CAPÍTULO 6
Servidores de Base de Datos y la Distribución

Patricia Bazán

Podemos deducir de los temas desarrollados hasta ahora en las clases que sin NOS y un mecanismo de comunicación que cuente con al menos el nivel de transporte, es imposible crear la ilusión de tener un sistema totalmente transparente de la distribución.

Desde el punto de vista de las aplicaciones no se puede hablar de cliente/servidor hoy sin tener en cuenta a los servidores de base de datos SQL o basados en SQL.

Las bases de datos SQL o servidores SQL son el modelo dominante para crear aplicaciones cliente/servidor. Varios productos comerciales han evolucionado y perfeccionado su funcionamiento en ese sentido.

La Figura 47 muestra una arquitectura cliente/servidor clásica en dos niveles. El servidor SQL acepta peticiones SQL como mecanismo de comunicación y retorna como respuesta un conjunto de resultado que satisface la condición fijada en la consulta.

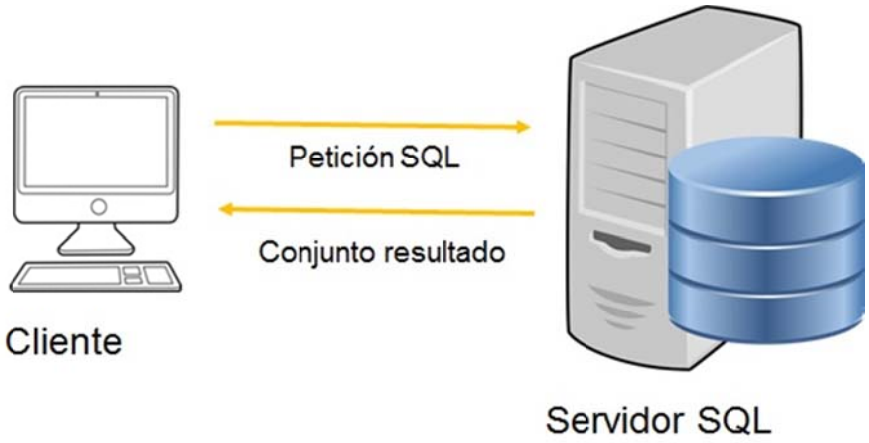

Fig. 47. Arquitectura Cliente/Servidor Clásico

El capítulo se organiza de la siguiente manera: en la Sección 1 se repasan algunos fundamentos del lenguaje SQL y bases de datos relacionales. En la Sección 2 se clasifican los motores de bases de datos SQL según su modo de almacenar datos o de procesar requerimientos. Luego, en la Sección 3 se analiza el impacto de procedimientos almacenados y disparadores en el modelo de distribución en dos niveles. La Sección 4 describe el manejo de transacciones en una base de datos como medio para asegurar la concurrencia de un sistema distribuido. La Sección 5 define el uso de transacciones distribuidas y en la Sección 6 se presentan variantes a dicha distribución mediante monitores de transacciones. Finalmente la Sección 7 arriba a algunas conclusiones.

6.1. Fundamentos de SQL y las bases de datos relacionales

El SQL como lenguaje de manipulación, definición y control de datos ha favorecido en gran medida la construcción de servidores de bases de datos de cualquier tamaño. SQL es un estándar ISO está orientado a los conjuntos, posee muy pocos comandos y fue creado para bases de datos que adhieren al modelo relacional.

Orígenes del SQL

El manejo de bases de datos que adhieren al modelo relacional fue creado por Codd en 1970 y se creó SQL como lenguaje de consulta front-end para la base de datos relacional prototipo System R. Es lenguaje orientado, al inglés, con firmes y sólidas raíces matemáticas se fundamenta en la teoría de conjuntos y el cálculo de predicados.

Con SQL se puede manipular un conjunto de datos que cumplen una determinada condición de recuperación.

Por otro lado, el modelo relacional hace una clara abstracción del almacenamiento físico de datos a través de las tablas, compuestas por filas y columnas. El modelo nos libera de concentrarnos en detalles relacionados con la forma en que está almacenada la información permitiendo un acceso puramente lógico.

Con SQL solo se especifican las tablas, las columnas y las filas involucradas en la operación. Además el SQL se ha transformado en el lenguaje predominante de mainframes, minicomputadoras y LAN servers. Las herramientas clientes SQL hacen una industria horizontal donde se mezclan herramientas front-end con servidores back-end.

¿Qué hace el SQL?

El lenguaje SQL se usa para realizar operaciones de datos complejas que insumiría cientos de líneas de código de un lenguaje tradicional.

- Los roles que cumple el SQL son:
- SQL es un lenguaje de consulta interactivo. Fue originalmente diseñado para usuarios finales y hoy se ve mejorado por las herramientas visuales para construcción de sentencias SQL sin escribir ningún comando.
- SQL es un lenguaje de programación de bases de datos. Puede estar embebido en otros lenguajes de programación para acceder a los datos o usar una conjunto de API como interface (X/Open). Por otra parte cada vendedor (Sybase, Oracle) proveen su propio dialecto de SQL para la programación.
- SQL es un lenguaje de definición y administración (manipulación) de datos. El lenguaje de definición de datos se utiliza para crear tablas simples, objetos complejos,

índices, vistas, restricciones de integridad referencial, seguridad y control de acceso. Todos los objetos definidos con SQL son almacenados en un diccionario conformado a su vez por tablas denominadas el catálogo del sistema.

- SQL ayuda a proteger los datos en un ambiente multiusuario. Provee excelentes características para validación de datos, control de integridad referencial, manejo de transacciones como unidades únicas de trabajo, cerramientos (*locks*) automáticos y detección de *deadlocks*.

La Figura 48 representa los distintos roles que puede cumplir el SQL como mecanismo de comunicación.

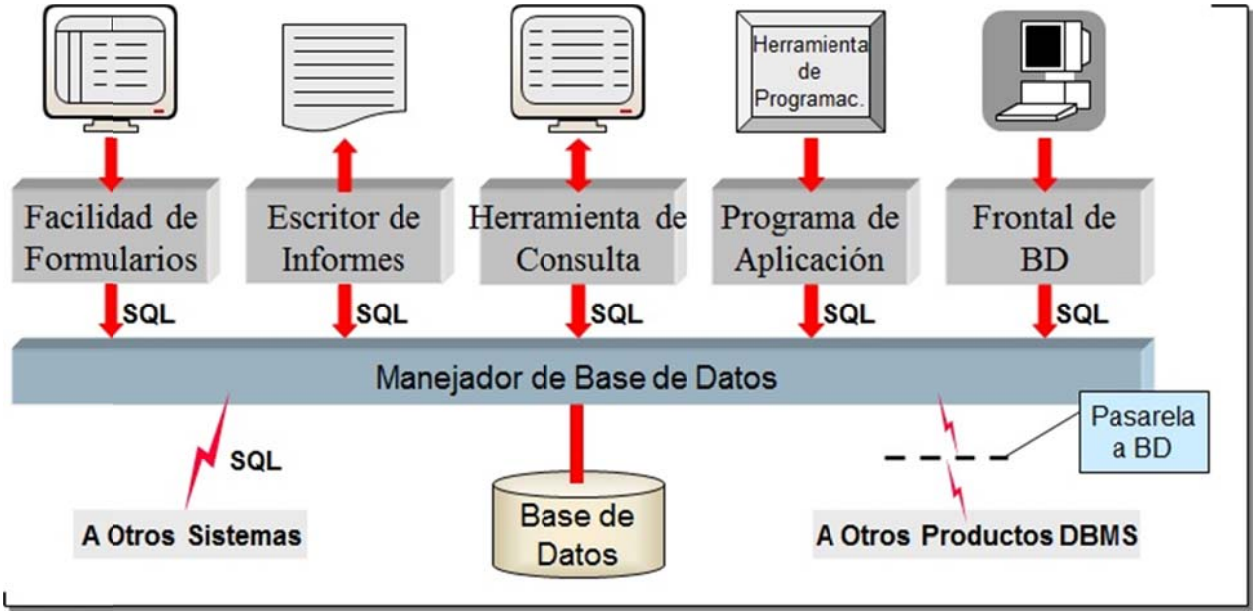

Fig. 48. Distintos Roles del SQL

¿Qué hace un servidor SQL?

En la arquitectura de aplicaciones cliente/servidor centradas en base de datos, una aplicación cliente requiere datos y servicios relacionados con ellos a un servidor de datos. El servidor de base de datos, normalmente llamado motor SQL, responde a los requerimientos del cliente y provee acceso seguro a datos compartidos. Una aplicación cliente puede recuperar datos y modificar un conjunto de datos con una petición simple.

- Un servidor SQL tiene las siguientes funciones:
- Controla y ejecuta comandos SQL.
- Provee una visión lógica y física de los datos.
- Genera planes optimizados de ejecución de los comandos SQL.
- Provee herramientas de administración y mantienen tablas de catálogos dinámicas de los objetos almacenados en dichos catálogos.
- Maneja la recuperación, concurrencia, seguridad y consistencia de la base de datos.
- Controla la ejecución de transacciones.

- Obtiene y libera *locks* durante la ejecución de la transacción en curso.
- Protege la base de datos de accesos no autorizados.

¿Qué es un servidor SQL?

Generalmente un servidor SQL es un híbrido del SQL estándar y el dialecto propio de vendedor.

Los vendedores de bases de datos ponen un gran esfuerzo en extender el SQL para otorgar nuevas funcionalidades más allá del modelo relacional fundamentalmente en lo relacionado con extensiones procedurales para los distintos sistemas operativos distribuidos, monitores de transacciones, bases de datos de objetos y administradores de objetos distribuidos.

6.2. Arquitecturas de Servidores SQL

La arquitectura de los servidores SQL pueden clasificarse conceptualmente desde dos puntos de vista: desde el punto de vista del almacenamiento y desde el punto de vista del procesamiento.

Arquitectura de Base de Datos Desde el Punto de Vista del Almacenamiento

Esta clasificación tipifica los servidores SQL según cómo organizan el repositorio físico de la información que gestionan y almacenan y se pueden encontrar do clases de variantes: 1- base de datos única - todos los objetos se almacenan en un mismo dispositivo físico o disco - y 2- base de datos múltiple - cada conjunto de objetos conforman una base de datos que ocupa cada una un dispositivo físico diferente -.

Las Figuras 49 y 50 muestran ejemplos de ambas arquitecturas.

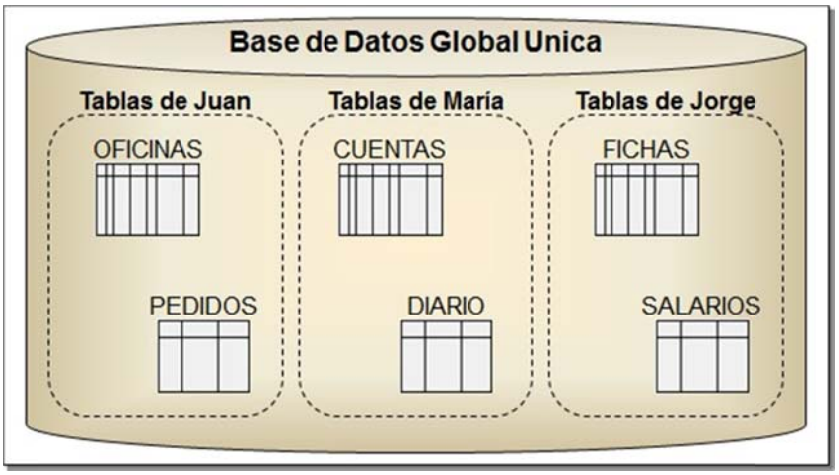

Fig. 49. Arquitectura de Base de Datos Única

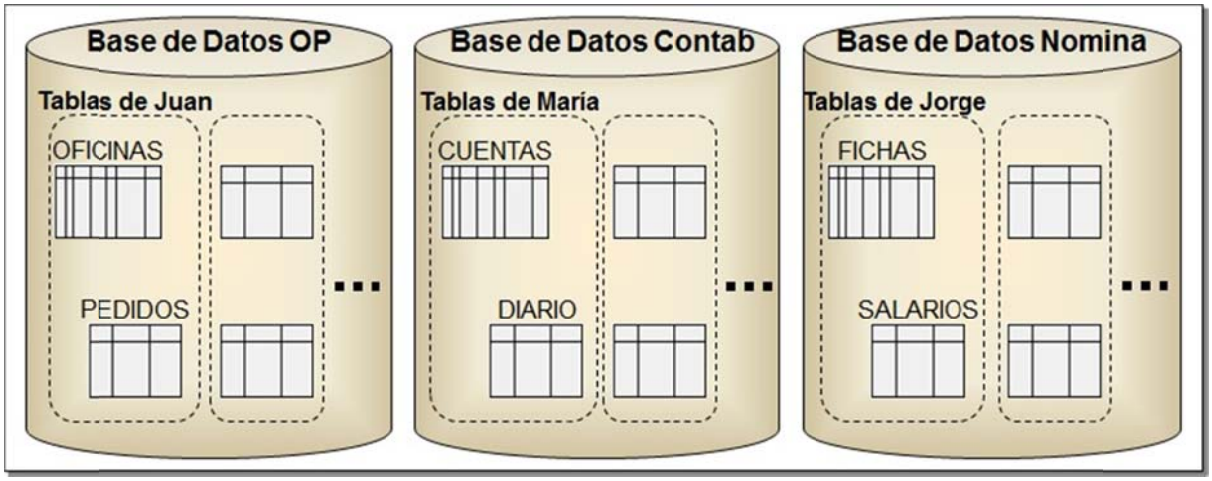

Fig. 50. Arquitectura de Base de Datos Múltiple.

Cada tipo de arquitectura posee sus ventajas y desventajas. En el caso 1, las tablas entre distintos sistemas de información pueden referenciarse fácilmente y también de manera más eficiente, debido a que el dispositivo de almacenamiento es único. El caso 2, por su parte, torna más complejo la referenciación de objetos que no son propios de su base de datos y además podría provocar el aislamiento de la información si no existe un buen diseño de datos. Por ejemplo, tablas con el mismo contenido (provincias de Argentina), pueden existir en dos bases de datos distintas con el mismo nombre, duplicando innecesariamente la información.

Por otra parte, el caso 1 resulta desventajoso y hasta ineficiente a la hora de realizar copias de resguardo y/o restaurar debido al crecimiento desmedido del dispositivo que almacena todos los objetos de la base de datos. Además, al ser un dispositivo único se debe realizar la copia del dispositivo completo cada vez, aunque haya información que no se modificó o su modificación es esporádica. El caso 2, por otra parte, facilita la administración de copias y su restauración pudiendo seleccionar diferentes frecuencias para cada dispositivo y además, al ser de menor tamaño, el copiado y restauración es más rápido.

Arquitectura de Base de Datos Desde el Punto de Vista del Procesamiento

Esta clasificación tipifica a los servidores de base de datos según la manera en que atienden las peticiones que provienen de sus clientes. En definitiva, según la manera en que resuelven el acceso concurrente y se pueden encontrar tres variantes: 1- proceso por cliente - a cada cliente que hace una petición se le asigna un proceso o espacio de memoria específico, 2- multihilada - se asigna un único proceso para atención de requerimientos y cada cliente ejecuta un hilo (thread) diferente y 3- híbrida - el servidor escucha múltiples hilos pero no pasa las peticiones directamente sino que cuenta con un componente intermedio (dispatcher) que encola las peticiones y administra su distribución balanceando la carga de trabajo.

Cada tipo de servidor, en este caso posee ventajas y desventajas:

Arquitectura Proceso por Cliente (Ej: DB2, Informix)	
Ventajas	Desventajas
Protege a los usuarios entre sí y al mismo motor de ejecución de fallas ajenas.	Consume más memoria y CPU
Cada proceso puede asignarse a un procesador si el hardware admite multiprocesamiento.	Degrada la performance debido al *overhead* que produce los cambios de contexto.

Arquitectura Multihilada (Ej: SQL Server)	
Ventajas	Desventajas
No requiere cambios de contexto y es por lo tanto más eficiente.	Un programa de usuario con errores puede provocar la caída del servidor completo
Las implementaciones de estos servidores son más portables porque no dependen del sistema operativo	La preemción de tareas necesarias para administrar los hijos nunca será realizada mejor que si se hiciese a nivel de sistema operativo.

Arquitectura Hibrida (Ej: Oracle a partir de versión 7)	
Ventajas	Desventajas
Restringe el acceso del espacio de cada usuario evitando la asignación permanente de recursos.	El balanceo de carga realizado por el dispatcher nunca será tan eficiente como un monitor de transacciones.

6.3. Procedimientos Almacenados, Disparadores y Reglas: su impacto en el Cliente/Servidor.

Las bases de datos relacionales cuentan hoy con extensiones procedurales como lo son los procedimientos almacenados, disparadores y reglas - En inglés, stored procedures, triggers y rules, respectivamente - que son extremadamente útiles pero no estándar. Los puristas del modelo relacional no ven con buenos ojos la existencia de estos mecanismos porque aseguran que van contra la esencia del modelo pues hacen que se pierda el sentido declarativo del lenguaje de los datos relacionales. Además tienen la contra de ser, en general, poco portables dado que se escriben en un dialecto de SQL propio del vendedor.

Pero más allá de todo, estas extensiones constituyen la primera aproximación al procesamiento distribuido dado que permite codificar procesos dentro del mismo motor de base de datos.

¿Qué es un stored procedure?

Muchos vendedores de base de datos ofrecen hoy un mecanismo de programación tipo RPC sobre la base de datos. Este mecanismo se conoce en general como *TP lite* o *stored procedures* (procedimientos almacenados).

Un procedimiento almacenado es una colección de sentencias SQL y lógica procedural (sentencias de control) que son compiladas, verificadas y almacenadas en el servidor de base de datos.

Un procedimiento almacenado es un objeto más de la base de datos y es almacenado en el catálogo junto a los demás (tablas, índices, vistas, etc.).

Un procedimiento almacenado acepta parámetros de entrada y puede ser usado a través de la red por múltiples clientes. El resultado de la ejecución es retornado y se gana enormemente en eficiencia y reducción de tráfico en la red.

El concepto de procedimiento almacenado fue introducido por Sybase en 1986 para mejorar la performance del SQL sobre redes. Son utilizados para asegurar reglas de negocios e integridad de datos, para realizar mantenimiento del sistema y también para administración pero la verdadera razón de ser es proveer inteligencia al servidor de base de datos y agregar funcionalidad de la lógica de las aplicaciones. Son óptimos para procesamiento de transacciones en línea (OLTP) donde básicamente se hace:

1. Recepción de un conjunto fijo de entradas desde clientes remotos.
2. Realización de múltiples comandos SQL previamente compilados contra la base de datos local.
3. Cumplimiento del trabajo (Commit).
4. Retorno de un conjunto fijo de resultados

Los procedimientos almacenados también proveen abstracción respecto de su sitio pues las modificaciones a las estructuras de datos son transparentes a las aplicaciones remotas que sigue solicitando el servicio sin importar cómo está implementado.

Para comprender las ventajas del uso de procedimientos almacenados, es importante tener presente las siguientes definiciones.

SQL dinámico y estático

El SQL estático son sentencias definidas en el código y compiladas en el momento de su definición. Si los objetos que referencia no existen, no compila.

El SQL dinámico es creado en tiempo de ejecución ofrecen más flexibilidad a expensas de velocidad de ejecución

SQL embebido y programado

SQL embebido son sentencias SQL que están incorporadas al código fuente del programa, entremezcladas con las otras sentencias del lenguaje de programación

SQL programado son las sentencias SQL se almacenan como objetos de la base de datos (triggers,Stored procedure, reglas)

Los procedimientos almacenados son un ejemplo de código SQL estático y programado, pero el SQL embebido también puede ser estático si las sentencias se compilan junto con el programa fuente en el que se encuentran o dinámico, si se incorporan a programas como cadenas de caracteres que son evaluadas en tiempo de ejecución.

A su vez, un procedimiento almacenado también puede ejecutar sentencias SQL de manera dinámica (es decir, evaluadas en tiempo de ejecución) utilizando la cláusula *dynamic*.

La Figura 51 muestra distintos aspectos evaluados en tres variantes, con procedimientos almacenados, con SQL embebido estático y con SQL embebido dinámico.

	Procedimientos almacenados	SQL embebido estático	SQL embebido dinámico
Nombres para las funciones	SI	NO	NO
Funciones compartidas	SI	NO	NO
Parametros de E/S	SI	NO	NO
Catalogado	SI	SI	NO
Logica procedural	En el objeto	Externa	Externa
Flexibilidad	Baja	Baja	Alta
Nivel de abstraccion	Alto	Bajo	Bajo
Estandarizacion	NO	SI	SI
Performance	Rapido	Mediano	Lento
Trafico	1 Request/Reply por varios comandos SQL	1 Request/Reply por cada comando SQL	1 Request/Reply por cada comando SQL

Fig. 51. Características de Funcionamiento de los Procedimientos Almacenados Respecto del SQL embebido.

Triggers y reglas

Un trigger o disparador, es una acción especial definida por el usuario (usualmente en la forma de procedimiento almacenado) que son automáticamente invocadas cuando ocurre un evento relacionado con los datos de la base de datos.

Una regla es un caso especial de trigger que se usa para chequeos simples sobre los datos.

Ambos están ligados a operaciones específicas sobre tablas específicas y realizan generalmente tareas relacionadas con auditoría de los datos, seteo de columnas por defecto y también control de integridad (de entidad o referencial).

Los triggers se habilitan por las operaciones de DELETE, INSERT y UPDATE sobre las tablas para las cuales están definidos, pueden llamar a otros triggers, e incluso puede generarse una recursión. Se diferencian de los procedimientos almacenados en que no tienen una invocación explícita sino que se disparan ante la ejecución de la operación para la que se hubieren definido.

Los procedimientos almacenados, los triggers o disparadores y las reglas constituyen código ejecutable que se cataloga y usa dentro del mismo servidor de base de datos, permitiendo que éste pueda incorporar capacidad de procesamiento y mejorar el balanceo de carga entre cliente y servidor para alojar parte de la lógica de la aplicación que de otro modo sólo podría ejecutarse en el cliente, debido a que en este modelo de distribución de dos niveles o capas no existe un componente que la aloje.

6.4. Manejo de transacciones en una base de datos

Los servidores de base de datos SQL heredan de dicho lenguaje la posibilidad de asegurar el acceso concurrente a los recursos mediante la definición de transacciones.

Una transacción en un servidor de base de datos SQL es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo para el servidor, quien asume que todas las sentencias deben completarse para asegurar la consistencia e integridad de los datos. Esto es posible debido a que SQL cuenta con las primitivas de lenguaje COMMIT y ROLLBACK.

Una transacción se inicia con cualquier sentencia SQL y finaliza exitosamente con COMMIT (que a su vez inicia un nueva transacción) o falla con ROLLBACK.

La Figura 52 muestra los tres escenarios posibles de finalización de una transacción estándar.

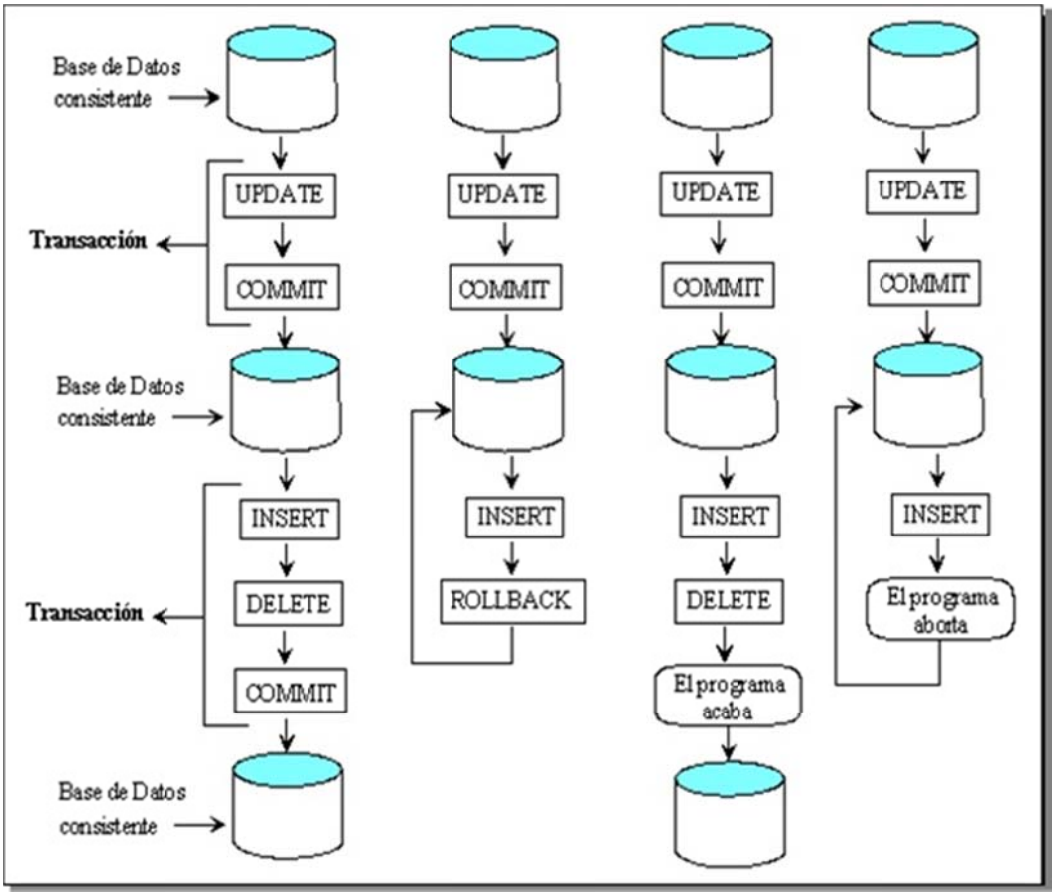

Fig. 52. Ejecución de Transacciones Estándar

Transacciones en el procesamiento multiusuario

Según Edgar Frank Codd, el creador de las bases de datos relacionales:

"Durante un transacción el usuario verá un vista completamente consistente de la base de datos. El usuario nunca verá modificaciones no cumplimentadas de otros usuarios, e incluso los cambios cumplimentados efectuados por otros, no afectarán a los datos examinados por el usuario en mitad de una transacción"

Este principio se basa en el concepto de aislamiento de las transacciones pero también requiere un mecanismo adicional que permita asegurarlo. Este mecanismo se logra estableciendo niveles de cerramiento. Los niveles de cerramiento pueden fijarse sobre toda la base de datos (lo que implicaría que la misma se transformaría en un recurso monousuario perdiendo la capacidad de acceso concurrente) o a nivel de sus distintos objetos.

Las alternativas, conceptualmente serían:

- Cerramiento a nivel de tabla: ningún usuario puede acceder a una tabla utilizada por otro usuario hasta que éste último libere el recurso. Si la tabla es muy grande esto podría provocar una larga cola de usuarios en espera, siendo que quizás no quieran acceder todos a la misma porción de la tabla.
- Cerramiento a nivel de fila de una tabla: ningún usuario puede acceder a la fila o conjunto de filas utilizadas por otro usuario hasta que éste último libere el recurso.

Este esquema de cerramiento es incierto dado que depende del conjunto de filas *cerrado* por el usuario.

La mayoría de las implementaciones de servidores de base de datos definen su propia unidad de cerramiento (generalmente denominadas *páginas*) con un tamaño fijo medido en KBytes y *cierran* el acceso a usuarios concurrentes esa porción de datos.

A su vez, para aumentar el acceso concurrente, se utilizan esquemas con más de un tipo de cierre, siendo habitual en tal sentido implementar al menos dos tipos de cierre: Cierre Compartido y Cierre Exclusivo.

Para comprender el funcionamiento, supongamos dos transacciones A y B y veamos si puede o no acceder una a los recursos cerrados por la otra, según cada una de ellas haya fijado un cierre compartido o exclusivo. La Figura 53 muestra lo que sucede en cada caso.

Transacción A		Transacción B		
	Acción	**No Cerrado**	**Cierre Compartido**	**Cierre Exclusivo**
	No Cerrado	SI	SI	SI
	Cierre Compartido	SI	SI	NO
	Cierre Exclusivo	SI	NO	NO

Fig. 53. Acceso permitido o restringido según tipo de cierre de las transacciones A y B

6.5. Modelos de Procesamiento de Transacciones. Transacciones Distribuidas

El concepto de transacción tal como se ha enunciado funciona adecuadamente cuando el servidor de base de datos SQL se encuentra centralizado y es el único que ejecuta las transacciones. Pero este modelo resulta insuficiente cuando la transacción debe ejecutarse en más de un servidor (transacción distribuida) o bien cuando, por su naturaleza, la transacción debe ser cumplimentada parcialmente debido a, por ejemplo, su longitud.

En cualquiera de estos casos, las transacciones planas resultan poco adecuadas o insuficientes para lograr el objetivo.

Transacciones Encadenadas

Una transacción encadenada introduce un concepto de control lineal para secuenciar transacciones. La forma más fácil de implementarla es con puntos de guarda dentro de una transacción plana que permite salvar cambios hasta el *commit* definitivo. Si se produce un *roll-back* se hace hasta el último punto de guarda, pero si el sistema se cae (*crash*) se pierden todos los puntos de guarda, es decir no cumplen la propiedad de Durabilidad.

La Figura 54 muestra el funcionamiento de transacciones encadenadas con puntos de guarda. Estos funcionan como COMMIT parciales pero no dividen la transacción en sub-transacciones.

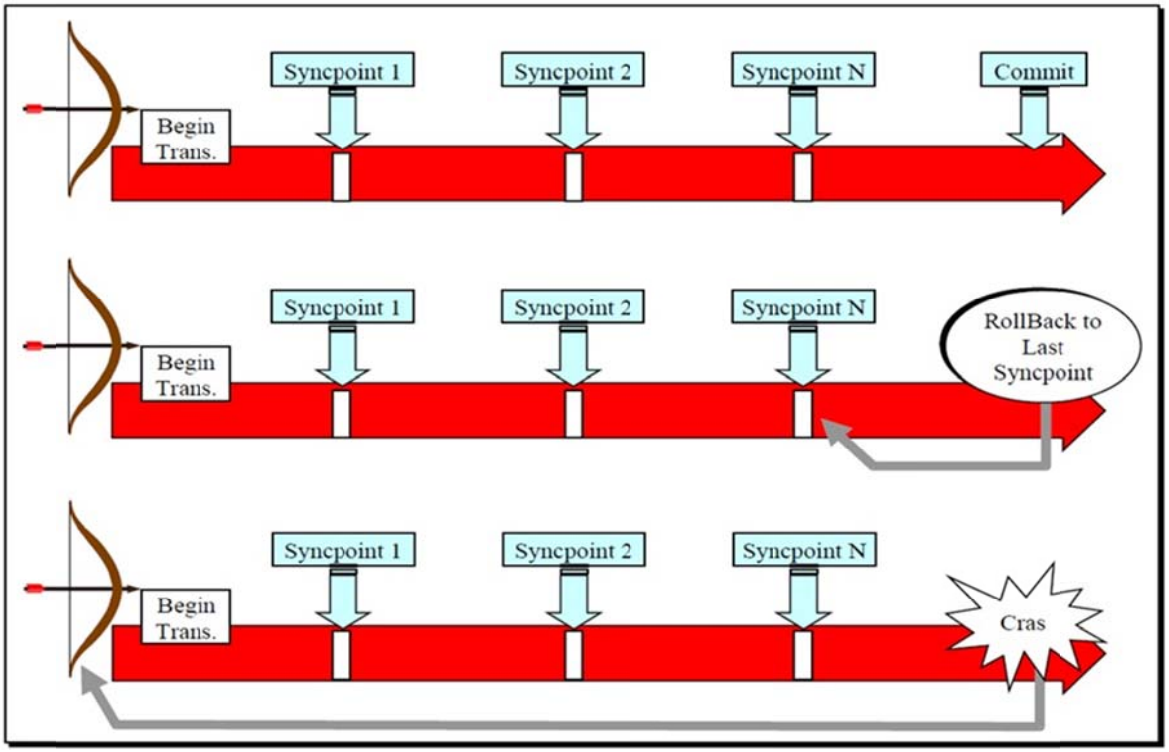

Fig. 54. Transacciones Encadenadas con Puntos de Guarda

Transacciones Anidadas

Las transacciones anidadas definen transacciones dentro de transacciones. Conservan la misma semántica que los procedimientos de un lenguaje de programación, estableciendo una relación jerárquica de sub-transacciones.

La transacción principal comienza las sub-transacciones y cada una de ellas puede hacer commit y rollback de sus piezas de trabajo locales. Cuando una sub-transacción termina, los resultados están sólo disponibles para el padre.

El commit de una sub-transacción se hace permanente después del commit local y el de todos sus ancestros.

La Figura 55 muestra el funcionamiento de las transacciones anidadas.

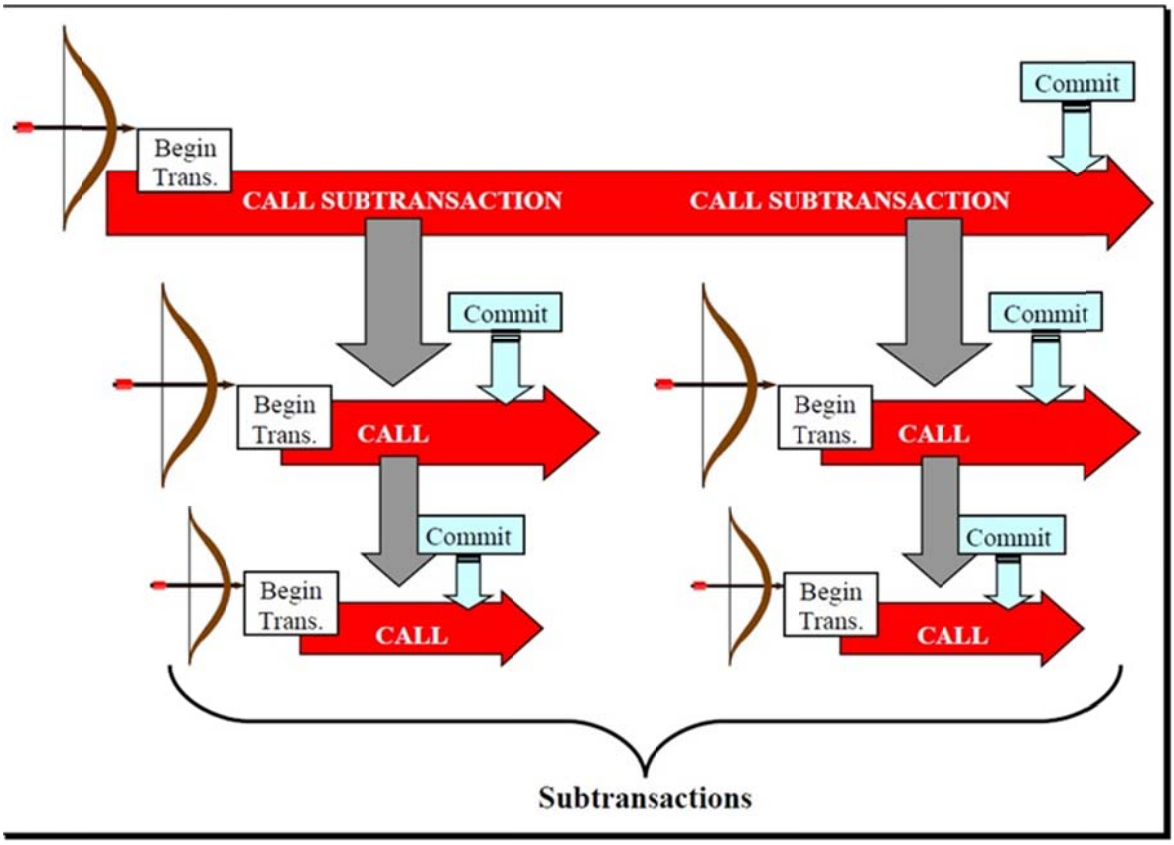

Fig. 55. Funcionamiento de Transacciones Anidadas

6.6. Monitores de Transacciones (TP Monitors). TP-Lite o TP-Heavy

Los monitores de transacciones son un concepto que apareció ya en los viejos mainframes brindando un ambiente de ejecución robusto y permitieron soportar aplicaciones OLTP (On Line Transaction Processing) a gran escala.

Con la migración de las OLTP a plataformas Cliente/Servidor clásico surgió la necesidad de introducir nuevamente la idea de monitores de transacciones y surgieron muchos productos comerciales para tal fin.

Los monitores de transacciones administran las transacciones desde su punto de origen a través de uno o más servidores. Cuando la transacción termina se garantiza un estado consistente del sistema. Además de ejecutar transacciones, los monitores de transacciones encaminan las mismas a través de la red, balancean la carga de su ejecución y las recuperan después de fallas. Se comportan como un sistema operativo para las transacciones y aseguran la posibilidad de contar con transacciones distribuidas.

TP-Lite y TP-Heavy

TP-Lite es simplemente la integración de funciones de un monitor de transacciones en los motores de base de datos (a través de procedimientos almacenados y Triggers).

TP-Heavy son los monitores de transacciones propiamente dichos que realizan la administración, balanceo de carga, sincronización y monitoreo de las transacciones. (Ejemplos: CICS, Tuxedo y Jaguar CTS).

A continuación analizamos distintas características y su comportamiento en TP-Lite y en TP-Heavy.

Alcance del Commit

En TP-Lite los procedimientos almacenados son definidos en un dialecto de SQL y almacenados como un objeto dentro de la base de datos, constituyendo una unidad transaccional pero no pudiendo participar con otras unidades transaccionales respecto de una transacción global. Si A llama a B, B es otro unidad transaccional y luego A muere, los commit hechos por B no pueden ser deshechos luego de la muerte de A, violando el *todo o nada*. Por lo tanto la única solución es que A y B estén en la misma transacción generando una unidad demasiado grande.

Por su parte, los procesos TP-Heavy son escritos usando lenguajes procedurales estándar (no SQL) permitiendo lograr el *todo o nada* y el concepto de transacción global.

Administración de recursos heterogéneos

Un procedimiento almacenado en TP-Lite solo puede persistir transacciones que afecten a recursos propios del vendedor de la base de datos. Los procedimientos TP-Heavy pueden persistir las transacciones sobre múltiples recursos de distinto tipo dentro del alcance de una transacción.

Administración de procesos

Un proceso almacenado en TP-Lite es invocado, se ejecuta transaccionalmente y puede quedar en memoria temporal para reusos futuros.

Un proceso TP-Heavy es manejado como una clase de servidor pudiendo manejar prioridades, utilizar barreras (firewalls) para que otros procesos no los interfieran y si la clase muere, la transacción puede ser reasignada a otra clase.

Invocaciones Cliente/Servidor

Una invocación a un procedimiento es totalmente no estándar, se realiza a través del mecanismo de RPC provisto por el vendedor.

En TP-Heavy el mecanismo de interacción es cualquiera (RPC o mensajes).

Performance

Los procedimientos almacenados poseen un mayor rendimiento en cuanto a tráfico de red respecto del SQL estático o embebido, pero son incapaces de mejorar la performance en otros aspectos, como por ejemplo el balanceo de carga.

TP-Heavy que tiene código precompilado permite multiplexar los requerimientos del cliente.

6.7 - Conclusiones del Capítulo

El capítulo presenta un análisis detallado del uso e impacto de los servidores de base de datos SQL en la construcción de aplicaciones distribuidas.

El lenguaje SQL, además de ser un poderoso lenguaje de consulta se transforma en un mecanismo de comunicación válido hacia los servidores de bases de datos. Estos a su vez han ampliado sus prestaciones hacia modelos distribuidos de más de dos niveles y además han adquirido suficiente versatilidad para comportarse como componentes de ejecución que permite alojar la lógica de negocio que hasta el momento sólo podía ser ejecutada por los clientes.