Table of Contents

SQL

Introducción

SQL (por sus siglas en inglés **Structured Query Language**; en español **lenguaje de consulta estructurada**) es un lenguaje específico del dominio utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales. Una de sus principales características es el manejo del álgebra y el cálculo relacional para efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como realizar cambios en ellas.

Originalmente basado en el álgebra relacional y en el cálculo relacional, SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. El alcance de SQL incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso a los datos. También el SQL a veces se describe como un lenguaje declarativo, también incluye elementos procesales.

SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar Frank Codd como se describió en su papel de 1970 *El modelo relacional de datos para grandes bancos de datos compartidos*. A pesar de no adherirse totalmente al modelo relacional descrito por Codd, pasó a ser el lenguaje de base de datos más usado.

ANSI SQL

SQL pasó a ser el estándar del Instituto Nacional Estadounidense de Estándares (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces, el estándar ha sido revisado para incluir más características. A pesar de la existencia de ambos estándares, la mayoría de los códigos SQL no son completamente portables entre sistemas de bases de datos diferentes sin ajustes. Orígenes y evolución.

Historia

Los orígenes de SQL están ligados a las bases de datos de las pc o móvil aun a los de las bases de datos relacionales. En 1970 E. F. Codd propone el modelo relacional y asociado a este un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de IBM definieron el lenguaje SEQUEL (Structured English Query Language) que más tarde fue ampliamente implementado por el sistema de gestión de bases de datos (SGBD) experimental System R, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un producto comercial.

El SEQUEL terminó siendo el predecesor de SQL, que es una versión evolucionada del primero. SQL pasa a ser el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y fue por fin estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, "SQL-86" o "SQL1". Al año siguiente este estándar es también adoptado por ISO.

Sin embargo, este primer estándar no cubría todas las necesidades de los desarrolladores e incluía funcionalidades de definición de almacenamiento que se consideró suprimirlas. Así que, en 1992, se lanzó un nuevo estándar ampliado y revisado de SQL llamado "SQL-92" o "SQL2".

En la actualidad SQL es el estándar *de facto* de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio.

El ANSI SQL sufrió varias revisiones y agregados a lo largo del tiempo:

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonuméricas.3
2005	SQL:2005		ISO/IEC 9075-14:2005 Define las maneras en las cuales SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y

		los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL:2008	Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.4
2011	SQL:2011	Datos temporales (PERIOD FOR). Mejoras en las funciones de ventana y de la cláusula FETCH.
2016	SQL:2016	Permite búsqueda de patrones, funciones de tabla polimórficas y compatibilidad con los ficheros JSON.

Características generales de SQL

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.5

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento" que, gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros —y no a registros individuales— permite una alta productividad en codificación y la orientación a objetos. De esta forma, una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros. SQL también tiene las siguientes características:

- **Lenguaje de definición de datos**: El LDD de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
- **Lenguaje interactivo de manipulación de datos**: El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.
- **Integridad**: El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
- **Definición de vistas**: El LDD incluye comandos para definir las vistas.
- **Control de transacciones**: SQL tiene comandos para especificar el comienzo y el final de una transacción.
- **SQL incorporado y dinámico**: Esto quiere decir que se pueden incorporar instrucciones de SQL en lenguajes de programación como: C++, C, Java, PHP, Cobol, Pascal y Fortran.
- **Autorización**: El LDD incluye comandos para especificar los derechos de acceso a las relaciones y a las vistas.

Tipos de datos

Algunos de los tipos de datos básicos de SQL son:

- **Varchar**: Recibe cadena de palabras compuestas de letras, números y caracteres especiales.
- **Date**: una fecha de calendario que contiene el año (de cuatro cifras), el mes y el día.
- **Time**: La hora del día en horas minutos segundos (el valor predeterminado es 0).

- **Datetime**: la combinación de Date y Time. Es decir, guarda o almacena una fecha con su respectiva hora

Optimización

Como ya se dijo antes, y suele ser común en los lenguajes de acceso a bases de datos de alto nivel, SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

El orden de ejecución interno de una sentencia puede afectar seriamente a la eficiencia del SGBD, por lo que se hace necesario que éste lleve a cabo una optimización antes de su ejecución. Muchas veces, el uso de índices acelera una instrucción de consulta, pero ralentiza la actualización de los datos. Dependiendo del uso de la aplicación, se priorizará el acceso indexado o una rápida actualización de la información. La optimización difiere sensiblemente en cada motor de base de datos y depende de muchos factores.

Los sistemas de bases de datos modernos poseen un componente llamado optimizador de consultas. Este realiza un detallado análisis de los posibles planes de ejecución de una consulta SQL y elige aquel que sea mas eficiente para llevar adelante la misma.

Existe una ampliación de SQL conocida como FSQL (Fuzzy SQL, SQL difuso) que permite el acceso a bases de datos difusas, usando la lógica difusa. Este lenguaje ha sido implementado a nivel experimental y está evolucionando rápidamente.

Lenguaje de definición de datos (DDL)

El lenguaje de definición de datos (en inglés *Data Definition Language*, o *DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.

CREATE | CREAR

Este comando permite crear objetos de datos, como nuevas bases de datos, tablas, vistas y procedimientos almacenados.

Ejemplo (crear una tabla)

```
CREATE TABLE 'CUSTOMERS';
```

ALTER | MODIFICAR

Este comando permite modificar la estructura de una tabla u objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplo (agregar columna a una tabla)

```
ALTER TABLE 'ALUMNOS' ADD EDAD INT UNSIGNED;
```

DROP | ELIMINAR

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplo

```
DROP TABLE 'ALUMNOS';
```

TRUNCATE | TRUNCAR

Este comando solo aplica a tablas y su función es borrar el contenido completo de la tabla especificada. La ventaja sobre el comando DROP, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Ejemplo

```
TRUNCATE TABLE 'NOMBRE_TABLA';
```

Lenguaje de manipulación de datos DML(Data Manipulation Language)

Definición

Un lenguaje de manipulación de datos (*Data Manipulation Language*, o *DML* en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

SELECT | SELECCIONAR

La sentencia **SELECT** nos permite consultar los datos almacenados en una tabla de la base de datos.

Forma básica

```
SELECT [ALL | DISTINCT ]
             <nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista>
        [{,<nombre_tabla>|<nombre_vista>}]
[WHERE <condición> [{ AND|OR <condición>}]]
```

```
[GROUP BY <nombre_campo> [{,<nombre_campo >}]]

[HAVING <condición>[{ AND|OR <condición>}]]

[ORDER BY <nombre_campo>|<indice_campo> [ASC | DESC]

                [{,<nombre_campo>|<indice_campo> [ASC | DESC ]}]]
```

SELECT	Palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.
ALL	Indica que queremos seleccionar todos los valores. Es el valor por defecto y no suele especificarse casi nunca.
DISTINCT	Indica que queremos seleccionar sólo los valores distintos.
FROM	Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula **WHERE**.
WHERE	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admite los operadores lógicos **AND** y **OR**.
GROUP BY	Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.
HAVING	Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de **WHERE** pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a **GROUP BY** y la condición debe estar referida a los campos contenidos en ella.
ORDER BY	Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con **ASC** (orden ascendente) y **DESC** (orden descendente). El valor predeterminado es **ASC.**

Ejemplo:

Para formular una consulta a la tabla Coches y recuperar los campos matricula, marca, modelo, color, numero_kilometros, num_plazas debemos ejecutar la siguiente consulta. Los datos serán devueltos ordenados por marca y por modelo en orden ascendente, de menor a mayor. La palabra clave **FROM** indica que los datos serán recuperados de la tabla Coches.

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas
FROM Coches
ORDER BY marca,modelo;
```

Ejemplo de Consulta simplificada a través de un comodín de Campos (*):

El uso del asterisco indica que queremos que la consulta devuelva todos los campos que existen en la tabla y los datos serán devueltos ordenados por marca y por modelo.

```
SELECT * FROM Coches ORDER BY marca, modelo;
```

Cláusula WHERE

La cláusula *WHERE* es la instrucción que nos permite filtrar el resultado de una sentencia **SELECT**. Habitualmente no deseamos obtener toda la información existente en la tabla, sino que queremos obtener sólo la información que nos resulte útil en ese momento. La cláusula **WHERE**

filtra los datos antes de ser devueltos por la consulta. Cuando en la Cláusula WHERE queremos incluir un tipo texto, debemos incluir el valor entre comillas simples.

Ejemplos:

En nuestro ejemplo, se desea consultar un coche en concreto, para esto se agregó una cláusula **WHERE**. Esta cláusula especifica una o varias condiciones que deben cumplirse para que la sentencia **SELECT** devuelva los datos. En este caso la consulta devolverá sólo los datos del coche con matrícula para que la consulta devuelva sólo los datos del coche con matrícula `MF-234-ZD` o bien la matrícula `FK-938-ZL` . Se puede utilizar la cláusula **WHERE** solamente, ó en combinación con tantas condiciones como queramos.

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas
FROM Coches
WHERE matricula = 'MF-234-ZD'
OR matricula = 'FK-938-ZL';
```

Una Condición WHERE puede ser negada a través del Operador Lógico NOT. La Siguiente consulta devolverá todos los datos de la tabla Coches, menos el que tenga la Matrícula `MF-234-ZD` .

```
SELECT matricula,marca, modelo, color, numero_kilometros, num_plazas
FROM coches
WHERE NOT matricula = 'MF-234-ZD';
```

La Siguiente consulta utiliza la condicional **DISTINCT**, la cual nos devolverá todos los valores distintos formados por los Campos *Marca y Modelo*. de la tabla *coches.*

```
SELECT DISTINCT marca, modelo FROM coches;
```

Cláusula ORDER BY

La cláusula *ORDER BY* es la instrucción que nos permite especificar el orden en el que serán devueltos los datos. Podemos especificar la ordenación ascendente o descendente a través de las palabras clave **ASC** y **DESC**. La ordenación depende del tipo de datos que este definido en la columna, de forma que un campo numérico será ordenado como tal, y un alfanumérico se ordenará de la A a la Z, aunque su contenido sea numérico. El valor predeterminado es ASC si no se especifica al hacer la consulta.

Ejemplos:

```
'SELECT' matricula,
marca,
modelo,
color,
numero_kilometros,
num_plazas
'FROM' coches
'ORDER BY' marca 'ASC', modelo 'DESC';
```

Este ejemplo, selecciona todos los campos matricula, marca, modelo, color, numero_kilometros y num_plazas de la tabla coches, ordenándolos por los campos marca y modelo, marca en forma ascendente y modelo en forma descendente.

```
SELECT matricula,
       marca,
       modelo,
       color,
       numero_kilometros, num_plazas
FROM  coches
ORDER BY 2;
```

Este ejemplo, selecciona todos los campos matrícula, marca, modelo, color, numero_kilometros y num_plazas de la tabla coches, ordenándolos por el campo *marca*, ya que aparece en segundo lugar dentro de la lista de campos que componen la **SELECT.**

SUBCONSULTAS

Una subconsulta es una sentencia SELECT que está embebida en una cláusula de otra sentencia SQL. También pueden utilizarse subconsultas en los comandos INSERT, UPDATE, DELETE y en la cláusula FROM7.

Las subconsultas pueden resultar útiles si necesitas seleccionar filas de una tabla con una condición que depende de los datos de la propia tabla o de otra tabla.

La subconsulta (consulta interna), se ejecuta antes de la consulta principal; el resultado de la subconsulta es utilizado por la consulta principal (consulta externa).

```
SELECT c.matricula, c.modelo
FROM   coches c
WHERE  c.matricula IN
                        (
                                SELECT m.matricula
                                FROM   multas m
                                WHERE  m.importe > 100
                );
```

En este ejemplo, se seleccionan las matriculas y los modelos de los coches cuyas multas superan los u$s 100.

INSERT | INSERTAR

Una sentencia *INSERT* de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica

```
 INSERT INTO 'tablatura' ('columnaA',['columnaB,... '])
VALUES ('valor1', ['valor2,...'])
```

```
O también se puede utilizar como:
```

```
INSERT INTO tablatura VALUES ('valor1','valor2')
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia `INSERT` deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo

```
 INSERT INTO agenda_telefonica (nombre, numero)
VALUES ('Roberto Jeldrez', 4886850);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
 INSERT INTO nombreTabla VALUES ('valor1', ['valor2,…'])
```

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
 INSERT INTO agenda_telefonica
VALUES ('Jhonny Aguilar', 080473968);
```

Formas avanzadas

Una característica de SQL (desde SQL-92) es el uso de *constructores de filas* para insertar múltiples filas a la vez, con una sola sentencia SQL:

```
INSERT INTO ''tabla'' (''columna1'', [''columna2,... ''])
VALUES (''valor1a'', [''valor1b,...'']),
(''value2a'', [''value2b,...'']),...;
```

Esta característica es soportada por DB2, PostgreSQL (desde la versión 8.2), MySQL, y H2.

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas en la tabla 'agenda_telefonica'):

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández', '4886850'),
('Alejandro Sosa', '4556550');
```

Que podía haber sido realizado por las sentencias

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández', '4886850');
INSERT INTO agenda_telefonica VALUES ('Alejandro Sosa', '4556550');
```

Notar que las sentencias separadas pueden tener semántica diferente (especialmente con respecto a los triggers), y puede tener diferente rendimiento que la sentencia de inserción múltiple.

Para insertar varias filas en MS SQL puede utilizar esa construcción:

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212'
UNION ALL
SELECT 'Peter Doe', '555-2323';
```

Tenga en cuenta que no se trata de una sentencia SQL válida de acuerdo con el estándar SQL (SQL: 2003), debido a la cláusula subselect incompleta.

Para hacer lo mismo en Oracle se usa la Tabla DUAL, siempre que se trate de solo una simple fila:

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM DUAL
UNION ALL
SELECT 'Peter Doe','555-2323' FROM DUAL
```

Una implementación conforme al estándar de esta lógica se muestra el siguiente ejemplo, o como se muestra arriba (no aplica en Oracle):

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM LATERAL ( VALUES (1) ) AS t(c)
UNION ALL
```

```
SELECT 'Peter Doe','555-2323' FROM LATERAL ( VALUES (1) ) AS t(c)
```

Copia de filas de otras tablas

Un INSERT también puede utilizarse para recuperar datos de otros, modificarla si es necesario e insertarla directamente en la tabla. Todo esto se hace en una sola sentencia SQL que no implica ningún procesamiento intermedio en la aplicación cliente. Un SUBSELECT se utiliza en lugar de la cláusula VALUES. El SUBSELECT puede contener JOIN, llamadas a funciones, y puede incluso consultar en la misma TABLA los datos que se inserta. Lógicamente, el SELECT se evalúa antes que la operación INSERT esté iniciada. Un ejemplo se da a continuación.

```
INSERT INTO phone_book2
SELECT *
FROM phone_book
WHERE name IN ('John Doe', 'Peter Doe')
```

Una variación es necesaria cuando algunos de los datos de la tabla fuente se está insertando en la nueva tabla, pero no todo el registro. (O cuando los esquemas de las tablas no son iguales.)

```
INSERT INTO phone_book2 ( [name], [phoneNumber] )
SELECT [name], [phoneNumber]
FROM phone_book
WHERE name IN ('John Doe', 'Peter Doe')
```

El SELECT produce una tabla (temporal), y el esquema de la tabla temporal debe coincidir con el esquema de la tabla donde los datos son insertados.

UPDATE

Una sentencia *UPDATE* de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo

```
UPDATE My_table SET field1 = 'updated value asd' WHERE field2 = 'N';
```

DELETE

Una sentencia *DELETE* de SQL borra uno o más registros existentes en una tabla.

Forma básica

```
DELETE FROM tabla WHERE columna1 = 'valor1';
```

Ejemplo

```
DELETE FROM My_table WHERE field2 = 'N';
```

Recuperación de clave

Los diseñadores de base de datos que usan una clave suplente como la clave principal para cada tabla, se ejecutará en el ocasional escenario en el que es necesario recuperar automáticamente la base de datos, generando una clave primaria de una sentencia SQL INSERT para su uso en otras sentencias SQL. La mayoría de los sistemas no permiten sentencias SQL INSERT para retornar fila de datos. Por lo tanto, se hace necesario aplicar una solución en tales escenarios.

Implementaciones comunes incluyen:

- Utilizando un procedimiento almacenado específico de base de datos que genera la clave suplente, realice la operación INSERT, y finalmente devuelve la clave generada.
- Utilizando una sentencia SELECT especifica de base de datos, sobre una tabla temporal que contiene la última fila insertada. DB2 implementa esta característica de la siguiente manera:

```
SELECT *
FROM NEW TABLE ( INSERT INTO phone_book VALUES ( 'Cristobal
Jeldrez','0426.817.10.30' ) ) AS t
```

- Utilizando una sentencia SELECT después de la sentencia INSERT con función específica de base de datos, que devuelve la clave primaria generada por el registro insertado más recientemente.
- Utilizando una combinación única de elementos del original SQL INSERT en una posterior sentencia SELECT.
- Utilizando un GUID en la sentencia SQL INSERT y la recupera en una sentencia SELECT.
- Utilizando la función de PHP mysql_insert_id() de MySQL después de la sentencia INSERT.
- Utilizando un INSERT con la cláusula RETURNING para Oracle, que sólo se puede utilizar dentro de un PL/SQL bloque, en el caso de PostgreSQL se puede usar también tanto con SQL como con PL/SQL.

```
INSERT INTO phone_book VALUES ( 'Cristobal Jeldrez','0426.817.10.30' )
RETURNING phone_book_id INTO v_pb_id
```

- En el caso de MS SQL se puede utilizar la siguiente instrucción:

```
Set NoCount On;

INSERT INTO phone_book VALUES ( 'Cristobal Jeldrez','0426.817.10.30' );
Select @@Identity as id
```

Disparadores

Los disparadores, también conocidos como desencadenantes (*triggers* en inglés) son definidos sobre la tabla en la que opera la sentencia INSERT, y son evaluados en el contexto de la operación. Los desencadenantes BEFORE INSERT permiten la modificación de los valores que se insertarán en la tabla. Los desencadenantes AFTER INSERT no puede modificar los datos de ahora en adelante, pero se puede utilizar para iniciar acciones en otras tablas, por ejemplo para aplicar mecanismos de auditoría Excel.

Sistemas de gestión de base de datos

Los sistemas de gestión de base de datos con soporte SQL más utilizados son, por orden alfabético:

- DB2
- Firebird
- HSQL
- Informix
- Interbase
- MariaDB
- Microsoft SQL Server

- MySQL
- Oracle
- PostgreSQL
- PervasiveSQL
- SQLite
- Sybase ASE

Interoperabilidad

El lenguaje de consultas de los diferentes sistemas de gestión de bases de datos son incompatibles entre ellos y no necesariamente siguen completamente el estándar. En particular, la sintaxis de fecha y tiempo, la concatenación de cadenas, nulas, y la comparación de textos en cuanto al tratamiento de mayúsculas y minúsculas varían de un proveedor a otro. Una excepción particular es PostgreSQL, que se esfuerza por lograr el cumplimiento del estándar.8

Las implementaciones populares de SQL omiten comúnmente soporte para funciones básicas de SQL estándar, como la de los tipos de dato `DATE` o `TIME`. Es el caso del manejador de bases de datos de Oracle (cuyo tipo `DATE` se comporta como `DATETIME`, y carece de un tipo `TIME`)9 y MS SQL Server (antes de la versión de 2008). Como resultado, el código SQL rara vez puede ser portado entre los sistemas de base de datos sin modificaciones.

Hay varias razones para esta falta de portabilidad entre sistemas de bases de datos:

- La complejidad y el tamaño del estándar SQL conlleva a que la mayoría de las implementaciones de SQL no sean compatibles con la norma completa.
- La norma no especifica el comportamiento de la base de datos en varias áreas importantes (por ejemplo, índices, almacenamiento de archivos, etc.), dejando a las implementaciones decidir cómo comportarse.
- El estándar SQL especifica con precisión la sintaxis que un sistema de base de datos conforme debe implementar. Sin embargo, no está tan bien definida la especificación en el estándar de la semántica de las construcciones del lenguaje, lo que lleva a ambigüedad.
- Muchos proveedores de bases de datos tienen grandes bases de clientes existentes, por lo que introducir cambios para adaptarse el estándar podría producir incompatibilidades en las instalaciones de los usuarios y el proveedor puede no estar dispuesto a abandonar la compatibilidad con versiones anteriores.
- Hay poco incentivo comercial para que un proveedor facilite a los usuarios el cambiar de proveedor de bases de datos.
- Los usuarios que evalúan el software de base de datos tienden a valorar más otros factores tales como el rendimiento más alto en sus prioridades sobre las conformidad al estándar.

El estándar ODBC (Open Database Connectivity) permite acceder a la información desde cualquier aplicación independientemente del sistema de gestión de base de datos (DBMS) en el que esté almacenada la información, desacoplando así la aplicación de la base de datos.

Bases de datos tolerantes a fallo

Hoy en día, una de las principales preocupaciones de las empresas reside en la necesidad de proteger la información que manejan. Por este motivo, invierten muchos recursos en disponer de sistemas cada vez más fiables y eficientes. Esta necesidad implícita que tienen las compañías de ofrecer una imagen de fortaleza ante sus clientes, lleva a las mismas a incorporar a su organización los últimos avances en tecnología.

Durante los últimos años, uno de los campos de investigación en los que se han invertido mayor cantidad de recursos, está relacionado con el desarrollo de bases de datos. En la actualidad, este tipo de sistemas suele integrarse dentro de una arquitectura distribuida que de soporte para tolerancia a fallos. Al tratarse de sistemas que requieren alta disponibilidad, debe ofrecerse continuidad en el servicio aún en presencia de fallo.

Es más, con la aparición de Internet, el problema se ha agravado aún más, pués sistemas accedidos hasta ahora por cientos de personas, han visto como, de la noche a la mañana, quedaban obsoletos e incapaces de dar respuesta al crecimiento inesperado de la demanda. Por ello, únicamente aquellos que hayan sido precavidos habrán sido capaces de compensar este incremento, los que no, tarde o temprano se verán obligados a adaptarse a los tiempos modernos.

Es en este sentido, que muchos fabricantes de servidores de bases de datos, ya incorporan en sus productos potentes mecanismos de replicación. Con ello, pretenden que quiénes utilicen sus servidores, tengan la seguridad de que la información que albergan va a estar siempre disponible (con independencia de los posibles fallos que puedan acontecer).

Características de un Sistema de Gestión de Bases de Datos

Un Sistema de Gestión de Bases de Datos (DBMS) es un conjunto de programas que permiten la construcción de una base de datos y de las aplicaciones que las emplean. Estas son las responsabilidades de un DBMS:

- **Creación de la base de datos**, generalmente en uno o más archivos almacenados en el disco duro de un ordenador. Algunos sistemas crean un único archivo dentro del cual crean a su vez una o más bases de datos. Los usuarios no tienen porque conocer la estructura interna de estos archivos ya que el DBMS proporciona todo lo necesario.
- **Proporcionar facilidades de consulta y actualización**. Un DBMS dispone de métodos para obtener datos que cumplan ciertos requisitos, como los pedidos realizados por un cliente concreto que todavía no se hayan servido. Antes de que se generalizase el uso de SQL, las consultas como esta se hacían de una forma distinta en cada sistema.
- **Mantener un registro de acciones**. Un DBMS mantiene un registro de todos los cambios realizados sobre los datos en un período de tiempo. Esto se puede utilizar más tarde para descubrir accesos no autorizados, pero su función más importante es, posiblemente, la reconstrucción de los datos si se produce un fallo en el sistema, como un corte no previsto de la corriente eléctrica. Generalmente una copia de seguridad y el registro de las acciones desarroladas desde que se realizó la copia servirán para reconstruir una base de datos.

- **Gestionar la seguridad de la base de datos**. Un DBMS permite controlar el acceso de forma que sólo los usuarios autorizados puedan manipular los datos de la base de datos así como su estructura. Generalmente, existe una jerarquía de usuarios en cada base de datos, desde un ``superusuario" o administrador de la base de datos, que puede cambiarlo todo pasando por los usuarios con permisos para añadir o borrar de datos, y hasta los usuarios que sólo pueden leer datos. El DBMS dispondrá de facilidades para añadir y eliminar usuarios, así como para indicar qué capacidades del sistema de base de datos puede emplear cada uno.

Además de estos servicios básicos, algunos DBMS, como los productos corporativos de Oracle y Microsoft, ofrecen toda clase de servicios y ayudas adicionales. Estos son algunos de los más importantes:

- **Transacción**. Cuando definimos una transacción, agrupamos una serie de eventos que deben completarse satisfactoriamente antes de que sus resultados sean permanentes. O todos los cambios realizados por la transacción tienen éxito o todos ellos fallan y la base de datos vuelve al estado en el que estaba antes de comenzar la transacción. El objetivo de las transacciones en una base de datos es llevar la base de datos de un estado consistente a otro estado consistente.

  El ejemplo clásico de esto es una transferencia bancaria. Si tenemos una cuenta en un banco y transferimos dinero de esta a otra cuenta, la aplicación del banco ejecutará dos sentencias SQL `UPDATE`, la primera para retirar la cantidad de dinero de una cuenta y la segunda para depositar esa misma cantidad de dinero en la otra cuenta. Pero, si el sistema se bloquea entre una sentencia y otra podrían surgir problemas. Las transacciones permiten agrupar estas dos sentencias de forma que se ejecuten las dos al mismo tiempo o no se ejecute ninguna. Si el banco retira el dinero de una cuenta pero no puede depositarlo en la otra, entonces lo devolverá a la cuenta inicial y todo quedará como antes.

  El funcionamiento interno de las transacciones varía de un sistema de base de datos a otro. Las transacciones son una de las características fundamentales de una base de datos, y una de las más complejas de realizar correctamente. Si señalamos una serie de operaciones como pertenecientes a una transacción, la base de datos tomará nota de las acciones y sus efectos y si alguna de ellas falla por alguna razón, entonces volverá hacia atrás para dejar la base de datos en el estado inicial.

- **Acceso concurrente**. Si una base de datos se emplea en varias aplicaciones o si varios usuarios acceden al mismo tiempo a ella, el DBMS se asegurará de que cada petición no afecta a las demás. Podemos decir que el DBMS debe asegurar que las transacciones que tengan lugar de forma concurrente no deben interferir unas con otras.

  De nuevo, la forma en la que esto tiene lugar varía mucho de un fabricante de sistemas de bases de datos a otro. La mayoría de las bases de datos emplean controles de bloqueo. Partamos de una transacción que calcula el valor total del inventario de una tienda en una tabla llamada `Inventory`. Mientras esta transacción tiene lugar, una segunda transacción quiere anotar la llegada de nuevos materiales y para ello insertará o actualizará el contenido de la columna `cantidad` de una fila concreta en la tabla `Inventory`. En este caso, se debería permitir que la primera transacción vea la actividad de la segunda transacción o no.

Una posible solución sería colocar un bloqueo en la tabla `Inventory` mientras tenga lugar la primera transacción. Esto significaría que la segunda transacción debería esperar a que terminase la primera. Obviamente el problema sería que nadie podría acceder a la tabla `Inventory` durante este tiempo. Cuantos más recursos se bloqueen de esta forma menor será el rendimiento de la aplicación y menos usuarios admitirá. Las bases de datos más potentes permiten disminuir este impacto ya que sólo bloquean una fila cada vez o, en el caso de Oracle, nada.

Si alguna vez ha programado código multitarea se habrá encontrado con el mismo problema. Algunas secciones del código, llamadas secciones críticas, sólo pueden ser ejecutadas simultáneamente por un hilo de ejecución. El resto de hilos que deseen acceder a la sección crítica tendrán que esperar a que se libere el bloqueo.

Sistemas de Gestión de Bases de Datos

Oracle

Oracle ofrece un completo paquete de herramientas y aplicaciones para desarrollar, desplegar, gestionar y poner a punto aplicaciones que requieran una alta disponibilidad. Una discusión completa de estas herramientas se extendería durante varios capítulos, por este motivo, esta sección se centra únicamente en explicar algunas de las características de alta disponibilidad ofrecidas:

- Clústeres reales de aplicaciones Oracle.
- Recuperación transparente de fallos de aplicación.
- *Oracle Data Guard*
- Replicación Avanzada.
- Combinación de múltiples soluciones.

Clústeres reales de aplicaciones Oracle

Oracle RAC (*Oracle Real Application Clusters*) es una configuración de base de datos que consiste en múltiples instancias de la base de datos con acceso compartido al mismo conjunto de archivos de datos. Como su predecesor, RAC está configurado en combinaciones de hardware/SO específicas capaces de soportar un entorno de clúster.

El Oracle RAC es una combinación de un número de distintos componentes hardware y software. La lista siguiente detalla los componentes individuales y sus funciones:

- **Hardware del servidor clúster**: El hardware físico en el que se aloja la base de datos RAC. Esto comprende dos o más nodos uniprocesador o multiprocesador conectados en una configuración clúster con una interconexión de alta velocidad para la comunicación entre nodos.
- **Discos compartidos del clúster**: Los discos compartidos en los que se almacenan los archivos de la base de datos. Los discos compartidos deben permitir accesos directos al disco simultáneos desde todos los nodos del clúster. Las configuraciones de discos compartidos pueden ser SAN (*Store Area Network*), RAID compartidos o NAS (*Network Attached Storage*), pudiendo conectarse tanto directamente a los nodos del clúster como a una interconexión de alta velocidad.

- **Gestor del clúster**: Las funciones OSD (*Operating System Dependent*) para el control de la funcionalidad del clúster. Esto se compone de dos módulos distintos. Primero, el demonio *Global Services* ejecuta tareas en nombre del software RAC, como el cierre, inicio y otras acciones de tipo administrativo en los nodos individuales del clúster. Segundo, el *Node Monitor* es responsable de mantener la comunicación entre los nodos del clúster y el software RAC. Esto incluye supervisión del estado de los nodos del clúster y el paso de esta información al software RAC.
- **Servicio global de cahé/colas**: Controla el acceso a los recursos de todos los nodos que forman parte del RAC. Durante las operaciones de bases de datos en las que debe accederse al mismo recurso desde múltiples sesiones, el servicio *Global Cache/Enqueue* se encarga de coordinar las operaciones entre los nodos del clúster. Operaciones como el acceso a bloques de datos en caché o la ejecución de operaciones de bloqueo, deben producirse de forma transparente independientemente de la sesión a la que esté asociado el nodo. Se utiliza un esquema conocido como dominio de recurso para permitir que la propiedad de un recurso ``flote" entre los nodos, dependiendo de dónde sea óptimamente referenciado.
- **Fusión de caché**: Una de las capacidades más poderosas de RAC. Utilizando los servicios globales de caché y colas, la fusión de cachés permite que las sesiones conectadas a un nodo del clúster hagan referencia a bloques de bases de datos que están en la memoria caché de un nodo distinto. Cuando se solicita la lectura de un bloque de base de datos, si el bloque de datos referenciado ya está en la caché de uno de los nodos del clúster, Oracle usa la interconexión de alta velocidad para copiar el bloque entre los nodos. Cuando se solicita la escritura de un bloque en la base de datos, si el bloque ya está en la caché de uno de los otros nodos del clúster, Oracle de nuevo copia los datos entre los nodos pero ahora la propiedad se asigna al nodo que está ejecutando la actualización del bloque de datos. Las operaciones de lectura adicionales sobre el bloque pueden continuar en otros nodos, pero cualquier escritura del bloque en base de datos se verá forzada a esperar hasta que la primera sesión que está actualizando efectúe una confirmación (commit) o un rechazo (rollback).

Siempre que se produce un fallo en uno de los nodos del clúster, se ponen en marcha las siguientes operaciones:

- El demonio de *Global Services* efectúa una **reorganización del clúster** que implica determinar qué nodos ya no forman parte del clúster. Esto se logra indicando a cada instancia de base de datos que actualice su entrada en el mapa de miembros contenido en el archivo de control. Al final del proceso, las entradas sin actualizar del mapa de pertenencia se juzgan como nodos con fallos.
- La **recuperación de la base de datos** implica determinar qué bloques de caché y bloqueos de objetos de la base de datos pertenecen a la instancia que ha fallado. Asimismo, es necesario reasignar la propiedad de los recursos.
- La **recuperación de transacción** conlleva la lectura de los archivos de registro para rehacer el nodo que ha fallado, para determinar las transacciones que deben aplicarse y las transacciones incompletas que deberían ser deshechas. Dependiendo de la cantidad de recuperación que deba efectuarse, este proceso puede tomar una cantidad de tiempo excesivo. La característica *fast-start-rollback* le permite ajustar la duración máxima del proceso de recuperación y facilita que la recuperación se efectúe en paralelo.

Las dos configuraciones de alta disponibilidad más simples que se pueden crear utilizando RAC usan un clúster de dos nodos con una configuración primario/secundario. Este método usa una base de datos primaria activa que da servicio a la mayoría de las solicitudes de conexión de la aplicación. La base de datos secundaria normalmente no responde a las solicitudes de la aplicación, aunque con la configuración de red apropiada, la instancia secundaria puede conectarse directamente para operaciones como el mantenimiento o los informes por lotes. En caso de fallo de la instancia primaria, la secundaria toma el control efectuando la devolución de transacción y recuperación de instancia. Todos los intentos de conexión siguientes son entonces dirigidos a la instancia secundaria. Los dos tipos de configuración RAC primaria/secundaria son:

- **Oyente dedicado básico de clúster real de aplicación** es una configuración en la que cada base de datos tiene su propio oyente. La aplicación se configura para conectar con el oyente de la primera base de datos. Cuando una solicitud de conexión no puede ser satisfecha por la primera base de datos, la aplicación conecta con la segunda base de datos. En ese caso, todas las transacciones que se encontraban en curso son canceladas. La figura muestra la configuración de oyentes para la configuración de oyente dedicado básico RAC.

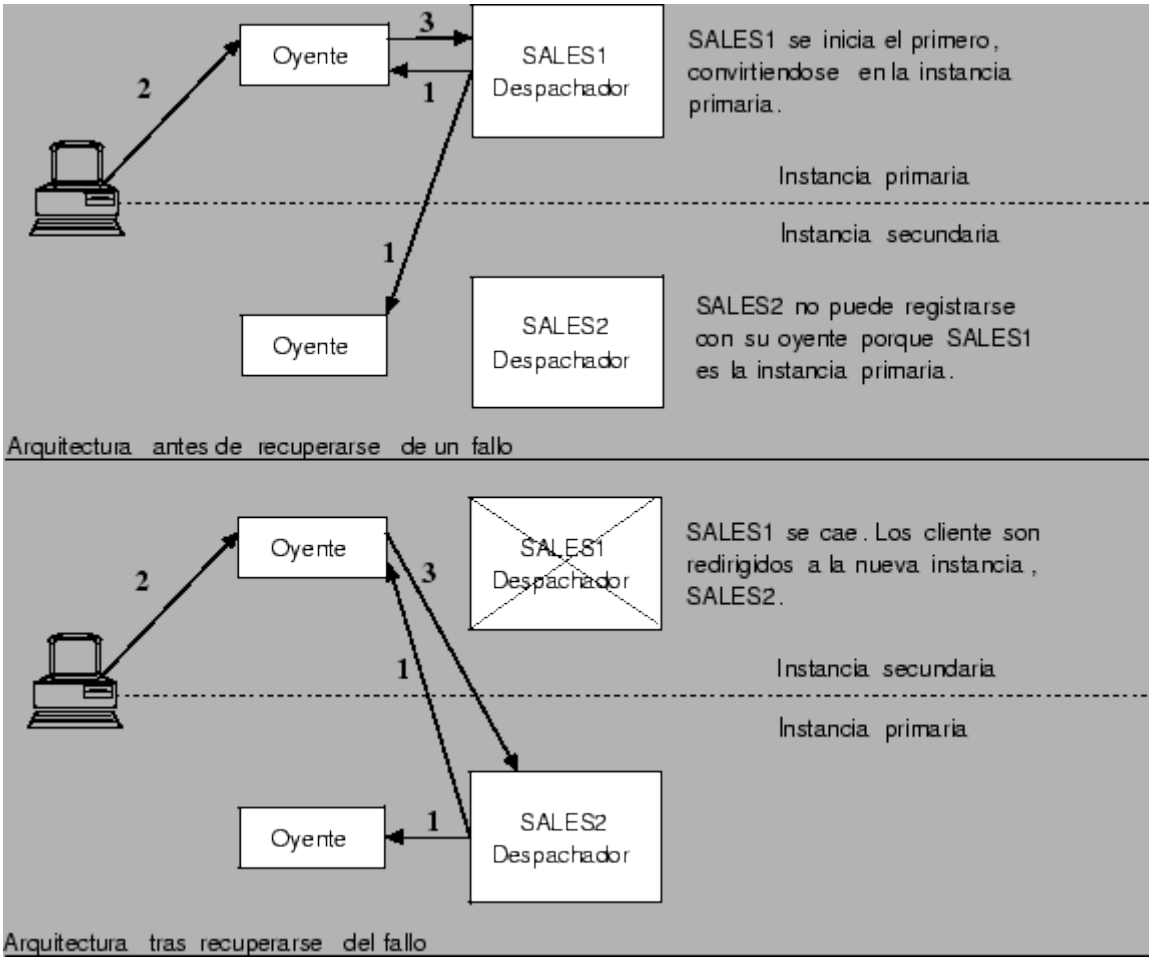


Figura : Oyente dedicado básico RAC

- **Oyente compartido básico de clúster real de aplicación**. Una configuración en la que las bases de datos comparten uno o más oyentes. Cuando se inicia la base de datos primaria, se registra con cada uno de los oyentes. La aplicación entonces se configura para conectar con cualquiera de los oyentes y las solicitudes son enviadas a la instancia primaria. Cuando la instancia primaria falla, la instancia secundaria efectúa la recuperación, se registra como

instancia primaria y acepta todas las conexiones siguientes. Cualquier transacción que estuviese en proceso cuando falló la instancia primaria es cancelada y la aplicación tiene que reconectar. La figura muestra la configuración de oyentes para la configuración oyente compartido básico RAC.

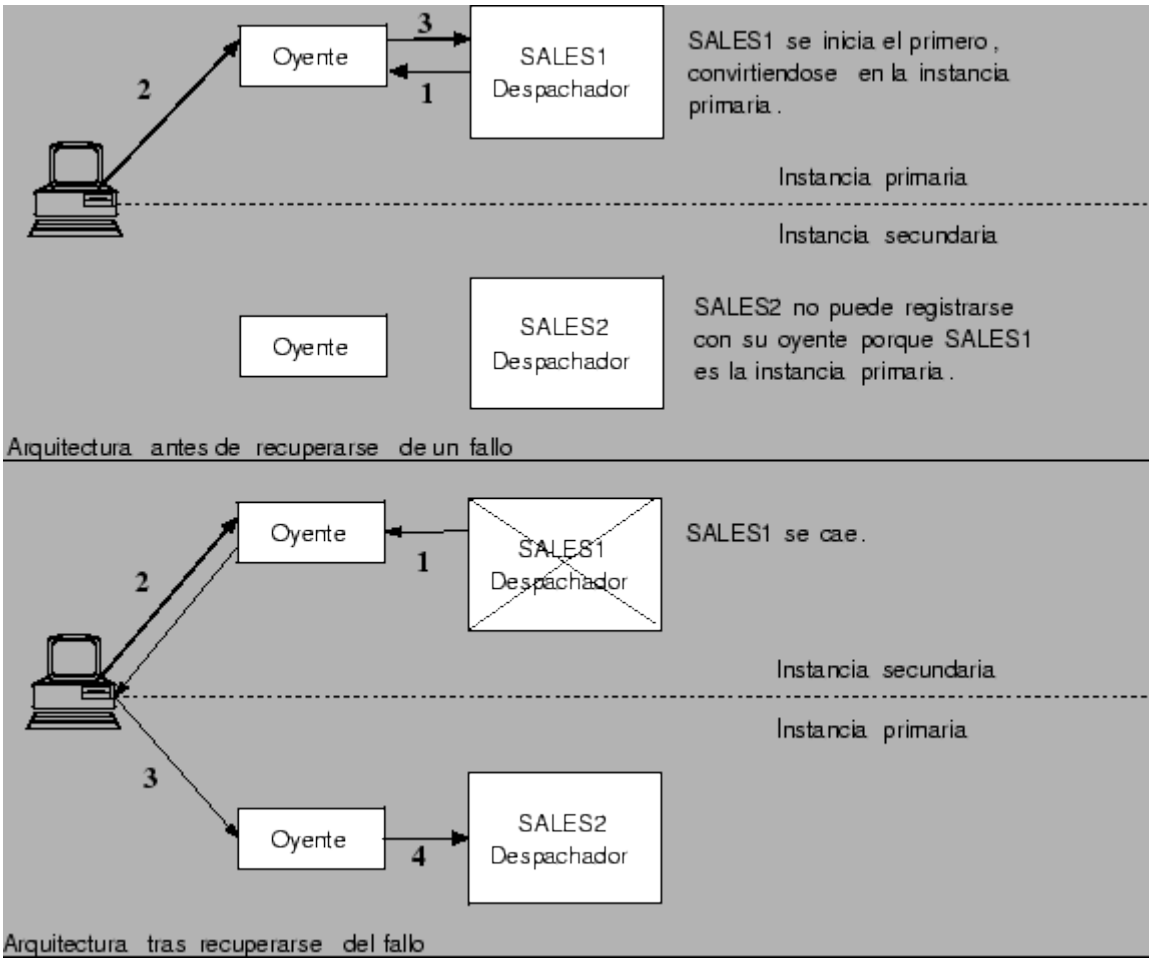


Figura : Oyente compartido básico RAC

Ventajas de los clústeres reales de aplicaciones

Los RAC (*Real Application Clusters*) ofrecen las siguientes ventajas:

- Son mínimos los cambios requeridos en la aplicación para aprovechar las características ofrecidas por la arquitectura del RAC.
- Crear una solución de alta disponibilidad es extremadamente fácil usando el RAC de dos nodos.
- En una configuración primario/secundario, la recuperación de instancias tras el fallo de un nodo es automática, mejorando así el tiempo medio de recuperación o MTTR (*mean time to recover*).
- Combinando el RAC con otras características, como *Oracle RAC Guard*, recuperación transparente de fallos de aplicación y *Oracle Data Guard* se puede crear una arquitectura extremadamente potente, altamente escalable así como flexible.

Desventajas de los clústeres reales de aplicaciones

Los RAC *Real Application Clusters*) tienen las desventajas que se exponen a continuación:

- Al compartir el subsistema de disco todos los nodos participantes, la configuración es todavía débil a los fallos de disco a menos que se den pasos específicos para prevenirlos. Estos pasos incluyen espejos por hardware.
- Dado que la interconexión de alta velocidad es el punto central de comunicación entre los nodos del clúster, a menos que se use redundancia puede convertirse en un punto de posibles fallos.
- RAC puede proteger tan sólo contra fallos localizados en caídas de los nodos del clúster. Por sí mismo, no ofrece ninguna redundancia remota como la que es necesaria para prevenir desastres como la pérdida completa de un centro de datos.

Pueden conseguirse mejoras adicionales de alta disponibilidad y escalabilidad combinando RAC con otras posibilidades de Oracle.

Recuperación transparente de fallos de aplicación

Una de las consideraciones de diseño más importantes de cualquier aplicación es la transparencia para el usuario final que nunca debería conocer la arquitectura de la aplicación. Desde la perspectiva del usuario, las interrupciones frecuentes del servicio pueden resultar frustrantes. Incluso la más elegante solución de alta disponibilidad puede parecer molesta a los usuarios si requiere que tengan que reconectar y reiniciar su trabajo.

TAF (*Transparent Application Failover*) es un componente de *Oracle Networking* que puede utilizarse para minimizar la interrupción causada por temas relacionados con conectividad. Cuando se pierde la conexión entre la aplicación y la base de datos, TAF ejecuta las siguientes funciones dependiendo de como haya sido configurado:

- **Restauración de sesiones de base de datos**: Se restaura la conexión a la base de datos usando el mismo identificador de usuario y clave utilizados previamente.
- **Continuación de operaciones SELECT**: Las sentencias SELECT emitidas previamente vuelven a ejecutarse, continuando a partir del mismo punto en que se produjo la desconexión.

Puesto que TAF no restaura transacciones dudosas, las variables de sesión de usuario perderán su estado tras la recuperación del fallo. Si se requiere transparencia completa en la aplicación, sin embargo, puede combinarse una composición de funciones de retorno para recuperación de fallos con la escritura de valores en una tabla temporal en la base de datos para restauración tras la recuperación del fallo.

A pesar de no ser estrictamente una característica de TAF, un componente adicional de *Oracle Networking* que puede utilizarse para múltiples instancias de bases de datos, como con RAC, para mejorar la transparencia de la aplicación, es el balanceo de cargas. Esto permite que las conexiones a la aplicación se repartan entre todas las instancias de bases de datos disponibles. El balanceo de cargas puede usarse también con bases de datos replicadas y bases de datos de reserva de *Oracle Data Guard*. Cuando se combina con la reconexión automática de TAF, esto ofrece una mejora significativa de alta disponibilidad minimizando la interrupción de sesiones.

Ventajas de la recuperación transparente de fallos de aplicación

TAF ofrece las siguientes ventajas:

- Las aplicaciones pueden reconectarse transparentemente a la base de datos sin intervención del usuario.
- Las sentencias SELECT previamente ejecutadas continúan donde se quedaron.
- TAF puede combinarse con el balanceo de cargas para mejorar la escalabilidad y disponibilidad de forma significativa.

Desventajas de la recuperación transparente de fallos de aplicación

TAF tiene las siguientes desventajas:

- No es posible la restauración de toda la información de sesión.
- Las transacciones previas son deshechas.

Oracle Data Guard

ODG (*Oracle Data Guard*) ofrece una solución integrada para la configuración y gestión de bases de datos de reserva. Una implementación de base de datos de reserva consiste en una o más bases de datos que son copias separadas de una base de datos en producción. A diferencia de las bases de datos RAC, las bases de datos de reserva pueden situarse en la misma localización dentro del mismo centro de datos que la base de datos en producción, localizarse remotamente en una red de área ancha (WAN) o una combinación de ambos. Algunos de los usos de las bases de datos de reserva incluye el mantenimiento de bases de datos separadas para informes, una base de datos fuera de sede para recuperación de desastres o una copia de la base de datos de producción usada para desarrollo o prueba de aseguramiento de la calidad.

La lista siguiente incluye algunos de los muchos componentes que son parte de la arquitectura ODG.

- **Base de datos primaria**: La base de datos en producción que es la fuente de datos para la instancia de reserva. Una sola base de datos primaria puede tener múltiples bases de datos de reserva.
- **Base de datos de reserva**: La base de datos que es copia de la base de datos primaria en producción.
- **Servicios de registro**: El método por el que los registros de rehacer son transferidos desde la base de datos primaria a las bases de datos de reserva. También controlan la frecuencia con que los archivos de registro para rehacer se aplican a las bases de datos de reserva.
- ***Data Guard Broker***: Es el componente software que se encarga de la creación, control y gestión de la configuración de base de datos primaria/secundaria.
- **Sede de *Data Guard***: La colección de una base de datos primaria y hasta nueve bases de datos de reserva.

Una base de datos de reserva puede crearse utilizando tanto el componente *Data Guard Manager* de *Oracle Enterprise Manager* como herramientas desde la línea de comandos. Crear y gestionar una sede ODB implica estos pasos:

- **Configurar la base de datos primaria**: La base de datos primaria debe estar funcionando en modo `ARCHIVELOG` y la misma plataforma hardware, versión de SO y versión del software RDBMS que el usado para la base de datos de reserva.
- **Determinación del modo de protección**. Cuando cree una base de datos de reserva, considerar qué modo de protección es el apropiado, podría ser importante. ODG permite utilizar un número de distintos modos de protección.
    - **Modo de protección garantizada**: Asegura que todos los cambios sean propagados a la sede de la base de datos de reserva después de que hayan sido confirmados en la base de datos primaria.
    - **Modo de protección instantánea**: Asegura que todos los cambios se propaguen a la base de datos de reserva tras ser confirmados en la base de datos primaria a menos que la conectividad de la red lo impida. Una vez que la conectividad de red se haya restablecido, los registros para rehacer se aplican para conseguir que la base de datos de reserva esté actualizada.
    - **Modo de protección rápida**: Procura que los cambios se propaguen a la base de datos de reserva tan pronto como sea posible, sin causar una degradación del rendimiento en la base de datos primaria.
    - **Modo de protección demorada**: Asegura la propagación de todos los cambios a la base de datos de reserva una vez ha transcurrido un pequeño periodo de tiempo. Con la espera del modo de protección demorado, el retraso en la propagación puede especificarse para permitir que la base de datos quede detrás de la base de datos en producción.
- **Determinar el modo de operación de reserva**: Utilizado para controlar la frecuencia con que los cambios se aplican a la base de datos de reserva tras la propagación desde la base de datos primaria. La base de datos puede estar funcionando en uno de los dos modos siguientes:
    - **Modo de recuperación gestionada**: Permite a la base de datos de reserva aplicar todos los cambios tan pronto como se propaguen a la sede de reserva. La base de datos no está disponible para uso, sin embargo, hasta que ha sido abierta.
    - **Modo de sólo lectura**: Permite que la base de datos permanezca abierta sólo para consultas de sólo lectura. Sin embargo, no se aplican cambios a la base de datos hasta que el modo se cambia a modo de recuperación gestionada.

La figura muestra de forma global la arquitectura de ODG.

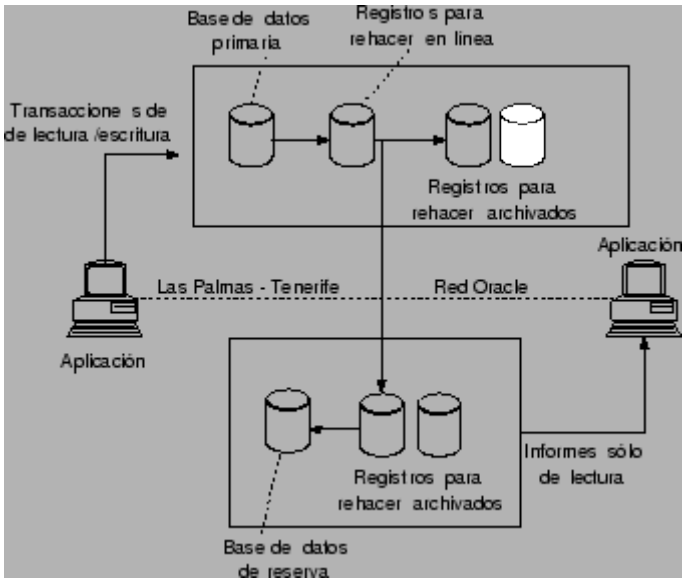


Figura: Arquitectura de *Oracle Data Guard*

Una vez se han creado y configurado las bases de datos de reserva, el *broker* ODG gestiona y supervisa la propagación y aplicación de registros para rehacer, dependiendo del modo de protección seleccionado. Ya que ODG permite la creación de múltiples instancias de reserva, la mejor configuración de protección podría ser tener un número de bases de datos de reserva, cada una de ellas con un modo de protección distinto.

Si la base de datos primaria queda innaccesible por problemas de hardware en el nodo de la base de datos primaria, puede iniciarse una recuperación del fallo con la base de datos de reserva. La cantidad de datos perdidos durante la operación de recuperación del fallo dependerá del modo de protección elegido. Una vez se haya producido la recuperación del fallo, la base de datos de reserva asume el papel de base de datos primaria.

Una vez resuelto el problema de hardware en el nodo de la base de datos primaria, volver a la configuración original requiere volver a crear una instancia de la base de datos copiando de nuevo la base de datos y reconfigurando el nodo de reserva con su estado original.

ODG permite el inicio manual de un intercambio desde la base de datos primaria a la base de datos de reserva sin necesidad de reinstanciar cuando se produzca la vuelta a la situación original. Esto puede ser útil para efectuar mantenimiento en los nodos de bases de datos.

Ventajas de Oracle Data Guard

ODG ofrece las siguientes ventajas:

- Puede ser utilizado conjuntamente con RAC y la recuperación transparente de fallos de aplicación.
- Dependiendo del modelo de protección elegido, el intercambio y recuperación de errores puede ocurrir con poca o ninguna pérdida de información.
- La configuración y administración se simplifican con *Data Guard Manager*.
- *Data Guard* ofrece recuperación de desastres si la base de datos de reserva se mantiene en una localización remota.
- *Data Guard* es flexible puesto que la base de datos de reserva puede mantenerse automáticamente o de forma manual mediante entradas de usuario y/o guiones.

- Las necesidades de ancho de banda son menos intensivas comparadas con otras soluciones de alta disponibilidad ya que sólo se propagan los archivos de registro guardados desde la base primaria a la que está en reserva.

Desventajas de *Oracle Data Guard*

ODG tiene las siguientes desventajas:

- Aunque ODG puede configurarse para funcionar en nodos locales, no ofrece la misma escalabilidad que los Oracle RAC.
- Dependiendo del modo de protección elegido, el intercambio y recuperación ante fallos pueden necesitar una cantidad de tiempo significativa.
- La frecuencia de la propagación depende mucho del ancho de banda disponible en la red. En un entorno con un significativo volumen de cambios en la base de datos, la cantidad de cambios propagados puede provocar un excesivo tráfico en la red.
- No existe un mecanismo fácil de vuelta atrás para volver a la máquina primaria original una vez que se ha activado la de reserva. La base de datos de reserva debe reconstruirse como si fuera la primaria e iniciarse una recuperación de fallo en la primaria original (ahora la de reserva).
- Mientras la base de datos de reserva está abierta en modo sólo de lectura, no puede mantenerse sincronizada con la base de datos primaria (esto es, no pueden aplicarse los archivos de reconstrucción guardados mientras está abierta).
- Los datos no presentes en los registros de reconstrucción guardados (como las tablas e índices creados mediante la opción `NOLOGGING/UNRECOVERABLE`) no se propagan a la reserva. Por ello, los comandos que desconectan la reconstrucción deben evitarse o tienen que implementarse medidas explícitas para registrar y propagar manualmente esos comandos. La implementación de un procedimiento robusto de parcheo de base de datos puede ayudar a través de los registros de reconstrucción archivados si no se detecta y rectifica a tiempo.
- La base de datos de reserva debe estar funcionando en el mismo hardware, versión de SO y versión de DBMS que la base de datos principal.

Replicación avanzada

AR (*Advanced Replication*) es un sofisticado mecanismo de replicación disponible en Oracle. AR permite que una o más bases de datos origen, con ciertos esquemas predeterminados, e incluso con segmentos específicos dentro de cada base de datos sean replicadas a una o más bases de datos destino en un esquema de replicación de sentido único o múltiple. Así, tanto las base de datos de origen como las de destino pueden gestionar de forma concurrente las lecturas y escrituras. Desde una perspectiva lógica, AR ofrece una base de datos de reserva ``en caliente".

A diferencia de un escenario convencional de base de datos de reserva, la base de datos destino está abierta y disponible para recuperación inmediata ante un fallo de la base de datos origen (ver figura). Idealmente, el destino es mantenido en una máquina separada en una situación remota para propósito de recuperación de desastres.

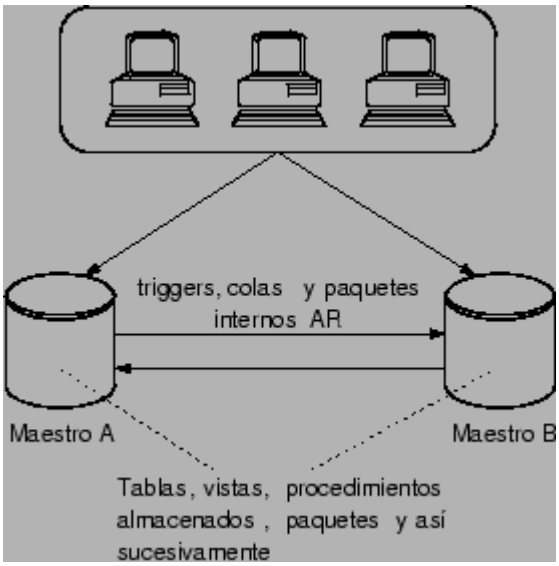


Figura: Arquitectura de replicación avanzada

Bajo AR, las copias múltiples de bases de datos o subconjuntos específicos son escritas concurrentemente y se mantienen sincronizadas casi en tiempo real. Una configuración así se denomina configuración multi-maestro (porque existen múltiples copias maestras). Si no se desea sincronización inmediata, puede tomarse un modelo dirigido por eventos para propagar las transacciones de un maestro a los otros. Por ejemplo, la propagación puede estar basada en tiempo, produciéndose durante horas específicas de poco uso. Si se está usando la replicación para propósitos de recuperación de fallos, sin embargo, la sincronización debería ser inmediata. Por ello, cada copia de la base de datos debería de mantenerse de manera *peer-to-peer*. Todas las escrituras inicialmente se almacenan de forma local, y luego se remiten a cada base de datos de destino utilizando el mecanismo *push*, en oposición a la replicación simple de instantáneas que se obtiene al tirar (*pull*) de los datos desde la base de datos origen.

Cada transacción se propaga de forma consistente para prevenir violaciones de la integridad de los datos. Si se producen violaciones o conflictos de integridad, se llevan a cabo los esquemas específicos para resolución de conflictos. Éstos pueden producirse por una variedad de razones. Por ejemplo, si distintos usuarios efectúan cambios en la misma fila en cada base de datos, existirá un conflicto. Una violación de clave única es otro ejemplo de estos conflictos. Los conflictos deben ser detectados y resueltos amigablemente. AR ofrece potentes algoritmos para la detección de conflictos y su resolución. La resolución de conflictos puede ser consistente o variar al nivel de segmento (tabla) o incluso columna. Hay disponibles múltiples técnicas de resolución de conflictos, como utilizar el último de los cambios, usar el primero de los cambios, utilizar los cambios específicos a una cierta sede/base de datos, etc.

A medida que las transacciones van remitiéndo, si una base de datos destino específica no está disponible, las transacciones son retenidas en el origen en la cola diferida. Cuando la base de datos de destino está disponible otra vez, se aplican en ella las transacciones. La falta de disponibilidad de una o más bases de datos destino no evitan que las transacciones sean propagadas a las restantes copias de la base de datos.

Tanto las sentencias DML como las DDL se propagan entre todas las maestras. AR tradicionalmente ha usado desencadenadores (*triggers*), colas asíncronas y varias tablas periódicas para implementar

distintos esquemas de replicación. La replicación basada en desencadenadores es efectiva en ciertos escenarios.

Tradicionalmente, el enfoque basado en desencadenadores en AR es disuasorio, evitando que las sedes con una fuerte actividad puedan usarlo eficientemente sin una degradación significativa del rendimiento. AR en Oracle usa desencadenadores y paquetes internos. Los desencadenadores y paquetes internos son módulos de código en C enlazados en el núcleo de Oracle. Esto hace el código relativamente más a prueba de manipulación (con seguridad elevada), así como ligero y eficiente, permitiendo que la implementación sea rápida y escalable. No se configuran ni mantienen componentes externos. Al no generarse paquetes ni disparadores, pueden ser instanciados y ejecutados con mayor rapidez. La propagación de datos es manejada mediante el protocolo de flujo directo, en lugar de mediante el igualmente fiable, pero menos eficaz, protocolo de confirmación en dos fases.

En el caso de AR, durante el fallo de una copia específica de la base de datos, la causa del fallo debe determinarse y la base de datos recuperarse/reonstruirse desde la última copia o desde salvaguardas. Como se indicó previamente, existe la posibilidad de una pérdida de datos (las escrituras más recientes) durante ese fallo. Siempre que una copia específica de la base de datos falla, las otras copias continúan funcionando a menos que se produzca un desastre mayor en cuyo caso todas las copias existentes son descartadas. Las posibilidades de que esto ocurra son remotas. Durante estas situaciones, el intervalo de fuera de servicio es significativo porque la base de datos tiene que reconstruirse usando la última salvaguarda disponible. Todo el tráfico de clientes/usuarios tiene que redirigirse a través de TAF o Net8 a copias supervivientes de la base de datos.

Ventajas de la replicación avanzada

AR ofrece las siguientes ventajas:

- Todas las bases de datos mantenidas por AR pueden mantenerse abiertas y en uso concurrente.
- AR puede utilizarse para recuperación de desastres teniendo una copia de la base de datos mantenida en una sede remota.
- Existe independencia de hardware ya que las máquinas en que se crean las bases de datos pueden ser de distintos fabricantes, plataformas y versiones de SO.
- Se han efectuado mejoras a *Oracle Enterprise Manager* para simplificar la configuración y administración de bases de datos replicadas.

Desventajas de la replicación avanzada

AR tiene las siguientes desventajas:

- La propagación podría no ser siempre inmediata. Los cambios de datos en una base de datos pueden no ser visibles inmediatamente en las otras bases de datos.
- Durante desastres, existe una posibilidad de alguna pérdida de datos porque la base de datos de una sede puede destruirse antes de que los últimos cambios se hayan propagado. Si la base de datos de una sede falla, pero no es destruida, la pérdida de datos pordría ser temporal y todas las copias de la base de datos pueden sincronizarse una vez que la base de datos se recupere del fallo.

- Los mecanismos de resolución de conflictos deben ser bien planificados y tener en cuenta escenarios en los que puede escribirse directamente en las copias de la base de datos para prevenir corrupción lógica de los datos.

Combinación de soluciones

Cada una de las opciones de alta disponibilidad ofrece protección contra ciertos tipos de fallos. A veces, la mejor opción es combinar dos o más de las soluciones y prevenir una amplia variedad de situaciones causantes de paros. Por ejemplo, usando RAC nos protegemos contra fallos de nodos locales solamente, pero usándolo con ODG obtenemos una protección adicional contra escenarios de desastres, como una pérdida completa del centro de datos.

Microsoft SQL Server

Disponibilidad es un término que describe los períodos que está operativo un determinado sistema informático. Así se pueden determinar las horas en las que se espera que los usuarios puedan tener acceso al sistema, teniendo en cuenta los períodos programados de no disponibilidad debidos al mantenimiento o actualización del sistema.

Una de las tareas clave en la administración de un sistema o aplicación consiste en tomar todas las precauciones para reducir al mínimo el tiempo de parada no programado. Los cortes de servicio no programados provocan la frustración de los usuarios, pueden reducir su productividad y, en algunos casos, pueden tener un efecto significativo en la marcha de la empresa. Por ejemplo, en el caso de una solución de comercio electrónico, la falta de disponibilidad producirá, casi con certeza, la pérdida de ingresos y, si se produce con frecuencia, puede hacer que los clientes dejen de acudir al sitio.

Los aspectos de disponibilidad se suelen tratar, a menudo, junto con las posibilidades de escalabilidad de una solución. Mientras que la disponibilidad trata de resolver el problema que supone proporcionar acceso a la aplicación dutante un período aceptable, la escalabilidad está relacionada con el problema asociado a proporcionar acceso simultáneo a un número aceptable de usuarios.

Hay dos métodos principales para aumentar la escalabilidad: el uso de procesadores más rápidos o el uso de un mayor número de procesadores.

Escalabilidad

Los métodos de crecimiento tratan de aumentar el número de usuarios simultáneos a los que puede dar servicio un servidor individual. Normalmente, para conseguir esto se agregan nuevos recursos hardware al servidor.

El aumento de la memoria RAM instalada en un servidor puede mejorar mucho su capacidad para procesar con mayor rapidez las solicitudes que recibe. A menudo, este sistema sencillo y relativamente económico puede mejorar el rendimiento y la escalabilidad del sistema.

El multiproceso permite ejecutar en paralelo varios subprocesos. La capacidad de *Microsoft Windows 2000* para usar multiproceso simétrico (SMP) permite instalar varios procesadores en el equipo.

Un método alternativo para aumentar la escalabilidad consiste en distribuir la carga de proceso entre varios servidores. Puede usar un sistema denominado balanceo de carga para asegurarse de que las tareas de proceso se distribuyen por igual entre todos ellos.

Hay varios sistemas para aumentar la escalabilidad de las soluciones de *SQL Server* por medio de la instalación de servidores adicionales. Los sistemas de replicación y los servidores de reserva mejoran la escalabilidad y la disponibilidad, mientras que las federaciones de servidores suponen una mejora en cuanto a escalabilidad, aunque disminuyen la disponibilidad.

La replicación puede usarse para distribuir los datos entre varios equipos de la empresa donde se ejecuta *SQL Server*. Esto permite que los usuarios muy alejados geográficamente puedan tener acceso a una aplicación de base de datos sin que necesiten una conexión permanente con la instalación central. Este sistema permite mejorar la escalabilidad global de una solución ya que distribuye la carga de proceso.

También puede mejorar la disponibilidad ya que permite que los usuarios trabajen con sus copias locales de datos, aun en el caso de que la base de datos central no esté disponible.

Los servidores de reserva de solo lectura son copias exactas de los servidores de producción de la base de datos. Se puede usar un servidor de reserva como origen de datos de sólo lectura para la generación de informes y otras operaciones de sólo lectura, con lo que se descarga de parte del trabajo del servidor de producción y se mejora la escalabilidad. También mejora la disponibilidad, ya que, en el caso de que el servidor de producción quede fuera de servicio, el servidor de reserva puede tomar su lugar.

Las federaciones de servidores están formadas por varios equipos con *SQL Server* donde cada uno contiene un subconjunto de una tabla de datos. Se pueden consultar y actualizar las tablas por separado por medio de una vista dividida y actualizable que muestra al usuario una única tabla virtual. Este sistema mejora la escalabilidad aunque tiene un efecto desfavorable en la disponibilidad ya que introduce varios puntos donde se pueden producir errores.

Aumentar la disponibilidad mediante Microsoft .NET Enterprise Server

Microsoft .NET Enterprise Server es una familia completa de aplicaciones de servidor que permite crear, distribuir y administrar soluciones Web integradas y escalables. Una aplicación se divide en tres capas o más, y se reparte entre varios equipos. Cada capa se puede mantener y configurar de forma independiente, lo que permite un alto grado de flexibilidad en cuanto al diseño y administración de la aplicación. Hay varias formas de optimizar la disponibilidad en cada una de las capas de la aplicación.

Figura: Modelo de 3 capas

Capa de presentación

La capa de presentación proporciona los servicios de software que permiten a los usuarios usar la aplicación. Puede tratarse de una aplicación Windows instalada en el equipo del usuario o una aplicación Web a la que tienen acceso los usuarios por medio de un explorador.

Una aplicación Windows instalada de forma local estará disponible siempre mientras la instalación no sufra ningún daño. En una red Windows 2000, puede usar *Windows Installer* para lograr que las instalaciones dañadas se reparen automáticamente y sea posible instalar a petición componentes adicionales.

Para mejorar la disponibilidad de una aplicación Web, puede distribuir el sitio Web en varios servidores. Puede usar balanceo de carga en la red Windows 2000 para crear un grupo de servidores que responda dinámicamente a una petición de cliente en función de la dirección IP de origen. En este caso se asigna a cada servidor del grupo un conjunto de direcciones IP a las que debe responder. Si se produce un error en el servidor, los restantes servidores se reparten las direcciones IP que quedan desatendidas, con lo que el sitio sigue estando disponible.

Capa de lógica empresarial

La capa de lógica empresarial contiene el software que implementa las reglas peculiares de una empresa en una aplicación. Normalmente, esta capa está compuesta por componentes del modelo de objetos componentes (COM, *Componente Object Model*) que contienen las reglas de la empresa.

Se puede lograr la máxima disponibilidad de la capa de lógica empresarial con grupos de servidores divididos en clústeres y administrados mediante *Application Center 2000*. Además, *Application Center 2000* proporciona balanceo de de la carga de componentes (CLB, *Component Load Balancing*), una de las características de Servicios de componentes, lo que permite distribuir los componentes en varios servidores y equilibrar de forma dinámica su carga en función de la disponibilidad y trabajo de cada uno.

En el caso de procesos asincrónicos de empresa que no precisen de una respuesta inmediata, se pueden usar los componentes de colas de los servicios de componentes para que sea posible el acceso a la lógica empresarial aunque el servidor de base de datos no esté disponible.

Capa de datos

En una aplicación de tres capas, las bases de datos SQL Server forman parte de la capa de datos. La disponibilidad de esta capa se puede mejorar de varias formas.

En el caso de una instalación de servidor único, se pueden emplear soluciones con tolerancia a fallos mediante tecnología RAID. Windows 2000 es compatible con RAID nivel 1 (discos espejo) y RAID nivel 5 (creación de bandas de disco con paridad). Ambos sistemas se pueden usar para garantizar la disponibilidad de la base de datos completa incluso en el caso de un error de disco.

Si hay más de un servidor instalado, se puede usar la replicación de datos para hacer que la base de datos esté disponible en varios puntos alejados o, incluso, al alcance de usuarios sin conexión. Así se permite que los usuarios usen la aplicación de base de datos cuando no se puede establecer una conexión con el servidor de publicación.

Para alcanzar un altísimo grado de disponibilidad se puede usar la organización por clústeres con conmutación por error. Este sistema se basa en las capacidades del uso de clústeres de Windows 2000 y permite crear un servidor virtual formado por varios equipos físicos. En el caso de que se produzca un error en un servidor, los restantes servidores del clúster hacen que la base de datos siga estando disponible.

Finalmente, se puede usar un servidor de reserva que contenga una copia completa del servidor de producción de la base de datos, que se podrá comenzar a utilizar en el caso de que se produzca un error en el servidor. Este servidor de reserva se mantiene sincronizado con el servidor de producción mediante un sistema denominado trasvase de registros en el que los registros de transacciones del servidor de producción se aplican al servidor de reserva.

Clústeres con conmutación por error

Un clúster está formado por dos o más equipos independientes que trabajan conjuntamente como si fueran un único equipo. La organización por clústeres constituye una eficaz forma de aumentar la disponibilidad ya que permite la conmutación en caso de error. Si un miembro del clúster tiene un error, los restantes miembros garantizan que las operaciones no se vean afectadas.

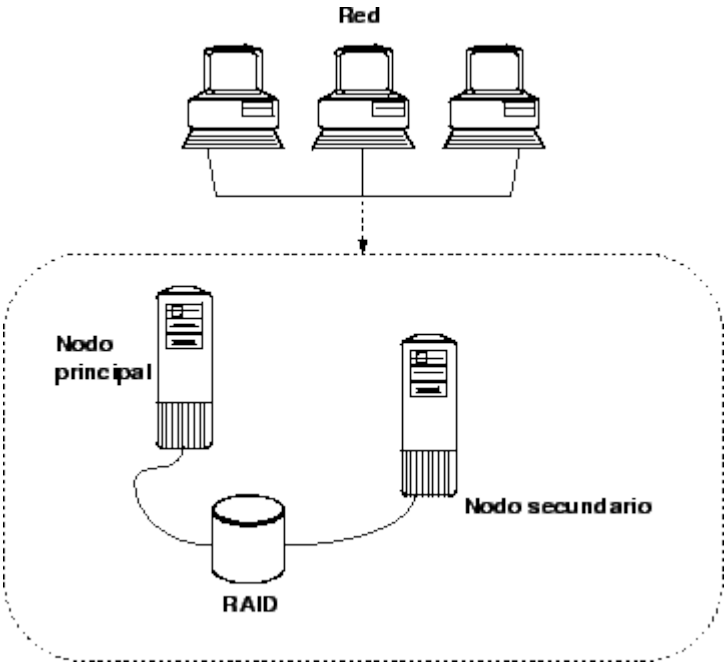


Figura: Organización por clústeres de Windows

Componentes de un clúster

Tenga en cuenta los siguientes hechos acerca de la organización por clústeres de Windows:

- Los servidores miembros de un clúster se conocen como *nodos*.
- Los nodos de un clúster deben estar vinculados entre sí por medio de un vínculo de comunicación de red independiente.
- Cada nodo debe disponer de una conexión a uno o varios discos compartidos (RAID), donde se almacenan los datos del clúster.
- Los recursos, como aplicaciones o unidades lógicas, se combinan en grupos de recursos a los que se asigna un nodo principal. Este nodo es el predeterminado para el acceso a esos recursos

Conmutación por error

Los nodos de un clúster usan software de organización por clústeres para mantener la disponibilidad y realizar la conmutación en caso de error. El Monitor de recursos realiza la comunicación entre el Servicio de clúster y los recursos de las aplicaciones en el clúster. El Servicio de clúster controla la comunicación entre los nodos y realiza la conmutación en caso de error.

En el caso de que haya un error en una aplicación debido a un error del equipo, el Servicio de clúster tratará de reiniciar la aplicación. Si no lo consigue, reiniciará la aplicación en otro nodo del clúster, al que moverá todos los recursos de esa aplicación.

Si se produce un error en un nodo, el Servicio de clúster reparte automáticamente el trabajo que realizaba ese nodo entre los restantes miembros del clúster.

Clústeres con conmutación por error

La organización por clústeres con conmutación por error de SQL Server está diseñada para que trabaje conjuntamente con la organización por clústeres de Windows. Cuando se instala la organización por clústeres con conmutación por error en un clúster, permite que varios equipos SQL Server aparezcan como un único *servidor virtual*.

Un clúster con conmutación por error de SQL Server 2000 puede incluir uno o varios servidores virtuales. Cada servidor virtual consta de uno o varios nodos. Puede crear servidores virtuales durante la instalación de SQL Server y puede instalar un servidor virtual para cada grupo de recursos del clúster subyacente de Windows.

En un clúster con conmutación por error de SQL Server, cada servidor virtual debe tener asignado un nombre de red de forma que los clientes puedan tener acceso a él a través de la red. Un servidor virtual puede tener asignadas varias direcciones IP. Puede asignar distintas direcciones IP para todas las subredes disponibles, con lo que se garantiza la conexión a la red aún en el caso de que ocurra un error en un adaptador de red o enrutador.

Organización por clústeres activo-pasivo

Puede configurar SQL Server en un clúster de forma que sólo esté activa una copia de SQL Server en cualquier momento. El otro servidor SQL Server está pasivo y sólo se activa cuando se produce un error en el nodo principal. En el caso de una conmutación por error, el nodo secundario se hace cargo de los recursos y carga de trabajo del nodo principal. Esta configuración ofrece protección frente a errores de sistema, al tiempo que proporciona un rendimiento óptimo.

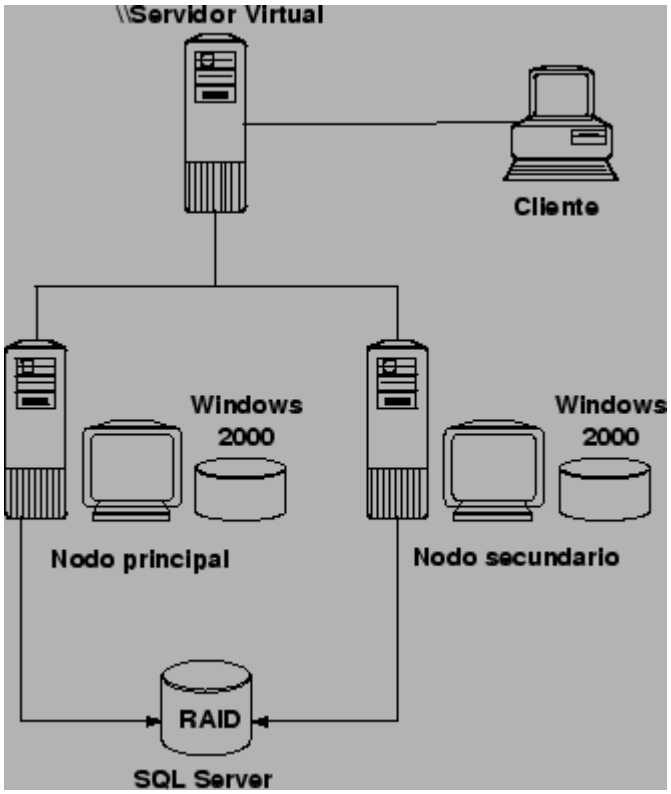


Figura: Organización por clústeres activo-pasivo

La configuración por clústeres con conmutación por error activo-pasivo se puede realizar siempre que se disponga de los siguientes componentes y se cumplan las condiciones mencionadas:

- El disco de cada nodo contiene una copia de *Windows 2000 Advanced Server* o *Windows 2000 DataCenter Server*. El disco también debe tener instalado Organización por clústeres de Windows.
- Una copia de SQL Server debe estar instalada en el disco SCSI o en la matriz de discos que comparten ambos nodos.
- Hay una conexión de red que vincula los dos servidores físicos.
- Se ha creado un servidor virtual que representa el conjunto formado por los dos servidores físicos. Puede crear un servidor virtual mediante el Asistente para clústeres con conmutación por error.

Los clientes que se conectan a este servidor virtual no saben a qué servidor físico activo están conectados.

Organización por clústeres activo-activo

En entornos en los que hay varias aplicaciones de bases de datos, se puede usar la organización por clústeres activo-activo para mejorar la disponibilidad de las aplicaciones al tiempo que se ejecutan en un host distinto par aumentar el rendimiento. En una configuración por clústeres con conmutación por error activo-activo, cada nodo ejecuta una o varias copias de SQL Server. De esta forma, es posible que cada nodo actúe como servidor principal de uno o varios servidores virtuales, y como servidor secundario del resto de los servidores.

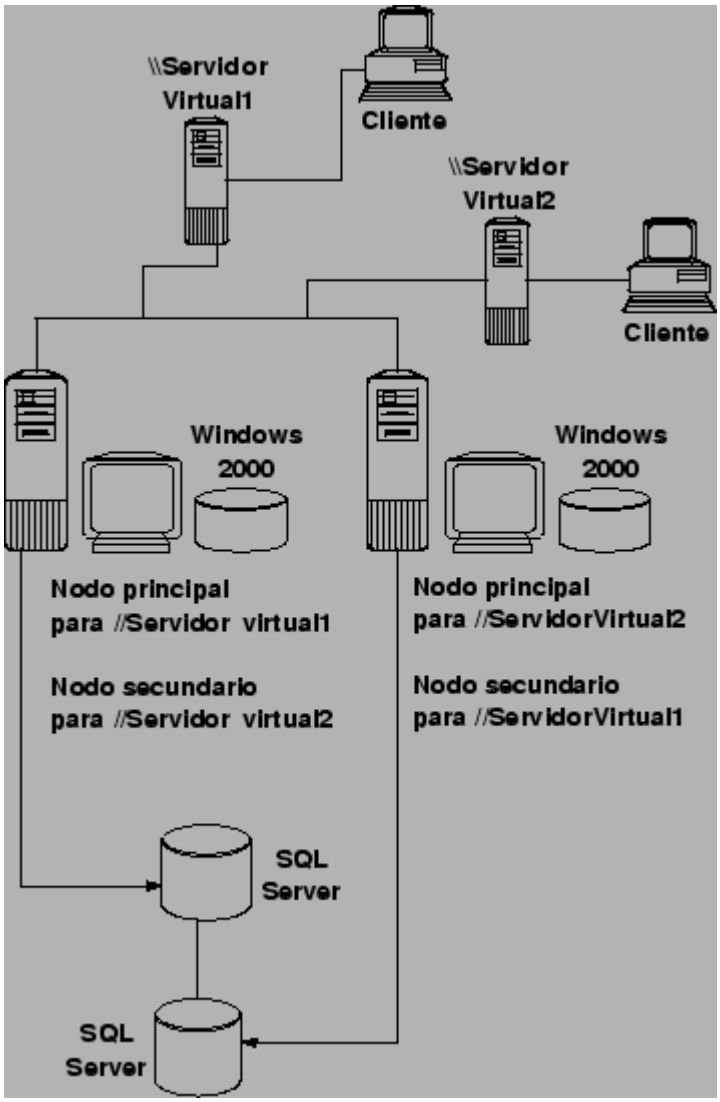


Figura: Organización por clústeres activo-activo

Esta configuración utiliza al máximo ambos nodos, aunque mantiene la posibilidad de conmutación por error. Sin embargo, si se produce la conmutación por error, el rendimiento general se verá afectado ya que un nodo debe hacer frente a la carga de trabajo de ambos nodos.

La configuración por clústeres con conmutación por error activo-activo se puede realizar siempre que se disponga de los siguientes componentes y se cumplan las condiciones mencionadas:

- El disco de cada nodo mantiene una copia de *Windows 2000 Advanced Server* o *Windows 2000 Datacenter Server*. El disco también debe tener instalada la organización por clústeres con conmutación por error.
- Debe haber dos discos físicos SCSI o matrices RAID de hardware compartidos en un bus SCSI compartido.
- Debe haber instalada una copia de SQL Server en cada uno de los discos compartidos.
- Debe haber al menos dos conexiones de red que vinculen los dos servidores físicos.
- Se deben haber instalado dos servidores virtuales, cada uno de los cuales debe representar al conjunto formado por los dos servidores físicos. Los clientes se pueden conectar a cualquiera de los servidores virtuales.

Servidores de reserva y trasvase de registros

Un *servidor de reserva* es un servidor físico independiente que contiene una copia exacta del servidor de producción. Si se produce un error en el servidor de producción, se puede realizar inmediatamente la conmutación al servidor de reserva, con lo que se reduce al mínimo el tiempo de inactividad y se alcanza un alto grado de disponibilidad.

Un servidor de reserva es un segundo servidor que refleja el contenido del servidor principal. Se puede usar el servidor de reserva para sustituir al servidor principal cuando se produce un error en éste. También se puede usar como una copia de sólo lectura de la base de datos, con lo que se descarga parte del trabajo del servidor principal.

Para mantener sincronizado el servidor de reserva con el servidor principal puede copiar y cargar los registros de transacciones del servidor principal al servidor de reserva. Debe comprobar que SQL Server utiliza el modelo de recuperación completa para que se realicen las copias de seguridad del registro de transacciones.

El transvase de registros automatiza el proceso de sincronización mediante el uso de trabajos de *SQL Server Agent*. La secuencia de sucesos que componen el transvase de registros incluye la realización de copias de seguridad de los registros de transacciones de la base de datos principal, la copia de éstos a una ubicación secundaria y su restauración en la base de datos secundaria. En concreto:

- *SQL Server Agent* realiza una copia de seguridad del registro de transacciones del servidor principal y le asigna un nombre basado en la fecha y hora actuales.
- El *SQL Server Agent* del servidor de reserva copia a una carpeta del del servidor secundario la copia de seguridad del registro de transacciones.
- El *SQL Server Agent* del servidor secundario ejecuta un trabajo de carga para restaurar el registro de transacciones en la base de datos del servidor de reserva.

MySQL

Un *Sistema Gestor de Bases de Datos Relacionales* (RDBMS) es una herramienta imprescindible en mucho entornos, desde los usos más tradicionales en los contextos de negocios, investigación y educación hasta las aplicaciones más novedosas, como el potenciamiento de los motores de búsquedas en Internet. Sin embargo, a pesar de la importancia de una buena base de datos para gestionar y acceder a los recursos de información, para muchas empresas están fuera del alcance de sus recursos financieros. Históricamente, los sistemas de bases de datos han sido una propuesta cara, ya que los proveedores añaden costes tanto por el software como por el mantenimiento y atención posterior, y debido a que los motores de bases de datos habitualmente requieren una potente configuración hardware para ejecutarse con un rendimiento razonable, el coste puede ser incluso mayor.

En los últimos años, la situación ha cambiado, tanto por el lado hardware como por el lado software. También el software de base de datos se ha hecho más accesible. Una de las entradas más recientes en el terreno de las bases de datos de coste mínimo es MySQL, un sistema gestor de bases de datos relacionales cliente-servidor SQL.

MySQL es portable y se ejecuta en sistemas operativos comerciales y en hardware que puede llegar hasta servidores empresariales. Además, su rendimiento rivaliza con cualquier sistema de base de

datos contra el que se desee compararlo, pudiendo gestionar bases de datos con millones de registros. De esta forma, empresas que únicamente podían soñar con disponer de la potencia de un RDBMS de alto rendimiento para su propio uso, ahora pueden disponer del mismo por un bajo coste.

Algunas de las características que nos han llevado a considerar MySQL como nuestro RDBMS son:

- **Velocidad**. MySQL es rápido. Los desarrolladores sostienen que MySQL es posiblemente la base de datos más rápida que se puede hallar.
- **Facilidad de uso**. MySQL es un sistema de base de datos de alto rendimiento, pero relativamente simple y mucho menos complejo de configurar y administrar que sistemas más grandes.
- **Coste**. MySQL es gratuito para la mayoría de usos internos.
- **Capacidad de gestión de lenguajes de consulta**. MySQL comprende SQL (*Structured Query Language*, Lenguaje de Consulta Estructurado), el lenguaje elegido para todos los sistemas de bases de datos modernos. También puede acceder a MySQL empleando aplicaciones que admitan ODBC (*Open Database Connectivity*, Conectividad de Base de Datos), un protocolo de comunicación de bases de datos desarrollado por *Microsoft*.
- **Capacidad**. Pueden conectarse muchos clientes simultáneamente al servidor. Los clientes pueden utilizar varias bases de datos simultáneamente. Puede acceder de forma interactiva a MySQL empleando diferentes interfaces que le permitan introducir consultar y visualizar los resultados: cliente de línea de comandos, navegadores web o clientes del sistema X Window.
- **Conectividad y Seguridad**. MySQL está completamente preparado para el trabajo en red y las bases de datos pueden ser accedidas desde cualquier lugar en Internet, por lo que puede compartir sus datos con cualquiera, en cualquier parte. Pero MySQL dispone de control de acceso, de forma que aquellos que no deberían ver sus datos, no los vean.
- **Portabilidad**. MySQL se ejecuta en muchas variantes de UNIX, así como otros sistemas no-UNIX como Windows y OS/2. MySQL se ejecuta en hardware que va desde PC hasta servidores de alta capacidad.
- **Distribución abierta**. MySQL es fácil de obtener a través de Internet. Además, si no se entiende cómo funciona alguna cosa o se tiene curiosidad por un algoritmo, se puede obtener el código fuente e investigarlo.

Características

A continuación se listan aquellas características que aún están en proceso de desarrollo en MySQL.

- **Subselecciones**. Una subselección es una sentencia `SELECT` que se anida dentro de otra, como sucede en la siguiente consulta:

```
SELECT * FROM score WHERE event_id
IN (SELECT event_id FROM event WHERE type=''T'')
```

- **Las transacciones y operaciones *commit-rollback***. Una transacción es un conjunto de sentencias SQL que se ejecutan como una unidad sin ser interrumpidas por otros clientes. La capacidad de *commit-rollback* le permite confirmar si las sentencias se van a ejecutar como una unidad o no. Si cualquier sentencia de la transacción falla, cada una de las sentencias que se hallan ejecutado hasta ese momento se deshace.

MySQL ejecuta automáticamente la búsqueda de sentencias SQL para que los clientes no interfieran entre ellos (por ejemplo, dos clientes no pueden escribir en la misma tabla simultáneamente). Además, puede usar `LOCK TABLES` y `UNLOCK TABLES` para agrupar las sentencias como una unidad, lo que le permitirá efectuar operaciones para las que no es suficiente el control de concurrencia de una sola sentencia. Si emite una transacción con MySQL no agrupará automáticamente las sentencias, y no podrá efectuar el *rollback* si una de ellas falla.

- **Claves extrañas e integridad referencial**. Una clave extraña le permite declarar que una clave de una tabla está relacionada con otra en otra tabla. Las claves extrañas le ayudan a mantener la consistencia de sus datos y, en cierto modo, proporcionan conveniencia. Las razones por las que MySQL no las apoya se deben principalmente a cierto efecto negativo que provocan en el funcionamiento y mantenimiento de la base de datos.
- **Procedimientos almacenados y *triggers***. Un procedimiento almacenado consiste en un conjunto de sentencias SQL previamente almacenadas en el DBMS. Cuando un cliente desea hacer uso de dicho procedimiento únicamente ha de invocarlo en el servidor, evitando tener que enviar las sentencias SQL que integran el mismo.
- **Vistas** Una vista es una entidad lógica que actúa como una tabla, pero que no lo es. Proporciona un modo de ver columnas de diferentes tablas como si formarán parte de una sola. En ocasiones se las llama tabla virtuales.

Copias de seguridad

Llevar a cabo copias de seguridad de las bases de datos evita que en caso de fallo del sistema se pierda toda la información almacenada. En ocasiones, un usuario poco prudente que ejecute una sentencia `DROP DATABASE` o `DROP TABLE` solicitará que se realice la recuperación de los datos. Igualmente, puede ser el propio administrador MySQL quien cause los daños.

Los dos métodos más importantes que dispone MySQL para hacer copias de seguridad de las bases de datos pasan por utilizar el programa `mysqldump` o copiar directamente los archivos de base de datos. Cada método tiene sus ventajas e inconvenientes:

- `mysqldump` opera en cooperación con el servidor MySQL. Los métodos de copia directa son externos al servidor, de manera que usted debe tomar las medidas oportunas para asegurar que ningún cliente modifique las tablas mientras usted las copia. El mismo problema se produce si una base de datos está siendo actualizada durante la copia de seguridad del sistema de archivos, los archivos de tabla que entran en dicha copia están en un estado poco consistente y no valen para restaurar la tabla más tarde.
- `mysqldump` genera archivos de texto transportables a otras máquinas (aún cuando presenten distintas arquitecturas hardware). Los archivos de copia directa no lo son, a no ser que las tablas que esté copiando usen el formato de almacenamiento MyISAM.
- Cuando se usa el método de copia directa, debe asegurarse de que las tablas no se están usando. En caso contrario, las copias no reflejarán el estado actual de la base de datos. El mejor modo de asegurar la integridad de sus copias es desactivar el servidor, copiar los archivos y reiniciar el servidor después.

Réplicas de bases de datos

En la sección anterior se ha descrito cómo se pueden realizar copias de seguridad de las bases de datos almacenados en un servidor MySQL:

- *Copia directa*
- *Utilización de la utilidad mysqldump*

Ambas técnicas requieren que la base de datos no sea modificada mientras que se realiza la copia de seguridad. Es más, en caso de fallo, el sistema quedará fuera de servicio hasta que no se recupere el estado de la base de datos.

El término *replicar* puede significar algo tan simple como duplicar la base de datos en otro servidor, o actualizar en vivo una base de datos secundaria según se hacen cambios en los contenidos de una base de datos *master*. El soporte para la replicación en MySQL, a diferencia de *Oracle* ó *Microsoft SQL Server*, se encuentra aún en fase de desarrollo. Por este motivo, se ha optado por este servidor a la hora de enfrentar el problema de ofrecer tolerancia a fallos.

API JDBC como interfaz de acceso a bases de datos SQL

En el mundo de la informática hay numerosos estándares y lenguajes, la mayoría de los cuales son incapaces de comunicarse entre sí. Afortunadamente, algunos de ellos son verdaderos referentes cuyo conocimiento es vital para los programadores. El **Structured Query Language** o SQL, se ha convertido en los últimos años en el método estándar de acceso a bases de datos. Se puede decir que cualquier Sistema de Gestión de Bases de Datos creado en los últimos años usa SQL. Esta es la principal virtud de SQL: es un lenguaje prácticamente universal dentro de las bases de datos.

Tener conocimientos de SQL es una necesidad para cualquier profesional de las tecnologías de la información (TI). A medida que el desarrollo de sitios web se hace más común entre personas sin conocimientos de programación, el tener una cierta noción de SQL se convierte en requisito indispensable para integrar datos en páginas HTML. No obstante, éste no es el único ámbito de SQL, puesto que la mayor parte de las aplicaciones de gestión que se desarrollan hoy en día acceden, en algún momento, a alguna base de datos utilizando sentencias SQL. La mayoría de los fabricantes ofrecen APIs nativas para acceder a sus bases de datos, sin embargo, el hecho de utilizar estas APIs, obliga a los programadores a adaptar cada programa a las características de un servidor determinado.

Para solventar este problema, Java introduce la **API JDBC**, compuesta por un conjunto de clases que unifican el acceso a las bases de datos. Gracias a la utilización de JDBC, un programa Java puede acceder a cualquier base de datos sin necesidad de modificar la aplicación. Sin embargo, para que esto sea posible es necesario que el fabricante ofrezca un driver que cumpla la especificación JDBC.

Un **driver JDBC** es una capa de software intermedia que traduce las llamadas JDBC a los APIs específicos del vendedor. Describir las características de la API JDBC es el objeto de este capítulo.

Programación de bases de datos con JDBC

La información contenida en un servidor de bases de datos es normalmente el bien más preciado dentro de una empresa. La API JDBC ofrece a los desarrolladores Java un modo de conectar con dichas bases de datos. Utilizando la API JDBC, los desarrolladores pueden crear un cliente que pueda conectar con una base de datos, ejecutar instrucciones SQL y procesar el resultado de esas instrucciones.

La API proporciona conectividad y acceso a datos en toda la extensión de las bases de datos relacionales. JDBC generaliza las funciones de acceso a bases de datos más comunes abstrayendo los detalles específicos de una determinada base de datos. El resultado es un conjunto de clases e interfaces, localizadas en el paquete `java.sql`, que pueden ser utilizadas con cualquier base de datos que disponga del driver JDBC apropiado. La utilización de este driver significa que, siempre y cuando una aplicación utilice las características más comunes de acceso a bases de datos, dicha aplicación podrá utilizarse con una base de datos diferente cambiando simplemente a un driver JDBC diferente.

Los vendedores de bases de datos más populares como Oracle, Sybase e Informix ofrecen APIs de su propiedad para el acceso del cliente. Las aplicaciones cliente escritas en lenguajes nativos pueden utilizar estos APIs para obtener acceso directo a los datos, pero no ofrecen una interfaz común de acceso a diferentes bases de datos. La API JDBC ofrece una alternativa al uso de estas APIs, permitiendo acceder a diferentes servidores de bases de datos, únicamente cambiando el driver JDBC por el que ofrezca el fabricante del servidor al que se desea acceder.

Paquete java.sql

JDBC ofrece el paquete `java.sql`, en el que existen clases muy útiles para trabajar con bases de datos.

Clase	Descripción
DriverManager	Para cargar un driver
Connection	Para establecer conexiones con las bases de datos
Statement	Para ejecutar sentencias SQL y enviarlas a las BBDD
PreparedStatement	La ruta de ejecución está predeterminada en el servidor de base de datos que le permite ser ejecutado varias veces
ResultSet	Para almacenar el resultado de la consulta

Tipos de Driver JDBC

Una de las decisiones importantes en el diseño, cuando estamos proyectando una aplicación de bases de datos Java, es decidir el driver JDBC que permitirá que las clases JDBC se comuniquen con la base de datos. Los driver JDBC se clasifican en cuatro tipos o niveles:

- **Tipo 1:** Puente JDBC-ODBC

- **Tipo 2:** Driver API nativo/parte Java
- **Tipo 3:** Driver protocolo de red/todo Java
- **Tipo 4:** Driver protocolo nativo/todo Java

Entender cómo se construyen los drivers y cuales son sus limitaciones, nos ayudará a decidir qué driver es el más apropiado para cada aplicación.

Tipo 1: Driver puente JDBC-ODBC

El puente JDBC-ODBC es un driver JDBC del tipo 1 que traduce operaciones JDBC en llamadas a la API ODBC. Estas llamadas son entonces cursadas a la base de datos mediante el driver ODBC apropiado. Esta arquitectura se muestra en la siguiente figura:

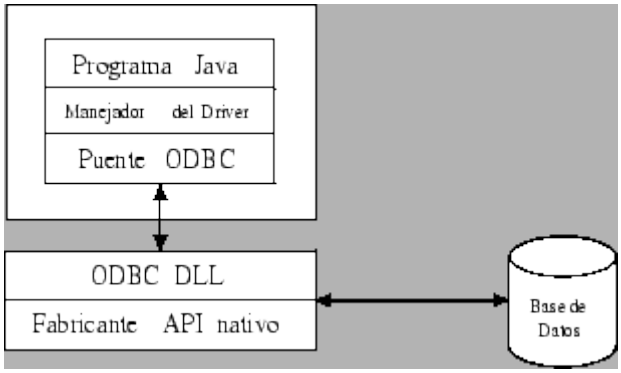


Figura: Driver JDBC Tipo 1

El puente se implementa como el paquete `sun.jdbc.odbc` y contiene una biblioteca nativa utilizada para acceder a ODBC.

Ventajas

A menudo, el primer contacto con un driver JDBC es un puente JDBC-ODBC, simplemente porque es el driver que se distribuye como parte de Java, como el paquete `sun.jdbc.odbc.JdbcOdbcDriver`.

Además tiene la ventaja de poder trabajar con una gran cantidad de drivers ODBC. Los desarrolladores suelen utilizar ODBC para conectar con bases de datos en un entorno distinto de Java. Por tanto, los drivers de tipo 1 pueden ser útiles para aquellas compañías que ya tienen un driver ODBC instalado en las máquinas clientes. Se utilizará normalmente en máquinas basadas en Windows que ejecutan aplicaciones de gestión. Por supuesto, puede ser el único modo de acceder a algunas bases de datos de escritorio, como MS Access, dBase y Paradox.

En este sentido, la ausencia de complejidad en la instalación y el hecho de que nos permita acceder virtualmente a cualquier base de datos, le convierte en una buena elección. Sin embargo, hay muchas razones por las que se desecha su utilización.

Desventajas

Básicamente sólo se recomienda su uso cuando se están realizando esfuerzos dirigidos a prototipos y en casos en los que no exista otro driver basado en JDBC para esa tecnología. Si es posible, debemos utilizar un driver JDBC en vez de un puente y un driver ODBC. Esto elimina totalmente la configuración cliente necesaria en ODBC.

Los siguientes puntos resumen algunos de los inconvenientes de utilizar el driver puente:

- Rendimiento: Como puede imaginar por el número de capas y traducciones que tienen lugar, utilizar el puente está lejos de ser la opción más eficaz en términos de rendimiento.
- Utilizando el puente JDBC-ODBC, el usuario está limitado por la funcionalidad del driver elegido. Es más, dicha funcionalidad se limita a proporcionar acceso a caraterísticas comunes a todas las bases de datos. No pudiendo hacer uso de las mejoras que cada fabricante introduce en sus productos, especialmente en lo que afecta a rendimiento y escalabilidad.
- El driver puente no funciona adecuadamente con applets. El driver ODBC y la interfaz de conexión nativa deben estar ya instalados en la máquina cliente. Por eso, cualquier ventaja de la utilización de applets en un entorno de Intranet se pierde, debido a los problemas de despliegue que conllevan las aplicaciones tradicionales.
- La mayoría de los browser no tienen soporte nativo del puente. Como el puente es un componente opcional del Java 2 SDK Standard Edition, no se ofrece con el navegador. Incluso si fuese ofrecido, sólo los applets de confianza (aquellos que permiten escribir en archivos) serán capaces de utilizar el puente. Esto es necesario para preservar la seguridad de los applet. Para terminar, incluso si el applet es de confianza, ODBC deve ser configurado en cada máquina cliente.

Tipo 2: Driver API Nativo / parte Java

Los drivers de tipo 2, del que es un ejemplo el driver JDBC/OCI de Oracle, utilizan la interfaz de métodos nativos de Java para convertir las solicitudes de API JDBC en llamadas específicas a bases de datos para RDBMS como SQL Server, Informix, Oracle o Sybase, como se puede ver en la siguiente figura:

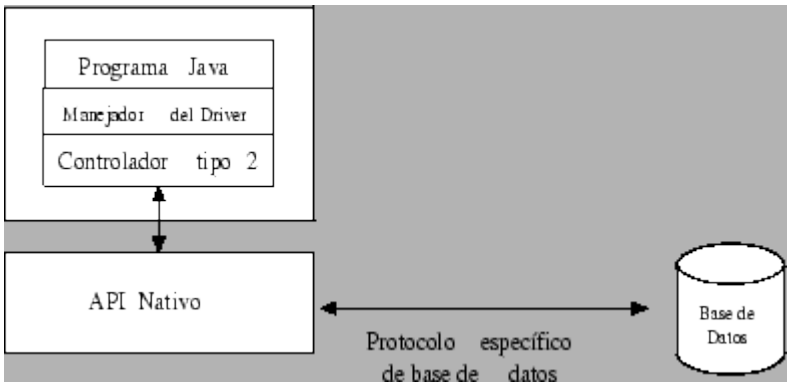


Figura 8.2: Driver JDBC Tipo 2

Aunque los drivers de tipo 2 habitualmente ofrecen mejor rendimiento que el puente JDBC-ODBC, siguen teniendo los mismos problemas de despliegue en los que la interfaz de conectividad nativa debe estar ya instalada en la máquina cliente. El driver JDBC necesita una biblioteca suministrada por el fabricante para traducir las funciones JDBC en lenguaje de consulta específico para ese servidor. Estos drivers están normalmente escritos en alguna combinación de Java y C/C++, ya que el driver debe utilizar una capa de C para realizar llamadas a la biblioteca que está escrita en C.

Ventajas

- El driver de tipo 2 ofrece un rendimiento significativamente mayor que el puente JDBC-ODBC, ya que las llamadas JDBC no se convierten en llamadas ODBC, sino que son directamente nativas.

Desventajas

- La biblioteca de la base de datos del fabricante necesita iniciarse en cada máquina cliente. En consecuencia, los drivers de tipo 2 no se pueden utilizar en Internet. Los drivers de tipo 2 muestran menor rendimiento que los de tipo 3 y 4.
- Un driver de tipo 2 también utiliza la interfaz nativa de Java, que no está implementada de forma consistente entre los distintos fabricantes de JVM por lo que habitualmente no es muy portable entre plataformas.

Tipo 3: Driver protocolo de red / todo Java

Los drivers JDBC de tipo 3 están implementados en una aproximación de tres capas por lo que las solicitudes de la base de datos JDBC están traducidas en un protocolo de red independiente de la base de datos y dirigidas al servidor de capa intermedia. El servidor de la capa intermedia recibe las solicitudes y las envía a la base de datos utilizando para ello un driver JDBC del tipo 1 o del tipo 2 (lo que significa que se trata de una arquitectura muy flexible).

La arquitectura en conjunto consiste en tres capas: la capa cliente JDBC y driver, la capa intermedia y la base o las bases de datos a las que se accede.

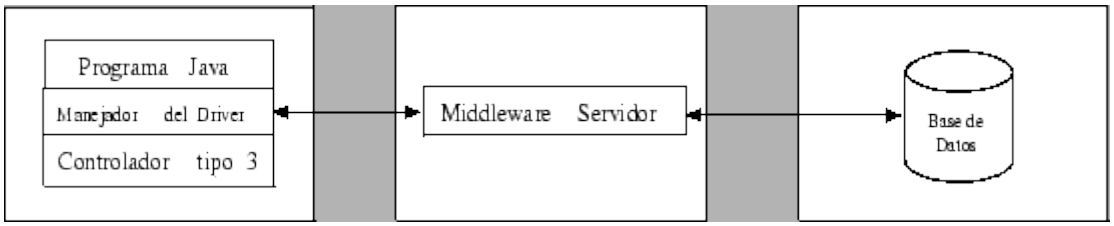


Figura: Driver JDBC Tipo 3

El driver JDBC se ejecuta en el cliente e implementa la lógica necesaria para enviar a través de la red comandos SQL al servidor JDBC, recibir las respuestas y manejar la conexión.

El componente servidor intermedio puede implementarse como un componente nativo, o alternativamente escrito en Java. Las implementaciones nativas conectan con la base de datos utilizando bien una biblioteca cliente del fabricante o bien ODBC. El servidor tiene que configurarse para la base o bases de datos a las que se va a acceder. Esto puede implicar asignación de números de puerto, configuración de variables de entorno, o de cualquier otro parámetro que pueda necesitar el servidor. Si el servidor intermedio está escrito en Java, puede utilizar cualquier driver en conformidad con JDBC para comunicarse con el servidor de bases de datos mediante el protocolo propietario del fabricante. El servidor JDBC maneja varias conexiones con la base de datos, así como excepciones y eventos de estado que resultan de la ejecución de SQL. Además, organiza los datos para su transmisión por la red a los clientes JDBC.

Ventajas

- El driver protocolo de red/todo Java tiene un componente en el servidor intermedio, por lo que no necesita ninguna biblioteca cliente del fabricante para presentarse en las máquinas clientes.
- Los drivers de tipo 3 son los que mejor funcionan en redes basadas en Internet o Intranet, aplicaciones intensivas de datos, en las que un gran número de operaciones concurrentes como consultas, búsquedas, etc., son previsibles y escalables y su rendimiento es su principal factor. Hay muchas oportunidades de optimizar la portabilidad, el rendimiento y la escalabilidad.
- El protocolo de red puede estar diseñado para hacer el driver JDBC cliente muy pequeño y rápido de iniciar, lo que es perfecto para el despliegue de aplicaciones de Internet.
- Además, un driver tipo 3 normalmente ofrece soporte para características como almacenamiento en memoria caché (conexiones, resultados de consultas, etc.), equilibrio de carga, y administración avanzada de sistemas como el registro.
- La mayor parte de aplicaciones web de bases de datos basadas en 3 capas implican seguridad, firewalls y proxis y los drivers del tipo 3 ofrecen normalmente estas características.

Inconvenientes

- Los drivers de tipo 3 requieren código específico de bases de datos para realizarse en la capa intermedia.
- Además, atravesar el conjunto de registros puede llevar mucho tiempo, ya que los datos vienen a través del servidor de datos.

Tipo 4: Driver protocolo nativo / todo Java

Este tipo de driver comunica directamente con el servidor de bases de datos utilizando el protocolo nativo del servidor. Estos drivers pueden escribirse totalmente en Java, son independientes de la plataforma y eliminan todo los aspectos relacionados con la configuración en el cliente. Sin embargo, este driver es específico de un fabricante determinado de base de datos. Cuando la base de datos necesita ser cambiada a un producto de otro fabricante, no se puede utilizar el mismo driver. Por el contrario, hay que reemplazarlo y también el programa cliente, o su asignación, para ser capaces de utilizar una cadena de conexión distinta para iniciar el driver.

Estos drivers traducen JDBC directamente a protocolo nativo sin utilizar ODBC o la API nativa, por lo que pueden proporcionar un alto rendimiento de acceso a bases de datos.

Ventajas

- Como los drivers JDBC de tipo 4 no tienen que traducir las solicitudes de ODBC o de una interfaz de conectividad nativa, o pasar la solicitud a otro servidor, el rendimiento es bastante bueno. Además, el driver protocolo nativo/todo Java da lugar a un mejor rendimiento que los de tipo 1 y 2.
- Además, no hay necesidad de instalar ningún software especial en el cliente o en el servidor. Además, estos drivers pueden bajarse de la forma habitual.

Desventajas

- Con los drivers de tipo 4, el usuario necesita un driver distinto para cada base de datos.

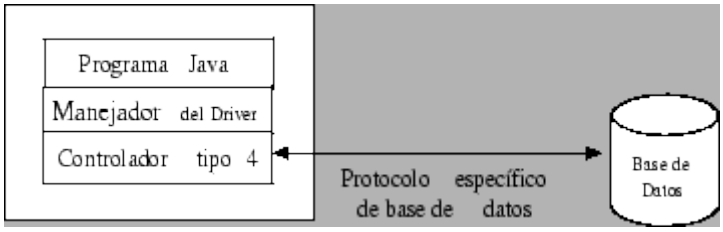


Figura: Driver JDBC Tipo 4

Arquitecturas para aplicaciones con bases de datos

Hay una gran variedad de arquitecturas posibles para las aplicaciones de bases de datos (dependiendo de los requisitos de la aplicación). Elegir el driver JDBC correcto es importante porque tiene un impacto directo en el rendimiento de la aplicación.

El puente JDBC-ODBC se podría considerar únicamente como una solución transitoria ya que no soporta todas las características de Java, y el usuario está limitado por la funcionalidad del driver ODBC elegido. Las aplicaciones a gran escala utilizarán drivers de los tipos 2, 3 o 4.

En las aplicaciones de Intranet es útil considerar los driver de tipo 2, pero estos drivers, como el puente ODBC, necesitan que ese código se instale en cada cliente. Por lo tanto, tienen los mismos problemas de mantenimiento que el driver puente JDBC-ODBC. Sin embargo, los drivers de tipo 2 son más rápidos que los de tipo 1 porque se elimina el nivel extra de traducción. Como los drivers de tipo 3 y de tipo 4 muestran mejor rendimiento que los drivers de tipo 2, la tendencia se dirige hacia un driver Java puro más robusto.

Para las aplicaciones relacionadas con Internet, no hay otra opción que utilizar drivers del tipo 3 o del tipo 4. Los drivers de tipo 3 son los que mejor funcionan con los entornos que necesitan proporcionar conexión a gran cantidad de servidores de bases de datos y a bases de datos heterogéneas. Los drivers de tipo 3 funcionan con aplicaciones intensivas de datos multiusuario, en las que se espera un alto número de operaciones concurrentes de datos, siendo el rendimiento y la escalabilidad el principal factor. El servidor puede proporcionar facilidades de registro y de administración, características del equilibrio de carga y puede soportar cachés de catálogo y de consultas.

Los drivers de tipo 4 están generalmente recomendados para aplicaciones que requieren un acceso rápido y eficiente a bases de datos. Como estos drivers traducen llamadas JDBC directamente a protocolo nativo sin utilizar ODBC o el API nativos, pueden aportar acceso a bases de datos de alto rendimiento.

Profundizando en la API JDBC

Hasta el momento se han presentado, por un lado, las características generales de la API JDBC y por otro, los distintos tipos de driver que se pueden utilizar. Sin embargo, la arquitectura JDBC está basada en un conjunto de clases Java que permiten conectar con bases de datos, crear y ejecutar sentencias SQL o recuperar y modificar la información almacenada en una base de datos. En las siguientes secciones se describirán cada una de estas operaciones:

- Cargar un driver de base de datos
- Establecer conexiones con bases de datos
- Crear y ejecutar instrucciones SQL

- Consultar bases de datos
- Realizar transacciones

Cargar un driver de base de datos y abrir conexiones

La interfaz `java.sql.Connection` representa una conexión con una base de datos. Es una interfaz porque la implementación de una conexión depende de la red, del protocolo y del vendedor. El API JDBC ofrece dos vías diferentes para obtener conexiones. La primera utiliza la clase `java.sql.DriverManager` y es adecuada para acceder a bases de datos desde programas cliente escritos en Java. El segundo enfoque se basa en el acceso a bases de datos desde aplicaciones J2EE (Java 2 Enterprise Edition). [2]

Consideremos cómo se obtienen las conexiones utilizando la clase `java.sql.DriverManager`. En una aplicación, podemos obtener una o más conexiones para una o más bases de datos utilizando drivers JDBC. Cada driver implementa la interfaz `java.sql.Driver`. Uno de los métodos que define esta interfaz es el método `connect()`, que permite establecer una conexión con la base de datos y obtener un objeto `Connection`.

En lugar de acceder directamente a clases que implementan la interfaz `java.sql.Driver`, el enfoque estándar para obtener conexiones es registrar cada driver con `java.sql.DriverManager` y utilizar los métodos proporcionados en esta clase para obtener conexiones. `java.sql.DriverManager` puede gestionar múltiples drivers. Antes de entrar en los detalles de este enfoque, es preciso entender cómo JDBC representa la URL de una base de datos.

Los URL de JDBC

La noción de un URL en JDBC es muy similar al modo típico de utilizar los URL. Los URL de JDBC proporcionan un modo de identificar un driver de base de datos. Un URL de JDBC representa un driver y la información adicional necesaria para localizar una base de datos y conectar a ella. Su sintaxis es la siguiente:

`jdbc:<subprotocol>:<subname>`

Existen tres partes separadas por dos puntos:

- Protocolo: En la sintaxis anterior, `jdbc` es el protocolo. Éste es el único protocolo permitido en JDBC.
- Subprotocolo: Utilizado para identificar el driver que utiliza la API JDBC para acceder al servidor de bases de datos. Este nombre depende de cada fabricante.
- Subnombre: La sintaxis del subnombre es específica del driver.

Por ejemplo, para una base de datos MySQL llamada ``Bank'', el URL al que debe conectar es:

```
jdbc:mysql:Bank
```

Alternativamente, si estuviéramos utilizando Oracle mediante el puente JDBC-ODBC, nuestro URL sería:

```
jdbc:odbc:Bank
```

Como puede ver, los URL de JDBC son lo suficientemente flexibles como para especificar información específica del driver en el subnombre.

Clase `DriverManager`

El propósito de la clase `java.sql.DriverManager` (gestor de drivers) es proporcionar una capa de acceso común encima de diferentes drivers de base de datos utilizados en una aplicación. En este enfoque, en lugar de utilizar clases de implementación `Driver` directamente, las aplicaciones utilizan la clase `DriverManager` para obtener conexiones. Esta clase ofrece tres métodos estáticos para obtener conexiones.

Sin embargo, `DriverManager` requiere que cada driver que necesite la aplicación sea registrado antes de su uso, de modo que el `DriverManager` sepa que está ahí.

El enfoque JDBC para el registro de un driver de base de datos puede parecer oscuro al principio. Fíjese en el siguiente fragmento de código que carga el driver de base de datos de MySQL:

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
    // Driver no encontrado
}
```

En tiempo de ejecución, el `ClassLoader` localiza y carga la clase `com.mysql.jdbc.Driver` desde la ruta de clases utilizando el cargador de clase de autoarranque. Mientras carga una clase, el cargador de clase ejecuta cualquier código estático de inicialización para la clase. En JDBC, se requiere que cada proveedor de driver registre una instancia del driver con la clase `java.sql.DriverManager` durante esta inicialización estática. Este registro tiene lugar automáticamente cuando el usuario carga la clase del driver (utilizando la llamada `Class.forName()`).

Una vez que el driver ha sido registrado con el `java.sql.DriverManager`, podemos utilizar sus métodos estáticos para obtener conexiones. El gestor de drivers tiene tres variantes del método estático `getConnection()` utilizado para establecer conexiones. El gestor de drivers delega estas llamadas en el método `connect()` de la interfaz `java.sql.Driver`.

Dependiendo del tipo de driver y del servidor de base de datos, una conexión puede conllevar una conexión de red física al servidor de base de datos o a un proxy de conexión. Las bases de datos integradas no requieren conexión física. Exista o no una conexión física, el objeto de conexión es el único objeto que utiliza una conexión para comunicar con la base de datos. Toda comunicación debe tener lugar dentro del contexto de una o más conexiones.

Consideremos ahora los diferentes métodos para obtener una conexión:

```
public static Connection getConnection(String url)
        throws SQLException
```

`java.sql.DriverManager` recupera el driver apropiado del conjunto de drivers registrados. El URL de la base de datos está especificado en la forma de `jdbc:subprotocol:subname`. Para poder obtener una conexión a la base de datos es

necesario que se introduzcan correctamente los parámetros de autentificación requeridos por el servidor de bases de datos.

```
public static Connection getConnection(String url,
        java.util.Properties info) throws SQLException
```

Este método requiere un URL y un objeto `java.util.Properties`. El objeto `Properties` contiene cada parámetro requerido para la base de datos especificada. La lista de propiedades difiere entre bases de datos. Dos propiedades comunmente utilizadas para una base de datos son `autocommit=true` y `create=false`. Podemos especificar estas propiedades junto con el URL como `jdbc:subprotocol:subname; autocommit=true;create=true` o podemos establecer estas propiedades utilizando el objeto `Properties` y pasar dicho objeto como parámetro en el anterior método `getConnection()`.

```
String url = "jdbc:mysql:Bank";
Properties p = new Properties();
p.put("autocommit", "true");
p.put("create", "true");
Connection connection = DriverManager.getConnection(url, p);
```

En caso de que no se adjunten todas las propiedades requeridas para el acceso, se generará una excepción en tiempo de ejecución.

La tercera variante toma como argumentos además del URL, el nombre del usuario y la contraseña. Fijese en el siguiente ejemplo, utiliza un driver MySQL, y requiere un nombre de usuario y una contraseña para obtener una conexión:

```
String url = "jdbc:mysql:Bank";
String user = "root";
String password = "nadiuska";
Connection connection = DriverManager.getConnection(url,
                                                    user,
                                                    password);
```

Observe que todos estos métodos están sincronizados, lo que supone que sólo puede haber un hilo accediendo a los mismos en cada momento. Estos métodos lanzan una excepción `SQLException` si el driver no consigue obtener una conexión.

Interfaz `Driver`

Cada driver debe implementar la interfaz `java.sql.Driver`. En MySQL, la clase `com.mysql.jdbc.Driver` implementa la interfaz `java.sql.Driver`.

La clase `DriverManager` utiliza los métodos definidos en esta interfaz. En general, las aplicaciones cliente no necesitan acceder directamente a la clase `Driver` puesto que se accederá a la misma a través de la API JDBC. Esta API enviará las peticiones al Driver, que será, quién en último término, acceda a la base de datos.

Establecer una conexión

Para comunicar con una base de datos utilizando JDBC, debemos en primer lugar establecer una conexión con la base de datos a través del driver JDBC apropiado. El API JDBC especifica la conexión en la interfaz `java.sql.Connection`.

El siguiente código muestra un ejemplo de conexión JDBC a una base de datos MySQL:

```
Connection connection;
String url              = "jdbc:mysql:Bank";
String login            = "root";
String password         = "nadiuska";

try {
     connection = DriverManager.getConnection(url,
                                              login,
                                              password);

     // Acceso a datos utilizando el objeto de conexión
     . . .
} catch (SQLException sqle) {
     // Tratar el error aquí
} finally {
     try {
          connection.close();
     } catch (SQLException e) {
          // Tratar el error aquí
     }
}
```

En este ejemplo, la clase `DriverManager` intenta establecer una conexión con la base de datos `Bank` utilizando el driver JDBC que proporciona MySQL. Para poder acceder al RDBMS MySQL es necesario introducir un *login* y un *password* válidos.

En el API JDBC, hay varios métodos que pueden lanzar la excepción `SQLException`. En este ejemplo, la conexión se cierra al final del bloque `finally`, de modo que los recursos del sistema puedan ser liberados independientemente del éxito o fracaso de cualquier operación de base de datos.

Crear y ejecutar instrucciones SQL

Antes de poder ejecutar una sentencia SQL, es necesario obtener un objeto de tipo `Statement`. Una vez creado dicho objeto, podrá ser utilizado para ejecutar cualquier operación contra la base de datos.

El siguiente método crea un objeto `Statement`, que podemos utilizar para enviar instrucciones SQL a la base de datos.

```
Statement createStatement() throws SQLException
```

La finalidad de un objeto `Statement` es ejecutar una instrucción SQL que puede o no devolver resultados. Para ello, la interfaz `Statement` dispone de los siguiente métodos:

- `executeQuery()`, para sentencias SQL que recuperen datos de un único objeto `ResultSet`.
- `executeUpdate()`, para realizar actualizaciones que no devuelvan un `ResultSet`. Por ejemplo, sentencias DML SQL (Data Manipulation Language) como `INSERT`, `UPDATE` y `DELETE`, o sentencias DDL SQL (Data Definition Language) como `CREATE TABLE`, `DROP TABLE` y `ALTER TABLE`. El valor que devuelve `executeUpdate()` es un entero (conocido como la cantidad de actualizaciones) que indica el número de filas que se vieron afectadas. Las sentencias que no operan en filas, como `CREATE TABLE` o `DROP TABLE`, devuelven el valor cero.

Ejemplo: Crear la tabla `BANK`

Para ilustrar el API JDBC, consideraremos la clase `CreateTableBank`. Esta clase nos ofrece los métodos `initialize()` y `close()` para establecer y liberar una conexión con la base de datos.

El método `createTableBank` crea la tabla `BANK`, utilizando para ello un objeto de tipo `Statement`. Sin embargo, dado que el método `executeUpdate()` ejecuta una sentencia SQL de tipo `CREATE TABLE`, ésta no actualiza ningún registro de la base de datos y por ello este método devuelve cero. En caso de ejecutar una sentencia de tipo `INSERT`, `UPDATE` o `DELETE`, el método devolvería el número de filas que resultasen afectadas por el cambio.

```
public class CreateTableBank {

     String driver          = "com.mysql.jdbc.Driver";
     String url             = "jdbc:mysql:Bank";
     String login           = "root";
     String password        = "nadiuska";
     String createTableBank =
                      "CREATE TABLE BANK (" +
                      "client VARCHAR(100) NOT NULL, "  +
                      "password VARCHAR(20) NOT NULL, " +
                      "balance Integer NOT NULL, "      +
                      "PRIMARY KEY(client))";
     Connection connection  = null;
     Statement  statement   = null;

     public void initialize() throws
          SQLException, ClassNotFoundException {

          Class.forName(driver);
          connection =  DriverManager.getConnection(url,
                                                    login,
                                                    password);

     }

     public void createTableBank() throws SQLException {

          statement = connection.createStatement();
          statement.executeUpdate(createTableBank);

     }

     public void close() throws SQLException {

         try {
             connection.close();
```

```
        } catch (SQLException e) {
            throw e;
        }

    }

}
```

Una vez creada la tabla BANK, el siguiente paso podría ser la introducción en la misma de los datos de los clientes. Si dichos datos están disponibles en fichero, el código para leerlos e introducirlos en la base de datos sería el siguiente:

```
public void insertData() throws SQLException, IOException {

    String client, password;
    int balance;
    BufferedReader br = new BufferedReader(
        new FileReader("clients.txt"));

    try {
        do {
            client = br.readLine();
            password = br.readLine();
            balance = Integer.parseInt(br.readLine());
            String sqlString =
                "INSERT INTO BANK VALUES('"
                + client + "','" + password + "','"
                + balance + "')";
            statement.executeUpdate(sqlString);
            statement.close();
        } while (br.readLine() != null);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        br.close();
    }
}
```

El formato del archivo de entrada es: nombre del cliente, clave de acceso y saldo, introducidos en líneas separadas, y seguidos de una línea separatoria como se muestra a continuación:

```
Iván Samuel Tejera Santana
nadiuska
1000000
-------------------------
```

En el código anterior, la única sentencia relevante es el método `statement.executeUpdate()`, invocado para insertar datos en la tabla BANK (dicho método devuelve el número de registros insertados).

Qué es una prepared statement?

Una Prepared Statement es una sentencia SQL de base de datos precompilada. Al estar precompilada, su ejecución será más rápida que una SQL normal, por lo que es adecuada cuando vamos a ejecutar la misma sentencia SQL (con distintos valores) muchas veces. Por ejemplo, estas dos inserciones MySQL normales

```
mysql> INSERT INTO person VALUES (null, 23, 'Pedro', 'Perez');
mysql> INSERT INTO person VALUES (null, 33, 'Rodrigo', 'Rodriquez');
```

Desde consola MySQL podrían hacerse por medio de una Prepared Statement de esta forma

```
mysql> PREPARE insertar FROM "INSERT INTO person VALUES (null,?,?,?)";

mysql> SET @edad=23;
mysql> SET @nombre='Pedro';
mysql> SET @apellido='Perez';
mysql> EXECUTE insertar USING @edad,@nombre,@apellido

mysql> SET @edad=33;
mysql> SET @nombre='Rodrigo';
mysql> SET @apellido='Rodriguez';
mysql> EXECUTE insertar USING @edad,@nombre,@apellido;

mysql> DEALLOCATE PREPARE insertar;
```

Uso desde java

Para poder usar una Prepared Statement real desde java, es necesario que tanto la base de datos como el driver java que usemos soporten las Prepared Statement. Independientemente de que lo soporten o no, nosotros podremos hacer el código java usando Prepared Statement, aunque si la base de datos o el driver no lo soportan, no obtendríamos la ventaja de la eficiencia.

Por ejemplo, MySQL sí soporta los Prepared Statement y los driver modernos de SQL para java también lo soportan. Sin embargo, debemos habilitarlos en el momento de establecer la conexión

```
Connection conexion = DriverManager.getConnection(
      "jdbc:mysql://servidor/basedatos?useServerPrepStmts=true",
      "usuario", "password");
```

Fíjate en que en la URL de conexión hemos añadido un parámetro `?useServerPrepStmts=true`. Para otras bases de datos deberás ver cómo es la cadena de conexión que habilite el uso de Prepared Statement si no lo están por defecto.

En la actualidad esta activado por defecto.

Código java

Para usar una Prepared Statement desde java, el código puede ser como este

```
PreparedStatement psInsertar = conexion.prepareStatement(
            "insert into person values (null,?,?,?)");

psInsertar.setInt(1, 23); // La edad, el primer interrogante, es un entero.
psInsertar.setString(2, "Pedro"); // El String nombre es el segundo interrogante
psInsertar.setString(3, "Perez"); // Y el tercer interrogante, un String
apellido.
psInsertar.exequteUpdate(); // Se ejecuta la inserción.
...
psInsertar.setInt(1, 12); // La edad, el primer interrogante, es un entero.
psInsertar.setString(2, "Juan"); // El String nombre es el segundo interrogante
psInsertar.setString(3, "Lopez"); // Y el tercer interrogante, un String
apellido.
psInsertar.exequteUpdate(); // Se ejecuta la inserción.
```

Símplemente ponemos interrogantes donde irán los valores concretos que vayamos a insertar. Cada interrogante se identifica luego con un número, de forma que el primer interrogante que aparece es el 1, el segundo el 2, etc. Luego, con los métodos `set()` correspondientes rellenamos esos valores.

El primer parámetro es el número del interrogante y el segundo el valor que queremos insertar. Finalmente llamamos a `executeUpdate()`. Luego, con la misma `PreparedStatement`, ponemos otros valores volviendo a llamar a los métodos `set()` y volvemos a llamar a `executeUpdate()`

Fíjate que a la hora de poner los interrogantes, no nos hemos preocupado de poner comillas en los valores de texto ni conversiones de ningún tipo. Las llamadas a `setInt()`, `setString()`, ya hacen todas las conversiones adecuadas para nosotros.

No es necesario verificar las cadenas

Y el que esta conversión sea automática es una ventaja adicional si los datos los ha introducido el usuario. Como comentamos en transacciones con base de datos, si usamos una `Statement` normal, nosotros tenemos que componer la cadena con la SQL con algo como esto

```
String sql = "insert into person values (null, " +
   edad + ",'" + nombre + "','" + apellido + "')";
```

y si algún desalmado mete en su nombre una comilla simple, como O'Donnell, entonces la cadena SQL quedaría

```
insert into person values (null, 23, 'O'Donnell', 'Perez')
```

que está mal construida por parecer que el nombre el 'O' y lo de Donnell que hay detrás no está bien construido en SQL. La ejecución daría un error.

Por ello, si los datos los va a introducir un usuario, es buena idea usar `PreparedStatement`, aunque no vayamos a usar la eficiencia real de las `PreparedStatement`. La otra opción es que nosotros con código verifiquemos la validez de esos datos.

Consultar la base de datos

El objeto `Statement` devuelve un objeto `java.sql.ResultSet` que encápsula los resultados de la ejecución de una sentencia `SELECT`. Ésta interfaz es implementada por los vendedores de drivers. Dispone de métodos que permiten al usuario navegar por los diferentes registros que se obtienen como resultado de la consulta.

El siguiente método, `executeQuery`, definido en la interfaz `java.sql.Statement` le permite ejecutar las instrucciones `SELECT`:

```
public ResultSet executeQuery (String sql) throws SQLException
```

La interfaz `java.sql.ResultSet` ofrece varios métodos para recuperar los datos que se obtienen de realizar una consulta:

```
getBoolean()
getInt()
getShort()
getByte()
getDate()
getDouble()
getfloat()
```

Todos estos métodos requieren el nombre de la columna (tipo `String`) o el índice de la columna (tipo `int`) como argumento. La sintaxis para las dos variantes del método `getString()` es la siguiente:

```
public String getString(int columnIndex) throws SQLException
public String getString(String columnName) throws SQLException
```

Regresando a la clase `CreateTableBank`, creemos un nuevo método `queryAll()` que recupere todos los datos de la tabla `BANK`.

```
public void queryAll() throws SQLException {

     String sqlString =
       "SELECT client, password, balance" +
       "FROM BANK";
     Statement statement = connection.createStatement();
     ResultSet rs = statement.executeQuery(sqlString);

     while (rs.next()) {
         System.out.println(rs.getString("client") +
                            rs.getString("password") +
                            rs.getInt("balance"));
     }

}
```

Este método crea un objeto `Statement`, utilizado para invocar el método `executeQuery()` con una instrucción SQL (`SELECT`) como argumento. El objeto `java.sql.ResultSet` contiene todas las filas de la tabla `BANK` que coinciden con la instrucción `SELECT`. Utilizando el método `next()` del objeto `ResultSet`, podemos recorrer todas las filas contenidas en el bloque de resultados. En cualquier fila, podemos utilizar uno de los métodos `getXXX()` descritos anteriormente para recuperar los campos de una fila.

La interfaz `ResultSet` también permite conocer la estructura del bloque de resultados. El método `getMetaData()` ayuda a recuperar un objeto `java.sql.ResultSetMetaData` que tiene varios métodos para describir el bloque de resultados, algunos de los cuales se enumeran a continuación:

```
getTableName()
getColumnCount()
getColumnName()
getColumnType()
```

Tomando un bloque de resultados, podemos utilizar el método `getColumnCount()` para obtener el número de columnas de dicho bloque. Conocido el número de columnas, podemos obtener la información de tipo asociada a cada una de ellas.

Por ejemplo, el siguiente método imprime la estructura del bloque de resultados:

```
public void getMetaData() throws SQLException {

    String sqlString = "SELECT * FROM BANK";
    Statement statement = connection.createStatement();
    ResultSet rs = statement.executeQuery(sqlString);
    ResultSetMetaData metaData = rs.getMetaData();
    int noColumns = metaData.getColumnCount();
```

```
    for (int i=1; i<noColumns+1; i++) {
        System.out.println(metaData.getColumnName(i)
                           + " " +
                           metaData.getColumnType(i));
    }

}
```

El método anterior obtiene el número de columnas del bloque de resultados e imprime el nombre y el tipo de cada columna. En este caso, los nombres de columna son `client`, `password` y `balance`. Observe que los tipos de columna son devueltos como números enteros. Por ejemplo, todas las columnas de tipo `VARCHAR` retornarán el entero 12, las del tipo `DATE`, 91. Estos tipos son constantes definidas en la interfaz `java.sql.Types`. Fíjese también en que los números de columnas empiezan desde 1 y no desde 0.

Transacciones

Si hay una propiedad que distingue una base de datos de un sistema de archivos, esa propiedad es la capacidad de soportar transacciones. Si está escribiendo en un archivo y el sistema operativo cae, es probable que el archivo se corrompa. Si está escribiendo en un archivo de base de datos, utilizando correctamente las transacciones, se asegura que, o bien el proceso se completará con éxito, o bien la base de datos volverá al estado en el que se encontraba antes de comenzar a escribir en ella.

Cuando múltiples instrucciones son ejecutadas en una única transacción, todas las operaciones pueden ser realizadas (convertidas en permanentes en la base de datos) o descartadas (es decir, se deshacen los cambios aplicados a la base de datos).

Cuando se crea un objeto `Connection`, éste está configurado para realizar automáticamente cada transacción. Esto significa que cada vez que se ejecuta una instrucción, se realiza en la base de datos y no puede ser deshecha. Los siguientes métodos en la interfaz `Connection` son utilizados para gestionar las transacciones en la base de datos:

```
void setAutoCommit(boolean autoCommit) throws SQLException
void commit() throws SQLException
void rollback() throws SQLException
```

Para iniciar una transacción, invocamos `setAutoCommit(false)`. Esto nos otorga el control sobre lo que se realiza y cuándo se realiza. Una llamada al método `commit()` realizará todas las instrucciones emitidas desde la última vez que se invocó el método `commit()`. Por el contrario, una llamada `rollback()` deshará todos los cambios realizados desde el último `commit()`. Sin embargo, una vez se ha emitido una instrucción `commit()`, esas transacciones no pueden deshacerse con `rollback()`.

Consideremos un caso práctico de empresa consistente en crear un pedido, actualizar el inventario y crear registro de envíos. La lógica de empresa puede requerir que todo sea efectivo o que falle.

El fracaso de la creación de un registro de envíos puede dictar que no cree un pedido. En tales casos, los efectos de las instrucciones SQL correspondientes a las dos primeras tareas (crear un pedido y actualizar el inventario) deben deshacerse.

El siguiente fragmento de código ilustra esta situación:

```
Connection connection = null;
```

```
// Obtener una conexión
...
try{
     // Iniciar una transacción
     connection.setAutoCommit(false);

     Statement statement = connection.createStatement();

     // Crear un pedido
     statement.executeUpdate(
          ``INSERT INTO ORDERS(ORDER ID,
          ``PRODUCT ID,...) VALUES(...)");

     // Actualizar el inventario
     statement.executeUpdate("UPDATE TABLE INVENTORY " +
                             "SET QUANTITY = QUANTITY-1 "
                             + ``WHERE PRODUCT ID = ...");

     // Crear un registro de envíos
     if (...) {
          // Operación éxitosa
          statement.execute(
               "INSERT INTO SHIP RECORD(...) " +
               "VALUES (...)");
          connection.commit();
     } else {
          // Deshacer operación
          connection.rollback();
     }
} catch (SQLException) {
     // Manejar excepciones aquí
} finally {
     // Cerrar instrucción y conexión
}
```

En este fragmento de código, una vez invocado el método `rollback()`, la base de datos restaura las tablas `ORDERS` e `INVENTORY` a su estado anterior. `commit()` convierte en permanentes los cambios efectuados por las instrucciones `INSERT` y `UPDATE`.

Programación en tres Capas con java

La programación de software por capas, que es una arquitectura en la que se separa el código o lógica que hace tareas de negocios (facturar, ventas) de la lógica de presentación gráfica y de datos. Modelo MCV, (vista, controlador, modelo).

La **programación de software por capas** es una arquitectura en la que buscamos separar el código o lógica que hace tareas de negocios (facturar, ventas) de la lógica de presentación gráfica y de datos. También le conocer como modelo MCV, (vista, controlador, modelo).

La ventaja de este estilo es que el desarrollo es la facilidad de reutilización y mantenimiento, ya que en caso de algún cambio, solo se modifica la capa necesaria sin tener que revisar todo el código.

Identificando las capas

Capa lógica de presentación

Hace referencia a como se va a presentar la información del programa al usuario. El objetivo es separar todo aquellos que se muestra al usuario, esta capa no tiene conexión a base de datos, ni realizar operaciones de ningún tipo solo muestra datos en pantalla, la capa de presentación solicita mediante funciones que se ejecutan en la capa de la lógica de negocio.

Capa de lógica de negocio

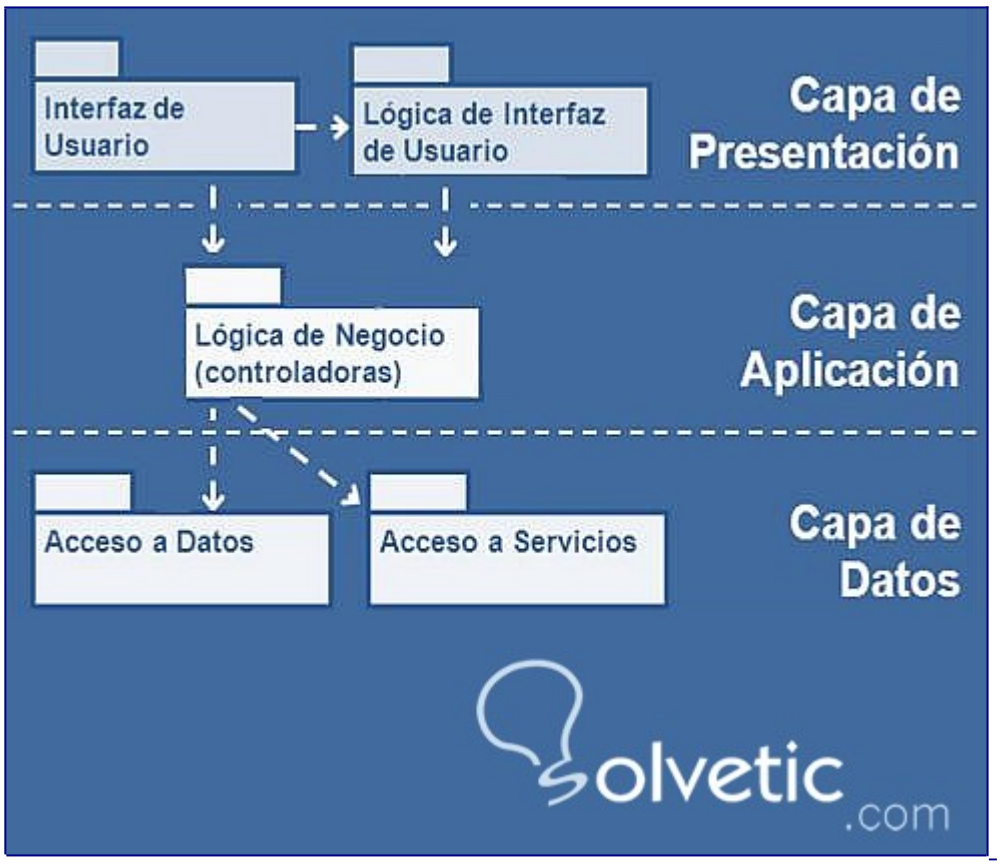
Aquí es donde se encuentran las funciones clases y funciones o procedimientos que serán invocados a través de la interfaz gráfica.

Recibe peticiones o eventos del usuario, procesa esas peticiones y luego envía la respuesta a la interfaz gráfica, si es necesario esta capa se comunicara con la capa de datos, pero la capa de negocios no se conecta a la base de datos, solo recibe datos o los procesa. Aquí se ejecutan e invocan reglas o funciones de negocios por ejemplo, facturar, listar productos, etc.

Capa de datos

Aquí tendremos clases y funciones que se conectan a la base de datos y es donde se realizan transacciones con sql para leer, insertar, modificar o eliminar información en la base de datos.

Aquí ejecutaremos consultas sql de forma que ninguna de las otras capas saben donde esta la base de datos, así la capa de presentación podría estar en un pc y las otras capas en un servidor como servicio se software Saas.



A modo ejemplo realizaremos una aplicación en java con tres capas:

Ventajas y Desventajas

La programación en capas es una metodologia que permite trabajar con total libertad, no es una técnica rígida que debe implementarse de una manera estructurada. Los desarrolladores de software e incluso de proyectos web tienen múltiples formas de implementarlas según sus necesidades y poder reutilizar muchas librerías de código desarrollado en capas, se puede implementar una capa en otro proyecto. La tendencia a utilizar el modelo de programación en N capas en grande proyecto con varios equipos de desarrolladores y cuando se trata principalmente de aplicaciones empresariales donde
se deben manejar gran cantidad de subsistemas y módulos.

Algunas desventajas de este modelo de programación es cuando se implementa se debe llegar a un balance entre la cantidad capas, recursos, subsistemas o subcapas, clases y sus interrelaciones. Debemos documentar muy bien un desarrollo en capas y debe ser fácilmente comprensible suficiente para realizar un trabajo específico con eficiencia y ser lo más modular e independiente posible, para que un cambio no afecte a todo el sistema.

Dentro de las desventajas tenemos que analizar la pérdida de eficiencia si se comienza a hacer trabajo redundante o se programan varias capas que hacen lo mismo pero con distintos datos o conexiones constantes a la base de datos sin sentido que ralentizan la aplicación además de consultas sql mal optimizadas que pueden volver el software muy lento.

Se debe evitar también caer en el error de generar mucha dependencia entre los objetos, métodos, propiedades y atributos de cada clase y cada capa que justamente contradice el objetivo de la programación en 3 o N capas.

Algunas tecnologías actuales que soportan la Programación en Capas. En la actualidad, la mayoría de los programas hacen uso de .Java, PHP y Net.

Microsoft Visual Studio.NET
Este framework implementa la posibilidad de programacion en N capas tanto en web como software de escritorio. La capa de datos se implementa mediante herramientas como Data Set y Data Reader.

Java
La capa de presentación se implementa por median te las herramientas JSP, Servlets, tambien en desarrollo de escritorio con las que se construyen y conectan las interfaces gráficas con la capa de lógica del negocio.

La capa de datos se implementa mediante herramientas como Data Set y Data Reader. Muy similar a C# en .net
La otra herramienta muy utilizada en Java es Hibernate, que se utiliza para mapear bases de datos con objetos creados en Java algo como lo que vimos en el ejemplo con la clase vehiculos salvo que el mapeo lo hicimos construyendo la clase en forma manual.