E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Tema 1

Desarrollo de aplicaciones

José Luis Pastrana Brincones
pastrana@lcc.uma.es

E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Tema 1. Contenido

1. Arquitectura de bases de datos para el patrón MVC (Modelo-Vista-Controlador)
2. Transformaciones entre el modelo Relacional y el modelo de Clases.
3. Diseño de transacciones en bases de datos
4. Conexiones dedicadas y compartidas: pool de conexiones.

José Luis Pastrana Brincones
pastrana@lcc.uma.es

E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

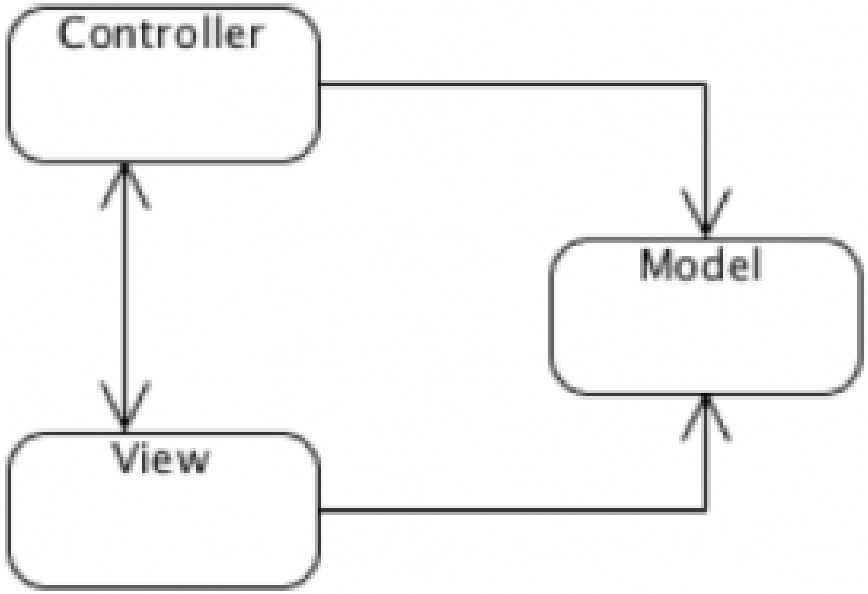
Tema 1.1

Arquitectura de bases de datos para el patrón MVC (Modelo-Vista-Controlador)

José Luis Pastrana Brincones
pastrana@lcc.uma.es

El Patrón Modelo-Vista-Controlador (MVC)

- El Modelo Vista Controlador es el patrón arquitectural más usado en la ingeniería del software.
- Separa el modelos de datos, el modelo de la capa de presentación (vista) y la parte de control.



El Patrón Modelo-Vista-Controlador (MVC)

- Descrito por primera vez en 1979 para Smalltalk
  - http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
- Utilizado en múltiples frameworks
  - Java Swing
  - Java Enterprise Edition (J2EE)
  - XForms (Formato XML estándar del)
  - GTK+ (Gnome para X Window)
  - ASP.NET MVC Framework (Microsoft)
  - Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
  - Apache Struts (framework para aplicaciones web J2EE)
  - Ruby on Rails (framework para aplicaciones web con Ruby)
  - Etc.

El Patrón Modelo-Vista-Controlador (MVC)

- El Modelo:
  - Gestiona la información
  - Advierte a las otras capas de cambios en sus datos.
  - Representa el dominio de datos.
- La vista:
  - Representa gráficamente el modelo para que el usuario pueda interactuar él.
  - Es la interfaz de datos.
- El controlador:
  - Recibe las peticiones de la vista.
  - Responde actualizando el modelo de datos.

El Patrón Modelo-Vista-Controlador (MVC)

- Modelo–Vista–Controlador
  - Un modelo
  - Varias vistas
  - Varios controladores
- Las vistas y los controladores suelen estar muy relacionados
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista)
- Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador

El Patrón Modelo-Vista-Controlador (MVC)

- Flujo de control

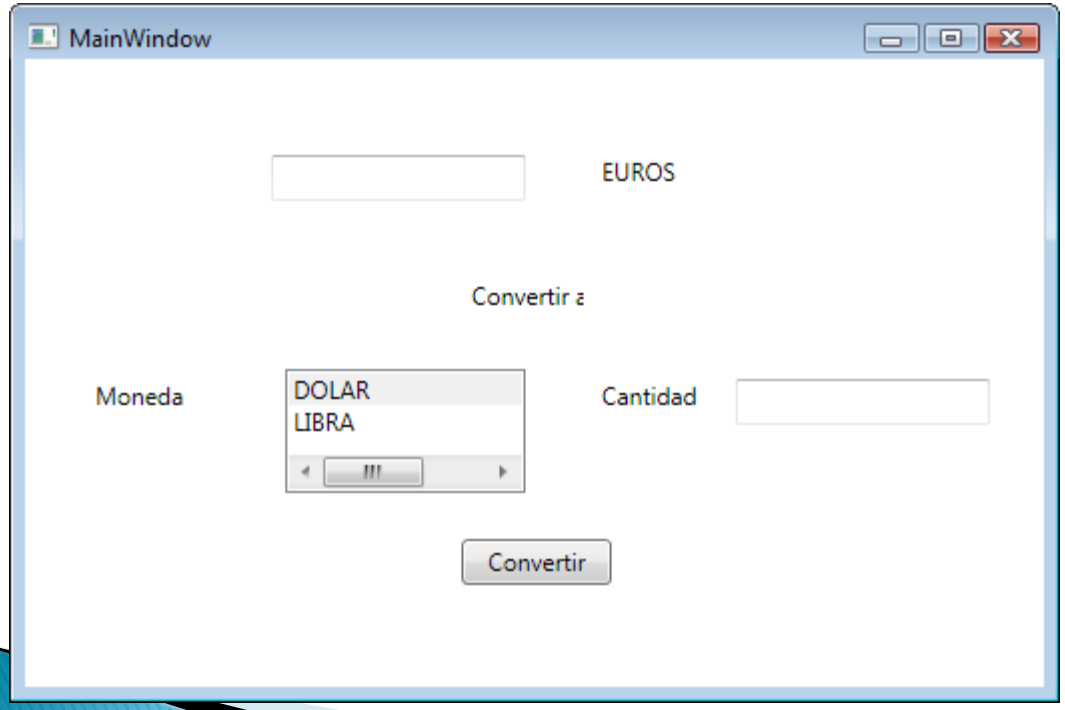
1. El usuario realiza una acción en la interfaz.
2. El controlador trata el evento de entrada.
3. El controlador notifica al modelo la acción.
   - Puede implicar un cambio del estado del modelo
4. Se genera/actualiza la vista.
   - La vista toma los datos del modelo
   - El modelo no tiene conocimiento directo de la vista
5. El interfaz de usuario espera otra interacción del usuario y comenzará otro nuevo ciclo.

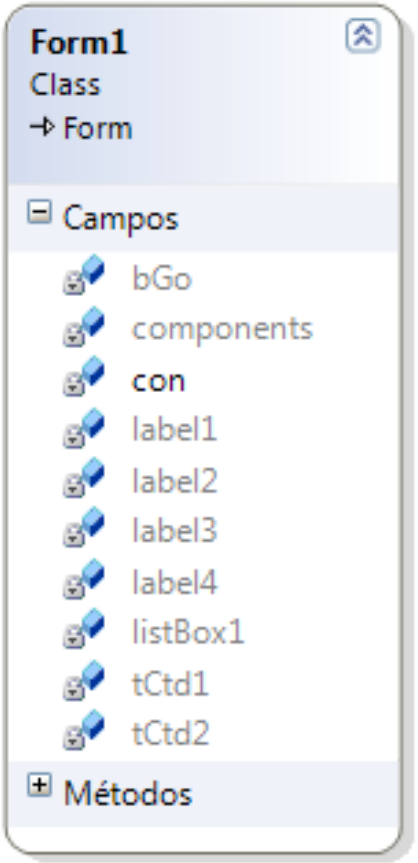
MVC en aplicaciones

- Vista:
  - página HTML, JFrame, Windows Form, WPF, …
- Controlador:
  - Hebra de tratamiento de eventos, que captura y propaga los eventos a la vista y al modelo
- Modelo:
  - Capa de Datos: información almacenada en una base de datos, en ficheros o en XML.
  - Reglas de negocio que transforman esa información.

MVC: Ejemplo

- Queremos desarrollar una aplicación para la conversión de euros a diferentes monedas.
- VISTA

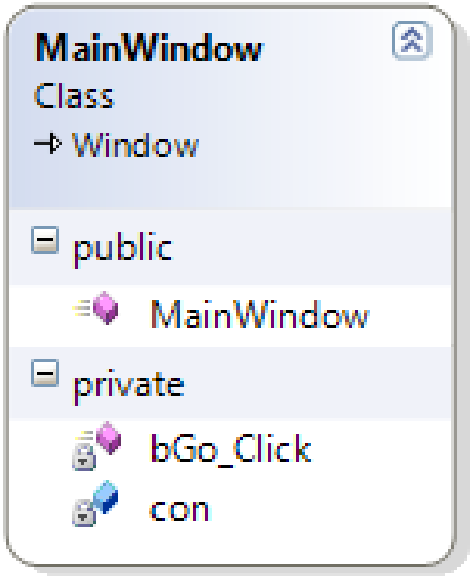




MVC: Ejemplo

- Controlador

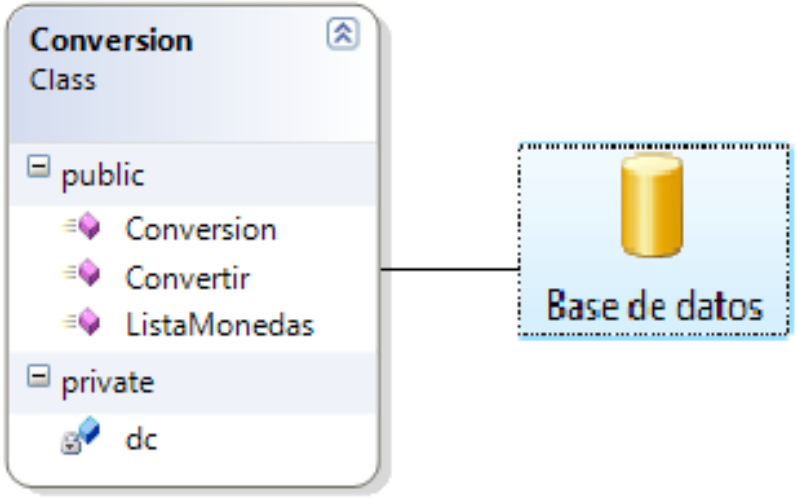
```
public partial class MainWindow : Window
{
    Conversion con;
    public MainWindow()
    {
        con = new Conversion();
        InitializeComponent();

        foreach (string m in con.ListaMonedas())
        {
            listBox1.Items.Add(m);
        }
        listBox1.SelectedIndex = 0;
    }
    private void bGo_Click(object sender, RoutedEventArgs e)
    {
        string moneda = listBox1.SelectedItem.ToString();
        double ctd = double.Parse(tCtd1.Text);
        double res = con.Convertir(moneda, ctd);
        tCtd2.Text = res.ToString();
    }
}
```



MVC: Ejemplo

- Modelo

```
class Conversion
{
    DataClasses1DataContext dc;
    public Conversion()
    {
        dc = new DataClasses1DataContext();
    }
    public string[] ListaMonedas()
    {
        . . .
    }
    public double Convertir(string moneda, double ctd)
    {
        var val = from conversion in dc.conversion
                  where (conversion.Moneda == moneda)
                  select conversion.valor;
        double cambio = val.First();
        return ctd * cambio;
    }
}
```



E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

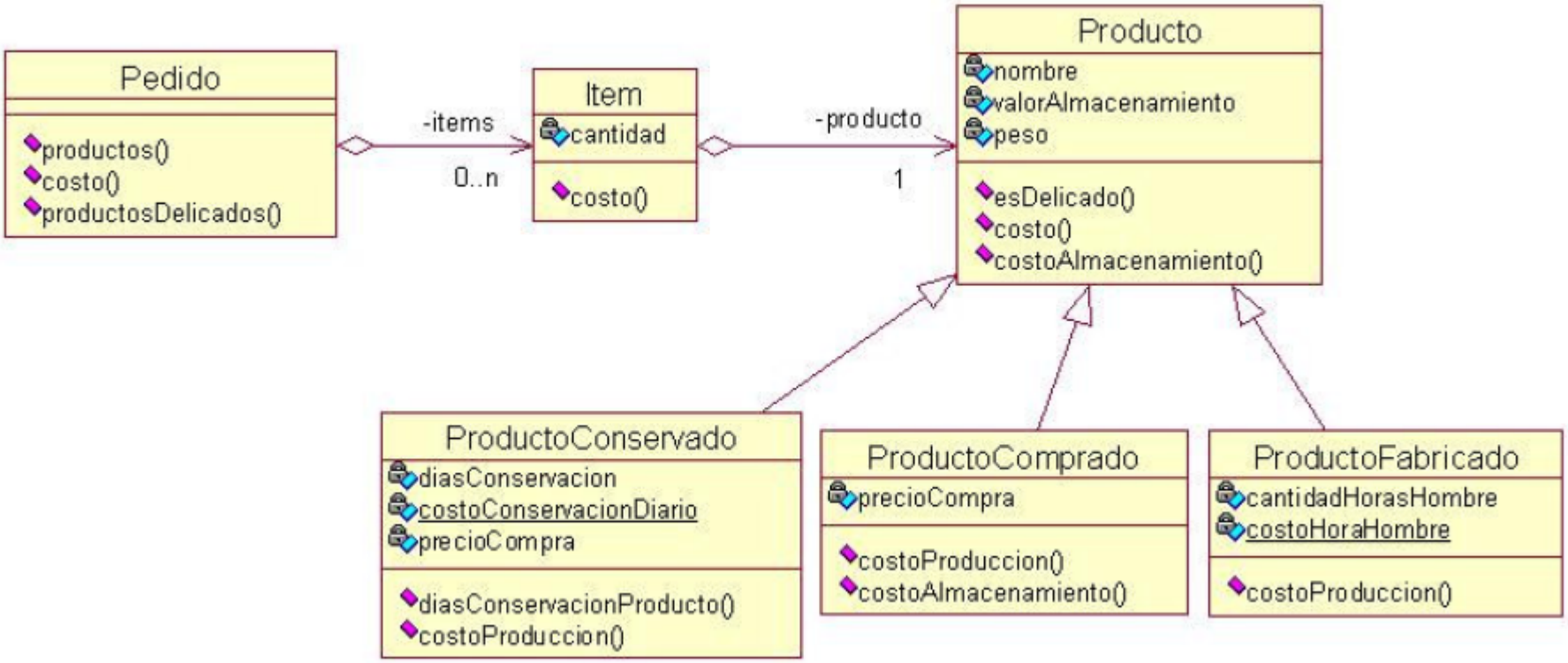
Tema 1.2

Transformaciones entre el modelo Relacional y el modelo de Clases.

José Luis Pastrana Brincones
pastrana@lcc.uma.es

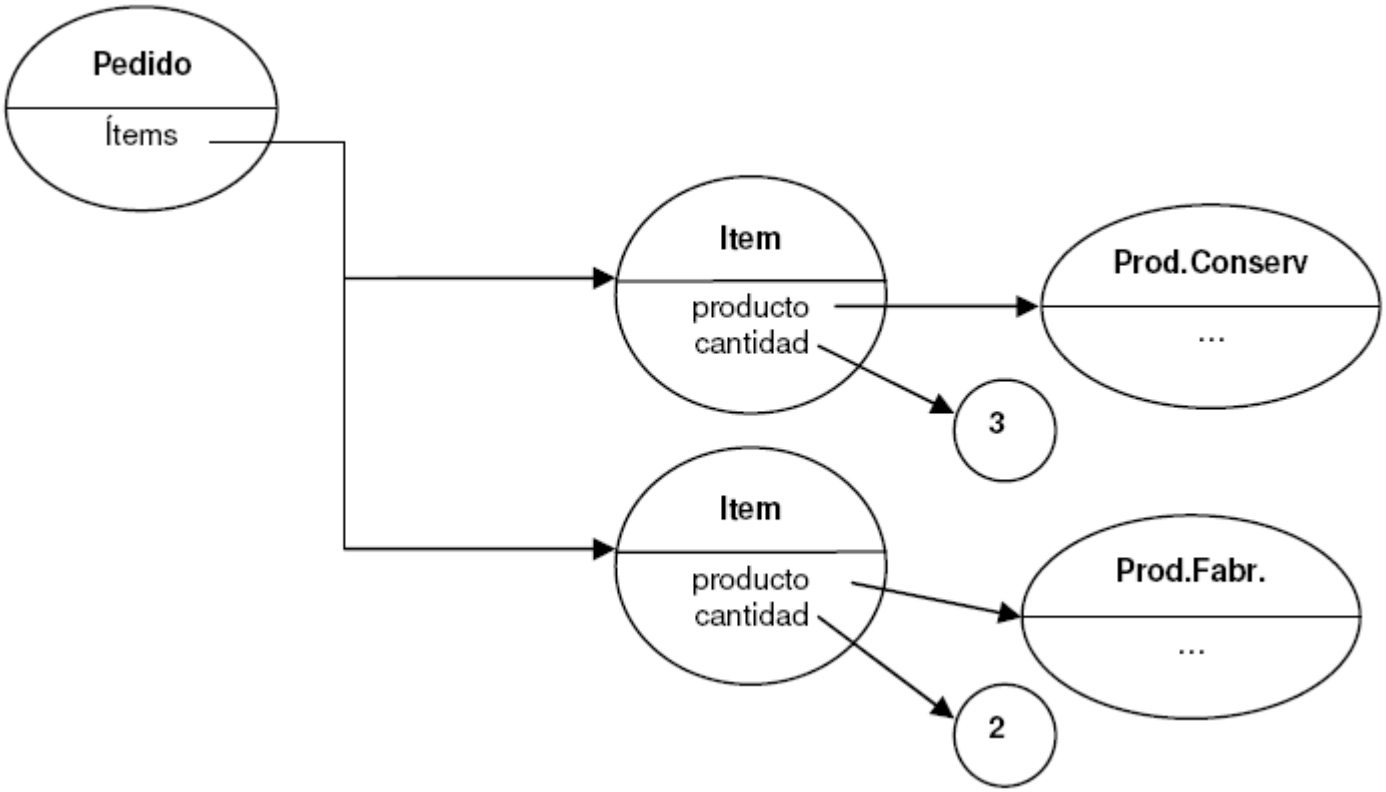
Introducción

- Supongamos que tenemos un modelo de objetos como el siguiente:



Introducción

- ¿Qué tipo de estructura siguen los objetos en memoria? ¿Cómo están almacenados?



Introducción

- La memoria es limitada y necesito poder almacenar los objetos en un medio que me permita recuperarlos (persistencia).

- La persistencia de la información es la parte más crítica en una aplicación de software.

- Opciones:
  - Puedo conservar la misma estructura en una base de objetos
  - Puedo utilizar una base de datos relacional como repositorio de información.

Introducción

- Si la aplicación está diseñada con orientación a objetos, la persistencia se logra mediante:
  - serialización del objeto (por ejemplo en XML)
  - o almacenando en una base de datos.

- El modelo de objetos difiere en muchos aspectos del modelo relacional.

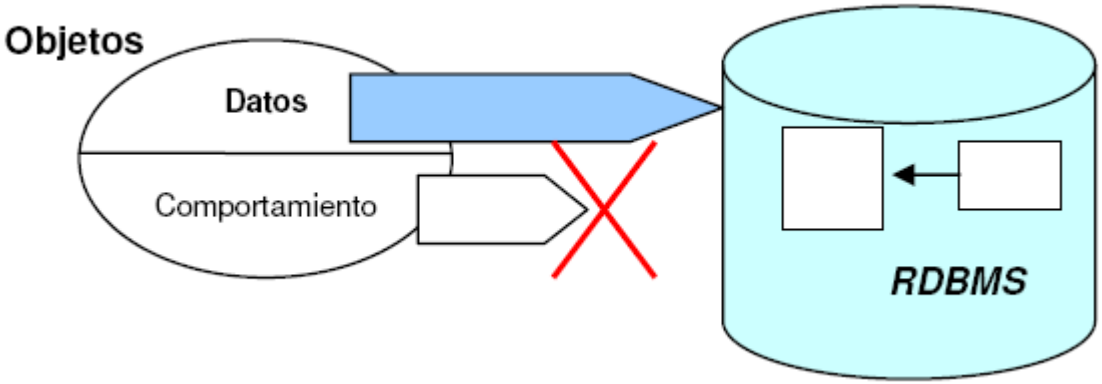
- La interface que une esos dos modelos se llama marco de Mapeo objeto-relacional (ORM en inglés).

Mapeo objeto-relacional

- Para persistir un objeto, normalmente:
  - se crea una conexión a la base de datos
  - se crea una sentencia SQL
  - Se asignan los parámetros
  - se ejecuta la transacción.
- Imaginemos que tenemos un objeto con varias propiedades, además de varias relaciones,
- ¿como las asociamos relacionalmente?
- ¿Cómo las almacenamos? ¿Automáticamente, manualmente? ¿Qué pasa con las claves?

Mapeo objeto-relacional

- Inicialmente podemos establecer una equivalencia entre el concepto de Tabla/Clase, y Registro/instancias de esa clase.



- Pero más adelante empezaremos a tener ciertas dificultades para que este mapping sea tan lineal.

Mapeo objeto-relacional

Objetos

- Tienen comportamiento.
- Los objetos encapsulan información para favorecer la abstracción del observador.
- Puedo generar interfaces.

Tablas

- controles de integridad o pequeñas validaciones (constraints o triggers).
- Una tabla no tiene esa habilidad.
- En el álgebra relacional la interfaz no se convierte en ninguna entidad

Mapeo objeto-relacional

- Los *Procedimientos Almacenados* no están asociados a una tabla, por lo que si toco un campo y necesito al menos recompilar todos los *Procedimientos Almacenados* asociados.
- La herencia es una relación estática que se da entre clases, que favorece agrupar comportamiento y atributos en común, pero en la implementación física las tablas no tienen el concepto de herencia.

Mapeo objeto-relacional

- En el álgebra relacional no hay polimorfismo ni vinculación dinámica.
- Al realizar una consulta sobre una tabla, necesito saber de qué tabla se trata.
- A todo esto es lo que se conoce como "*Impedance mismatch*" : las dificultades de traducción entre los dos mundos.
- Estamos usando la base de datos relacional sólo como repositorio para almacenar los datos de los objetos.
- El comportamiento sigue estando en los objetos cuando se instancian. Por ese motivo tenemos la posibilidad de adaptar el modelo relacional al de objetos

Mapeo objeto-relacional

- Hace algunos años teníamos que hacer manualmente ese mapeo objetos-relacional.
- Para persistir un objeto había que hacer un INSERT o un UPDATE
- Cuando queríamos recuperar un objeto había que hacer un SELECT de la tabla, y a partir de ahí por cada registro teníamos que construir el objeto.
- Hoy tenemos varios frameworks que manejan el mapeo Objetos-Relacional (OR/M Object Relational Mapping): Hibernate, OJB, JDO, LINQ, etc, que se han transformado en estándares de facto.
- La existencia de estos frameworks facilitaron la aceptación por parte del mercado de lo que tenemos hoy en día: reglas de negocio implementadas en una tecnología orientada a objetos + un framework de persistencia OO contra una base de datos relacional.

Ejemplo Mapeo objeto-relacional

- Supongamos que un Pedido tiene una fecha de pedido. Creamos la entidad Pedido:

Modelo relacional: Tabla Pedido

PEDIDO

PEDIDO_ID
FECHA_PEDIDO

Modelo de objetos: Clase Pedido

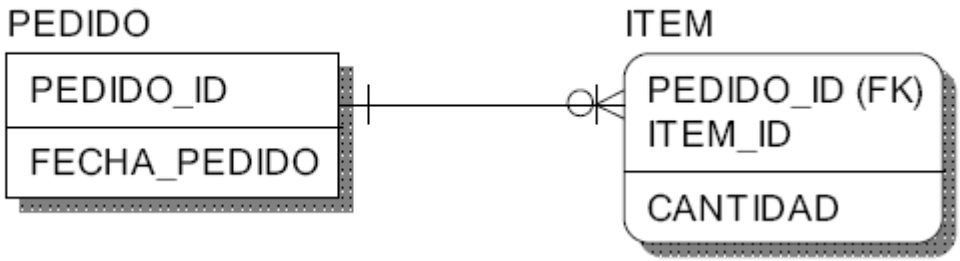
PedIdo
-fecha

- En el modelo relacional necesitamos una clave que identifique unívocamente a cada registro de la tabla Pedido. Estamos hablando del mismo registro si tienen el mismo PEDIDO_ID.
- En POO manejamos el concepto de identidad: Cuando yo referencio a un objeto no necesito identificarlo Hablamos del mismo objeto si objeto1 == objeto2.

Ejemplo Mapeo objeto-relacional

- Tenemos que manejar la relación Pedidos-Items-Productos.

Modelo relacional:

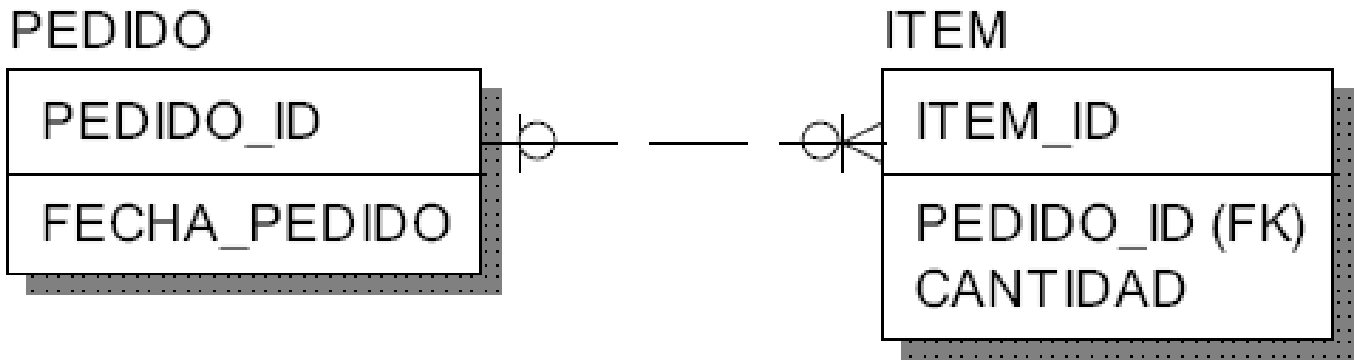


- Como hay una relación de 1 a muchos la clave principal de ITEM se compone del Identificador del Pedido (FK) y el del Identificador del Item.

Pedido_Id	Item_Id	Cantidad
1	1	...
1	2	...
2	1	...
2	2	...

Ejemplo Mapeo objeto-relacional

- Otra opción es definir al ítem como autoincremental

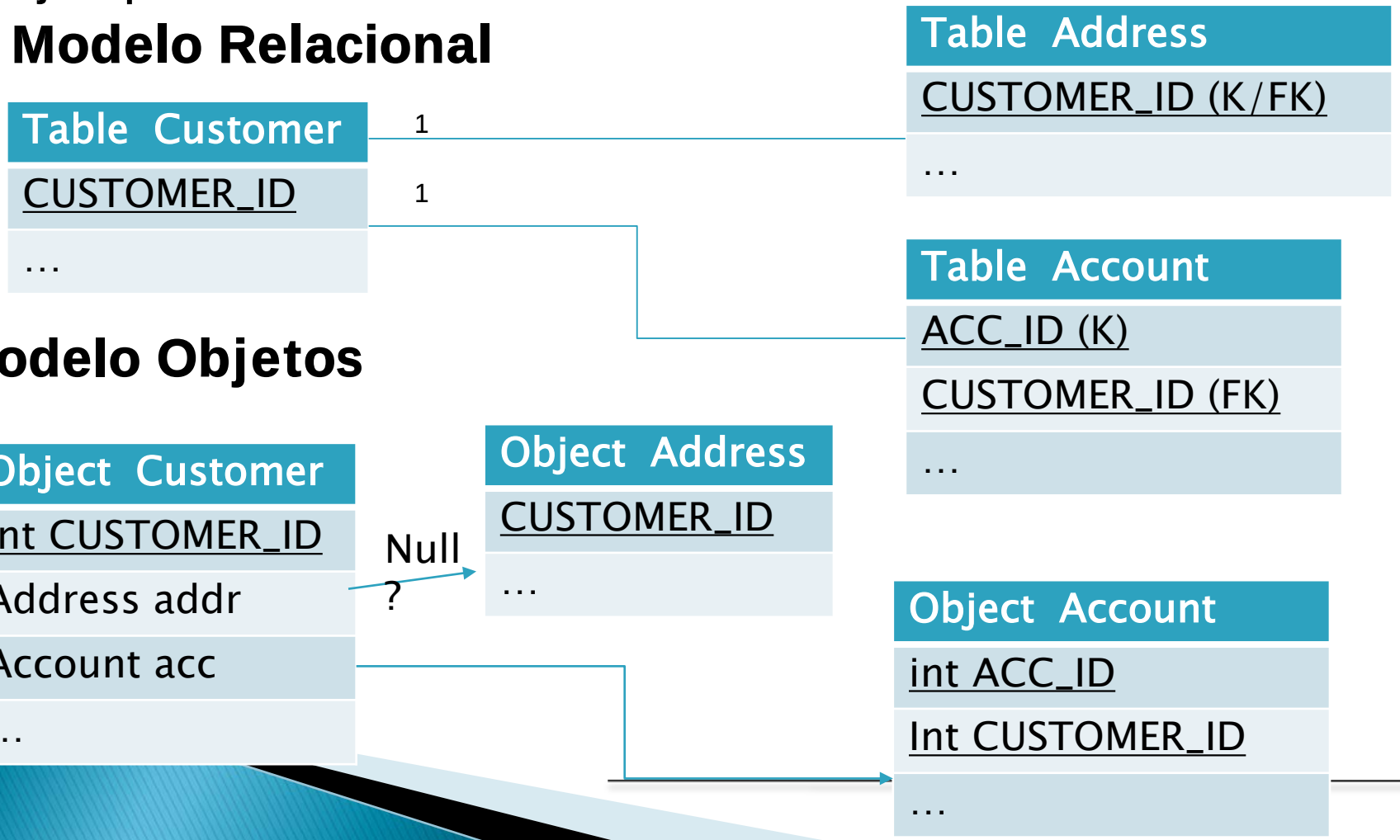


Item_Id	Pedido_Id	Cantidad
1	1	...
2	1	...
3	2	...
4	2	...

Ejemplo Mapeo objeto-relacional

- Ejemplo: Relaciones 1 ... 0/1

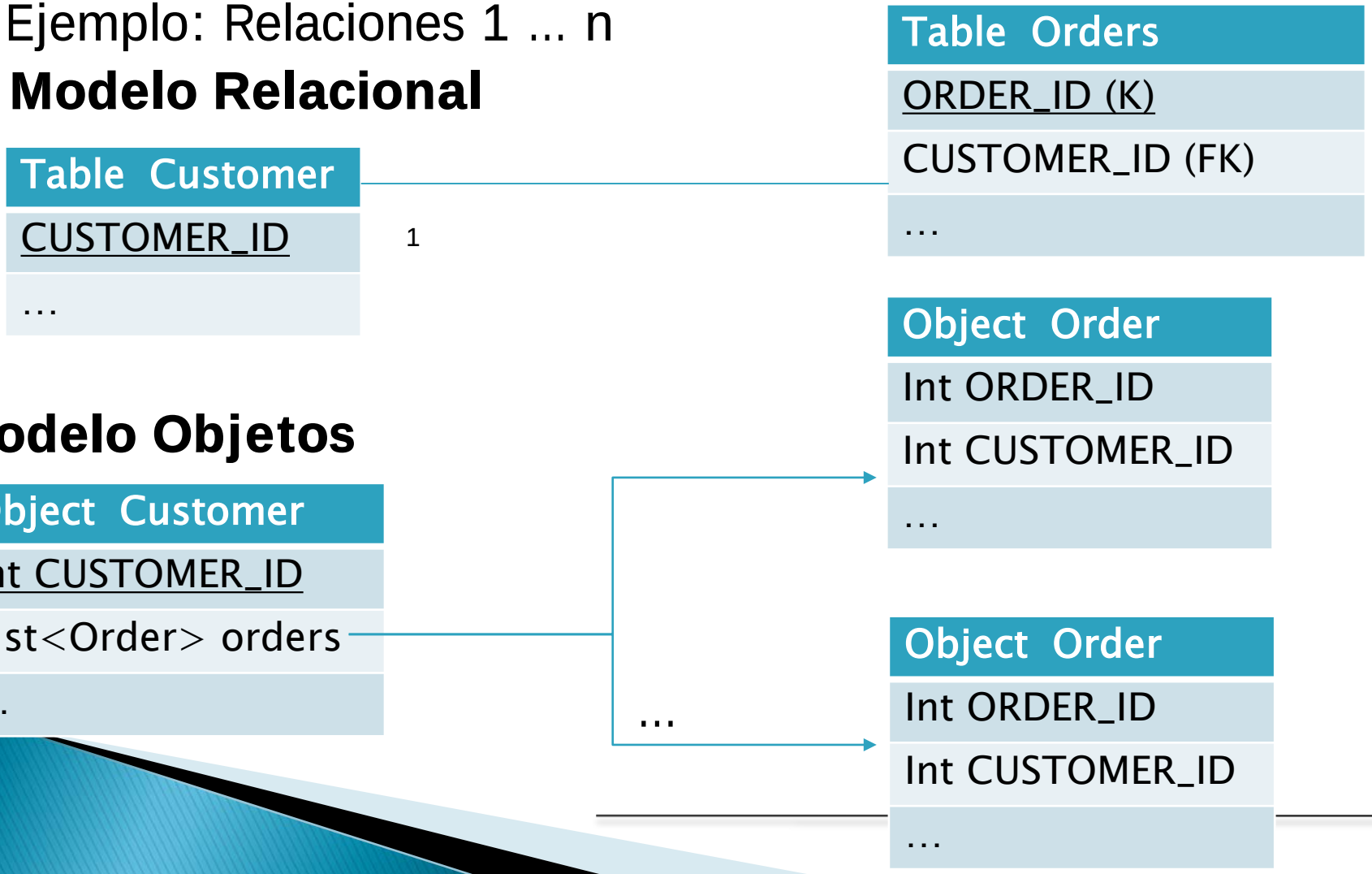
Modelo Relacional



Ejemplo Mapeo objeto-relacional

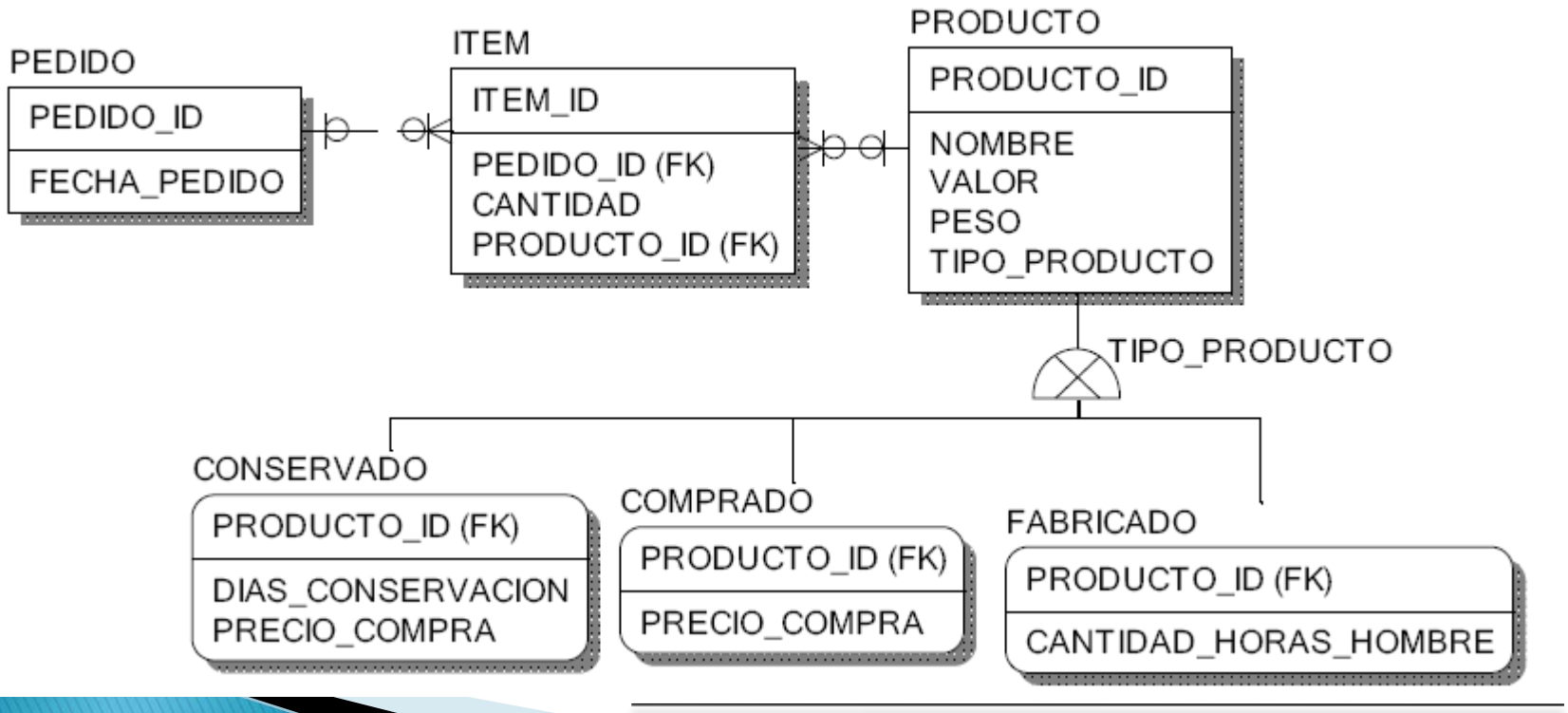
- Ejemplo: Relaciones 1 ... n

Modelo Relacional



Ejemplo Mapeo objeto-relacional

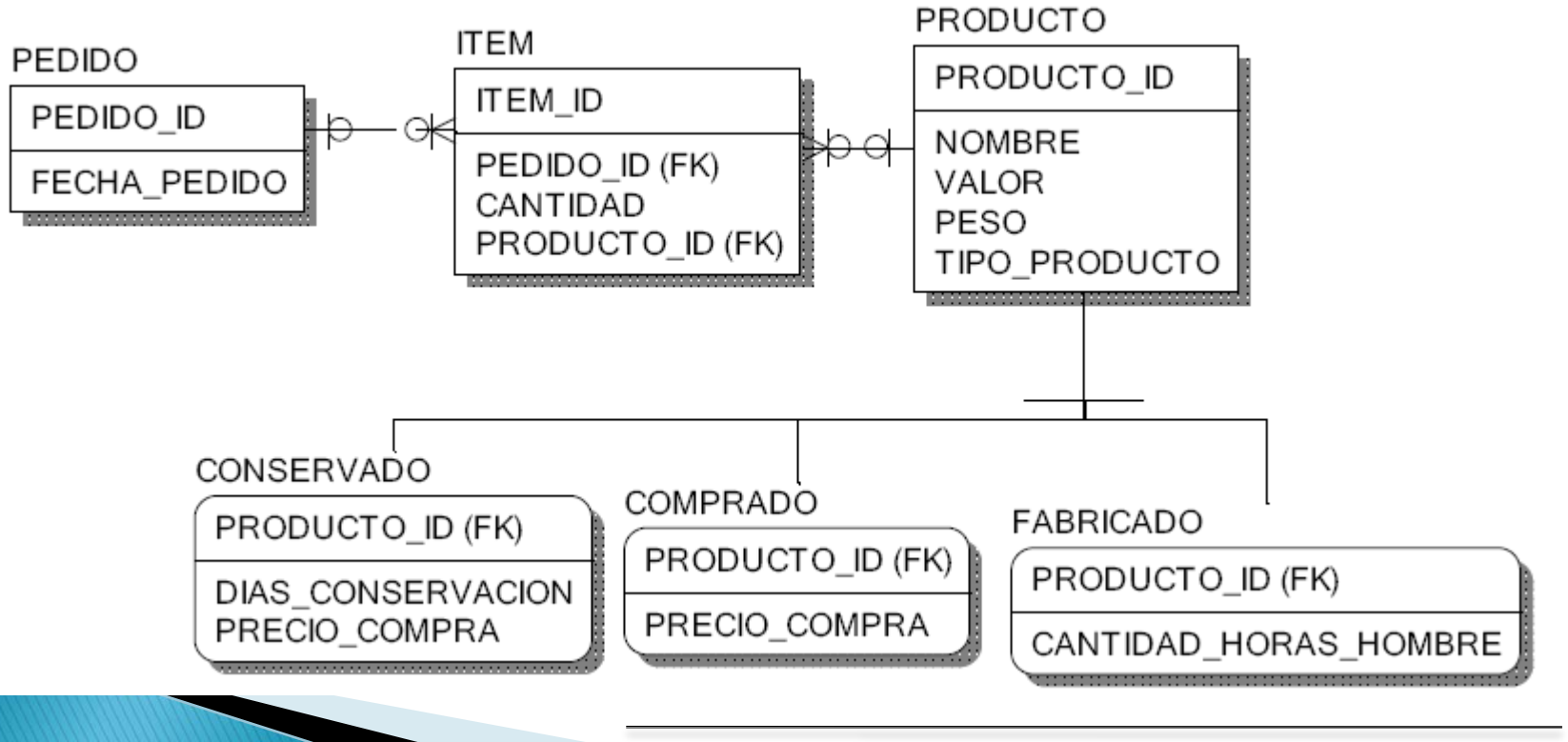
- Vamos a ver la entidad Producto.
- El modelo lógico equivalente a la clase Producto con sus subclases sería:



Ejemplo Mapeo objeto-relacional

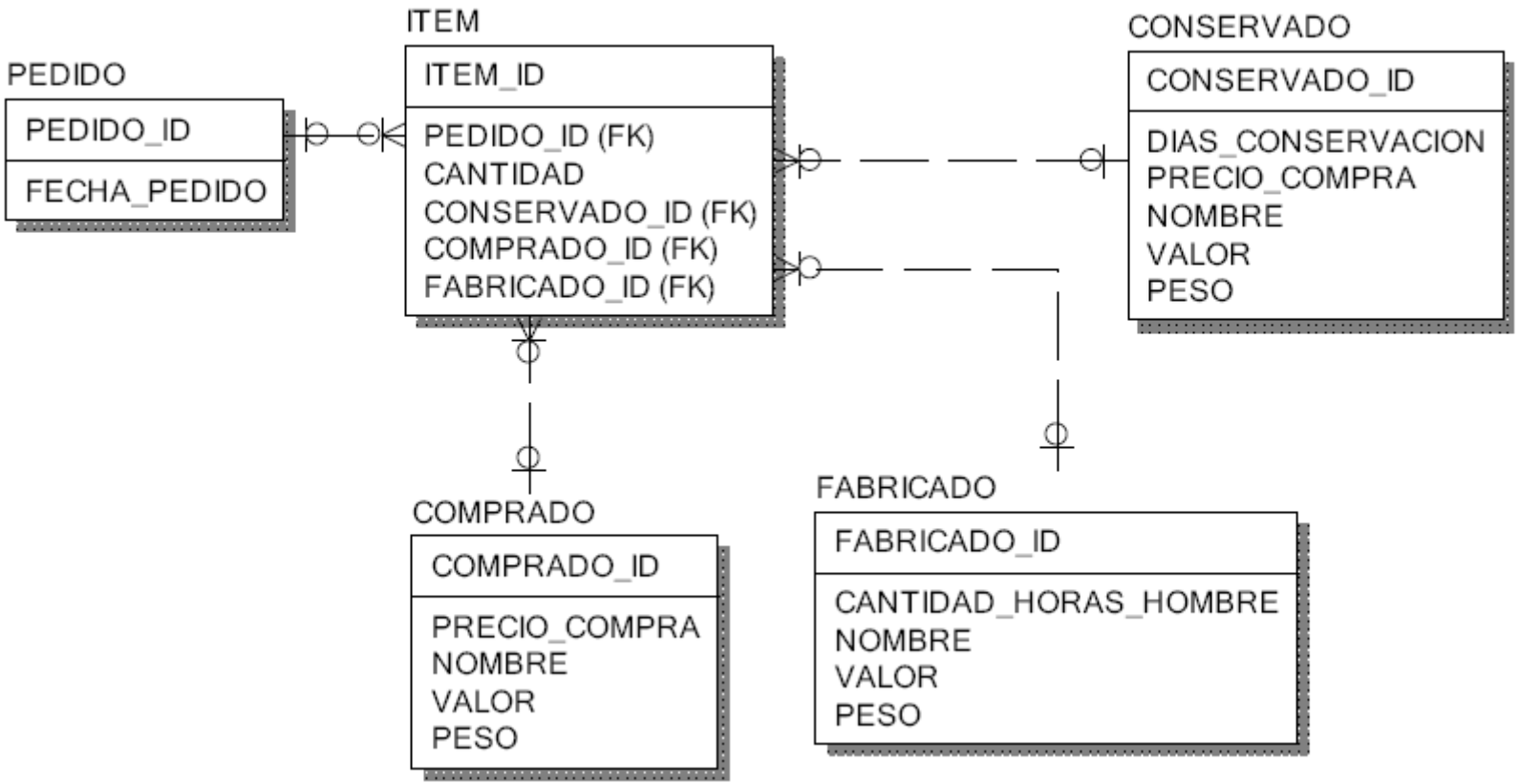
- Al implementar físicamente este modelo podemos elegir 3 opciones:

1. Implementar una tabla por cada clase:



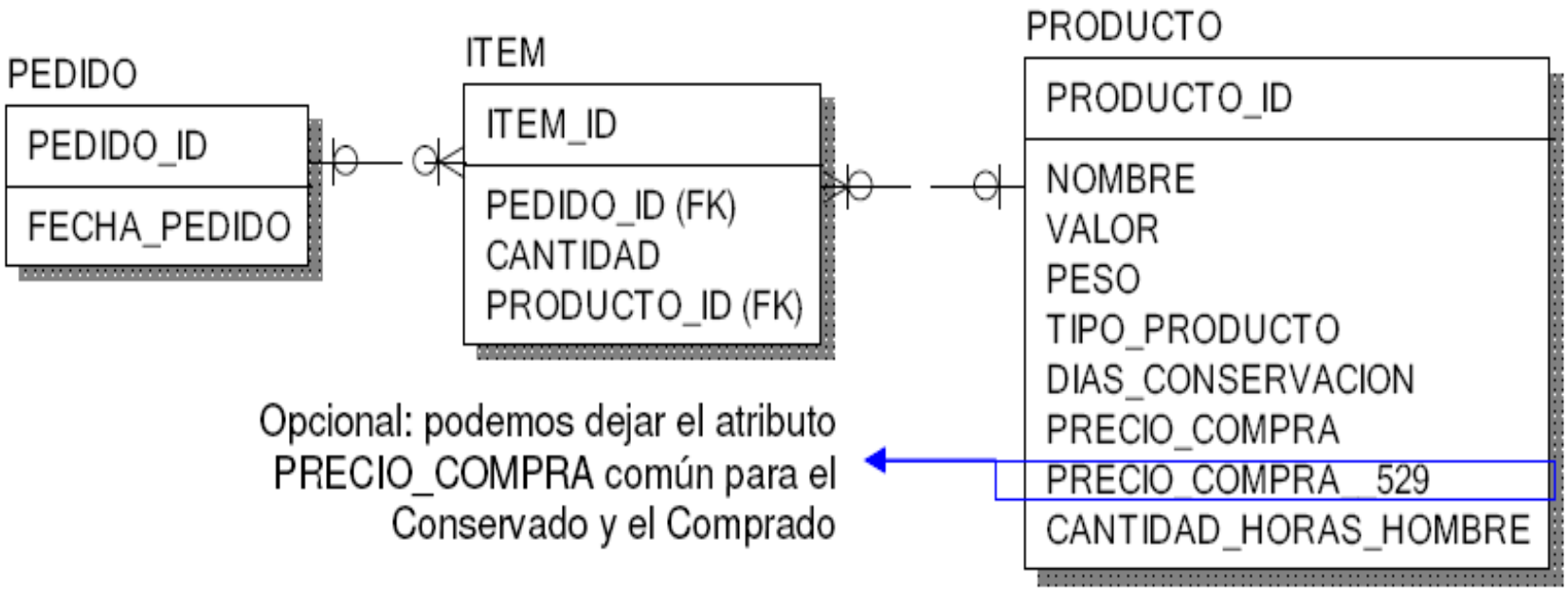
Ejemplo Mapeo objeto-relacional

2. Implementar una tabla por subclase:



Ejemplo Mapeo objeto-relacional

3. Implementar una única tabla:



Revisión Mapeo objeto-relacional

Solución	A favor	En contra
1 tabla	• Facilidad/Performance • Evito generar muchas tablas	• Campos no utilizados • Tentación de “reutilizar” atributos para cosas distintas
1 tabla por cada clase (n + 1)	• Es el modelo “ideal” según las reglas de normalización • No necesito saber en qué tabla está la información. • Permite establecer campos no nulos para cada subclase	• Es la opción que más entidades requiere crear • Requieren hacer LEFT JOIN contra todas las tablas que representan las subclases
1 tabla por subclase (n)	• Permite establecer campos no nulos para cada subclase • No requiere tener un campo discriminador (TIPO_PEDIDO)	• Cada subclase repite atributos “heredados” de la superclase

Revisión Mapeo objeto-relacional

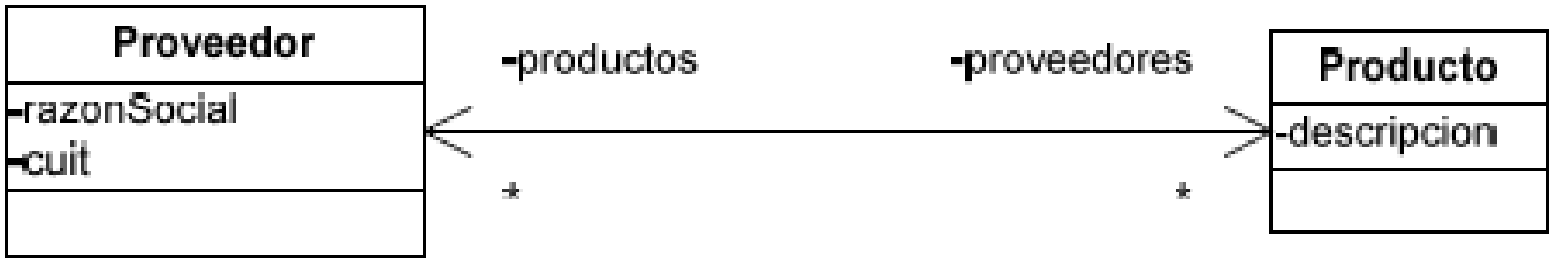
Solución	Cuándo conviene
1 tabla	• Cuando las subclases comparten muchos atributos en común. • Cuando se necesita simplificar las consultas.
1 tabla por cada clase (n + 1)	• Cuando hay muchos atributos comunes entre las subclases pero también muchos atributos propios en cada subclase.
1 tabla por subclase (n)	• Es la técnica utilizada para las entidades independientes. • Cuando las subclases comparten muy pocos atributos entre sí • Cuando no me interesa trabajar con consultas polimórficas (trabajo en forma independiente cada subclase)

Relaciones

Relaciones muchos a muchos

- Ejemplo 1: “Un proveedor vende muchos productos y un producto es vendido por muchos proveedores”.
- ¿Cómo lo modelo en objetos? Son colecciones desde los objetos raíces (proveedor y producto respectivamente).

Diagrama de clases



Navegación entre objetos

- El grafo de objetos puede navegarse en cualquier sentido porque todos los objetos están en memoria.
- Tengo un cliente, le pido el total y el cliente tiene acceso a sus pedidos. Para conocer el importe total, cada pedido tiene acceso a sus ítems, y cada ítem conoce su cantidad y tiene una referencia al producto.
- Si el Pedido tiene Items y cada Item tiene Producto, por un lado es cómodo traer toda la estructura del objeto Pedido para poder navegarlo libremente.
- Por otro lado, quizás arme toda una estructura sólo para averiguar la fecha de un pedido.

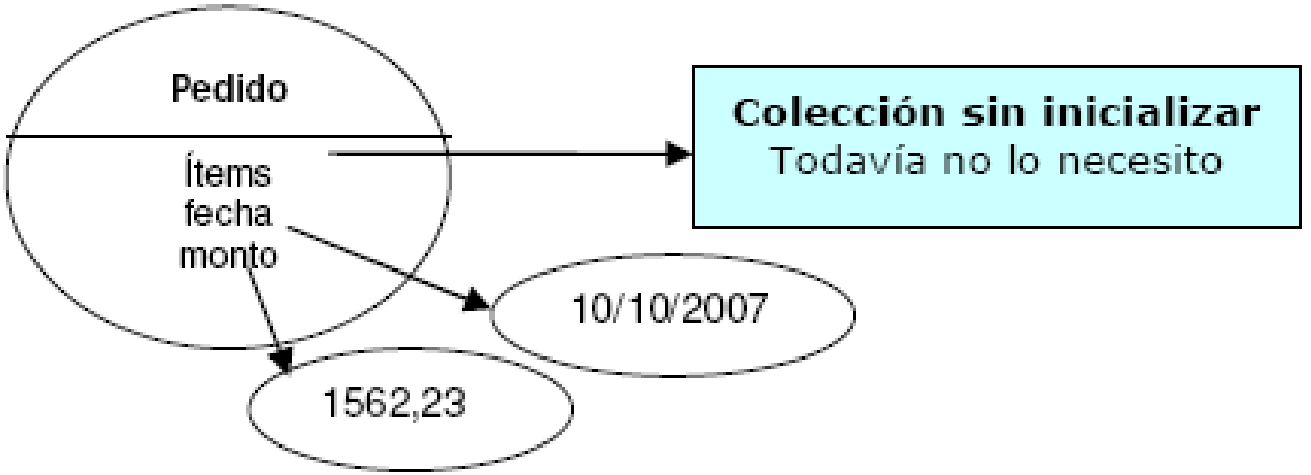
Navegación entre objetos

1. Obtengo sólo el pedido.

Queremos conocer todos los pedidos de un cliente:

```
SELECT * FROM PEDIDO WHERE CLIENTE_ID = 122
```

Esto se transforma en un objeto Pedido:



Navegación entre objetos

2. Obtengo el pedido y todas las relaciones asociadas:

Ahora tenemos que recuperar los pedidos del cliente:

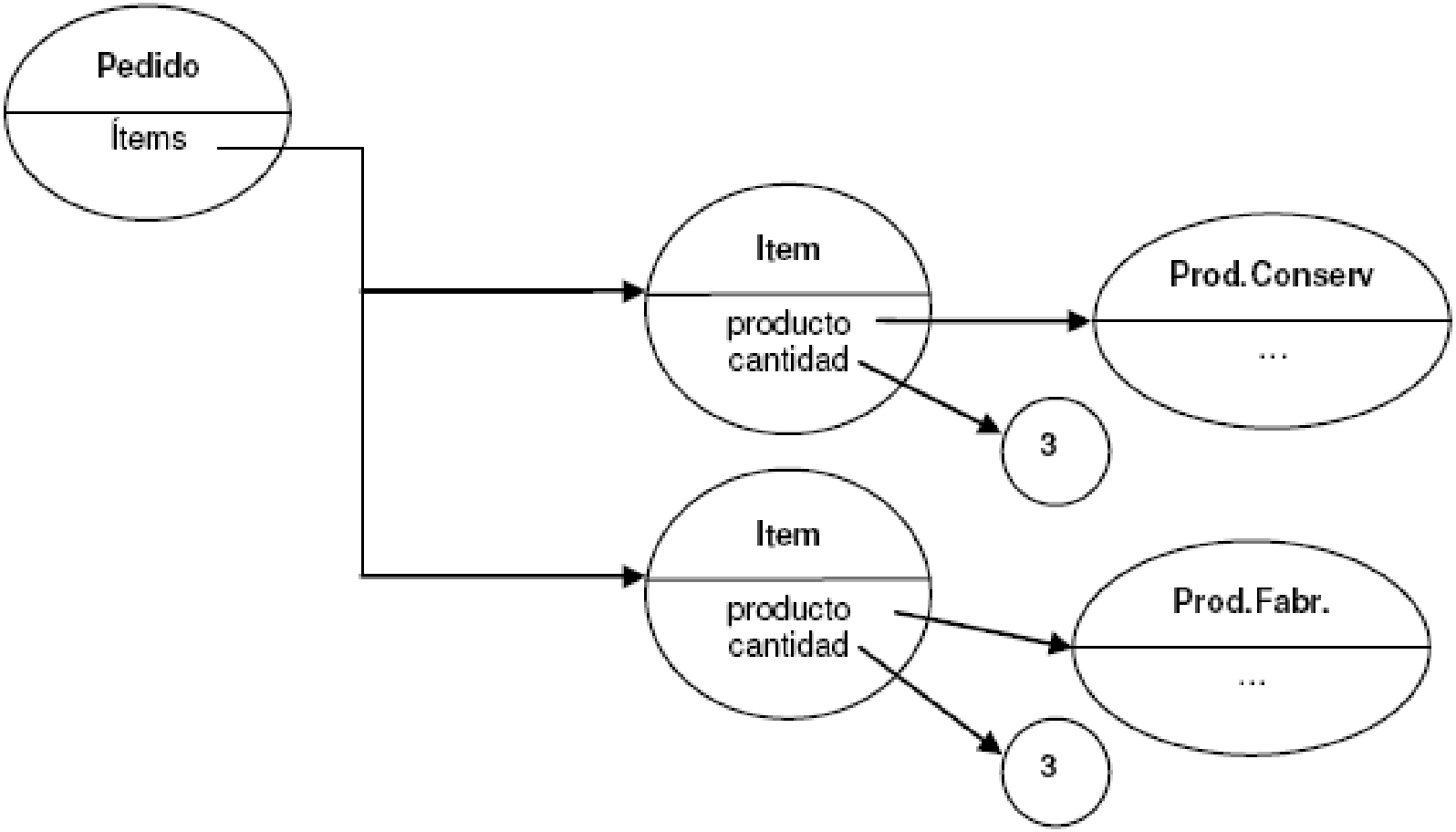
```
SELECT * FROM PEDIDO WHERE CLIENTE_ID = 122
```

Pero también tenemos que generar los ítems y los productos. Por cada ID_PEDIDO obtenido en la consulta anterior hacemos:

```
SELECT * FROM ITEM i INNER JOIN PRODUCTO p
ON p.ID_PRODUCTO = i.ID_PRODUCTO
WHERE i.ID_PEDIDO = ?
```

Navegación entre objetos

Con esto generamos:



Discusión

- ¿Me conviene (1) o (2)?
- En general los frameworks que hacen mapeo O/R ofrecen la posibilidad de definir hasta dónde voy a convertir.
- **Lazy association**: una asociación lazy, es aquella donde voy a traer la información bajo demanda ("sólo cuando lo necesite").

Discusión ¿O/R vs R/O?

- Una vez adoptada la decisión de trabajar con bases de datos relacionales, la pregunta es qué hacer primero:
  a) Generamos el modelo relacional y luego adaptamos el modelo de objetos en base a las tablas generadas
  b) Generamos el modelo de objetos y en base a éste se crean las tablas.
- La opción a) supone que es más importante la forma en que guardo los datos que las reglas de negocio que modifican esos datos.
- En la opción b) el desarrollo con objetos no se ve ensuciado por restricciones propias de otra tecnología.

E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

Tema 1.3

Diseño de transacciones en bases de datos

José Luis Pastrana Brincones
pastrana@lcc.uma.es

Diseño de Transacciones

Las transacciones en una BD se utilizan para representar cómo evoluciona el sistema.

- *Operaciones de acceso a los datos:*
  - **inserción de ocurrencias de entidad o** relación.
  - **borrado de ocurrencias de entidad o** relación.
  - **modificación del valor de los atributos** de ocurrencias de entidad o relación.

En su nivel más bajo consiste en definir cómo se insertan, borran y modifican ocurrencias de todos los objetos del sistema

Diseño de Transacciones

- Una **transacción es una secuencia de operaciones de** acceso a los datos que constituye una unidad lógica de ejecución.

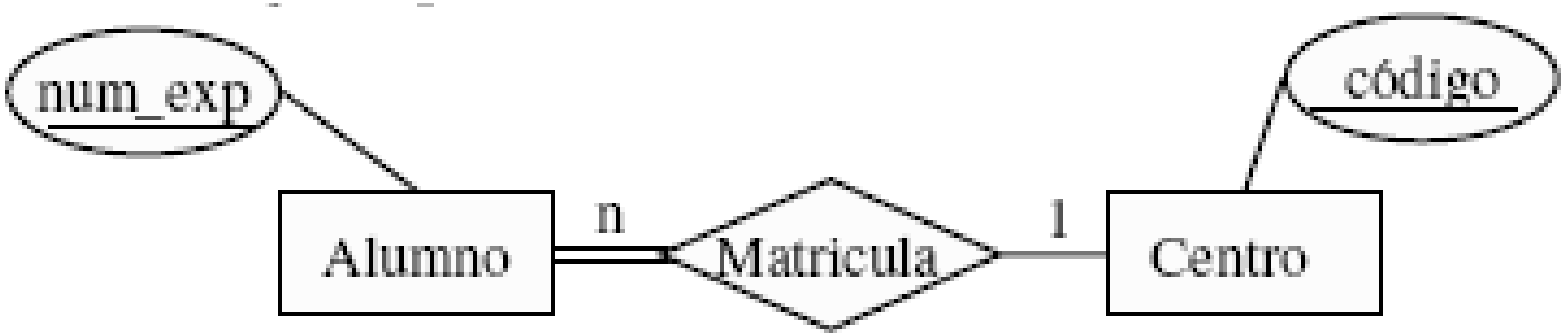
- Atomicidad, *(Atomicity)*
- Consistencia, *(Consistency)*
- Aislamiento, *(Isolation)*
- Persistencia *(Durability)*

ACID

ISO/IEC 10026-1

Diseño de Transacciones

- ¿Por qué son necesarias las transacciones?



Transacción Nuevo_alumno

- Insertar en *Alumno*
- Insertar en *Matricula*

(Previamente debe existir el centro en el que se matricula)

Diseño de Transacciones

Metodología de diseño de transacciones:

- **Objetivo: Obtener un conjunto de** *transacciones mínimas a partir del* diagrama ER.

- Este conjunto incluirá, para cada entidad y para cada relación del diagrama, una transacción mínima de **inserción, una de borrado y varias de modificación.**

- Transacciones mínimas: única forma permitida de modificar la información almacenada. Podrán ser utilizadas en transacciones más complejas.

Diseño de Transacciones

Transacción mínima:

- Incluye todas las operaciones que son necesarias para que la modificación del sistema de información sea **válida.**
- Constará siempre de **una operación básica (de inserción, borrado o** modificación) sobre el objeto (entidad o relación) para el que se está definiendo la transacción.
- Puede incorporar operaciones sobre otras entidades o relaciones con las que esté vinculado.

Diseño de Transacciones

- **Transacción Nuevo_alumno**
  - Insertar en *Alumno*
  - Insertar en *Estudia*

Transacción Mínima

Diseño de Transacciones

Transacción Nuevo_alumno

- Insertar en *Alumno*
- Insertar en *Estudia*
- Mientras se quiera seguir
  - Insertar en *Estudia*
- Fin_mientras

Transacción No Mínima

Diseño de Transacciones

Análisis de transacciones:

- Se decidirá, para cada entidad o relación del DER, qué operaciones sobre otros objetos serán necesarias para conseguir añadir, borrar o modificar una ocurrencia de esa entidad o relación.
- La determinación de dichas operaciones **se deduce lógicamente del DER**, al considerar las restricciones representadas en el mismo.
- En diagramas que no tienen restricciones de integridad añadidas, se concreta de la forma siguiente:

Diseño de Transacciones

Inserción

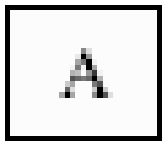
- ***Entidades:*** *Si una entidad tiene* restricción de existencia en alguna relación R, hay que insertar en R. En casos particulares hay que insertar en varias entidades a la vez.

- ***Relaciones:*** *Si la relación R en la* que se está insertando participa como objeto agregado con restricción de existencia en otra relación S, hay que insertar en S.

Diseño de Transacciones

- Transacciones de inserción

Entidades:



Transacción Nuevo_A

* Insertar en A

Transacción Nuevo_A

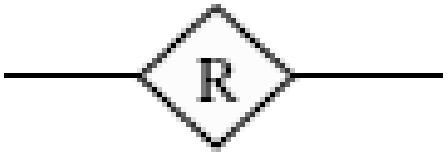
* Insertar en A

* Insertar en R

Diseño de Transacciones

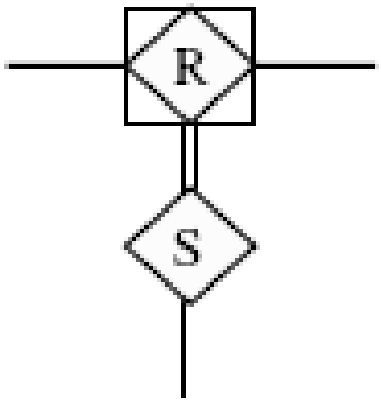
- Transacciones de inserción

Relaciones:



Transacción Nuevo_R

* Insertar en R



Transacción Nuevo_R

* Insertar en R

* Insertar en S

Diseño de Transacciones

Borrado: Se considerarán dos tipos: restrictivo y en cascada

Entidades:

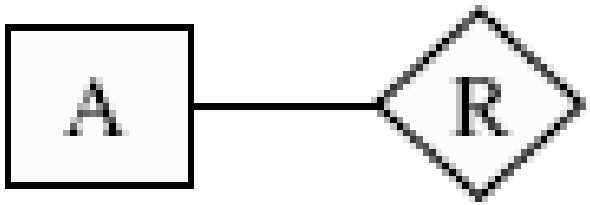
- ***Restrictivo respecto a una relación R:*** la transacción sólo incluye la operación de borrado de la entidad.

- ***En cascada respecto a una relación R:*** la transacción incluye, además de la operación de borrado de la entidad, el borrado de las ocurrencias de R en las que interviene la entidad. Si la entidad tiene restricción de existencia respecto a R, el borrado siempre tendrá que ser de este tipo.

Diseño de Transacciones

- Transacciones de borrado

Entidades:



Restrictivo

Transacción Borrar_A

* Borrar en A

Cascada

Transacción Borrar_A

* Borrar en A

* Borrar en R

Diseño de Transacciones

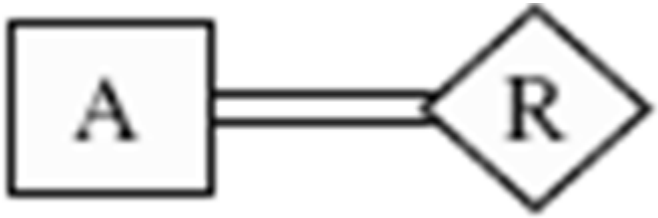
Relaciones:

- ***Restrictivo** respecto a una entidad o relación: no hay que añadir ninguna* operación.
- ***En cascada:** si alguna de las entidades* que participan en la relación tienen restricción de existencia en ella, hay que incluir también el borrado de dicha entidad para los casos en los que la ocurrencia de relación borrada sea la última en la que participaba la entidad.
- Además, si la relación participa como objeto agregado en otra relación, sea S, en la transacción se incluye el borrado de S para las ocurrencias en las que interviene el agregado.

Diseño de Transacciones

- Transacciones de borrado

Relaciones:



Restrictivo

Transacción Borrar_R

* Borrar en R

Cascada

Transacción Borrar_R

* Borrar en R

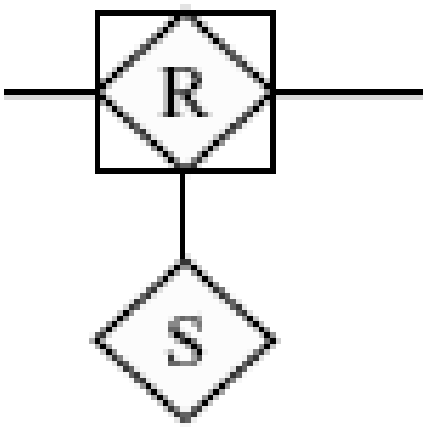
* Borrar en A

(si era el último en R)

Diseño de Transacciones

- Transacciones de borrado

Relaciones:



Restrictivo

Transacción Borrar_R

* Borrar en R

Cascada

Transacción Borrar_R

* Borrar en R

* Borrar en S

Diseño de Transacciones

- En el borrado en cascada o en la inserción de entidades con restricciones de existencia, las operaciones añadidas a la transacción pueden requerir a su vez una propagación, que también hay que incluir en la transacción.

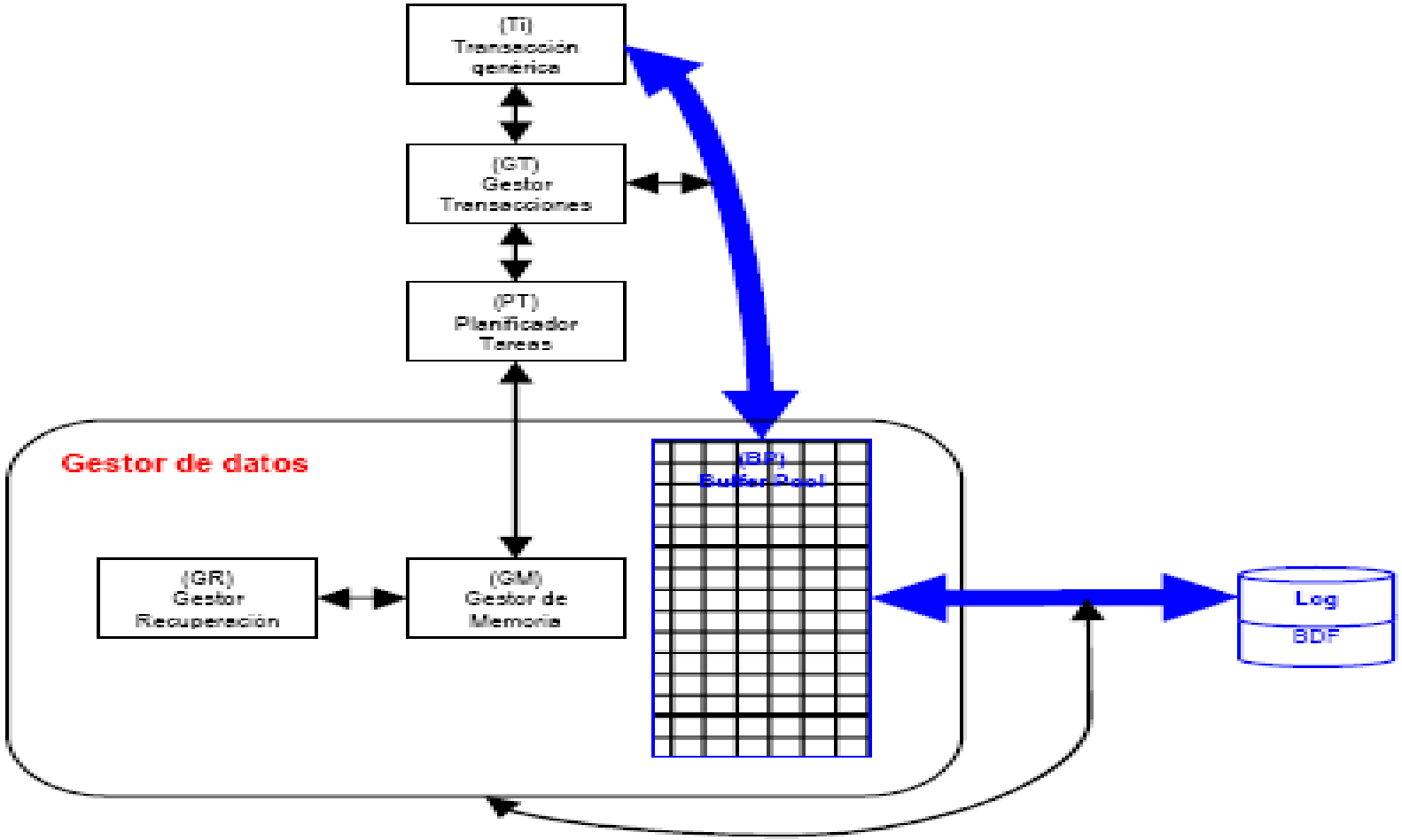
Modificación:

- Puede haber muchas transacciones de modificación diferentes.
- Deben diseñarse las transacciones que posibiliten modificar cualquier atributo de cualquier objeto que sea necesario

Arquitectura de gestión de transacciones

- El SGBD actúa como interfaz entre los programas(transacciones) y el sistema operativo (métodos de acceso a ficheros) para realizar la recuperación física de las páginas o bloques físicos de datos (unidades mínimas de transferencia entre la memoria externa y la memoria principal).
- En el esquema siguiente se presentan los distintos subsistemas en que puede descomponerse la funcionalidad del SGBD para dar servicios a estas transacciones y, simultáneamente, garantizar la consistencia de representación de estados en la base de datos física.

Arquitectura de gestión de transacciones



Arquitectura de gestión de transacciones

Se diferencian los siguientes subsistemas:

- (GT) Gestor de transacciones.
- (PT) Planificador de tareas o gestor de transacciones concurrentes.
- (GM) Gestor de memoria caché.
- (GR) Gestor de recuperación de la base de datos.

Arquitectura de gestión de transacciones

(GT) Gestor de transacciones

- Recibe las peticiones de las transacciones (lecturas, inserciones, actualizaciones, eliminaciones, etc.)
- Es capaz de servir los datos a dichas transacciones como retorno a su petición así como los códigos de estado (códigos de retorno) que informan del nivel de finalización de dicha petición.

Arquitectura de gestión de transacciones

(PT) Planificador de tareas

- Gestiona el ámbito de concurrencia de dichas transacciones .
- En función de las políticas y algoritmos implementados para resolver los conflictos de concurrencia que pueden originarse
  - Bloqueo
  - Métodos de ordenación de transacciones
  - Gestión optimista de transacciones
  - etc.
- El Planificador (Scheduler) puede decidir, y en qué orden, si acepta o rechaza determinadas peticiones que recibe del gestor de transacciones, implementando primitivas específicas de cancelación (ABORT) o reinicio (RESTART) de transacciones.

Arquitectura de gestión de transacciones

(GM) Gestor de memoria caché.

- Trata de optimizar el rendimiento global de transacciones por segundo) del sistema.
- Para ello la política se basa en acaparar la memoria rápida disponible (buffer pool(BP) del SGBD).
- En el (BP) se alojan los gránulos o páginas intercambiados con el almacenamiento externo.
- El (GM) procura retener estas páginas una vez cargadas en el (BP)

Arquitectura de gestión de transacciones

- Las transacciones que generan modificaciones de estas páginas generan en el (BP) una nueva versión de dicha página denominada página sucia (dirty page).
- Por otro lado, dado que el (BP) se aloja en memoria volátil, es imposible mantener indefinidamente una página sucia en el mismo, teniendo lugar, antes o después, una descarga de dichas páginas a la memoria externa o base de datos física (BDF).
- El (GM) tiene que gestionar todas estas operaciones buscando el mejor rendimiento global del sistema.

Arquitectura de gestión de transacciones

(GR) Gestor de recuperación de la base de datos.

- Tiene que implementar los procedimientos adecuados para garantizar la representación de estados persistentes (almacenados en la BDF) consistentes.

- (GR) requiere de una estructura adicional de datos en memoria externa para garantizar sus actuaciones:
  - el diario (Log) de transacciones,
    - manteniendo en el mismo un registro de la actividad de todas las transacciones
    - y de los procesos de descarga masiva de páginas del (BP) a la (BDF).

Arquitectura de gestión de transacciones

Los registros del Log están representados por:

- Puntos de sincronismo de transacciones: Begin(T), Commit(T), Abort(T). Marcan el inicio y la terminación, normal o anormal, de una transacción.
- Puntos de control del sistema(CheckPoint): Marcan la actividad de descarga masiva de páginas sucias del (BP) a la (BDF).
- Preimágenes de páginas: Es el estado de una página antes de la actualización. Las preimágenes permitirán devolver, en caso de fallo, el estado de la base de datos al estado consistente inicial o anterior al inicio de la transacción (Backward Recovery).

Arquitectura de gestión de transacciones

- Postimágenes de páginas: Es el estado de una página (página sucia en principio) después de la actualización (inserción, modificación o eliminación) generada por una transacción.
- Las postimágenes permitirán reconstruir, en caso de fallo, la actividad de una transacción hacia delante (Forward recovery) hasta el punto de confirmación de la misma, si este se produjo.
- Todos estos registros se almacenan en el Log cronológicamente, asociando a los mismos la transacción correspondiente.

E.T.S. INGENIERÍA INFORMÁTICA

Gestión de la Información

Grado en Ingeniería del Software

UNIVERSIDAD DE MÁLAGA

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

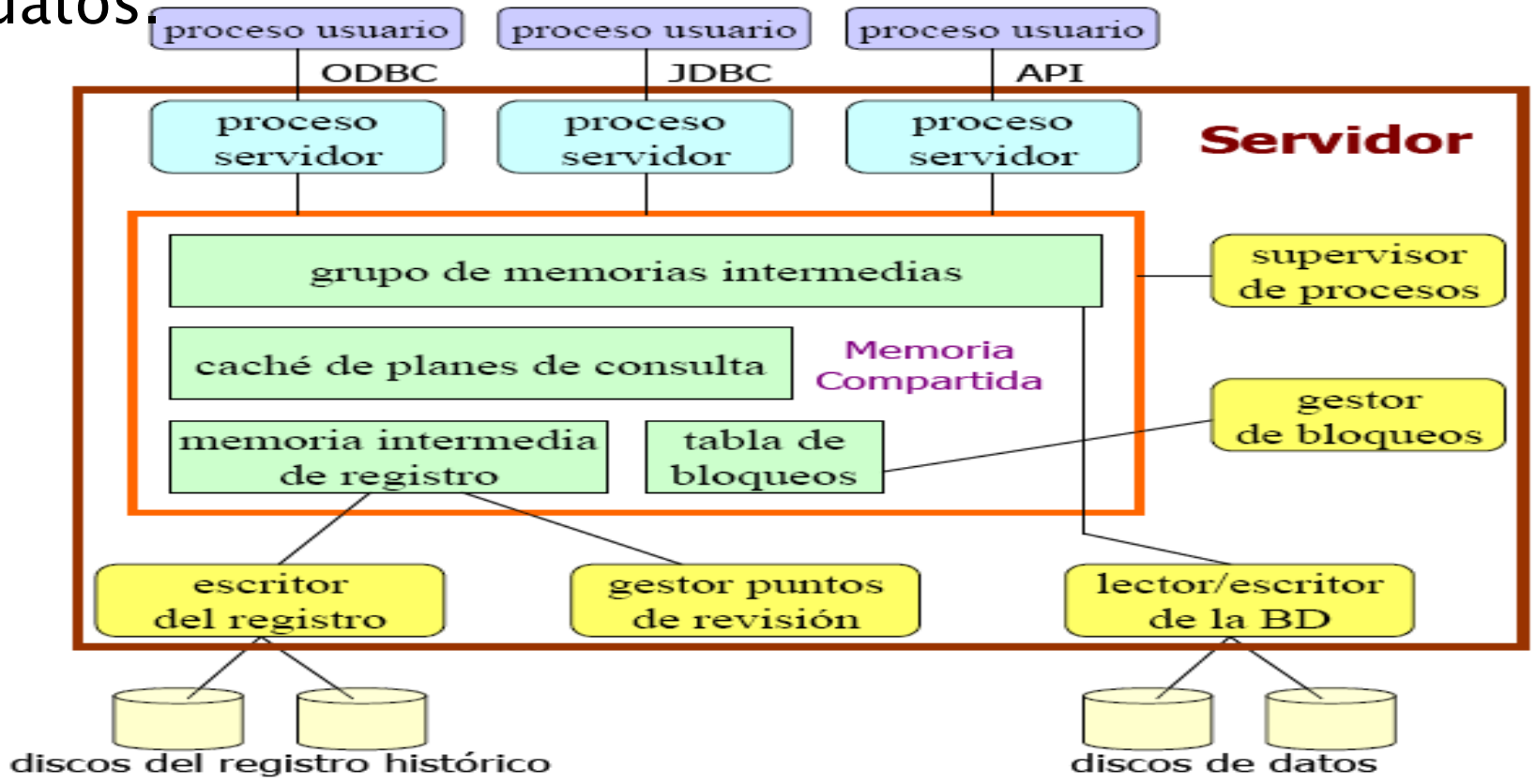
Tema 1.4

Conexiones dedicadas y compartidas: pool de conexiones.

José Luis Pastrana Brincones
pastrana@lcc.uma.es

Conexiones a Una Base de Datos

- Las conexiones de bases de datos son vínculos activos a una base de datos que permite leer y escribir datos, y crear objetos SQL en la base de datos.



Servidores Dedicados vs Servidores Compartidos

- En un servidor dedicado, se crea un proceso servidor por cada petición de conexión.
  - Según vayan aumentando el numero de conexiones el numero de procesos servidor ira aumentando y con ello la reserva de memoria RAM de nuestro sistema.
- En un servidor compartido, aparece el concepto dispatcher.
  - El dispatcher es el proceso que recibe y gestiona las conexiones.
  - Pasa las transacciones a una cola de procesado.
  - Una vez ejecutadas por el proceso servidor, éste depositara los resultados en una cola de respuesta, siendo el dispatcher el que responda al cliente con el resultado.

Servidores Dedicados vs Servidores Compartidos

- Usar un servidor dedicado es el método más común para conectar a una base de datos.
- Es más fácil de configurar y ofrece muchos beneficios cuando se hace debug sobre un proceso.

- Un servidor compartido posee una configuración más compleja pero puede ayudar a poder aceptar más conexiones usando el mismo hardware.

Servidores Dedicados vs Servidores Compartidos

- Las ventajas que ofrece un servidor compartido son:
  - Permite un mayor numero de conexiones sin tener que aumentar los requerimientos de hardware.
  - Permite desconectar a clientes que llevan mucho tiempo inactivos.
  - Permite balanceo de conexiones entre los diferentes procesos servidor.

- Las desventajas respecto al servidor dedicado son:
  - Si la cantidad de información que hay que devolver al cliente aumenta, el dispatcher podría llegar a saturarse.
  - Al utilizar un servidor compartido, algunas funciones de administración no están permitidas: startup, shutdown, algunas funciones de recuperación, recreación de índices o análisis de tablas.

Conexiones a un servidor dedicado

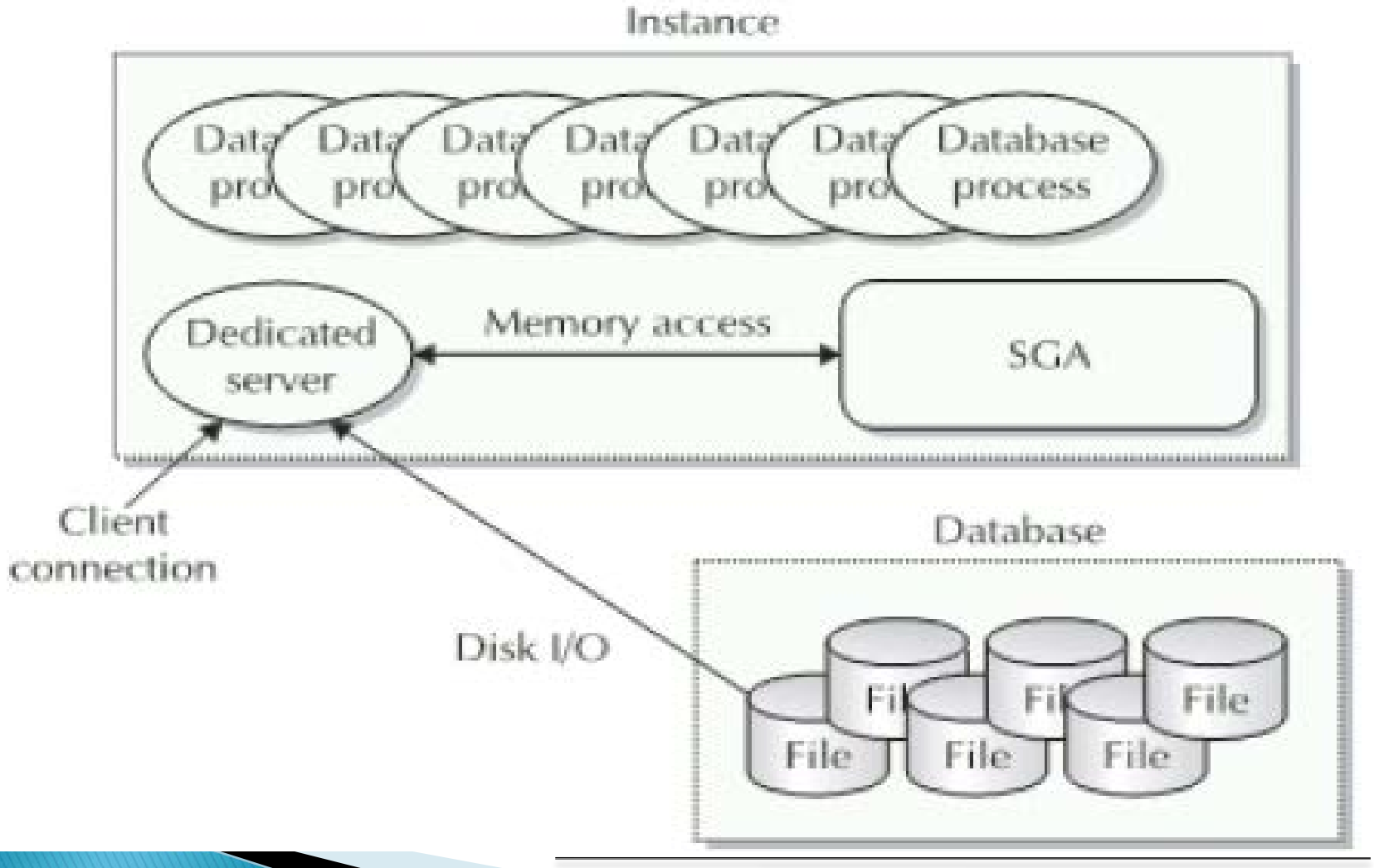
Pasos en una conexión a un servidor dedicado

- Cuando nos conectamos a un servidor dedicado a través de la red seguimos estos pasos:
  1. Nuestro proceso de cliente se conecta por la red al listener que normalmente está en el servidor de base de datos. Hay situaciones (como en el Oracle Cluster o RAC) en donde el listener no estará en la misma máquina que la instancia de base de datos.
  2. El listener crea un nuevo proceso dedicado (Unix) o solicita a la base de datos la creación de un nuevo thread dedicado (Windows) para nuestra conexión.
  3. El listener traspasa la conexión al nuevo proceso o nuevo thread.
  4. Nuestro cliente envía las peticiones al servidor dedicado el cual las procesa y nos retorna los resultados.

Conexiones a un servidor dedicado

- La figura de la siguiente transparencia muestra cómo funciona la configuración de servidor dedicado.
- La instancia de la Base de Datos será un único proceso con threads (Windows) o cada burbuja será un proceso físico separado (Unix).
- En una configuración de servidor dedicado cada cliente tiene su propio proceso asociado a él.
- El servidor dedicado recibirá nuestra sentencia SQL y la ejecutará. Leerá los archivos de datos y pondrá datos en cache o mirará en la cache por si ya están los datos, ejecutará sentencias UPDATE y ejecutará nuestro código SQL.
- Su único objetivo es responder a las llamadas SQL que le enviemos.

Conexiones a un servidor dedicado



Conexiones a un servidor dedicado

- En el modo servidor dedicado hay un mapeo uno-a-uno entre procesos de la base de datos y procesos de cliente.
- Cada conexión tiene un servidor dedicado designado exclusivamente para ella.
- Esto no significa que cada sesión tenga un servidor dedicado (es posible para una aplicación tenga muchas conexiones activas a la base de datos). Sin embargo por normal general cuando se trabaja en modo servidor dedicado hay una relación uno-a-uno entre una sesión y un servidor dedicado.

Conexiones a un servidor dedicado

Ventajas de las conexiones dedicadas

- Es fácil configurarlo. De hecho no requiere ninguna configuración.
- Es el modo más rápido de operar en la base de datos al usar el camino más corto.
- Debido a que los archivos de traza están ligados a un proceso las utilidades como SQL_TRACE se benefician de este modo.
- Todas las tareas administrativas están disponibles.
- La memoria para el Área Global de Usuario (UGA), que es una zona de memoria específica por sesión no necesita ser configurada.

Conexiones a un servidor dedicado

Desventajas de las conexiones dedicadas

- El cliente tiene un proceso/thread consumiendo recursos (memoria, CPU. etc.) del servidor desde que la sesión empieza hasta que termina;
- Al aumentar el número de usuarios el sistema operativo puede sobrecargarse debido al manejo de sus procesos/threads.
- Se tiene un control limitado sobre la cantidad de sesiones que pueden estar activas simultáneamente.
- Si se tiene un ratio muy alto de conexiones/desconexiones a los datos, los procesos de creación y destrucción de procesos/ threads pueden ser molestos.

Para la mayoría de sistemas los beneficios sobrepasan a las desventajas. En los servidores convencionales (máquinas con 2 CPU) se necesitarían de 200 a 400 conexiones concurrentes antes de considerar un tipo diferente de conexión. En máquinas con más potencia el límite es mucho más alto.

Conexiones a un servidor Compartido

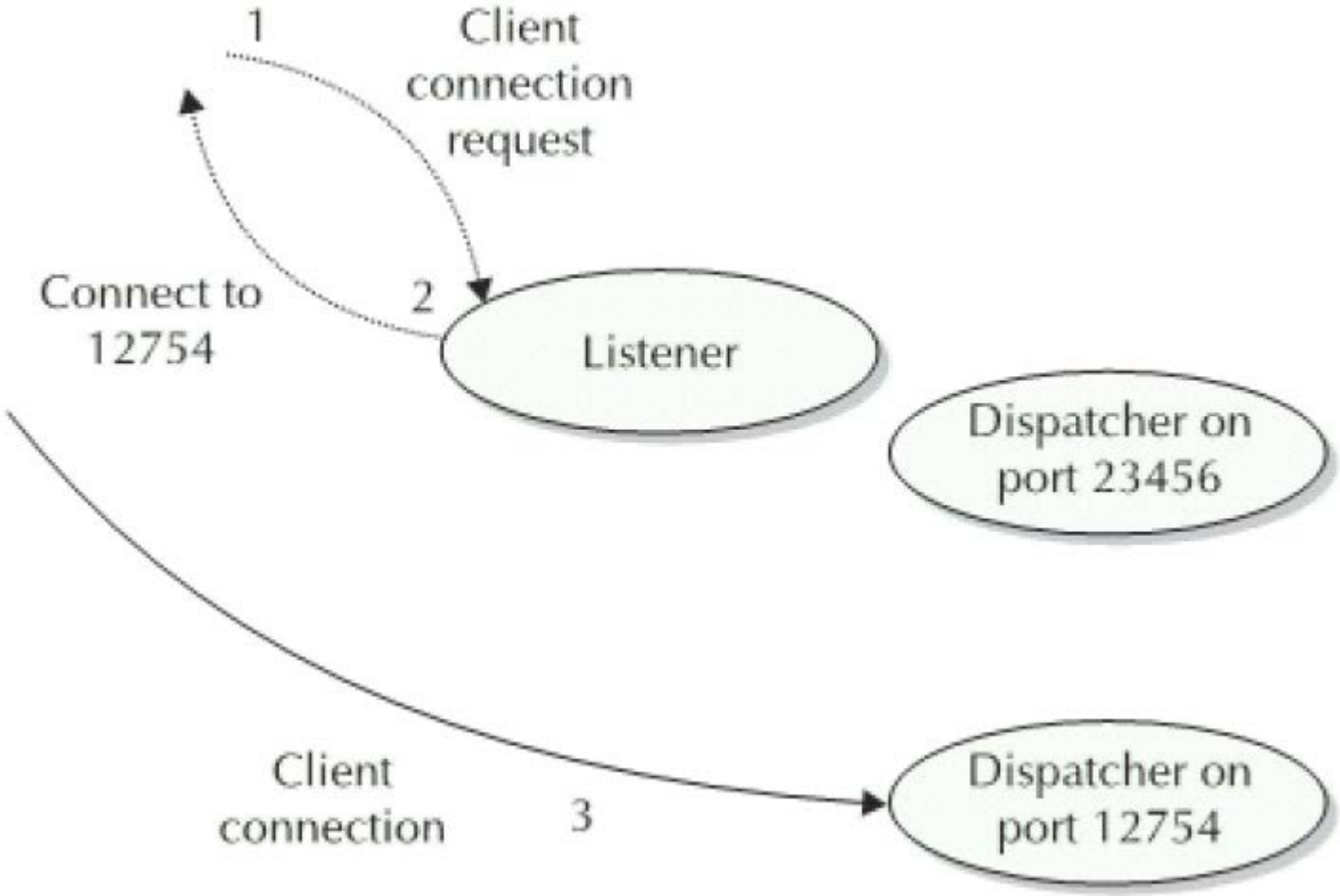
- El protocolo de conexión para una conexión a un servidor compartido es muy diferente del usado para una configuración de servidor dedicado.
- En el modo "shared server" no hay un mapeo uno-a-uno entre clientes (sesiones) y procesos/ threads del servidor.
- Hay un “repositorio" (= pool) llamado "shared servers” que realiza las mismas operaciones que un servidor dedicado, pero las realizan para múltiples sesiones en vez de para una sesión sólo.
- Los clientes estarán conectados a un "despachador" (dispatchers) durante toda la vida de su sesión.
- El dispatcher facilitará el manejo de sus peticiones hacia un "shared server" y les retornará la respuesta.
- Cuando nos conectamos el "listener" nos redirigirá hacia el "dispatcher" disponible.

Conexiones a un servidor Compartido

Pasos en una conexión a un servidor compartido

- Cuando nos conectamos vía conexiones de servidor compartido se suceden estos pasos:
  1. El proceso cliente se conecta a través de la red al "listener" funcionando en la base de datos. Este "listener" elegirá un "dispatcher" desde el “pool” de "dispatchers" diponibles para que se conecte.
  2. El "listener“ retornará la dirección del "dispatcher" elegido para que el cliente se conecte a él, o también podrá ser redirigido directamente al "dispatcher". Esto sucede a nivel TCP/ IP y depende del sistema operativo. Si no hemos sido conectados directamente al "dispatcher" nuestro cliente se desconectará del "listener" y se conectará al "dispatcher".
  3. El proceso de cliente envía su petición al "dispatcher".

Conexiones a un servidor Compartido



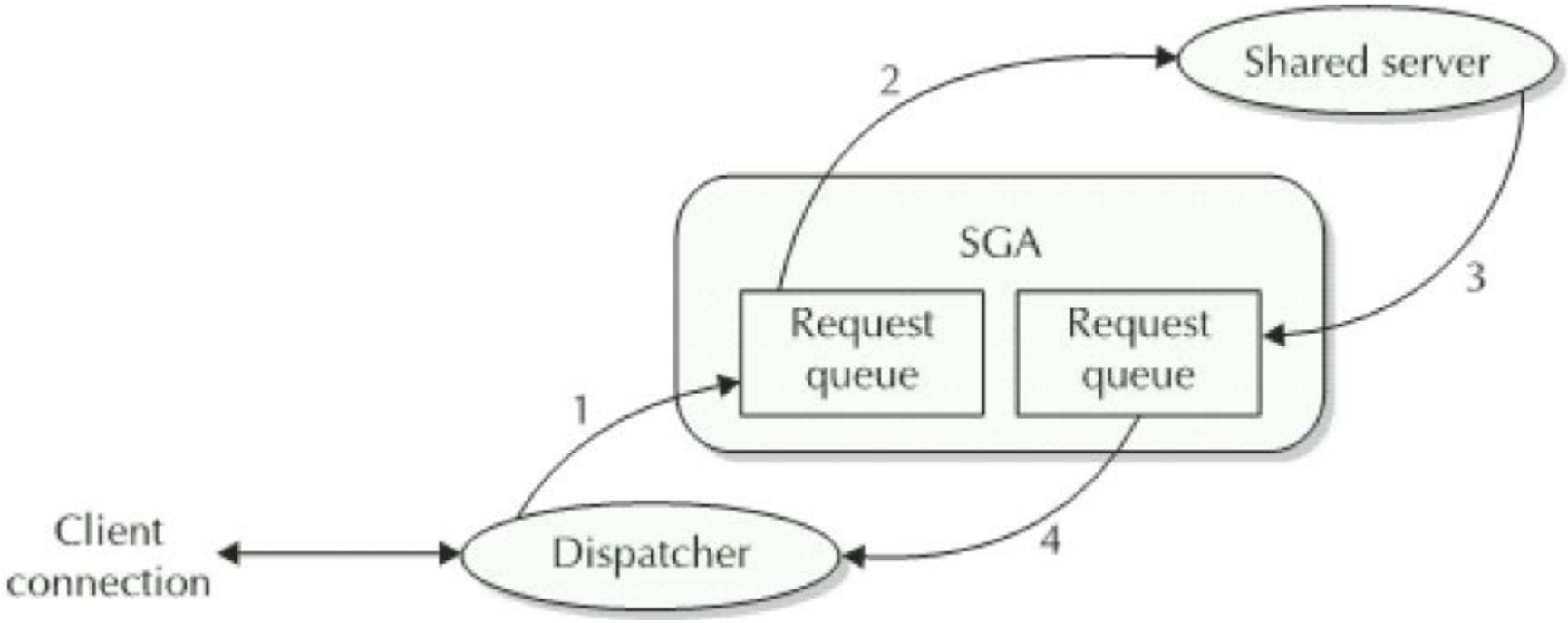
Conexiones a un servidor Compartido

Procesamiento de comandos en un servidor compartido

1. El "dispatcher" coloca la petición en una cola común (compartida por todos los "dispatchers") de la SGA (System Global Area).
2. El primer "shared server" disponible coge y procesa esa petición.
3. El "shared server" coloca la respuesta a esa petición en su cola privada de respuestas de la SGA.
4. El "dispatcher" al que estamos conectados cogerá esa respuesta y nos la retornará.

Conexiones a un servidor Compartido

Procesamiento de comandos en un servidor compartido



Conexiones a un servidor dedicado

Ventajas de las conexiones compartidas

- Es tan escalable así como necesitemos aumentarlo. Una máquina que pueda físicamente manejar 100 conexiones de servidor dedicado será capaz de manejar 1.000 ó más conexiones a servidor compartido, dependiendo de la aplicación.
- Notad que este escalamiento se refiere sólo a la CPU (todavía necesitamos memoria suficiente para esas 1.000 conexiones).
- Permite un fino control sobre cuantas sesiones pueden estar activas de forma concurrente;.
- Impone un límite "duro" al número total de sesiones activas.

Conexiones a un servidor dedicado

Ventajas de las conexiones compartidas

- Usa marginalmente menos memoria que la configuración de servidor dedicado.
- Permite que haya N+1 usuarios conectados cuando en un servidor dedicado sería imposible.
- Si tenemos un alto ratio de conexiones/desconexiones el uso de servidores compartidos puede ser más rápido que servidores dedicados.
- La creación y destrucción de procesos y threads es muy cara y puede sobrecargar al sistema.

Conexiones a un servidor dedicado

Desventajas de las conexiones compartidas

- Es más compleja de preparar que la configuración de servidor dedicado, aunque las herramientas GUI proporcionadas con la base de datos la ocultan generalmente.
- En un sistema donde las conexiones de servidor dedicado trabajan bien, las conexiones de servidor compartido pueden ser más lentas.
- La ruta hacia un "shared server" es por definición más larga que la ruta hacia un servidor dedicado.
- Algunas de las características de la base de datos no están disponibles cuando nos conectamos vía "shared server".

Conexiones a un servidor dedicado

Desventajas de las conexiones compartidas

- Por ejemplo usando conexiones "shared server" no podemos detener o iniciar la instancia,
- realizar recuperaciones o usar "Log Miner".
- La memoria total de todas las sesiones concurrentes debe ser computada y configurada como parte del parámetro LARGE_POOL.
- El traceado (usando SQL_TRACE) no es algo aplicable de forma cómoda a las conexiones "shared server".
- Los archivos de traza son específicos de un proceso, y en un servidor compartido un proceso no es específico de una sesión.

Conexiones a un servidor dedicado

Desventajas de las conexiones compartidas

- Existe el peligro de "deadlocks" artificiales.
- Ocurren porque una vez que un "shared server" inicia el procesamiento de una petición no retornará nada hasta que haya acabado con ella.
- Por lo tanto si el "shared server" está enviando una petición UPDATE y la fila ya está bloqueada, se bloqueará.
- Supongamos que la sesión bloqueante estuvo sin hacer nada durante un período de tiempo y el resto de los otros "shared servers" están bloqueados por esta sesión.

Conexiones a un servidor dedicado

Desventajas de las conexiones compartidas

- Cuando la sesión que mantiene el bloqueo intenta realizar un COMMIT o ROLLBACK (lo cual liberaría los bloqueos), se colgará porque no existen más "shared server" libres.

- Esta es una situación de "deadlock" artificial donde el bloqueante no puede liberar a los bloqueados porque éstos tienen copados al resto de servidores compartidos.

Conexiones a un servidor dedicado

Desventajas de las conexiones compartidas

- Hay un riesgo potencial de monopolización del "shared server".
- Esto ocurre por la misma razón que los "deadlocks" artificiales.
- Si usamos un "shared server" con transacciones muy largas nuestras sesiones monopolizarán un recurso compartido, previniendo a los otros de usarlo.
- Si tenemos muchas de estas transacciones largas el rendimiento del sistema caerá de forma abismal porque las transacciones pequeñas deberán esperar a que las largas se completen.

Falsos mitos sobre los servidores dedicados

Los servidores compartidos usan muchísima menos memoria.

- La memoria necesitada dependerá del número de procesos utilizados.
- En el caso de un servidor dedicado estará determinada por el número de procesos dedicados y en el caso de un servidor compartido por el número de dispatchers.
- La configuración "shared server" puede ahorrar memoria pero no tanta como mucha gente cree.

Falsos mitos sobre los servidores dedicados

Se debe usar siempre el servidor compartido con servidores de aplicaciones.

- Si tenemos una granja inmensa de servidores de aplicación, donde cada uno tiene un gran "pool" de conexiones conectando a una única instancia de base de datos, debemos considerar usar "shared server".
- Si tenemos un gran número de conexiones y estamos sobrecargando el servidor de base de datos, es mejor usar conexiones "shared server" para reducir el número de procesos físicos.
- En el caso de que no tenemos una gran cantidad de sesiones y no necesitamos rápidos ratios de conexión/desconexión, lo mejor es usar servidores dedicados.

Falsos mitos sobre los servidores dedicados

Los servidores compartidos sólo son para aplicaciones cliente/ servidor.

- No se debería usar "shared server" para un servidor de aplicaciones dado que él ya hace un "connection pooling".
- El servidor de aplicación es sólo un cliente. Tenemos clientes (conexiones del servidor de aplicaciones) y un servidor (la base de datos).
- Desde la perspectiva de la base de datos todo son conexiones.

Conclusión

Debemos mirar el número de conexiones concurrentes a la base de datos. Si se generan más procesos de los que el sistema puede manejar las conexiones "shared server" serán la solución.

En caso contrario, lo mejor será usar un servidor dedicado.