Tema 1 - Arquitecturas Distribuidas

1. El problema: cuándo son preferibles las bases de datos distribuidas

En primer lugar, definimos un **proceso local** como aquel que utiliza datos generados en el mismo centro de actividad, y por consiguiente, un **proceso global**, será aquel que utilice datos procedentes de más de un centro.

Por otra parte, vamos a hacer una distinción entre lo que es un **dato** y lo que es un **valor**. De una forma poco formal, los datos son básicamente los campos o atributos de las tablas de la base de datos, y los valores, lo que contiene cada tupla (son especificaciones de los datos). Cabe mencionar que los datos constituyen la **parte intensional** de una base de datos (porque es poco probable que cambie), y los valores, la **parte extensional**.

Pues bien, cuando la mayoría de los valores se usan solamente en procesos locales, o bien la mayoría de las veces, todos los valores se usan en procesos locales, o una mezcla de las dos situaciones, será preferible una **Base de datos distribuida y/o replicada** (porque algunos datos pueden estar repetidos en más de un centro).

El problema es gestionar de manera unificada una base de datos de la que hay almacenadas distintas partes en varios centros de procesos de datos. Entonces, nos planteamos la pregunta:

¿cuándo son **preferibles** las **bases de datos distribuidas**?

- ➢ Cuando en cada centro, la mayoría de los procesos utilizan valores generados en el propio centro. De esta forma, si falla algún centro o se queda aislado, los demás pueden seguir funcionando. Salvo que todo se averíe, siempre hay posibilidad de que el sistema siga trabajando → garantiza fiabilidad.
- ➢ Cuando el número de accesos remotos es pequeño en comparación con una arquitectura centralizada.

¿cuándo **no son preferibles** las bases de datos distribuidas?

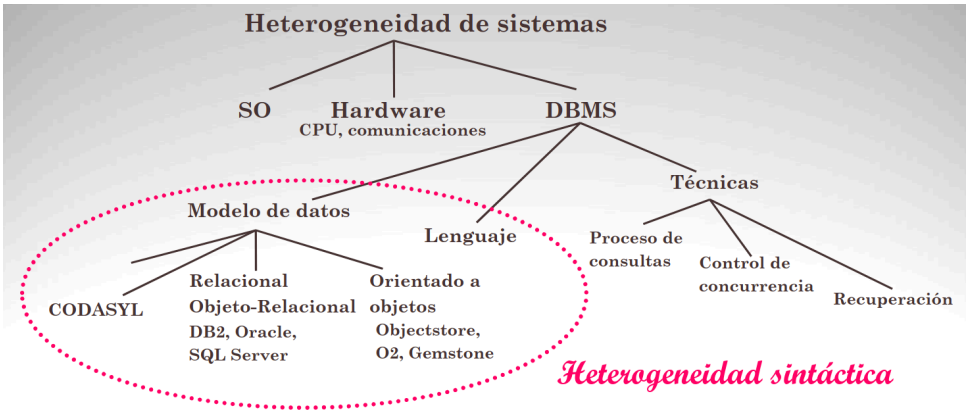
- ➢ Cuando hay un bajo porcentaje de procesos que utilizan sólo datos locales, frente a un gran porcentaje que usa datos de los demás centros. O simplemente si un centro hace muchos más accesos que los demás. La base de datos centralizada garantiza que cada centro, si no falla el ordenador que tiene los datos, puede trabajar al 100%.

➢ Cuando el número de accesos remotos es pequeño en comparación con una arquitectura distribuida
➢ Si todos los procesos son locales → Bases de datos independientes.

2. Conceptos básicos

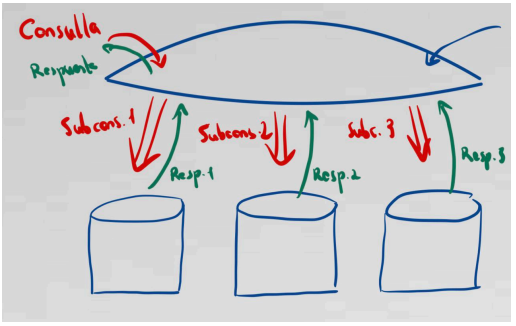
- **Localidad:** forma de denominar a un centro de datos. Se le llama también Nodo, Instalación, o computador.
- **Sistema distribuido:** conjunto de localidades capaces de operar interconectadas y capaz, cada una de ellas, de operar autónomamente.
- **Base de datos distribuida:** colección de datos almacenados en diferentes localidades de un sistema distribuido. Características:
    - Cada localidad puede ejecutar aplicaciones locales.
    - Cada localidad participa, al menos, en la ejecución de una aplicación global que requiere acceso a datos de varias localidades.
    - Los programas de aplicación no necesitan saber en qué localidad están los datos para acceder a ellos. Esto es lo que se conoce como "transparencia".
- **Punto de almacenamiento:** localidad que almacena partes de la base de datos.
- **Punto de acceso**: localidad desde la que se hacen consultas.
- **Fragmento:** conjunto de datos almacenados en una localidad. Es una parte de la base de datos ubicada en un punto de almacenamiento. Está gestionado por un DBMS ordinario.
- **Sistema de Gestión de Bases de datos Distribuidas (DDBMS):** Sistema que asume, para una base de datos distribuida, funciones análogas a las que lleva a cabo para una base de datos centralizada un Sistema de Gestión de Bases de Datos (DBMS).
  Dichas funciones son:
    1. Integración con el sistema de ficheros; traduce instrucciones DML (Data Management Language (Insert, Update, Delete)) a lenguaje de bajo nivel.
    2. Implantación de la integridad: procurar que todos los datos sean íntegros y cumplan las condiciones definidas.
    3. Implantación de la seguridad.
    4. Copia de seguridad y recuperación.
    5. Control de concurrencia: que la base de datos no quede inconsistente cuando dos usuarios accedan concurrentemente.

  Existen dos tipos:
    - **Homogéneos:** Si los DBMSs que gestionan los diferentes fragmentos (los locales) son todos iguales o al menos "se entienden entre ellos".
    - **Heterogéneos:** Si al menos uno de los DBMSs es diferente de los demás y no puede "entenderse" con los demás; cuando hay cierta heterogeneidad.



La heterogeneidad de sistemas puede darse en el ámbito del SO, del Hardware o del Sistema de Gestión de Bases de Datos (DBMS). Dentro de la heterogeneidad del DBMS, la principal es la del Modelo de Datos: podemos tener DBMSs relacionales, orientados a objetos o el CODASYL (específico del modelo en red).

- Hay dos tipos de **BDDs:**
  - **Autonómicas:** Se diseñan, desde el principio, como una base de datos distribuida.
  - **Federadas:** No se piensan desde el principio como distribuidas. Nacen de la agregación de varias bases de datos independientes, que se continuarán gestionando de manera relativamente independiente



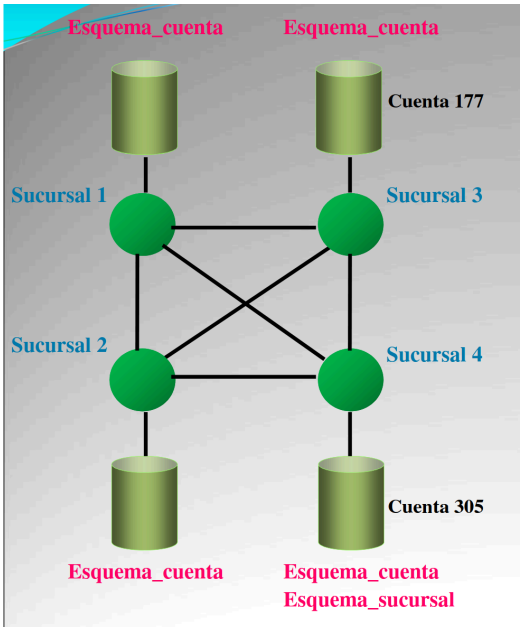
Esto es un esquema federado (software) que integra las distintas BD. Permite consultas locales y globales que impliquen varias BD. Se define como una vista de los esquemas que están “abajo” en las BD.

¿Esta distinción entre autonómicas y federadas es independiente del tipo de DDBMS? La respuesta es sí, es independiente. Sin embargo suele ser normal que para una base de datos distribuida autonómica, se use un DDBMS homogéneo, pero no tiene porqué.

Ejemplo de Base de datos distribuida:

Supongamos que tenemos un banco con cuatro sucursales (Sursal 1, Sucursal 2, Sucursal 3, Sucursal 4), tales que todas ellas almacenan datos de las cuentas bancarias de sus clientes (cada localidad almacena un esquema;

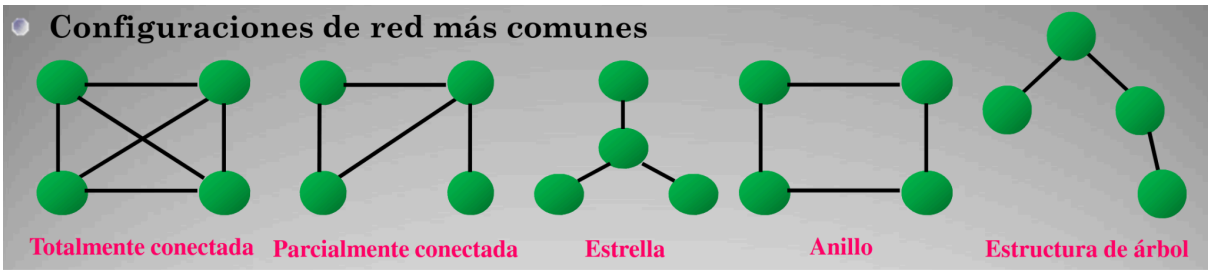
Esquema_cuenta = (nombre_sucursal, número_cuenta, nombre_cliente, saldo)), aunque la sucursal 4, también almacena la información de todas las sucursales **(Esquema_sucursal = (nombre_sucursal, activo, ciudad_sucursal)**)



Supongamos que desde la sucursal 3 se ingresa 100 euros a la cuenta 177 (**Transacción local**) y desde la 3 se transfiere 100 euros de la cuenta 177 a la cuenta 305, que está en la sucursal 4 (**Transacción global**).

Como los datos del banco están almacenados físicamente en localidades distintas, y además se llevan a cabo procesos (transacciones) locales y globales, concluimos que estamos ante una **Base de Datos Distribuida**.

3. Bases de datos distribuidas y redes de computadores



Estas son las configuraciones de red más comunes para una base de datos distribuida. Estas configuraciones se diferencian en:

- **Coste de instalación:** coste de conectar físicamente las localidades del sistema. Cuantas más conexiones tenga la configuración de red, más costosa será (como es el caso de la Totalmente conectada).
- **Coste de comunicación:** coste, en tiempo y dinero, de enviar mensajes de una localidad a otra.
- **Fiabilidad:** frecuencia de fallo de una línea de comunicación o de una localidad. Dependiendo de la configuración de red, cuando una conexión falla, puede dar lugar

al aislamiento de una localidad, o si falla una localidad, puede dar lugar a la incomunicación de las demás (como es el caso estrella, si fallase la del centro).

- ➢ **Disponibilidad:** Posibilidad de acceder a datos a pesar de fallos en algunas localidades o líneas de comunicación.

4. Ventajas y desventajas de las Bases de datos distribuidas

VENTAJAS	**DESVENTAJAS**
Utilización compartida de los datos (necesario para hacer transacciones globales)	**Coste de desarrollo software** (tanto en tiempo como en dinero)
Distribución del control (en cada localidad, hay un DBA. Un DBA global que controla y gestiona la BDD en su conjunto y distribuye el control de la misma a los BDD locales)	**Mayor posibilidad de errores** (se usan algoritmos paralelos, que son más complejos → mayor probabilidad de error)
Fiabilidad y disponibilidad	**Mayor tiempo de procesamiento (**el tiempo de procesamiento es mayor porque se calcula como $\Sigma_i tiempoSubconsulta_i$. Sin embargo, el tiempo de respuesta en una BDD es menor, por la ejecución paralela de las subconsultas (coincide con el tiempo de la subconsulta que más ha tardado en ejecutarse)
Agilización del procesamiento de consultas (dada una consulta, esta se subdivide en subconjuntos que se ejecutan en paralelo (que no concurrentemente), lo cual agiliza bastante las consultas	

5. Transparencia y Autonomía

Transparencia: grado hasta el cual los usuarios del sistema y los programas de aplicación ignoran los detalles de la distribución. Una BDD es totalmente transparente.

Autonomía local: grado hasta el cual el diseñador o administrador de una localidad puede ser independiente del resto del sistema distribuido. En cada localidad tenemos un administrador (local). Cuantas más funciones desempeñe dicho administrador, mayor autonomía local.

Ambos conceptos los podemos interpretar desde el punto de vista de:

Nombre de los datos

Todo **elemento** de información debe tener un **nombre único**. Existe un programa que genera nombres de todos los elementos de la base de datos, y es transparente. Este software es lo que se conoce como **Asignador de nombres**. Tiene los inconvenientes de que se puede convertir en un cuello de botella , si se cae, puede que ninguna localidad pueda trabajar, porque todos necesitan acceder a dicho programa, y se reduce la autonomía local, es decir, se le ha quitado una función al administrador local.

Por otra parte, existe la posibilidad de **Agregar un identificador de localidad**. Esto es, agregar sufijos para identificar copias y fragmentos de un elemento de información. Para ello, se sigue el siguiente convenio:

I*dentificador de localidad.Nombre de elemento.Identificador de fragmento.Identificador de copia*

Por **ejemplo**: Sucursal 3.Cuentas.f3.r2
Con esto se **reduce** el problema de la **pérdida de autonomía local**, pero **se pierde transparencia.**

Localización de fragmentos y réplicas

Para cada usuario se genera un conjunto de alias que el sistema traduce a nombres completos.

Repetición y fragmentación de los datos

- ➢ El sistema debe determinar a qué copia debe acceder cuando se solicita la lectura de un elemento y debe modificar todas las copias cuando se solicite su escritura.
- ➢ El sistema debe permitir que las consultas y las actualizaciones se hagan en términos de elementos sin fragmentar.

Independencia de datos

Inmunidad de las aplicaciones a cambios en la definición y organización de los datos (que no usa). También se controla en Bases de datos centralizadas (es lo único que también se controla en BDC).

Por **ejemplo**: Sea T=(A,B,C,D) y consideremos una aplicación que usa los cuatro atributos de T. Por alguna cuestión, se modifica la tabla T y se añade un atributo nuevo: T=(A,B,C,D,E,). La aplicación no se ve afectada por la actualización de la estructura de datos (inmunidad). Sin embargo, esa independencia se perdería si quitamos un atributo que usa la aplicación.

En función de la definición de los datos, distinguiremos entre **independencia lógica y física**. La definición de datos se hace a dos niveles:

- **Definición del esquema (estructura lógica):** Aquí se define cómo están organizados los datos desde el punto de vista lógico, es decir, qué entidades, relaciones, y atributos existen en el sistema.
  Está asociado a la **Independencia lógica**, que significa que los cambios en esta estructura lógica no deberían afectar a las aplicaciones que usan la base de datos. Por ejemplo, si se añade una nueva tabla o se crea una relación adicional entre tablas, las aplicaciones no deberían requerir modificaciones, siempre que los datos que usan no sean alterados directamente.

- **Descripción física (estructura física):** Se refiere a cómo se almacenan físicamente los datos en el sistema de almacenamiento (disco, SSD, etc.), incluyendo aspectos como la organización en discos, los índices, las estructuras de acceso y otras optimizaciones de almacenamiento.
  Está asociada a la **Independencia física**, que implica que las aplicaciones son inmunes a los cambios en la estructura física. Por ejemplo, si se decide reorganizar los datos en el almacenamiento o cambiar el tipo de indexación, las aplicaciones deberían seguir funcionando sin modificaciones. Además, las aplicaciones no necesitan conocer ni gestionar los detalles del almacenamiento físico ni los métodos de acceso, ya que estos están encapsulados en el sistema de gestión de la base de datos (DBMS).
  Esto es clave para la independencia física, pues permite modificar la estructura física sin afectar a las aplicaciones que consultan o manipulan los datos.

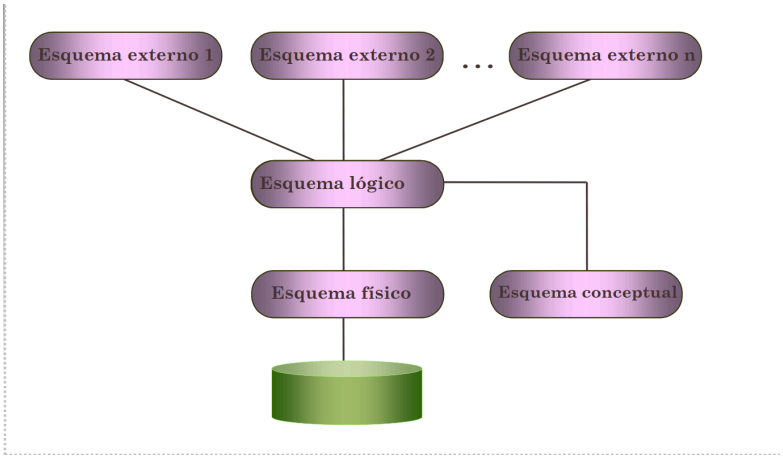
Sistema de gestión de bases de datos

El sistema de gestión de bases de datos distribuido (**DDBMS**), **oculta** el hecho de que los **DBMSs** locales pueden ser **distintos** (utilizan modelos de datos diferentes). Este es uno de los objetivos fundamentales de los DDBMs heterogéneos.
Además, es necesario disponer de un mecanismo que **traduzca** los **modelos de datos** y los **lenguajes** entre los diferentes sistemas. La solución más usual, es usar un modelo de datos (y su lenguaje asociado) como **referencia**. Por ejemplo, si tenemos como modelo de datos de referencia el relacional, las consultas, actualizaciones, etc., se van a traducir al relacional.

6. Arquitecturas

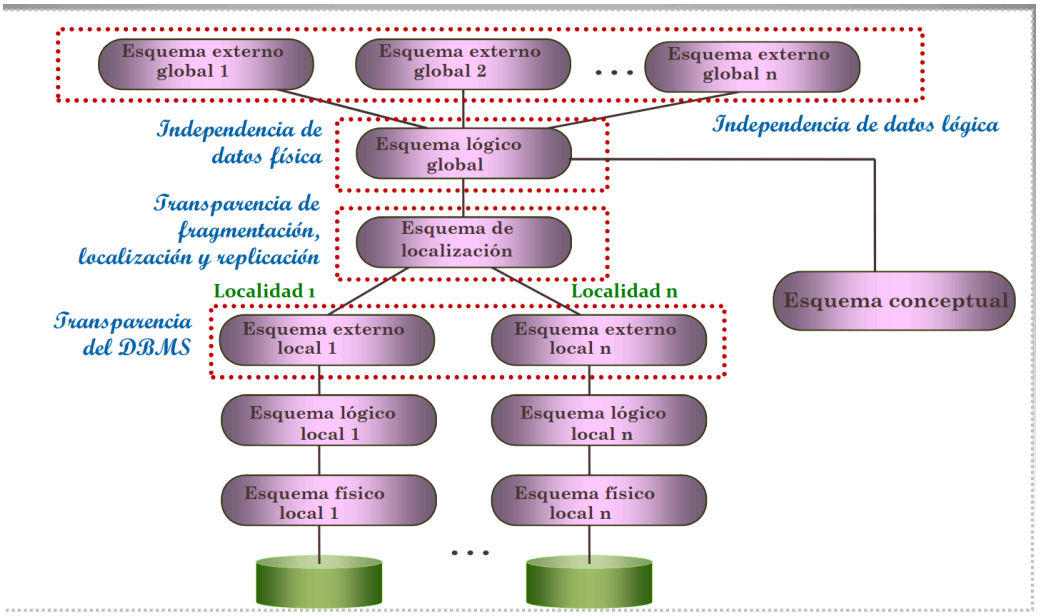
6.1 De referencia para bases de datos centralizadas

La arquitectura de referencia para las bases de datos centralizadas es la **arquitectura ANSI-SPARC de tres niveles**. Consiste en:



donde los **esquemas externos** no son más que subconjuntos del esquema conceptual; lo que se conoce como "vistas". El esquema **lógico** describe la estructura lógica de la base de datos (en el caso relaciones, pues el esquema de tablas), derivado del esquema **conceptual,** que contiene descripciones de datos y relaciones de toda la BD. El esquema **físico**, por su parte, informa de cómo y dónde se almacenan físicamente los datos.

6.2 De referencia para bases de datos distribuidas



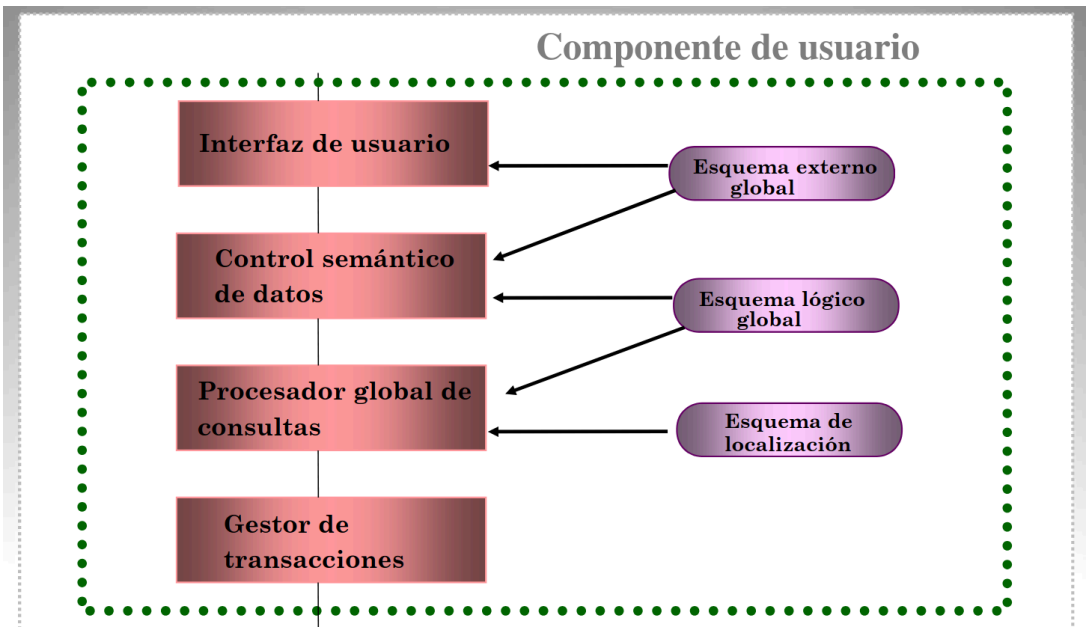
Los **esquemas externos globales** son las vistas. No difieren mucho de los de la BDC y permiten implementar la independencia de datos lógica. El complemento "globales" es porque son esquemas que engloban a todas las localidades de la BDD.
El **esquema lógico global,** implementa la independencia de datos física. El **esquema conceptual** no es diferente del de una BDC; contiene la definición y descripción de los datos y relaciones. Por su parte, el **esquema de localización**, que contiene casi toda la información relativa a cómo está distribuida a BDD, proporciona transparencia de fragmentación y localización y replicación.
Para cada localidad:

- Hay un **esquema externo local**, que, aunque se llamen externos, no tienen que ver con los de arriba. Estos esquemas traducen objetos del esquema de localización en objetos entendibles en la localidad. Son traductores que ponen en comunicación a las localidades. Proporcionan transparencia del DBMS (los que están en la localidad i, y hacen una consulta, no saben el tipo de DBMS de la localidad j.
- Hay un **esquema lógico local**, que depende del modelo de datos que use el DBMS de la localidad.
- Hay un **esquema físico local,** análogo a un esquema físico de una BDC. Cada localidad que contiene datos tiene un esquema físico. No hay un esquema físico global de la BDD.

6.3 Funcional para bases de datos distribuidas

Distinguimos dos componentes:

- **Componente de usuario:** Partimos de una **interfaz de usuario**, que se encarga de recibir la consulta/aplicación. Analiza de forma léxica y sintáctica la consulta para determinar si la consulta está escrita correctamente y en el orden correcto. Para ello, la interfaz utiliza el **esquema externo global**. Seguidamente, se pasa al **control semántico de datos**, que por medio de la vista y el esquema lógico global (donde están definidas las restricciones de integridad), controla si se cumplen o no las restricciones de seguridad (autorizaciones,...), además de si una actualización cumple o no las restricciones de integridad que hay definidas. Una vez hecho esto, se pasa al **procesador global de consultas**, que, utilizado el **esquema lógico global y el esquema de localización** (lo usa para dividir las consultas en subconsultas que se ejecutarán en distintas localidades), traduce una consulta a un lenguaje de bajo nivel (en el caso de Oracle, es el álgebra relacional). Finalmente, pasa al **gestor de transacciones,** mediante el cual se realiza la comunicación con las distintas localidades (comunicación entre componente usuario y componente de datos); manda las subconsultas.



➢ **Componente de datos:** recibe las subconsultas y hace transformaciones para optimizarlas. El **procesador local de consultas** recibe la subconsulta, apoyándose de los esquemas **externo local**, **lógico local** e **interno local** y se pasa al **gestor de transacciones**, que, si lo que se quiere ejecutar a través de la componente de usuario es una transacción, se divide en subtransacciones que se ejecutarán en diferentes localidades. Envía avisos a los demás gestores de transacciones para abortar la subtransacción en caso de fallo de alguna de ellas. Finalmente, junto con el **esquema interno local,** se realiza el **acceso local a datos**; se accede a la memoria secundaria de la localidad.