UNIDAD DE TRABAJO 1: ALMACENAMIENTO DE LA INFORMACIÓN

TEMA 3: BASES DE DATOS CENTRALIZADAS. BASES DE DATOS DISTRIBUIDAS

3.1. INTRODUCCIÓN

La arquitectura de un sistema de base de datos está influenciada en gran mediad por el sistema informático subyacente en el que se ejecuta la base de datos. En la arquitectura de un sistema de base de datos se reflejan aspectos como la conexión en red, el paralelismo y la distribución.

- La conexión en red de varios ordenadores permite que algunas tareas se ejecuten en un sistema servidor y que otras se ejecuten en los sistemas clientes. Esta división de trabajo ha conducido al desarrollo de sistemas de bases de datos *cliente-servidor.*
- El procesamiento paralelo dentro del ordenador permite acelerar las actividades del sistema de base de datos, proporcionando a las transacciones unas respuestas más rápidas, así como la capacidad de ejecutar más transacciones por segundo.
- La distribución de datos a través de las distintas sedes o departamentos de una organización permite que estos datos residan donde han sido generados o donde sean necesarios, pero continuar siendo accesibles desde otros lugares o departamentos diferentes. El hecho de guardar varias copias de la base de datos en diferentes sitios permite que puedan continuar las operaciones sobre la base de datos aunque algún sitio se vea afectado por algún desastre. Se han desarrollado los sistemas de bases de datos distribuidos para manejar datos distribuidos geográficamente o administrativamente a lo largo de múltiples sistemas de bases de datos.

3.2. SISTEMAS CENTRALIZADOS

Los sistemas de bases de datos centralizados son aquellos que se ejecutan en un único sistema informático. Tales sistemas comprenden el rango desde los sistemas de BD monousuario ejecutándose en ordenadores personales hasta los sistemas de bases de datos de alto rendimiento ejecutándose en grandes sistemas.

Un ordenador de propósito general consiste en un una unidad (o unas pocas unidades) de procesamiento y un número determinado de controladores para los dispositivos que se encuentran conectados a un bus común, el cual proporciona acceso a la memoria compartida. Las UCP poseen memorias caché locales donde se almacenan copias de ciertas partes de la memoria para acelerar el acceso a los datos. Cada controlador de dispositivo se encarga de un tipo específico de dispositivo (p.e. una unidad de disco, una tarjeta de sonido, o un monitor). La UCP y los controladores de dispositivos pueden estar ejecutándose concurrentemente, compitiendo así por el acceso a memoria. La memoria caché reduce la disputa por el acceso a memoria, ya que la UCP necesita acceder a la memoria compartida un número de veces menor.

Se distinguen dos formas de utilizar los ordenadores: como sistemas monousuario o multiusuario.

Un *sistema monousuario* típico es una unidad de sobremesa utilizada por una única persona que dispone de una sola UCP, de un disco (o dos) y que trabaja con un solo S.O. que solo permite un único usuario.

Un *sistema multiusuario* tiene más discos, más memoria, puede disponer de varias UCP y trabaja con un S.O. multiusuario. Se encarga de dar servicio a un gran número de usuarios que están conectados a través de terminales o de PC. Estos sistemas se denominan *servidores.*

En relación con las BD, los ordenadores de hoy en día de propósito general disponen de unos pocos procesadores (tienen un paralelismo de *grano grueso)* que comparten la misma memoria principal. Las bases de datos que se ejecutan en estas máquinas no intentan dividir una consulta simple entre los distintos procesadores, sino que ejecutan cada consulta en un único procesador, posibilitando la **concurrencia** de varias consultas. Así, estos sistemas permiten ejecutar un mayor número de transacciones por segundo.

Por el contrario, las máquinas paralelas de *grano fino* tienen un gran número de procesadores y los sistemas de bases de datos que se ejecutan sobre ellas intentan paralelizar las tareas simples (consultas, por ejemplo).

3.3. SISTEMAS CLIENTE-SERVIDOR

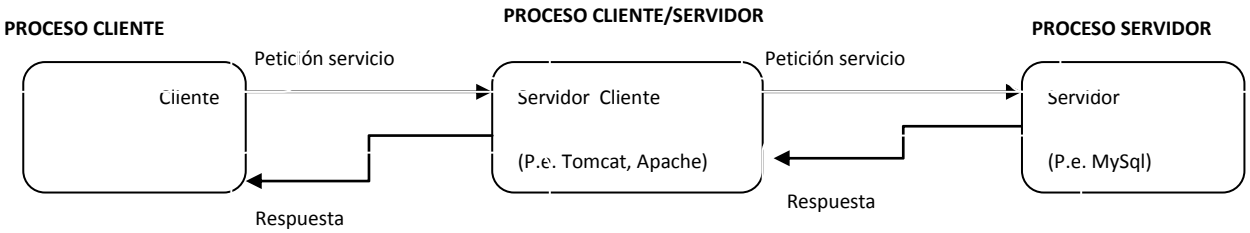
En la década de los años 60 el procesamiento de la información se realizaba con mainframes, máquinas grandes y costosas, principalmente utilizadas por las grandes empresas para soportar sus necesidades de negocio. En la década de los 70, los miniordenadores y el tiempo compartido incrementaron la accesibilidad del poder de procesamiento, pero aún así, éste seguía siendo centralizado en máquinas individuales.

A principio de los 80 comenzaron a aparecer los primeros ordenadores personales (PC) con capacidad de procesamiento de bajo coste que rápidamente se propagaron por las empresas. Esto provocó la gestión de datos de manera local que dio lugar a la aparición de multitud de islas de datos y, como consecuencia, surgieron problemas de actualización y de integridad de datos. La tecnología cliente/servidor (C/S) surge en esta década ante las necesidades de un control centralizado de los datos así como la posibilidad de su acceso generalizado.

El concepto de C/S hace referencia a la conexión de ordenadores por medio de una red para descentralizar el procesamiento y utilizar fuentes de datos centralizadas. El modelo cliente/servidor es un concepto que sirve para describir las comunicaciones entre procesos que se clasifican como consumidores de servicios (clientes) y proveedores de servicio (servidores).

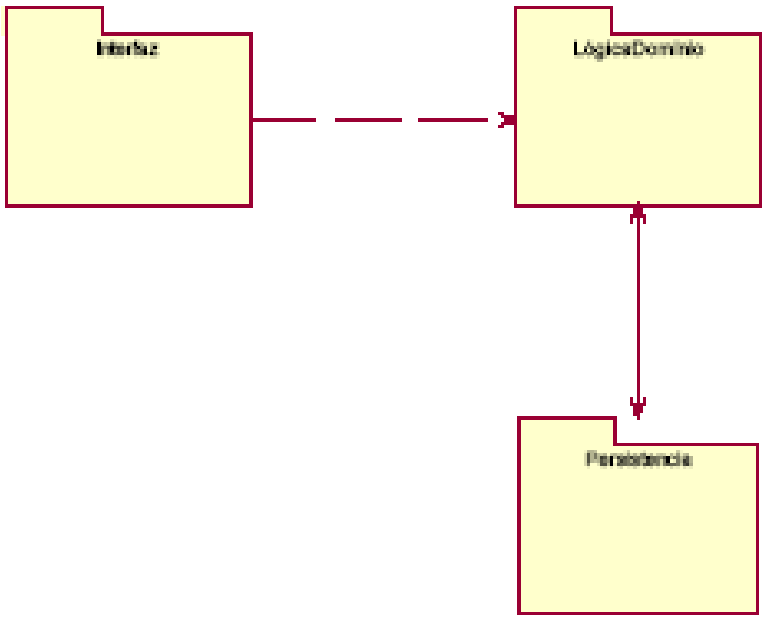
Un **cliente** es un proceso (programa en ejecución) que envía una petición a un proceso servidor solicitándole la realización de una tarea o servicio. Los clientes o programas clientes normalmente gestionan aspectos de la interfaz de usuario (por ejemplo, presentación de datos solicitados al servidor, validación de datos de entrada y, en algunos casos, lógica de negocio). Sin embargo no todo cliente necesariamente debe residir en la máquina en donde se presente la interfaz de usuario. Un cliente puede ser un proceso que reside en un servidor y realiza peticiones a otro proceso servidor en una misma máquina servidora. (Por ejemplo, un servidor de aplicaciones Tomcat (Linux, Windows, Solaris (Unix)) atacando a un servidor de base de datos: por ejemplo un servlet (que es una clase de java) que corre en el Tomcat se conecta vía JDBC a un servidor de base de datos (Oracle, MySql, SQL Server…) para insertar una transacción.

Un proceso **servidor** (el término "servidor" se usa tanto para el proceso que responde a una petición de un cliente como a la máquina que responde a la petición), es un proceso que responde a la petición de un proceso cliente para realizar la tarea solicitada. Normalmente ejecutan acciones sobre bases de datos en consulta y actualizaciones, gestionan la integridad de datos y los devuelven al cliente que los solicita. Estos procesos pueden residir en cualquier servidor conectado a la red.



Como vemos, los procesos clientes y servidores pueden residir en una máquina con capacidad de procesamiento o en varias máquinas conectadas en red.

La lógica de negocio de un sistema puede dividirse en diferentes tipos de procesamiento. Por ejemplo, la *lógica de presentación*, encargada de presentar la interfaz de usuario, la *lógica de negocio*, encargada de procesar la funcionalidad del sistema en cuestión, y la *lógica de datos*, encargada de devolver y actualizar los datos.



Para referirnos a la evolución de los sistemas C/S normalmente se suele hacer referencia al número de niveles físicos y al grado de acoplamiento entre los distintos tipos de procesamiento. De este modo, las arquitecturas de **un nivel físico** se corresponden con una máquina que aglutina toda la lógica de presentación, negocio y datos de una forma fuertemente acoplada como es el caso de los mainframes.

La primera generación de sistemas cliente/servidor fue de **dos niveles físicos**. Esta arquitectura se orientaba a la conexión de ordenadores personales (PC) con servidores

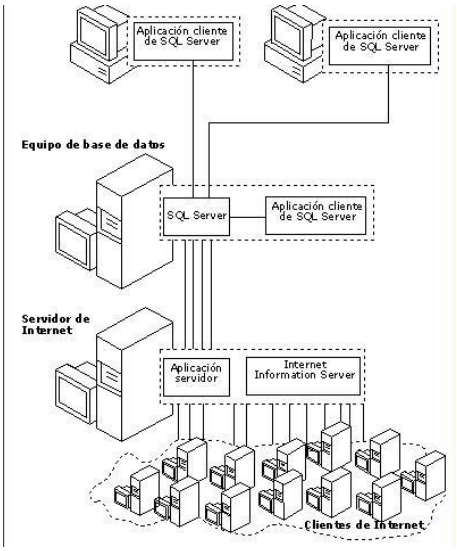
conectados a una red. El caso más común de este tipo de arquitectura es el de las que presentan la lógica de presentación y la lógica de negocio en la máquina cliente y la lógica de datos en el servidor (por ejemplo, una arquitectura C/S que la capa de presentación se realice en un entorno visual, como Visual Basic, Delphi, etc. y se realice el acceso a base de datos a través de drivers específicos como ODBC).

En función del grado de procesamiento un cliente (o máquina cliente) es ligero (Thin Client) cuando el procesamiento se limita a la lógica de presentación o pesado (Fat Client o Thick Client) cuando el procesamiento incluye, además, lógica de negocio.

Este tipo de arquitecturas se siguen utilizando muy frecuentemente en la actualidad, pero siguen presentando un fuerte acoplamiento entre los tipos de procesamiento, lo que produce una serie de inconvenientes:

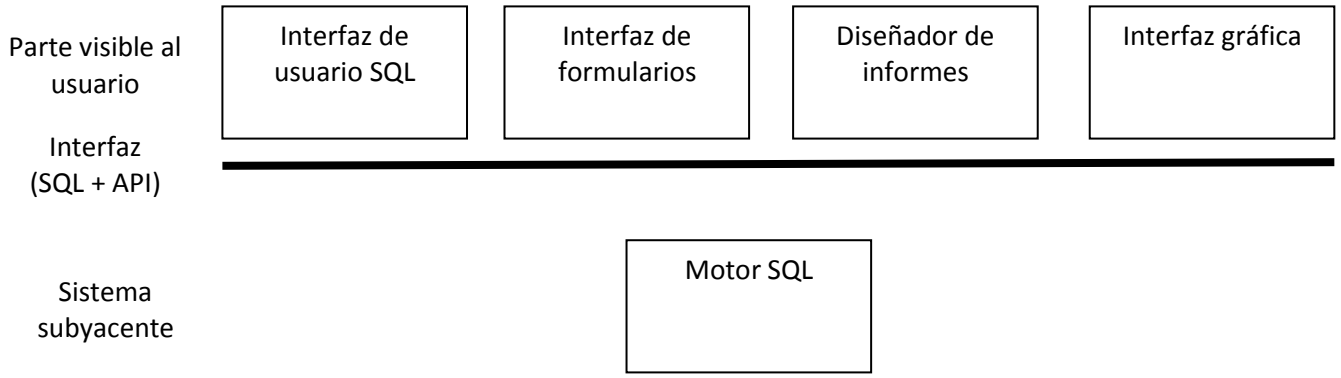
- **Falta de flexibilidad de los sistemas**: debido al acoplamiento existente entre las aplicaciones (escritas en el lado del cliente) con la BD, la modificación de ésta hace necesaria la modificación de todas las aplicaciones asociadas. Esta desventaja se hace crítica en sistemas que evolucionan rápidamente por medio de nuevas funcionalidades, lo que da lugar a que la carga de mantenimiento sea creciente.
- **Dificultad de migración**: es dificultosa la migración del sistema por la dependencia del fabricante del mecanismo de persistencia (SGBD, sistemas de directorios, etc.). La migración de un mecanismo de persistencia a otro puede ser una actividad larga y tediosa dependiendo del tipo de aplicación.
- **Problemas de escalabilidad**: en la práctica se está comprobando que este tipo de arquitecturas no es recomendable para sistemas que necesitan un alto número de usuarios interactivos. Para escalar cientos o miles de usuarios es aconsejable moverse a arquitecturas de tres niveles.
- **Dificultad de reutilización de componentes**: este problema se debe al acoplamiento existente entre el nivel de presentación (interfaz de usuario) y los SGBD.
- **Dificultad de actualización del software**: realizamos una aplicación en Visual Basic, por ejemplo, y la instalamos en 200 equipos de una fábrica. Se detecta un fallo y se saca una nueva versión del software. En esta arquitectura, habría que reinstalar la aplicación nueva (la actualización) en todos los equipos, tarea muy tediosa.

Con el objetivo de paliar esta problemática aparece una nueva generación de sistemas C/S centrada en independizar los tipos de procesamiento entre sí mediante interfaces definidas, lo que facilita el despliegue de los tipos de procesamiento en diferentes máquinas. Así surge la arquitectura C/S de **tres niveles (Three Tier)**, en donde una máquina cliente realiza peticiones a una máquina servidora y ésta a su vez a otros servidores para satisfacer la petición original. Esta arquitectura se caracteriza por el hecho de que los niveles lógicos descritos anteriormente son independientes pudiendo, cada nivel lógico, desplegarse en una máquina física diferente. Un ejemplo de arquitectura a tres niveles puede ser una máquina cliente que realiza peticiones a un servidor de aplicaciones y ésta, a su vez, hace peticiones a un servidor de base de datos.



Se habla también de Arquitecturas **n-Tier** en donde cada nivel físico se responsabiliza de una función del sistema. Un ejemplo podría ser una arquitectura de cuatro niveles físicos en el que es encuentra un cliente ligero (por ejemplo un PC con un navegador web), un servidor web, un servidor de lógica de negocio y un servidor de bases de datos.

Con respecto a las Bases de Datos, la funcionalidad de una BD se puede dividir, a grandes rasgos, en dos partes: la parte visible al usuario y el sistema subyacente. El sistema subyacente gestiona el acceso a las estructuras, la evaluación y optimización de consultas, el control de concurrencia y la recuperación. La parte visible al usuario de base de datos está formada por herramientas como *formularios, diseñadores de informes y facilidades gráficas de interfaz de usuario.* La interfaz entre la parte visible al usuario y el sistema subyacente puede ser SQL o una aplicación.



3.4. ALMACENAMIENTO DISTRIBUIDO DE DATOS

Los sistemas distribuidos de bases de datos consisten en emplazamientos poco acoplados que no comparten componentes físicos. Más aún, los sistemas de bases de datos que se ejecutan en cada emplazamiento pueden tener un grado significativo de independencia mutua.

Cada emplazamiento puede participar en la ejecución de transacciones que tienen acceso a los datos en uno o varios emplazamientos. La diferencia principal entre los sistemas de bases de datos centralizados y los distribuidos es en que los primeros los datos residen un una sola ubicación mientras que en los últimos los datos residen en varias ubicaciones. Esta distribución de los datos es causa de muchas dificultades en el procesamiento de transacciones y consultas. Veremos cómo tratar estas dificultades.

Consideremos una relación *r* que deba guardarse en una base de datos. Hay varios enfoques del almacenamiento de esta relación en una base de datos distribuida:

- **Réplica**: el sistema conserva varias réplicas (copias) idénticas de la relación. Cada réplica se guarda en un emplazamiento diferente, lo que da lugar a la duplicidad de datos.
- **Fragmentación**: la relación se divide en varios fragmentos. Cada fragmento se guarda en un emplazamiento diferente.
- **Réplica y fragmentación**: la relación se divide en varios fragmentos. El sistema conserva varias réplicas de cada fragmento.

3.4.1. Réplicas de datos

Si se replica la relación r, se guardará una copia de la misma en dos o más emplazamientos. En el caso más extremo se tendrá una réplica completa en la que se guarda una copia en cada emplazamiento del sistema.

La réplica presenta ventajas e inconvenientes:

- **Disponibilidad:** si falla uno de los emplazamientos que contienen la relación *r*, ésta aún se podrá encontrar en otro emplazamiento. Por tanto, el sistema puede seguir procesando las consultas que impliquen a *r*, a pesar del fallo.
- **Aumento del paralelismo:** en el caso de que la mayor parte de los accesos a la relación *r* sólo den por resultado la lectura de la misma, varios emplazamientos pueden procesar en paralelo la consultas que impliquen a *r*. Cuantas más replicas haya, mayor será la posibilidad de que el dato buscado se halle en el emplazamiento en que se esté ejecutando la transacción. Por tanto, la réplica de los datos minimiza el tráfico de datos entre los emplazamientos.
- **Aumento de la sobrecarga en las actualizaciones:** el sistema debe asegurarse de que todas las réplicas de la relación *r* sean consistentes. En caso contrario, puede dar lugar a resultados erróneos. Por tanto, siempre que se actualice *r*, la actualización debe extenderse a todos los emplazamientos que contengan réplicas. La consecuencia es un aumento en la sobrecarga. Por ejemplo, en un sistema bancario en el que la información sobre las cuentas se replica en varios emplazamientos, hay que asegurarse de que el saldo de una cuenta concreta coincida en todos los emplazamientos.

En general, la réplica mejora el rendimiento de las operaciones **leer** y aumenta la disponibilidad de los datos para las transacciones de sólo lectura. Sin embargo, las transacciones de actualización suponen una sobrecarga mayor. El control de las actualizaciones concurrentes de los datos replicados por varias transacciones es más complicado que cuando se utiliza un enfoque centralizado. Se puede simplificar la administración de las réplicas de la relación *r* escogiendo una de ellas como *copia principal de r.*

3.4.2. Fragmentación de datos

Si la relación *r* está fragmentada, se dividirá en cierto número de fragmentos $r_1, r_2, \ldots, r_n$. Estos fragmentos contienen suficiente información como para permitir la reconstrucción de la relación original *r*. Hay dos esquemas diferentes para fragmentar una relación: *fragmentación horizontal y fragmentación vertical.*

La *fragmentación horizontal* divide la relación asignando cada tupla de *r* a uno o varios fragmentos. La *fragmentación vertical* divide la relación descomponiendo el esquema R de la relación *r* de un modo especial.

Fragmentación horizontal: la relación *r* se divide en cierto número de subconjuntos, $r_1, r_2, \ldots, r_n$. Cada tupla de la relación *r* debe pertenecer al menos a uno de los fragmentos, de modo que se pueda reconstruir la relación original cuando sea necesario.

Nombre-sucursal	Número-cuenta	saldo
Guadarrama	c-305	1000
Guadarrama	c-226	640
Cercedilla	c-177	410
Cercedilla	c-402	20000
Guadarrama	c-155	120
Cercedilla	c-408	2246
Cercedilla	c-639	1500

Un fragmento se podrá definir como *una selección de la relación global r*. Por ejemplo, podemos dividirla en fragmentos cada uno de los cuales consiste en tuplas de cuentas que pertenecen a una sucursal concreta. Si, como en este caso, sólo tenemos dos sucursales, entonces sólo tendremos dos fragmentos diferentes:

Nombre-sucursal	Número-cuenta	Saldo
Guadarrama	c-305	1000
Guadarrama	c-226	640
Guadarrama	c-155	120

Nombre-sucursal	Número-cuenta	Saldo
Cercedilla	c-177	4100
Cercedilla	c-402	2000
Cercedilla	c-408	2246
Cercedilla	c-639	1500

$\mathbf{r = r_1 \cup r_2}$

Fragmentación vertical: implica la definición de varios subconjuntos de atributos $R_1, R_2, \ldots, R_3$

La fragmentación debe hacerse de modo que se pueda reconstruir la relación *r* a partir de los fragmentos tomando la reunión natural. Un modo de asegurar que la relación *r* pueda reconstruirse es incluir los atributos de la clave primaria de *R* en cada uno de los R_i. Normalmente se añade un atributo especial en cada uno de los R_i, se suele llamar id-tupla, es un valor único, por tanto es una clave candidata del esquema ampliado.

Nombre-sucursal	Número-cuenta	Nombre-cliente	saldo
Guadarrama	c-305	García	1000
Guadarrama	c-226	Cordero	640
Cercedilla	c-177	Cordero	410
Cercedilla	c-402	Obeso	20000
Guadarrama	c-155	Obeso	120
Cercedilla	c-408	Obeso	2246
Cercedilla	c-639	Badorrey	1500

Esquema1: (nombre-sucursal, nombre-cliente, id-tupla)
Esquema2: (nombre-sucursal, saldo, id-tupla)

El atributo de reunión es *id-tupla*. Este atributo no deberá ser visible a los usuarios, dado que es un mecanismo interno de la aplicación.

Nombre-sucursal	Nombre-cliente	id-cliente
Guadarrama	García	1
Guadarrama	Cordero	2
Cercedilla	Cordero	3
Cercedilla	Obeso	4
Guadarrama	Obeso	5
Cercedilla	Obeso	6
Cercedilla	Badorrey	7

Número-cuenta	saldo	id-cliente
c-305	1000	1
c-226	640	2
c-177	410	3
c-402	20000	4
c-155	120	5
c-408	2246	6
c-639	1500	7

3.5. TRANSPARENCIA DE LA RED

Hemos visto como una relación *r* puede guardarse de muchas maneras en un sistema distribuido de bases de datos. Es fundamental que el sistema minimice el grado en que los usuarios necesitan conocer la manera en que se ha guardado la relación. Los sistemas pueden ocultar los detalles de la distribución de los datos. Esto es lo que se denomina *transparencia de la red.*

Se consideran lo aspectos de la transparencia desde los puntos de vista de:

- La denominación de los elementos de datos.
- La réplica de los elementos de datos
- La fragmentación de los elementos de datos.
- La ubicación de los fragmentos y de las réplicas.

3.5.1. Denominación de los elementos de datos

Los elementos e datos (relaciones, fragmentos y réplicas) deben tener nombres únicos. En las bases de datos distribuidas hay que poner cuidado para asegurarse de que los emplazamientos no utilicen el mimo nombre para distintos elementos de datos.

1. Una solución es exigir a todos los nombres que se registren en un servidor de nombres central. Esta solución, sin embargo, presenta dos inconvenientes, en primer lugar el servidor de nombres puede convertirse en un cuello de botella y en segundo lugar, si el servidor de nombres se avería puede que no sea posible que funcione ningún emplazamiento del sistema distribuido.

2. Un enfoque alternativo es exigir que cada emplazamiento añada como prefijo su propio identificador de emplazamiento a todos los nombres que genere. Esta solución, sin embargo, no logra conseguir la transparencia de la red, ya que los identificadores de los emplazamientos están ligados a los nombres. Por tanto, puede que se haga referencia a la relación *cuenta* como *emplazamiento17.cuenta*.
   Para solucionar este problema se pueden usar *alias*, de esta forma el usuario puede referirse a los elementos de datos mediante nombres sencillos que el sistema traducirá a los nombres completos. Con los alias el usuario permanece ignorante de la ubicación física de los elementos de datos.

Cada réplica y cada fragmento de un elemento de datos debe tener también un nombre único. Se adopta el convenio de añadir como sufijo "*.f1*", "*.f2*", …, "*fn*" a los fragmentos de un elemento de datos y "*.r1*", "*.r2*", …, "*.rn*" a las réplicas. Por tanto podríamos encontrarnos con algo así:

emplazamiento17.cuenta.f3.r2

que hace referencia a la réplica 2 del fragmento 3 de *cuenta* e indica que este elemento lo generó el emplazamiento 17.

No es deseable que un usuario haga referencia a réplicas concretas de un elemento. Por el contrario, el sistema debe determinar la réplica a la que debe hacer referencia una petición de **lectura** y debe actualizar todas las réplicas ante una petición de **escritura**.

3.5.2. Transparencia y actualizaciones

Proporcionar transparencia a los usuarios que actualizan la base de datos es algo más difícil que hacerlo para los que se limitan a leerla. Los problemas principales son asegurar que todas las réplicas de un elemento de datos y que todos los fragmentos afectados se actualicen.

Si se realiza una actualización de una relación replicada, la actualización debe aplicarse a todas las réplicas. Este requisito supone un problema si hay acceso concurrente a la relación, dado que es posible que se actualice una réplica antes que otra.