llamado User2 en la instancia oracle de la máquina virtual copiada con permiso CREATE SESSION y el rol RESOURCE.

6. Crea una tabla llamada Cliente(Código, Nombre, Dirección) en la máquina virtual original en el esquema User1 e inserta dos registros.
 7. Crea una tabla llamada Mascotas(Código, Tipo, Raza, Nombre) en la máquina virtual copiada en el esquema User2 e inserta dos mascotas para cada Cliente.
 8. Ejecuta una consulta que devuelva un listado con las mascotas de cada cliente incluyendo la dirección y el nombre del cliente.
 9. Crea ahora la misma consulta, pero utilizando una tabla derivada para traer solamente las mascotas del primer cliente.
 10. Ejecuta la consulta anterior utilizando los hints NO_MERGE y DRIVING_SITE comparando su plan de ejecución con el comando explain plan.
-
-

Práctica 5.4: Replicación en MySQL

Se va a proceder a la instalación de un servidor con virtualización y balanceo de carga que contendrá una base de datos replicada. Sigue los siguientes pasos:

1. Instala una máquina virtual con linux (Ubuntu) y configura MySQL.
 2. Genera una base de datos con dos tablas e inserta unos cuantos registros.
 3. Copia la máquina virtual en otro directorio y llámala esclavo.
 4. Configura la red y comprueba que haya conectividad entre las dos máquinas.
 5. Sigue el procedimiento del Apartado 5.7 para realizar la replicación entre el servidor maestro y el esclavo.
-
-

5.9. Amplía tus conocimientos

1. Busca documentación sobre los problemas típicos encontrados en las transacciones distribuidas y cómo resolverlos.
2. Prueba la replicación en Oracle a través de vistas materializadas.
3. Es muy fácil y barato montar un cluster activo-pasivo con una base de datos MySQL con tres máquinas virtuales. Una máquina virtual será una SAN virtual (Storage Area Network) y puedes cargar dos Windows Server en las otras dos máquinas virtuales. Estas dos máquinas virtuales harán de nodos activo y pasivo. A través del software de Cluster de Windows Server se puede configurar el servicio servidor de MySQL para que cuando caiga un nodo, el otro nodo se haga cargo del servicio de la base de datos.
4. Apache Cassandra es un SGBD distribuido de código libre. Prueba a descargarlo de la página <http://cassandra.apache.org/> y experimenta con este SGBD.
5. Relacionado con las BBDD distribuidas, existe el proyecto DiGIR (Distributed Generic Information Retrieval). Busca en internet información sobre esto.
6. Los *Data WareHouses* pueden ser vistos como una especie de replicación asíncrona. Investiga un poco qué son estos tipos de bases de datos y cómo pueden ser implementados en los SGBD típicos.
7. Investiga qué es la replicación multimaster y prueba a crear una réplica en Oracle.
8. Prueba en motor de base de datos FEDERATED de MySQL para crear base de datos distribuidas en este SGBD.

Administración de Sistemas Gestores de Bases de Datos

Este texto es la continuación del libro Gestión de Bases de Datos de la misma editorial y autores. Siguiendo la filosofía del primer libro, el contenido del libro tiene una orientación práctica, con actividades, consejos y prácticas especialmente diseñadas para facilitar al lector su completo seguimiento.

El libro está dividido en cinco capítulos, ordenados por dificultad. Los dos primeros capítulos abarcan tareas de administración sencillas como la instalación y configuración de un SGBD, mantenimiento y gestión de usuarios. El capítulo 3 es de dificultad media y trata sobre la automatización de las tareas de administración. Los dos últimos capítulos son de un nivel de dificultad avanzada, tratando temas tan complejos como la optimización (tunning) de BBDD y la organización de la información en bases de datos distribuidas.

Los tres primeros capítulos se han abordado de forma genérica, haciendo hincapié en operaciones con MySQL, Oracle y DB2, mientras que los dos últimos, debido a su complejidad, tan solo se han tratado desde el punto de vista de Oracle.

El objetivo del libro no es ser una guía de administración de un solo SGBD, sino la formación de administradores de bases de datos actualizados, versátiles y competentes.

Existe multitud de material disponible para la realización de las prácticas propuestas en el blog bbdd-asir.blogspot.com y en la página web www.garceta.es.

ISBN 978-84-9281-284-2

www.garceta.es

