

# Curso de Programación para Internet con Java EE (Parte III – Database Management)

Antonio Santos Ramos

[antoniosantosramos@gmail.com](mailto:antoniosantosramos@gmail.com)

Curso de Programación para Internet con Java EE

(29 Octubre 2018 – 14 Diciembre 2018)<sub>v14</sub>

Java (Lucatic) L-J(8h-17:30h) V(8h-14h) Puente 2/11, 9/11 y 07/12)

© Antonio Santos Ramos 2018

1

# Curso de Programación para Internet con Java EE (Parte III - Database)

T0 - Índice

© Antonio Santos Ramos 2018

2

# Contenido

## Curso de Programación para Internet con Java EE (Parte III - Database)

### Temario

- T01 – BBDD Modeling
- T02 – JDBC
- T03 – MySQL
- T04 – PL/SQL

© Antonio Santos Ramos 2018

3

# Curso de Programación para Internet con Java EE (Parte III - Database)

## T01

### BBDD Modeling

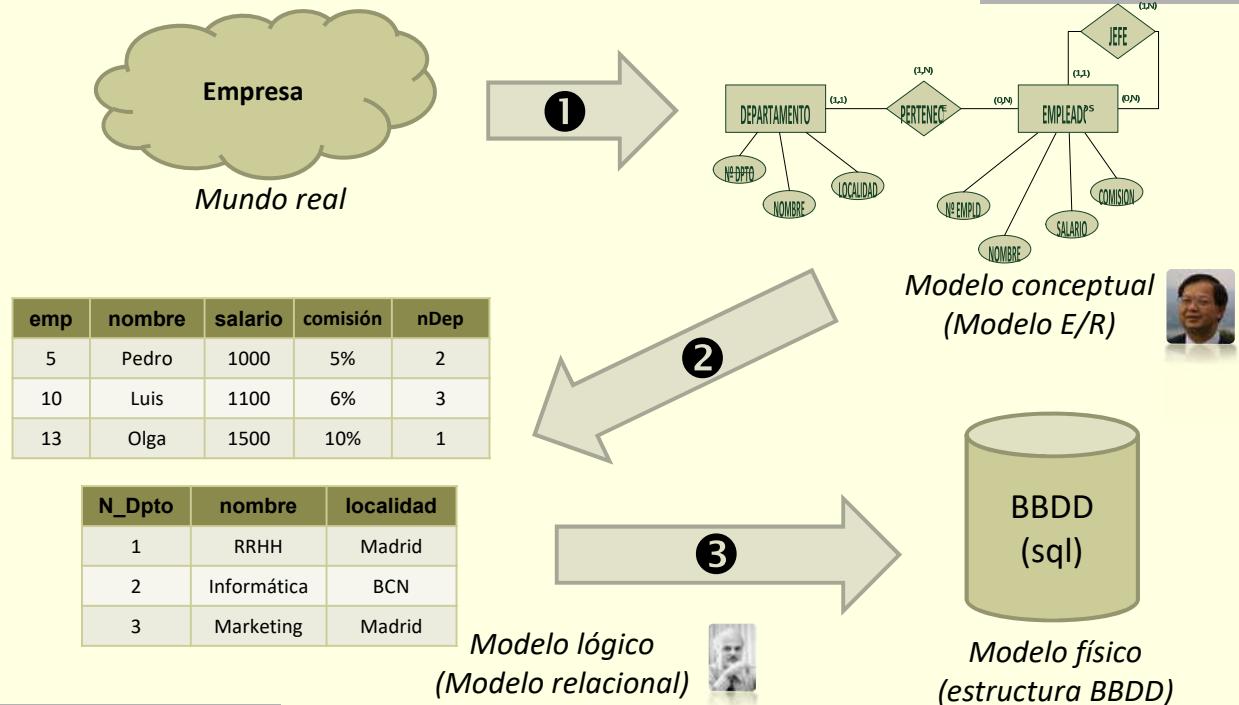
# Bases de datos. Aspecto

EDITORIAL				
LIBRO				
AUTOR				
nombre	...	nombre	apellidos	...
Obelisco	...	Paulo	Coelho	...
Alfaguara	...	Oscar	Wilde	...
Planeta	...	Michael	Ende	...
Alianza	...	...	...	...
...	...	...	...	...
id	nombre	teléfono		
2	Julia Ibáñez	555123456		
10	Eva Andrés	555654321		
...	...	...		
3	Cristina Prats	555987654		
1	Ginés Soriano	555221122		
PERSONA				
PRESTAMO				
idPer	idLib	fecha		
10	8408049003	23/9/03		
3	8408048783	1/10/03		
10	8420464988	2/3/03		
1	8420432261	10/8/02		
...	...	...		

© Antonio Santos Ramos 2018

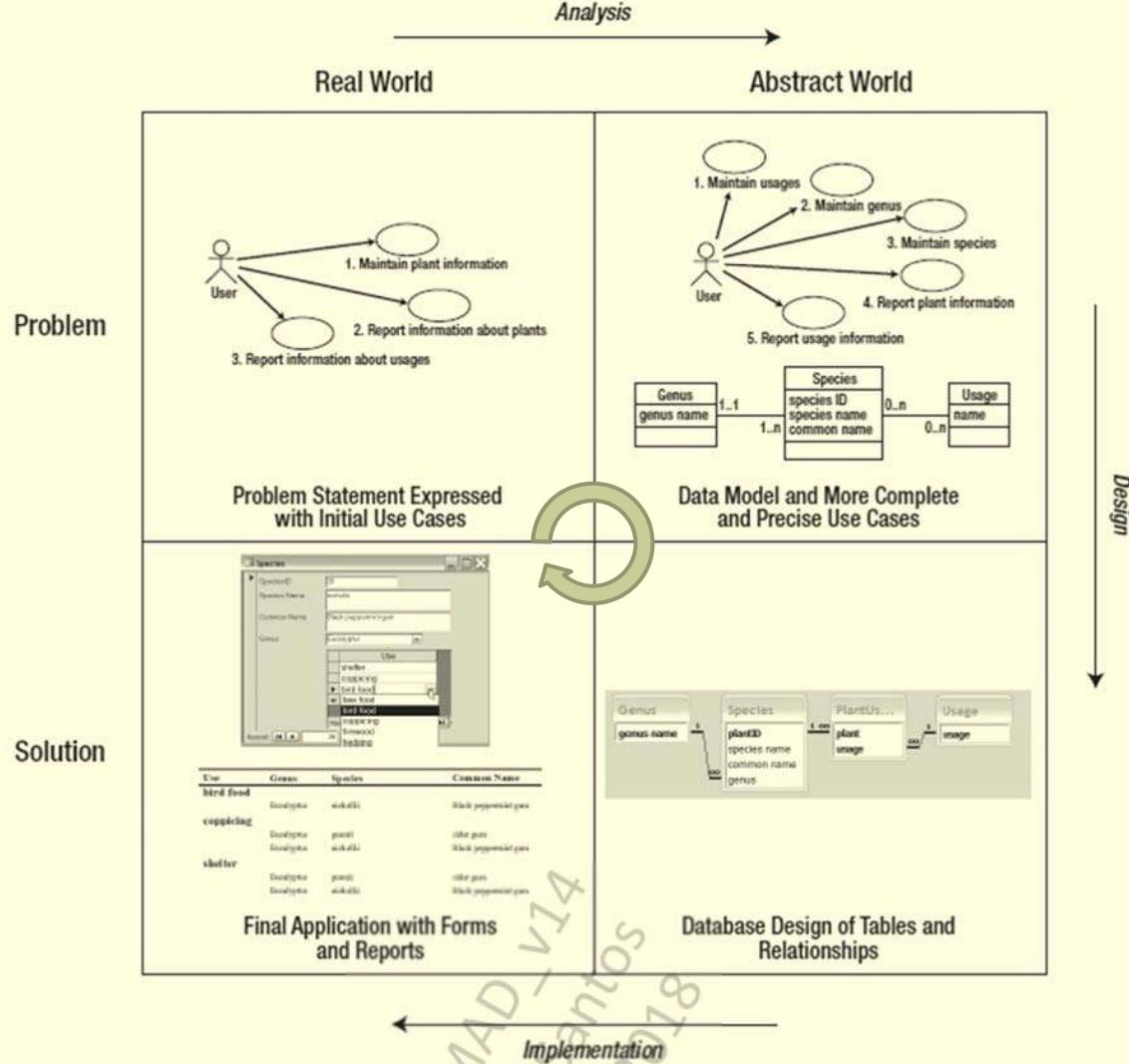
5

## Creación de una Base de Datos (I/II)



© Antonio Santos Ramos 2018

6

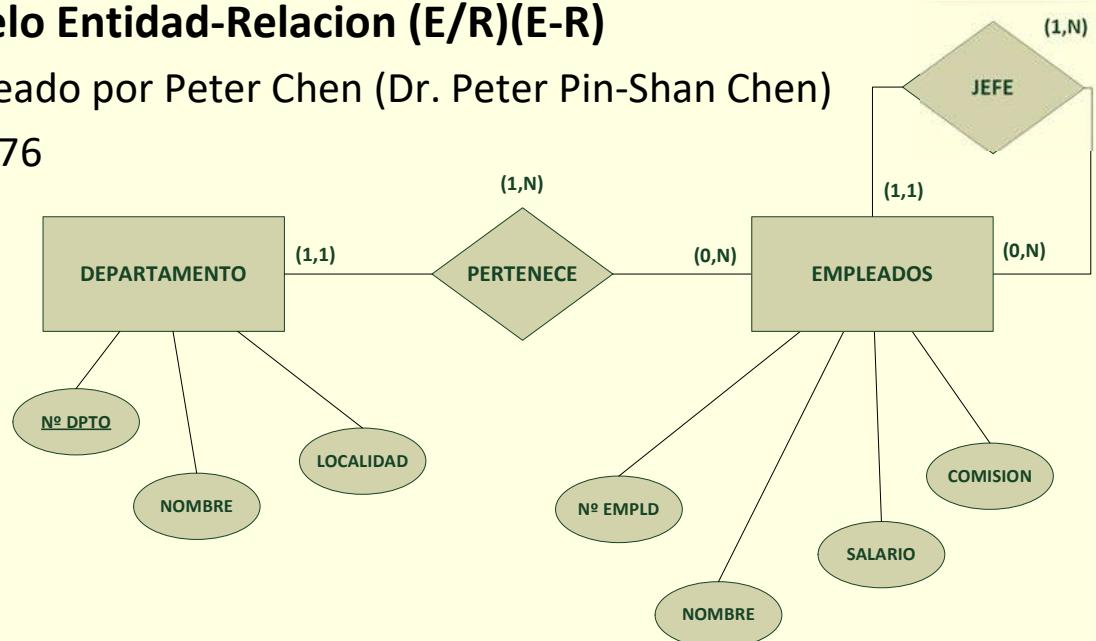


## Modelo Entidad Relación (Modelo conceptual)



### ■ Modelo Entidad-Relacion (E/R)(E-R)

- Creado por Peter Chen (Dr. Peter Pin-Shan Chen)
- 1976



[http://en.wikipedia.org/wiki/Comparison\\_of\\_data\\_modeling\\_tools](http://en.wikipedia.org/wiki/Comparison_of_data_modeling_tools)

# Modelo Entidad Relación (II/VIII)

## Entidad

- Definición
  - **Entidad** : objeto (real o abstracto) con características diferenciadoras que lo distinguen de otros objetos.
- Representación
  - Se representa por medio de un rectángulo.
  - En el interior se escribe el identificador
- Características
  - Identifica objetos de los cuales almacenaremos información
  - Normalmente se asocia con un sustantivo
- Tipos
  - **Entidad Fuerte**: su existencia no depende de otra entidad
  - **Entidad Débil**: existencia condicionada a otra entidad

© Antonio Santos Ramos 2018

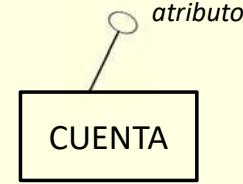
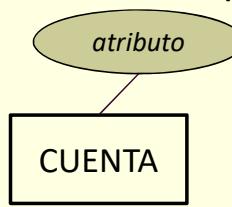
CUENTA

MOVIMIENT

# Modelo Entidad Relación (III/VIII)

## Atributos

- Definición
  - **Atributo**: propiedad o característica asociada a una determinada entidad. Se utilizan para describir la entidad
- Representación
  - Elipse o círculos
- Características
  - Un atributo es común a todas las ocurrencias de esa entidad.
    - Ej.- Entidad Alumno: Nombre, Apellidos, Expediente, ...
  - **Dominio** de un Atributo: conjunto de valores permitidos
    - Ej.- atributo COLOR
      - {NARANJA, BLANCO, AZUL y NEGRO}



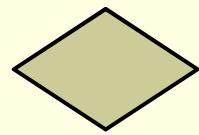
© Antonio Santos Ramos 2018

# Modelo Entidad Relación (IV/VIII)

## Relaciones (I/V)

### Definición

- Relación:** asociación entre dos o más entidades.



### Características

- Entre dos entidades puede existir más de una relación
- Suelen asociarse con verbos y pueden tener atributos propios. Ej.- fecha

### Propiedades:

- Nombre**

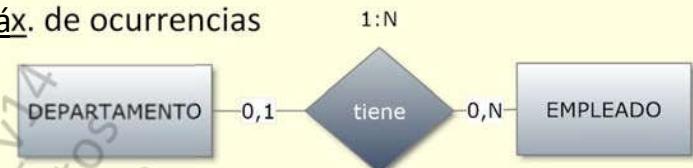
- Grado:** Nº de entidades que participan en la relación.

  - Puede ser reflexiva (1), binaria (2) o ternaria (3)

- Cardinalidad / Multiplicidad:** nº máximo y mínimo de ocurrencias

- Tipo de Correspondencia:** Nº máx. de ocurrencias

  - Puede ser 1:1 1:N M:N



# Modelo Entidad Relación (V/VIII)

## Relaciones (II/V)

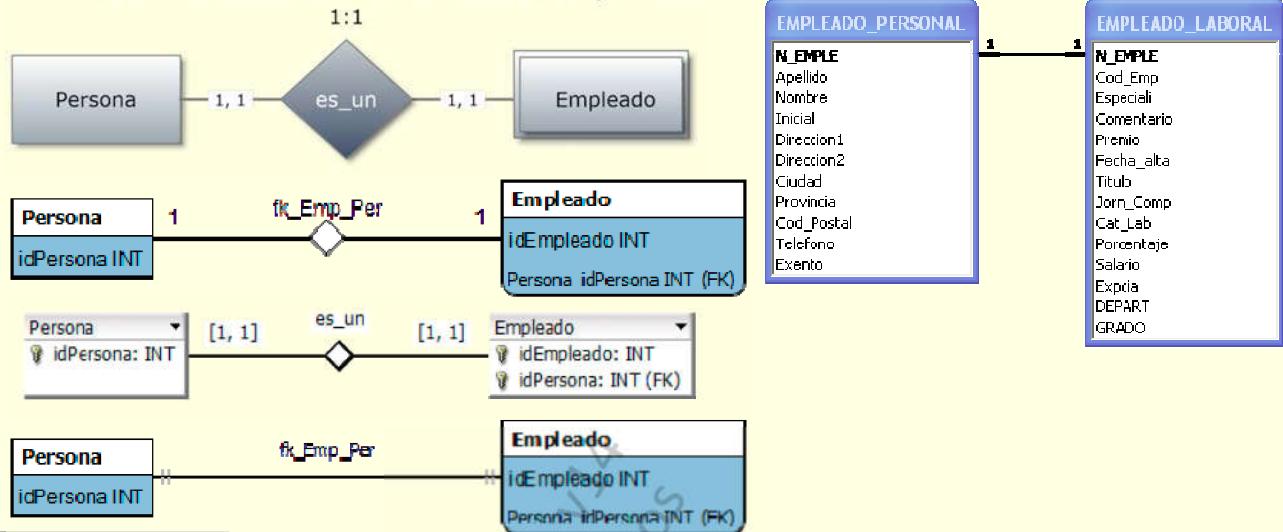
		Left to Right	Right to Left									
Example 2-1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">Plant</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">Usage</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">0..*</td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">0..*</td> <td style="padding: 5px;"></td> </tr> </table>	Plant		Usage		0..*			0..*		<i>One particular plant may have no usages or it may have any number of usages.</i>	<i>One particular usage may have no plants associated with it, or it may have a number of plants associated with it.</i>
Plant		Usage										
	0..*											
	0..*											
Example 2-2	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">Person</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">Interest</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">1..*</td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">0..*</td> <td style="padding: 5px;"></td> </tr> </table>	Person		Interest		1..*			0..*		<i>One person may have lots of interests or may have none.</i>	<i>Each interest has at least one person associated with it and maybe several people.</i>
Person		Interest										
	1..*											
	0..*											
Example 2-3	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">Customer</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">Transaction</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">1..1</td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">0..*</td> <td style="padding: 5px;"></td> </tr> </table>	Customer		Transaction		1..1			0..*		<i>One customer may have several transactions but doesn't necessarily have any at all.</i>	<i>Each transaction is for exactly one customer.</i>
Customer		Transaction										
	1..1											
	0..*											
Example 2-4	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 5px;">Visit</td> <td style="padding: 5px;"></td> <td style="padding: 5px;">Sample</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">1..1</td> <td style="text-align: center; padding: 5px;">1..*</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="text-align: center; padding: 5px;">1..*</td> <td style="padding: 5px;"></td> </tr> </table>	Visit		Sample		1..1	1..*		1..*		<i>Each visit has at least one sample associated with it and maybe several.</i>	<i>Each sample comes from exactly one visit.</i>
Visit		Sample										
	1..1	1..*										
	1..*											

# Modelo Entidad Relación (VI/VIII)

## Relaciones (III/V)

### ■ Relación 1:1 Ej.- Ayuntamiento (A) /ciudad (B)

- Cada ocurrencia de la entidad A está asociada con 0 ó 1 ocurrencias de la entidad B y viceversa.



© Antonio Santos Ramos 2018

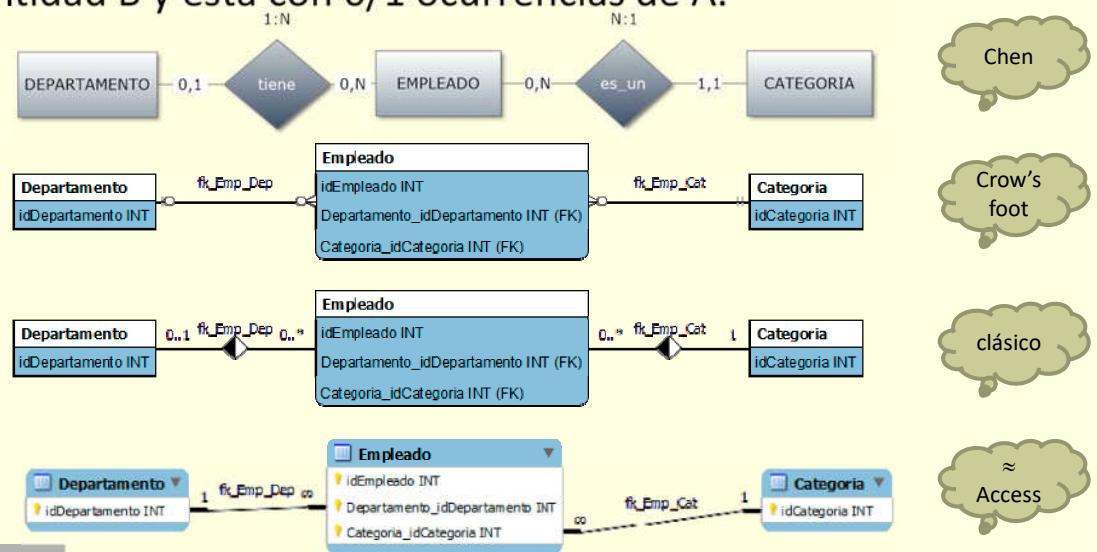
13

# Modelo Entidad Relación (VII/VIII)

## Relaciones (IV/V)

### ■ Relación 1:M (Ej.- Periodista (A) /noticia (B))

- Cada ocurrencia de la entidad A está asociada con 0/1/M de la entidad B y ésta con 0/1 ocurrencias de A.



© Antonio Santos Ramos 2018

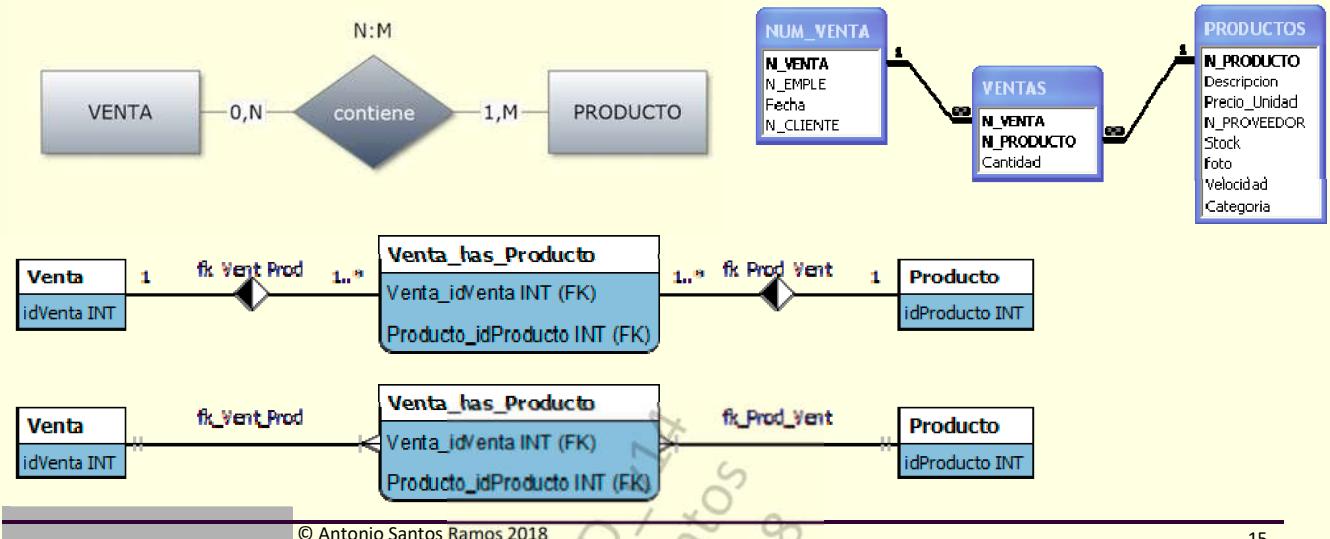
14

# Modelo Entidad Relación (VIII/VIII)

## Relaciones (V/V)

### ■ Relación M:N (Ej.- Pedido (A) / Artículo (B))

- Cada ocurrencia de la entidad A está asociada con 0/1/M de la entidad B y ésta con 0/1/N ocurrencias de A.



© Antonio Santos Ramos 2018

15

# Modelo Relacional (I/IV)

## Tabla



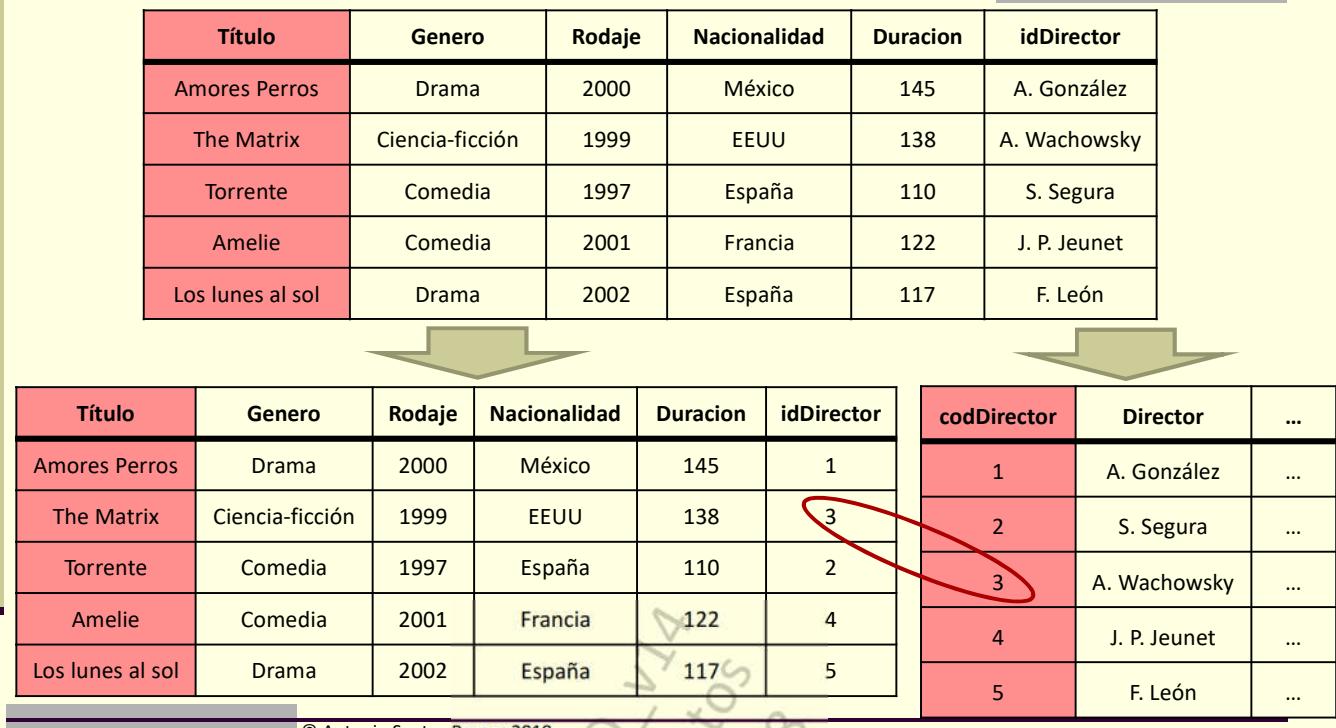
Codd  
1970

The diagram illustrates the relational model. At the top, six clouds represent domains: 'Títulos', 'Nombres', 'Géneros' (listing 'Ciencia-ficción, Drama, Thriller, Comedia...'), 'Años' (listing '2002, 1997, 1999, 2001, 1994, 1972...'), 'Países' (listing 'Italia, Argentina, España, EEUU, Francia, Japón...'), and 'Tiempo'. Below them is a table with columns: Título, Director, Género, Rodaje, Nacionalidad, and Duración. Data rows include: 'Amores Perros', 'A. González', 'Drama', '2000', 'México', '145'; 'The Matrix', 'A. Wachowsky', 'Ciencia-ficción', '1999', 'EEUU', '138'; 'Torrente', 'S. Segura', 'Comedia', '1997', 'España', '110'; 'Nos miran', 'N. López', 'Policíaco', '2001', 'España', '118'; 'Amelie', 'J. P. Jeunet', 'Comedia', '2001', 'Francia', '122'; and 'Los lunes al sol', 'F. León', 'Drama', '2002', 'España', '117'. To the left, vertical arrows indicate cardinality: an upward arrow for 'dominios' and a downward arrow for 'cardinalidad'. To the right, double-headed arrows indicate relationships: one for 'Filas, Tuplas, registros' pointing to the table, another for 'Atributos, campos' pointing to the columns, and a third for 'grado' pointing to the width of the table. Arrows also point from the clouds to the corresponding table columns.

Título	Director	Género	Rodaje	Nacionalidad	Duración
Amores Perros	A. González	Drama	2000	México	145
The Matrix	A. Wachowsky	Ciencia-ficción	1999	EEUU	138
Torrente	S. Segura	Comedia	1997	España	110
Nos miran	N. López	Policíaco	2001	España	118
Amelie	J. P. Jeunet	Comedia	2001	Francia	122
Los lunes al sol	F. León	Drama	2002	España	117

# Modelo Relacional (II/IV)

## Tablas



# Modelo Relacional (III/IV)

## Claves

### ■ Concepto de Clave

#### ■ Clave primaria o principal (Primary Key o PK)

- Es la clave candidata seleccionada por el diseñador de la BD.
  - Identifica un registro
  - La PK no puede tener valores nulos
  - No debe variar con el tiempo
- Puede ser simple o compuesta (Ej consulta médico)

#### ■ Clave ajena o foránea (Foreign Key o FK)

- Atributo o conjunto de atributos de una tabla (entidad) que forman la clave primaria en otra entidad.
- Se utilizan para generar las relaciones entre tablas
- Una FK debe ser PK en otra tabla

# Modelo Relacional (IV/IV)

## Representación de tablas

### ■ Representación de tablas

- Se escribe el nombre de la tabla
- Se escriben a continuación sus atributos (entre paréntesis)
- El primer atributo debe ser la clave (precedido por un #).
- La PK y las FK suelen subrayarse y/o enlazarse

### ■ Ejemplo

- |            |  |
|------------|--|
| ■ Profesor | <u>(#idprofesor</u> , nombre, apellidos, ...)                  |
| ■ Alumno   | ( <u>#idalumno</u> , nombre, apellidos... <u>#idprofesor</u> ) |

# Transformación de un modelo E/R a un modelo relacional (I/III)

### ■ Transformaciones de atributos, entidades y relaciones

Elemento en el Modelo E/R		Elemento en el Modelo relacional
Entidad	se transforma en	Tabla
Atributo	se transforma en	Columnas de la tabla
		Filas
Atributo identificador	se transforma en	Clave Principal (PK)
Relaciones	se transforma en	Nuevas tablas o claves FK

# Transformación de un modelo E/R a un modelo relacional (II/III)

## ■ Transformación de las relaciones normales

### ■ Relación 1:1 Ej.- Ayuntamiento (A) /ciudad (B)

- Se crea una tabla de cada entidad.
- O bien cada tabla incluye un FK de la otra o bien una de ellas lleva el FK de la otra o bien se hace una 3<sup>a</sup> tabla

### ■ Relación 1:M Ej.- Periodista (A) /noticia (B)

- Se hace una tabla de cada entidad.
- La tabla B lleva un FK de la tabla A
- Si la relación tuviera atributos, podría tratarse como relación N:M

Ej.- Jefe-evalúa-empleado

### ■ Relación M:N (o M:N:P) Ej.- Pedido (A) / Artículo (B)

- Se hace una tabla de cada entidad y una tercera para la relación
- La tercera tabla lleva el FK de la tabla A y el FK de B (y los atributos de la relación)

# Transformación de un modelo E/R a un modelo relacional (III/III)

## ■ Transformación de las relaciones reflexivas

### ■ Relación 1:1 Ej.- Hombre -conyuge- mujer

- Opcion 1) Se hace una tabla de la entidad repitiendo la FK.
- Opcion 2) Se crea una tabla nueva con los dos PK

### ■ Relación 1:M Ej.- Empleado -dirige- jefe

- Opción 1) Se hace una tabla de la entidad repitiendo la FK.
- Opción 2) Se crea una tabla nueva con los dos FK

### ■ Relación M:N Ej.- Empleados -dirige- jefes

- Es como el caso M:N (binario)
- Se hace una tabla de la entidad y una segunda para la relación
- La segunda tabla lleva los FK de ambas entidades (y los atributos de la relación)

# Curso de Programación para Internet con Java EE (Parte III - Database)

## T02

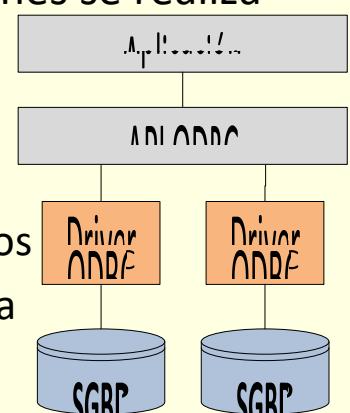
### Acceso a las bases de datos con JDBC

© Antonio Santos Ramos 2018

23

## DBMS (SGBD)

- La base de datos debe ser independiente de las aplicaciones que accedan a ella.
- La independencia se logra a través del módulo DBMS (Data Base Management System).
- La comunicación entre el DBMS y las aplicaciones se realiza utilizando el lenguaje SQL, que permite:
  - Definir la base de datos mediante tablas
  - Almacenar información en las tablas
  - Seleccionar información en base a unos criterios
  - Realizar cambios en la información y estructura
  - Combinar y calcular datos



# ¿Qué es el JDBC? (I/II)

- JDBC: acrónimo de Java DataBase Connectivity
  - API que incorpora un conjunto de clases, interfaces y métodos para que cualquier programa Java pueda acceder a sistemas de bases de datos.
- Ofrece un estándar de conexión a cualquier base de datos disponible en el mercado.
- Empleando un lenguaje SQL, permite obtener los datos de forma ágil en entornos cliente-servidor que se encuentren en Internet/Intranet.
- La versión incluida en Java SE 8 es JDBC 4.2.

<http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

<http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>

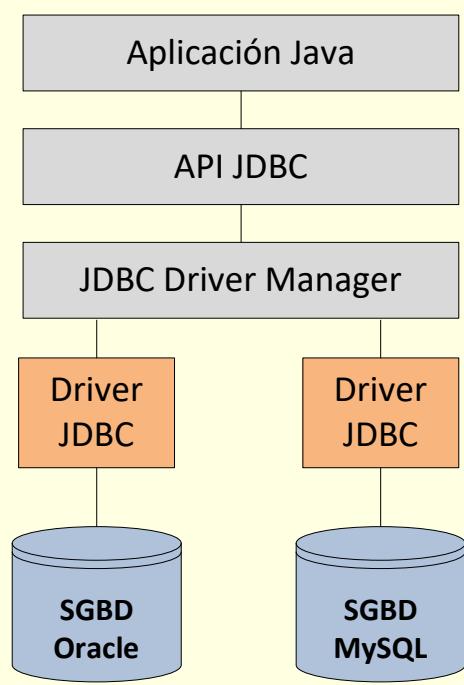
© Antonio Santos Ramos 2018

25

# ¿Qué es el JDBC? (II/II)

- El API JDBC se encuentra en los paquetes **java.sql** y **javax.sql**.
 

```
import java.sql.*;
import javax.sql.*;
```
- Se puede utilizar para:
  - Establecer conexiones con bases de datos
  - Enviar sentencias SQL a dichas BDs
  - Procesar los resultados de estas sentencias.
- JDBC permite ejecutar instrucciones SQL (Structured Query Language)

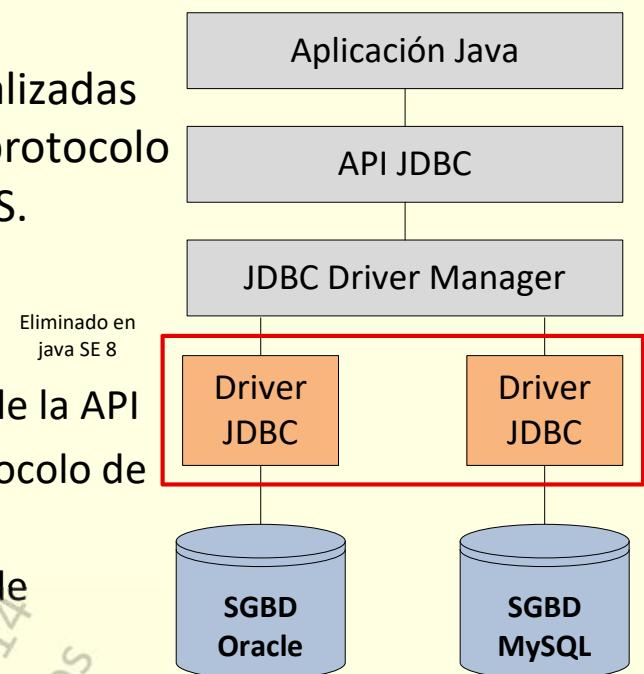


© Antonio Santos Ramos 2018

26

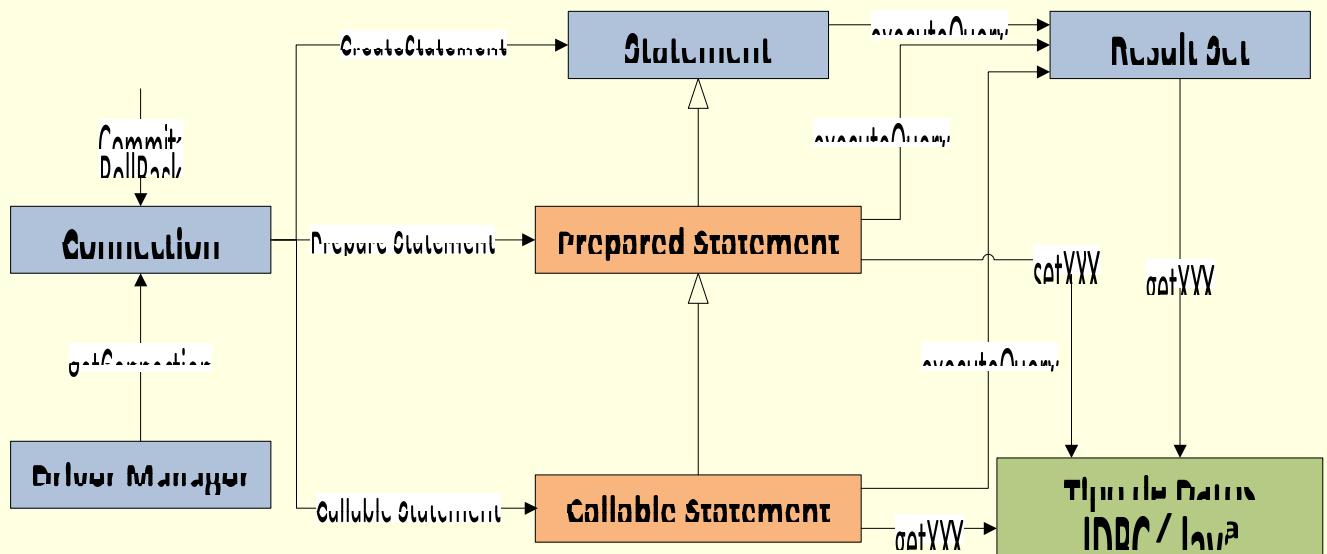
# Controladores JDBC

- Se instalan en el cliente.
- Convierten las peticiones realizadas desde programas Java a un protocolo que pueda entender el DBMS.
- Pueden ser de cuatro tipos:
  - Tipo 1: Puente JDBC-ODBC
  - Tipo 2: Controlador nativo de la API
  - Tipo 3: Controlador de protocolo de red
  - ✓ Tipo 4: Controlador nativo de protocolo



## Clases e interfaces

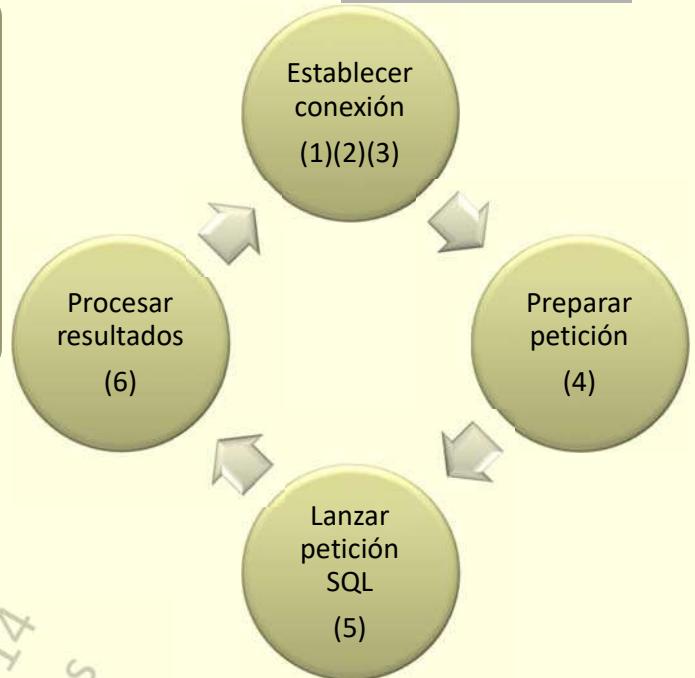
Java.sql



- Solo DriverManager es una clase. Las demás son interfaces

# Pasos para utilizar JDBC

1. Crear una instancia del JDBC driver .
2. Especificar la url de la base de datos.
3. Establecer una conexión usando el driver que crea el objeto Connection.
4. Crear un objeto Statement, usando Connection.
5. Lanzar la petición SQL
6. Recibir los resultados y procesarlos



© Antonio Santos Ramos 2018

29

## Paso 1: crear instancia

- Comprobamos la existencia del driver JDBC.
- Si se produce una excepción (ClassNotFoundException) debe comprobarse que el fichero físico existe y se encuentra en el classpath.

```

Connection con = null
Statement st = null;
ResultSet rs = null;
try
{
    String driverClassName = "com.mysql.jdbc.Driver";
    String driverUrl = "jdbc:mysql://localhost/empresa";
    String user = "antonio";
    String password = "asantosr";
    Class.forName(driverClassName);
    con = DriverManager.getConnection(driverUrl, user, password);
    ....
}
  
```

Sólo se registra 1 vez

<https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-api-changes.html>

Entre esos cambios está que ahora es com.mysql.cj.jdbc.Driver

Con JDBC 4.0 (Java SE 6) ya no es necesario usar Class.forName... pero a veces...

© Antonio Santos Ramos 2018

30

# Paso 2: indicar la url de la BB.DD.

- El aspecto general de la conexión para mysql es:  
**jdbc:mysql://<host>/<database>**

```

Connection con = null;
Statement st = null;
ResultSet rs = null;
try
{
    String driverClassName = "com.mysql.jdbc.Driver";
    String driverUrl = "jdbc:mysql://localhost/empresa";
    String user = "antonio";
    String password = "asantosr";
    Class.forName(driverClassName);
    con = DriverManager.getConnection(driverUrl, user, password);
    ....
}

```

© Antonio Santos Ramos 2018

31

## Algunos driver JDBC (old)

- **MySQL**
  - Clase Driver : com.mysql.jdbc.Driver
  - URL de Conexión: jdbc:mysql://<host>/<database>
- **DB2**
  - Clase Driver : com.ibm.db2.jdbc.app.DB2Driver
  - URL de Conexión:: jdbc:db2:<database>
- **Sybase**
  - Clase Driver : com.sybase.jdbc2.jdbc.SybDriver
  - URL de Conexión: jdbc:sybase:Tds:<host>
- **Oracle**
  - Clase Driver : oracle.jdbc.driver.OracleDriver
  - URL de Conexión: jdbc:oracle:thin:@ <host>:<port>:<sid>
- **SQLServer**
  - Clase Driver : com.microsoft.jdbc.sqlserver.SQLServerDriver
  - URL de Conexión: jdbc:microsoft:sqlserver://localhost:1433
- **PostgreSQL**
  - Clase Driver: org.postgresql.Driver
  - URL de Conexión: jdbc:postgresql://<server>:<port>/<database>

## Paso 3: establecer conexión

- Si se establece la conexión sin problema, devuelve un objeto de tipo **Connection**.
- Si hay problemas, se genera una SQLException.

```
Connection con = null
Statement st = null;
ResultSet rs = null;
try
{
    String driverClassName = "com.mysql.jdbc.Driver";
    String driverUrl = "jdbc:mysql://localhost/empresa";
    String user = "antonio";
    String password = "asantosr";
    Class.forName(driverClassName);
    con = DriverManager.getConnection(driverUrl, user, password);
    ....
```

## Paso 4: crear un objeto Statement (I/II)

- Objeto **java.sql.Statement**
  - Representa una instrucción de procesamiento SQL.
- Método **createStatement**
  - Devuelve un objeto de tipo Statement.
    - El objeto clase Statement permite enviar peticiones SQL a la Base de Datos.
  - La sentencia que se emplee a posteriori con el objeto Statement, puede ser cualquier petición válida en el lenguaje SQL del SGBD (Insert, Delete, Create Table, etc..)
  - Si hay problemas, se genera una SQLException.

# Paso 4: crear un objeto Statement (II/II)

```

import java.sql.*;
import paquete.driver.*;

...
Connection con = null
Statement st = null;
ResultSet rs = null;
try
{
    ...
    con = DriverManager.getConnection(driverUrl,user, password);
    ...
    st = con.createStatement();
    String query = "SELECT * FROM Empleados;";
    rs = st.executeQuery(query);
    ...
}

```

# Paso 5: lanzar la petición SQL (I/IV)

- Se realiza la sentencia SQL. Hay tres opciones
  - (I) Consultas SQL (**executeQuery**)
    - Devuelve un objeto de tipo ResultSet que contiene los resultados de la query SQL.

```
ResultSet rs = st.executeQuery("Select SQL");
```
  - (II) Actualizaciones SQL (**executeUpdate**)
    - Devuelve un entero con el nº de registros de la tabla afectados por la sentencia SQL (UPDATE, INSERT, DELETE, CREATE u otra sentencia SQL).

```
int num = st.executeUpdate("sentencia SQL");
```
  - Devuelve 0 si es consulta y 1 al insertar.
  - También se podría emplear executeInsert, executeDelete, ...

## Paso 5: lanzar la petición SQL (II/IV)

- (III) El método **execute (String sentSQL)** puede ejecutar cualquier tipo de sentencia
  - Si es una SELECT devolverá true
    - El ResultSet se deberá conseguir con `getResultSet ()`
  - Si es un UPDATE, DELETE, ... devolverá false
    - Mediante `getUpdateCount ()` se consigue el número de filas modificadas
  - Ambos métodos (`getResultSet/getUpdateCount`) apuntan al primer resultado.
    - Para obtener los resultados sg. se utiliza `getMoreResults ()`.
      - Devuelve `true` si el sg. resultado es de tipo ResultSet y `false` si es una actualización o no hay más resultados.

© Antonio Santos Ramos 2018

37

## Paso 5: lanzar la petición SQL (III/IV)

```
import java.sql.*;
import paquete.driver.*;
...
Connection con = null
Statement st = null;
ResultSet rs = null;
try
{
    ...
    con = DriverManager.getConnection(driverUrl,user, password);
    ...
    st = con.createStatement();
    String query = "SELECT * FROM Empleados";
    rs = st.executeQuery(query);
```

Si hay problemas,  
se genera una  
SQLException.

© Antonio Santos Ramos 2018

38

# Paso 5: lanzar la petición SQL (IV/IV)

## ■ Creación de tabla

- ```
st.executeUpdate("create table alumno (nombre varchar(32)
NOT NULL, matricula integer, direccion varchar(40),
annoNac integer, PRIMARY KEY (matricula));")
```

## ■ Insertar registro

- ```
st.executeUpdate( "insert into Alumnos " +
"values ('910339-0', 'Antonio Herranz',
'Informatica',...)");
```

## ■ Actualizar registro

- ```
st.executeUpdate("update alumnos set direccion = '?????'
where nombre = 'Antonio Herranz' ");
```

# Paso 6: procesar los resultados (I/III)

## ■ El conjunto de registros, resultado de la query, se almacena en el objeto **ResultSet**.

- El objeto **ResultSet** inicialmente apunta “a antes” de la primera fila.
- El método **next ()** de **ResultSet** mueve a la siguiente fila.
- Dentro de una fila los métodos **getXXX ()** de **ResultSet** permiten acceso a las columnas de la fila apuntada.

```
while(rs.next()) {
    System.out.println("Col1: " + rs.getString(1));
    System.out.println("Col2: " + rs.getString(2));
}
```

- Recorro fila a fila (en modo lectura)

Recuerda que las columnas  
son los atributos

# Paso 6: procesar los resultados (II/III)

```

import java.sql.*;
import paquete.driver.*;
...
Statement st = null;
ResultSet rs = null;
try {
    ...
    String query = "SELECT * FROM Empleados;";
    rs = st.executeQuery(query);
    while (rs.next()) {
        //se puede obtener una columna por posición
        String nombre = rs.getString(1);
        //se puede obtener una columna por nombre
        Date fecha = rs.getDate("fechaIngreso");
        System.out.println("Nombre: " + nombre + " | Fecha: " + fecha);
    }
}

```

| nombre  | fechaIngreso |
|---------|--------------|
| Antonio | 13/11/1975   |
| Herranz | 1/9/1990     |

Con getString y getObject se puede recuperar sin problemas cualquier “cosa”

Si no sé el tipo, puedo usar getType() que devuelve int

# Paso 6: procesar los resultados (III/III)

- Los métodos getXXX para procesar los resultados deben ajustarse al tipo de datos devueltos
- El argumento de los métodos getXXX puede ser:
  - ✓ El número de la columna de la tabla empezando con el registro 1.
  - El nombre de la columna

| Método          | Tipo devuelto        |
|-----------------|----------------------|
| getASCIIStream  | Java.io.InputStream  |
| getBigDecimal   | Java.math.BigDecimal |
| getBinaryStream | Java.io.InputStream  |
| getBoolean      | boolean              |
| getByte         | byte                 |
| getBytes        | byte[ ]              |
| getDate         | Java.sql.Date        |
| getDouble       | double               |
| getArray        | Java.sql.Array       |
| getFloat        | float                |
| getInt          | int                  |
| getLong         | long                 |
| getObject       | Object               |
| getShort        | short                |
| getString       | java.lang.String     |
| getTime         | java.sql.Time        |
| getTimestamp    | Java.sql.Timestamp   |
| getBlob         | java.sql.Blob        |

# Otras operaciones (I/IV)

## Errores y excepciones (I/II)

- Se pueden producir errores por:
  - Fallos en la conexión
  - No existencia de la base de datos
  - No disponer de permisos sobre la BBDD
  - Errores de sintaxis en la sentencia SQL
  - Operaciones no permitidas
- Si hay fallos, se producirá una excepción en el programa.
  - Normalmente será SQLException.
  - Puede ser ClassNotFoundException si no se encuentra el driver.

# Otras operaciones (II/IV)

## Errores y excepciones (II/II)

```
[...]
try{
    ...
} catch (ClassNotFoundException e) {
    System.out.println("No se encuentra el driver");
} catch (SQLException e) {
    System.out.println("Excepcion SQL: " + e.getMessage());
    System.out.println("Estado SQL: " + e.getSQLState());
    System.out.println("Código del Error: " + e.getErrorCode());
}
http://www.tutorialspoint.com/jdbc/jdbc-exceptions.htm
https://docs.oracle.com/javase/tutorial/jdbc/basics/sqlexception.html
```

- Se puede usar getWarnings() para acceder a los avisos (al no ser un error, no para la ejecución y no necesita catch)

# Otras operaciones (III/IV)

## Liberar recursos (I/II)

- Es imprescindible cerrar las conexiones tan pronto como se pueda mediante Connection.close().
  - Al cerrar una conexión, se cierran todos sus Statements asociados.
  - Al cerrar un Statement, se cierran todos sus ResultSets asociados.
  - Aunque no se llame a Connection.close, cuando el Garbage Collector elimine la conexión, se invocará al método close().
- Es mejor cerrar por “voluntad propia” los Statement y los ResultSet
  - Ej.- imagina una aplicación multi-thread que solicita muchas conexiones por minuto (ej.: una aplicación Internet)
  - Puede existir bugs en algunos drivers, de manera que no cierren los Statements asociados a una conexión.

# Otras operaciones (IV/IV)

## Liberar recursos (II/II)

```
...
try {
    ...
} catch ...
...
finally {
    try {
        if (rs != null) rs.close();
        if (st != null) st.close();
        if (con != null) con.close();
    } catch (SQLException e) {e.printStackTrace();}
}
```

En JDBC 4.1 (Java 7) se puede cerrar de forma automática un statement al cerrar el try

Las fases descritas anteriormente

# Optimizaciones (I/XII)

## MetaData (I/III)

### ■ Problema:

- ¿Qué ocurre si no sé cuantas columnas se devuelven?
- ¿Qué ocurre si necesito saber datos de la BBDD?

### ■ Solución: los metadatos

- **DatabaseMetaData**: info sobre la BBDD, nombre, tablas existentes, columnas de una tabla, etc.
  - getColumns, getPrimaryKeys, getTables, etc.
- **ResultSetMetaData**: info sobre la consulta, tipos de cada columna, propiedades, nombres, etc.
  - getColumnCount(), getColumnTypeName(n), getColumnType(n), etc.

<http://www.chuidiang.com/java/mysql/ResultSet-DataBase-MetaData.php>

[http://docstore.mik.ua/orely/java-ent/jenut/ch02\\_09.htm](http://docstore.mik.ua/orely/java-ent/jenut/ch02_09.htm)

© Antonio Santos Ramos 2018

47

# Optimizaciones (II/XII)

## MetaData (II/III)

```
DatabaseMetaData dbmd = con.getMetaData();
System.out.println("Driver Name: "+dbmd.getDriverName());
System.out.println("Driver Version: "+dbmd.getDriverVersion());
System.out.println("UserName: "+dbmd.getUserName());
System.out.println("Database Name: "+dbmd.getDatabaseProductName());
System.out.println("Database Version: "+dbmd.getDatabaseProductVersion());

//(catalogo, esquema, patrón para elegir tablas, tipos de tablas (vistas, system..)
ResultSet allTables = dbmd.getTables(null,null,null,null);
while(allTables.next()) {
    String table_name = allTables.getString("TABLE_NAME");
    System.out.println("Table Name: " + table_name);
    System.out.println("Table Type: " + allTables.getString("TABLE_TYPE"));
    System.out.println("Indexes: ");
    ResultSet indexList = dbmd.getIndexInfo(null,null,table_name,false,false);
    while(indexList.next()) {
        System.out.println(" Index Name: "+indexList.getString("INDEX_NAME"));
        System.out.println(" Column Name: "+indexList.getString("COLUMN_NAME"));
    }
    indexList.close();
}
```

© Antonio Santos Ramos 2018

48

# Optimizaciones (III/XII)

## MetaData (III/III)

```
[...]
ResultSet rs = stmt.executeQuery("SELECT * FROM "+ table);
ResultSetMetaData rsmd = rs.getMetaData();
int columnCount = rsmd.getColumnCount();
for(int col = 1; col <= columnCount; col++) {
    System.out.print(rsmd.getColumnLabel(col));
    System.out.print(" (" + rsmd.getColumnTypeName(col)+")");
    if(col < columnCount) { System.out.print(", ");}
}
System.out.println();

while(rs.next()) {
    for(int col = 1; col <= columnCount; col++) {
        System.out.print(rs.getString(col));
        if(col < columnCount) { System.out.print(", ");}
    }
    System.out.println();      idPersona (SHORT), nombre (VARCHAR), dirección (VARCHAR)
}                                1, Antonio Herranz, 27 Stevens St
                                2, Virginia Wolf, 45 Morrison Lane
                                3, Olga Wong, 13 Times Square
```

© Antonio Santos Ramos 2018

49

# Optimizaciones (IV/XII)

## Número de filas retornadas

- Por defecto ResultSets se almacenan de forma completa en memoria. En general es la opción más inteligente
- ¿Problema?
  - ¿Y si se devuelven muchas filas de golpe? ¿Y si necesito controlar los bloques de filas que van llegando?
- ¿Solución?
  - Método **setFetchSize(int filas)**

rs.next() irá pidiendo de la caché.  
Cuando e acaben las filas, se piden más

```
Connection connection = DriverManager.getConnection("");
Statement statement = connection.createStatement();
statement.setFetchSize(1000); // configure the fetch size
ResultSet resultSet = statement.executeQuery("");
```

<http://examples.javacodegeeks.com/core-java/sql/set-prefetch-size-of-sql-query-example/>  
<http://webmoli.com/2009/02/01/jdbc-performance-tuning-with-optimal-fetch-size/>

# Optimizaciones (V/XII)

## Procesos Batch

- Se pueden realizar procesos por lotes
  - `addBatch (String sql)`: Añade una nueva operación dentro del proceso de órdenes.
  - `clearBatch ()`: Borra las operaciones a realizar dentro del proceso por lotes.
  - `executeBatch ()`: Ejecuta el proceso de órdenes creado.

```
st.addBatch("INSERT INTO Valores VALUES ('valor1')");
st.addBatch("INSERT INTO Valores VALUES ('valor2')");
st.addBatch("INSERT INTO Valores VALUES ('valor3')");
st.addBatch("INSERT INTO Valores VALUES ('valor4')");
int [] actualizadas=st.executeBatch();
```

[http://www.tutorialspoint.com/jdbc/jdbc\\_batch\\_processing.htm](http://www.tutorialspoint.com/jdbc/jdbc_batch_processing.htm)

© Antonio Santos Ramos 2018

51

# Optimizaciones (VI/XII)

## Peticiones precompiladas (I/III)

- Peticiones precompiladas ≡ Prepared Statements
- Cada vez que se envía una query a la BBDD, ésta:
  - La analiza sintácticamente y ...
    - ... construye un plan para ejecutarla
  - Conclusión: Statement es ineficiente si lanzamos varias queries iguales en las cuales sólo cambian los parámetros
- ¿Solución? Emplear **PreparedStatement**
  - PreparedStatement es una subinterface de Statement
  - Permite cambiar valores dentro de la petición (la precompila).
  - Puede usarse el carácter ? como “hueco” para el parámetro.
  - El valor del parámetro se especifica con los métodos setXXX.

© Antonio Santos Ramos 2018

52

# Optimizaciones (VII/XII)

## Peticiones precompiladas (II/III)

```

// establecer la conexión
...
String [] nombreEmp={...};
Date [] fechaIngEmp={...}; // de la misma long. que el anterior
PreparedStatement preparedStatement = null;

String query = "INSERT INTO Empleados (nombreEmp,fechaIng) VALUES (?,?);";
PreparedStatement preparedStatement = con.prepareStatement(query);

//insertamos los empleados en la BD
for (int i=0; i<nombreEmp.length; i++) {

    // rellenamos el parámetro 1
    preparedStatement.setString(1, nombreEmp[i]);

    // rellenamos el parámetro 2
    preparedStatement.setDate(2, fechaIngEmp[i]);

    // ejecutar la consulta
    int filas = preparedStatement.executeUpdate();
    if (filas != 1) { throw new SQLException("Problemas insertando "+ nombreEmp[i]);}
}

http://www.tutorialspoint.com/jdbc/preparstatement-object-example.htm

```

© Antonio Santos Ramos 2018

53

# Optimizaciones (VIII/XII)

## Peticiones precompiladas (III/III)

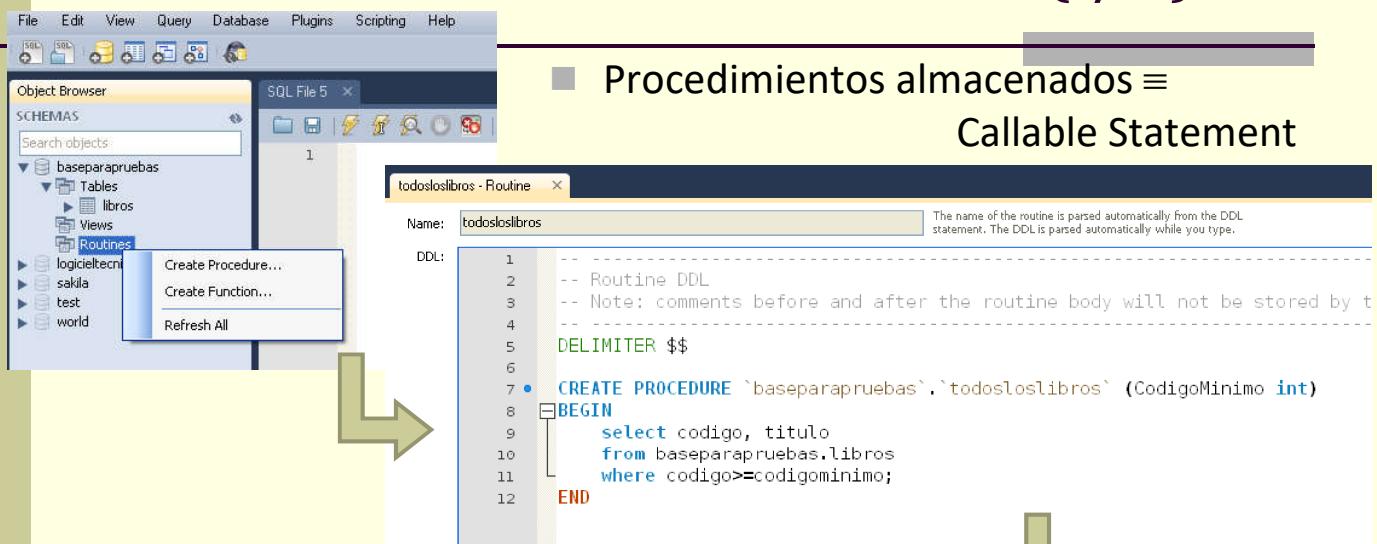
### ■ Métodos setXXX para “Prepared Statements”

| Método          | Tipo SQL      |
|-----------------|---------------|
| setASCIIStream  | LONGVARCHAR   |
| setBigDecimal   | NUMERIC       |
| setBinaryStream | LONGVARBINARY |
| setBoolean      | BIT           |
| setByte         | TINYINT       |
| setBytes        | VARBINARY     |
| setDouble       | DOUBLE        |
| setArray        | Array         |
| setFloat        | FLOAT         |
| setInt          | INTEGER       |
| setLong         | LONG          |
| setShort        | SHORT         |
| setString       | VARCHAR       |

| Método       | Tipo SQL         |
|--------------|------------------|
| setObject    | --se convierte-- |
| setDate      | DATE             |
| setTime      | TIME             |
| setTimestamp | TIMESTAMP        |
| setBlob      | BLOB             |

# Optimizaciones (IX/XII)

## Procedimientos almacenados (I/II)



### ■ Procedimientos almacenados ≡ Callable Statement

- Representa procedimientos almacenados en el servidor de BB.DD.

<http://campus.almagro.ort.edu.ar/informatica/prog/articulo/393795/procedimientos-almacenados-y-funciones-almacenadas>

© Antonio Santos Ramos 2018

55

# Optimizaciones (X/XII)

## Procedimientos almacenados (II/II)

- Permite optimizar procesos y tiempos de respuesta porque las operaciones se realizan más cerca de los datos

```
String sql = "{call return_num_seats(?, ?, ?, ?)}";
CallableStatement st = conn.prepareCall(sql);
```

return\_num\_seats  
es el proc.  
almacenado

```
String planeID = "1990";
st.setString(1, planeID); //cod del avión
st.registerOutParameter(2,java.sql.Types.INTEGER); //First Class
st.registerOutParameter(3,java.sql.Types.INTEGER); //Business Class
st.registerOutParameter(4,java.sql.Types.INTEGER); //Economic Class

st.execute();
```

```
int FCSeats = st.getInt(2); //First Class
int BCSeats = st.getInt(3); //Business Class
int ECSeats = st.getInt(4); //Economic Class
```

Los parámetros  
de salida hay  
que registrarlos

© Antonio Santos Ramos 2018

56

<http://www.tutorialspoint.com/jdbc/callablestatement-object-example.htm>

[http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=usando\\_CallableStatements\\_to\\_execute\\_stored\\_procedure](http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=usando_CallableStatements_to_execute_stored_procedure)

# Optimizaciones (XI/XII)

## Transacciones (I/II)

### ■ Transacción:

- Conjunto de ordenes SQL (y por tanto operaciones), agrupadas por la lógica del dominio, vistas como una única operación.
- La transacción se realiza o NO se realiza.
- Permite cumplir las restricciones de integridad de la BBDD.
  - Ejemplo: Transferencia
    - Dos operaciones distintas.
      - Se decrementa el saldo de la cuenta origen
      - Se incrementa el saldo de la cuenta destino.
    - O se realizan las 2 operaciones, o no se realiza ninguna
- Lo lógico es emplear excepciones.
  - Si ocurre un problema se lanza la excepción y se ejecuta **rollback()**
  - Si no hay problemas se ejecuta la transacción (**commit**)

© Antonio Santos Ramos 2018

57

# Optimizaciones (XII/XII)

## Transacciones (II/II)

```
Connection conn = null;
Producto p1 = new Producto("leche");
Producto p2 = new Producto("carne");
Producto p3 = new Producto("jamon");
try {
    conn = ConexionesHandler.getConnection();
    //Indico que hay transacción de varias acciones
    conn.setAutoCommit(false);
    Insertar(p1, conn);
    Insertar(p2, conn);
    Insertar(p1, conn);
    conn.commit();

} catch (Exception e) {
    try {
        conn.rollback();
    } catch (Exception e) {
        ...
    } finally {
        ...
    }
}
```

<http://www.tutorialspoint.com/jdbc/jdbc-transactions.htm>  
<http://www.tutorialspoint.com/jdbc/commit-rollback.htm>  
<http://www.mkyong.com/jdbc/jdbc-transaction-example/>  
<http://blog.rolandopalermo.com/2012/10/transactions-jdbc.html>

© Antonio Santos Ramos 2018

58

# Pool de conexiones (I/II)

- Cada vez que un programa cliente necesita comunicarse con una BBDD, establece una conexión y la mantiene abierta el tiempo que dura la ejecución del programa.
  - Abrir una conexión puede ser costoso de tiempo
  - El servidor puede limitar el nº de conexiones abiertas de forma simultanea
  - Este esquema, no es apropiado para aplicaciones multitarea.
    - Ej.- servicio de páginas web a varios usuarios
- Un JDBC **Connection pool** es un grupo de conexiones que el servidor mantiene para una BBDD concreta.
  - La agrupación de conexiones reduce el tiempo de transacción

# Pool de conexiones (II/II)

- **Funcionamiento** de un pool de conexiones
  - Se crea un grupo de conexiones abiertas
  - Cuando un cliente necesita una, la solicita, la usa y la devuelve
  - Cuando una aplicación cierra una conexión, regresa al pool.
  - La conexión se crea una única vez y se puede utilizar muchas veces
- Para utilizarla, emplea la interface javax.sql.DataSource
  - [http://cursohibernate.es/doku.php?id=patrones:pool\\_conexiones](http://cursohibernate.es/doku.php?id=patrones:pool_conexiones)
  - <http://christmo99.wordpress.com/2008/04/09/pool-con/>
  - <http://wiki.netbeans.org/PoolConexionesGlassfishNetBeans>
  - <http://gcoronelc.blogspot.com.es/2011/08/pool-de-conexiones-con-mysql-glassfish.html>

# Patrón DAO

## ■ Patrón DAO (Data Access Object)

- Crea un protocolo para controlar la persistencia de una información y ocultar la implementación de esta persistencia.
- Abstacta y encapsula los accesos, gestiona la conexiones a la fuente de datos y obtiene los datos almacenados.
- El cliente se centra en los datos que necesita y se olvida de cómo se trabaja con la fuente de almacenamiento.

[http://chuwiki.chuidiang.org/index.php?title=Patrón\\_DAO](http://chuwiki.chuidiang.org/index.php?title=Patrón_DAO)

<http://ict.udlap.mx/people/carlos/is346/admon05.html>

[http://www.tutorialspoint.com/design\\_pattern/data\\_access\\_object\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm)

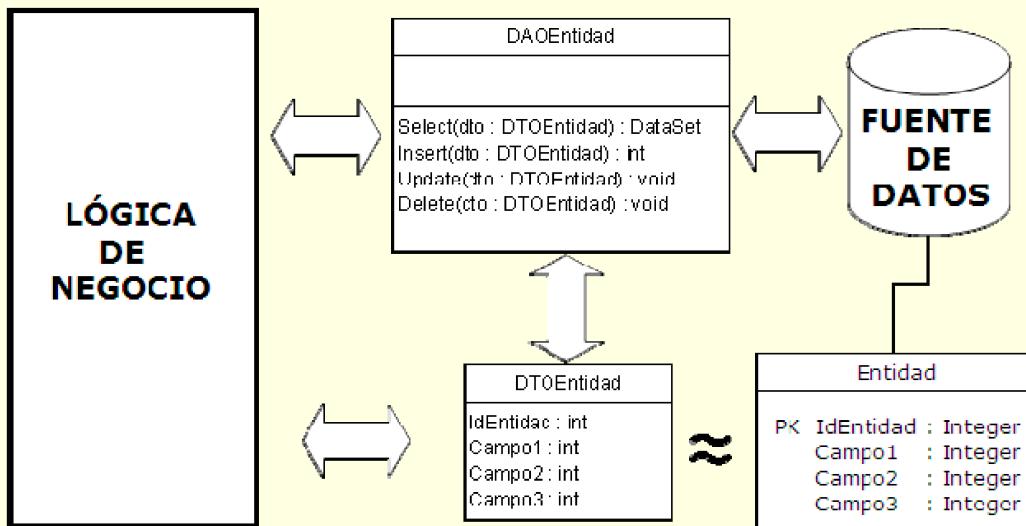
## ■ Suele realizarse una interface y utilizarlo con patrón factoría

<http://balusc.omnifaces.org/2008/07/dao-tutorial-data-layer.html>

© Antonio Santos Ramos 2018

61

# DAO. Esquema



Patrón DAO (Data Access Object)

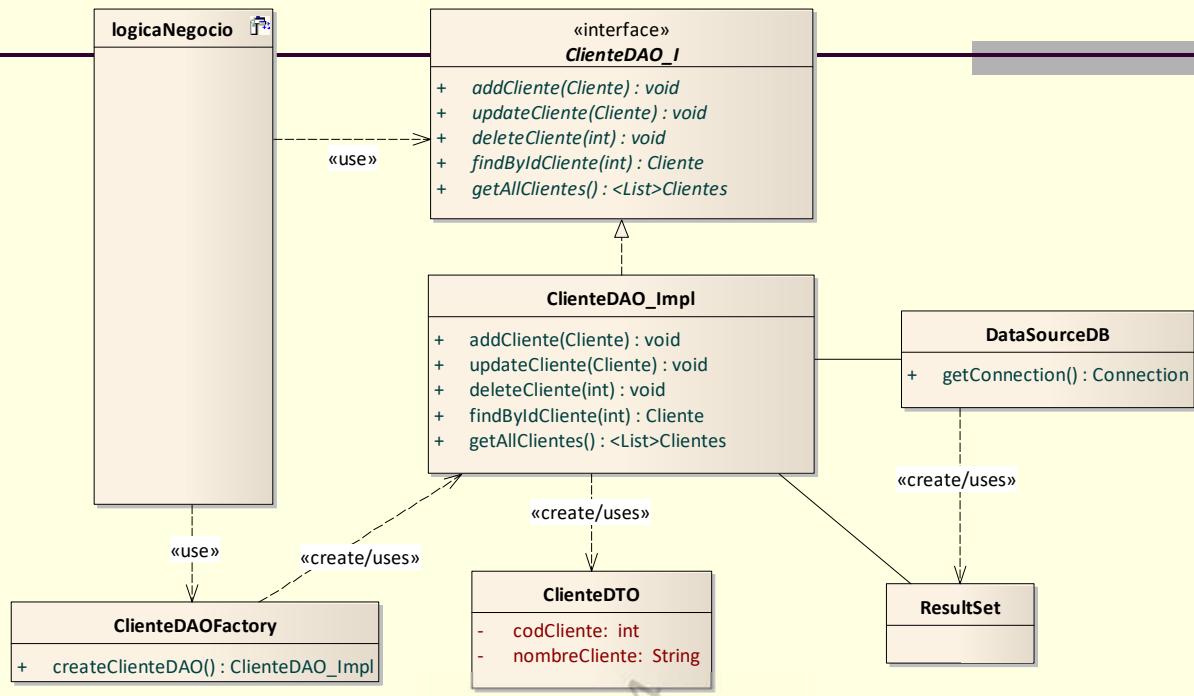
Patrón DTO (Data Transfer Object)

<http://www.informit.com/articles/article.aspx?p=1398621&seqNum=3>

<http://gcoding.blogspot.com.es/2010/03/tres-amigos-de-persistencia-dao.html>

# DAO. Esquema mejorado

T02 – ACCESO A LAS BASES DE DATOS CON JDBC



En el método devuelve un new ClienteDAO\_Impl  
ocultando así su implementación

ClienteDAOfactory f = new ClienteDAOfactory();  
ClienteDAO\_I dao = f.createClienteDAO();

© Antonio Santos Ramos 2018

63

# Curso de Programación para Internet con Java EE (Parte III - Database)

T03

SQL y MySQL

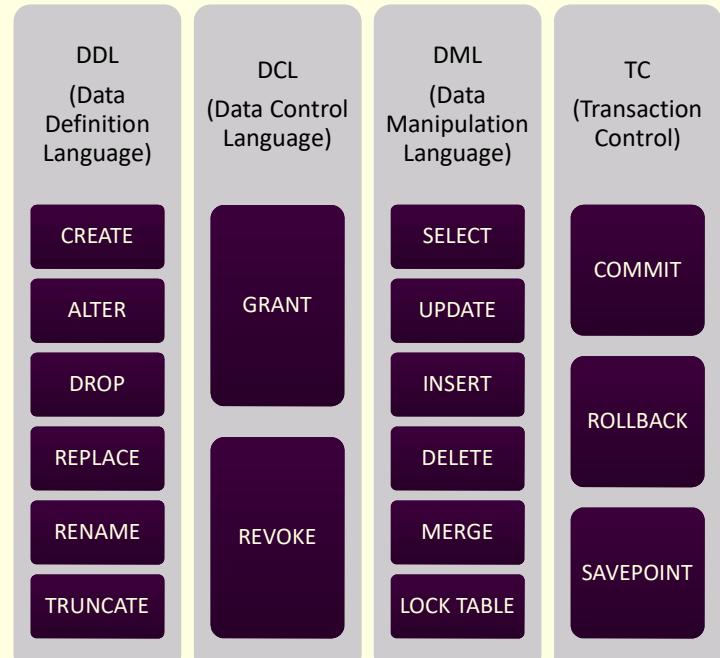
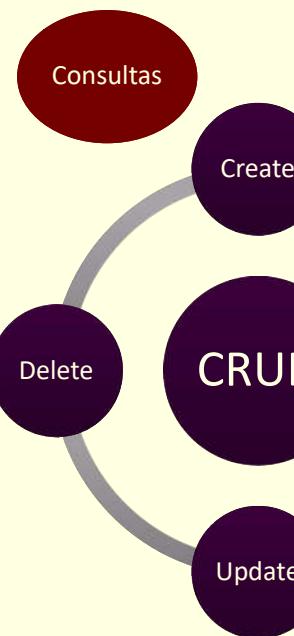
© Antonio Santos Ramos 2018

64

# SQL. Introducción

- SQL: Structured Query Language
- Es el lenguaje más utilizado para BB.DD. relacionales
- Lenguaje declarativo de alto nivel
- Desarrollado por IBM (1974-1977)
  - 1979: Adquiere el lenguaje la compañía Relational Software INC. (Posteriormente Oracle) y lo incorpora a su BBDD.
- Se convirtió en un standard definido por :
  - ANSI (American National Standards Institute) e
  - ISO (International Standards Organization)
- El estándar actual es SQL:2016. La base es SQL-92.

## Operaciones y comandos



# MySQL. Normas

## ■ Normas para escribir

- Da lo mismo escribir en mayúsculas o en minúsculas
  - Las palabras claves y los comandos suelen escribirse en mayúsculas
  - El resto de las palabras (nombres de tabla y columnas) suelen escribirse en minúsculas
- Las sentencias SQL pueden constar de una o varias líneas.
  - Una cláusula es parte de una sentencia:
    - Ej.- SELECT empno, ename;
  - Normalmente las cláusulas se escriben en diferentes líneas por legibilidad y facilidad de edición.
  - Colocar un punto y coma (;) al final de la última cláusula.

# MySQL. Acceso y estructura (I/III)

## ■ Mostrar las bases de datos

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| mysql    |
| test     |
+-----+
2 rows in set (0.00 sec)
```

## ■ Crear bases de datos

```
mysql> CREATE DATABASE empresa;
Query OK, 1 row affected (0.00 sec)
```

## ■ Seleccionar la base de datos sobre la que se va a trabajar.

```
mysql> USE empresa;
Database changed
```

| EMPNO | ENAME  | JOB       | MGR  | HIREDATE  | SAL  | COMM | DEPTNO |
|-------|--------|-----------|------|-----------|------|------|--------|
| 7839  | KING   | PRESIDENT |      | 17-NOV-81 | 5000 |      | 10     |
| 7698  | BLAKE  | MANAGER   | 7839 | 01-MAY-81 | 2850 |      | 30     |
| 7782  | CLARK  | MANAGER   | 7839 | 09-JUN-81 | 1500 |      | 10     |
| 7566  | JONES  | MANAGER   | 7839 | 02-APR-81 | 2975 |      | 20     |
| 7654  | MARTIN | SALESMAN  | 7698 | 28-SEP-81 | 1250 | 1400 | 30     |
| 7499  | ALLEN  | SALESMAN  | 7698 | 20-FEB-81 | 1600 | 300  | 30     |
| 7844  | TURNER | SALESMAN  | 7698 | 08-SEP-81 | 1500 | 0    | 30     |
| 7900  | JAMES  | CLERK     | 7698 | 03-DEC-81 | 950  |      | 30     |
| 7521  | WARD   | SALESMAN  | 7698 | 22-FEB-81 | 1250 | 500  | 30     |
| 7902  | FORD   | ANALYST   | 7566 | 03-DEC-81 | 3000 |      | 20     |
| 7369  | SMITH  | CLERK     | 7902 | 17-DEC-80 | 800  |      | 20     |
| 7788  | SCOTT  | ANALYST   | 7566 | 09-DEC-82 | 3000 |      | 20     |
| 7876  | ADAMS  | CLERK     | 7788 | 12-JAN-83 | 1100 |      | 20     |
| 7934  | MILLER | CLERK     | 7782 | 23-JAN-82 | 1300 |      | 10     |

EMP

| DEPTNO | DNAME      | LOC      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |

DEPT

| GRADE | LOSAL | HISAL |
|-------|-------|-------|
| 1     | 700   | 1200  |
| 2     | 1201  | 1400  |
| 3     | 1401  | 2000  |
| 4     | 2001  | 3000  |
| 5     | 3001  | 9999  |

SALGRADE

© Antonio Santos Ramos 2018

69

# MySQL

## Acceso y estructura (III/III)

- Mostrar las tablas de una base de datos.

```
mysql> SHOW TABLES;
```

- Examinar la estructura de una tabla.

```
mysql> DESC contactos;
```

```
mysql> SHOW FIELDS FROM contactos;
```

| Field         | Type     | Null | Key | Default | Extra          |
|---------------|----------|------|-----|---------|----------------|
| employee_id   | int(11)  | NO   | PRI | NULL    | auto_increment |
| first_name    | char(20) | YES  |     | NULL    |                |
| last_name     | char(25) | NO   |     | NULL    |                |
| email         | char(25) | NO   | UNI | NULL    |                |
| salary        | int(11)  | NO   |     | NULL    |                |
| hire_date     | date     | NO   |     | NULL    |                |
| job_id        | char(10) | NO   |     | NULL    |                |
| department_id | int(11)  | YES  |     | NULL    |                |

# MySQL

## Sentencia base para consultas

```
SELECT [ALL/DISTINCT] select_list
FROM table [table alias] [,...]
[WHERE condition]
[GROUP BY column_list]
[HAVING condition]
[ORDER BY column_name [ASC/DESC] [,...]
[LIMIT NUM]
```

# comentario  
-- comentario

Para conocer como MySQL procesa las sentencias SQL mediante sus índices y uniones, se puede emplear el método EXPLAIN

<http://www.sergiquinonero.net/explain-mysql-o-como-optimiza-sql.html>

© Antonio Santos Ramos 2018

71

# MySQL

## La clausula SELECT

```
SELECT *
FROM dept;
```

```
SELECT DISTINCT deptno
FROM emp;
```

```
SELECT deptno, loc
FROM dept;
```

```
SELECT ename, sal, sal+300
FROM emp;
```

```
SELECT ename AS name, sal salary
FROM emp;
```

```
SELECT ename, deptno, sal
FROM emp
ORDER BY deptno, sal DESC;
```

# MySQL Operadores

## ■ Operadores de comparación básicos

| Operador | Significado         |
|----------|---------------------|
| =        | Igual a             |
| >        | Mayor que           |
| >=       | Mayor que o igual a |
| <        | Menor que           |
| <=       | Menor que o igual a |
| <>       | No igual a          |

## ■ Operadores de comparación

| Operador            | Significado                   |
|---------------------|-------------------------------|
| BETWEEN ... AND ... | Entre dos valores (inclusive) |
| IN (list)           | Lista de valores              |
| LIKE                | Se ajusta a un patrón         |
| IS NULL             | Es un valor nulo              |

## ■ Operadores lógicos

- AND
- OR
- NOT

# MySQL Uso de la Cláusula WHERE (I/III)

```
SELECT ename, job, deptno
FROM emp
WHERE job='CLERK' ;
```

```
SELECT ename, sal, comm
FROM emp
WHERE sal<=comm;
```

```
SELECT ename, sal, comm
FROM emp
WHERE sal<=comm LIMIT 2;
```

```
SELECT ename, sal
FROM emp
WHERE sal BETWEEN 1000 AND 1500 ;
```

```
SELECT empno, ename, sal, mgr
FROM emp
WHERE mgr IN (7902, 7566, 7788) ;
```

# MySQL

## Uso de la Clausula WHERE (II/III)

```
SELECT empno, ename, job, sal
FROM emp
WHERE sal>=1100
AND job='CLERK';
```

```
SELECT empno, ename, job, sal
FROM emp
WHERE sal>=1100
OR job='CLERK';
```

```
SELECT ename, job
FROM emp
WHERE job NOT IN ('CLERK', 'MANAGER', 'ANALYST');
```

```
SELECT ename, sal
FROM emp
WHERE sal NOT BETWEEN 1000 AND 3000;
```

# MySQL

## Uso de la Clausula WHERE (III/III)

```
SELECT ename
FROM emp
WHERE ename LIKE 'O1%';
```

Empieza por O1

```
SELECT ename
FROM emp
WHERE ename LIKE '_A%';
```

A como segundo carácter

```
SELECT ename, mgr
FROM emp
WHERE mgr IS NULL;
```

# MySQL

## Funciones con registros agrupados (I/III)

```
SELECT      COUNT (*)
FROM        emp
WHERE       deptno = 30;
```

```
SELECT      COUNT (comm)
FROM        emp
WHERE       deptno = 30;
```

```
SELECT      AVG (sal) , MAX (sal) ,
            MIN (sal) , SUM (sal)
FROM        emp
WHERE       job LIKE 'SALES%';
```

```
SELECT AVG (comm)
FROM   emp;
```

Las Funciones de Grupo ignoran los valores nulos de las columnas. No los contabilizan

```
SELECT AVG (COALESCE (comm, 0))
FROM   emp;
```

AVG(nvl(comm,0))

# MySQL

## Funciones con registros agrupados (II/III)

| DEPTNO | SAL  |
|--------|------|
| 10     | 2450 |
|        | 5000 |
|        | 1300 |
| 20     | 800  |
|        | 1100 |
|        | 3000 |
|        | 3000 |
|        | 2975 |
| 30     | 1600 |
|        | 2850 |
|        | 1250 |
|        | 950  |
|        | 1500 |
|        | 1250 |

2916.6667

2175

1566.6667

EMP

“media de salarios en EMP para cada departamento”

| DEPTNO | AVG (SAL) |
|--------|-----------|
| 10     | 2916.6667 |
| 20     | 2175      |
| 30     | 1566.6667 |

# MySQL

## Funciones con registros agrupados (III/III)

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```

```
SELECT deptno, job, sum(sal)
FROM emp
GROUP BY deptno, job;
```

```
SELECT max(avg(sal))
FROM emp
GROUP BY deptno;
```

```
SELECT deptno, max(sal)
FROM emp
GROUP BY deptno
HAVING max(sal)>2900;
```



Lo podemos resolver con subconsultas

HAVING filtra grupos así como el WHERE filtra registros.

## Enlaces con ejemplos

### Funciones para cadenas de texto y números

#### ■ Funciones para trabajar con fechas

- <http://www.geeksengine.com/database/single-row-functions/date-time-functions-1.php>
- <http://www.geeksengine.com/database/single-row-functions/date-time-functions-2.php>
- <http://www.geeksengine.com/database/single-row-functions/date-time-functions-3.php>

#### ■ Funciones para trabajar con String

- <http://www.geeksengine.com/database/single-row-functions/string-functions-1.php>
- <http://www.geeksengine.com/database/single-row-functions/string-functions-2.php>
- <http://www.geeksengine.com/database/single-row-functions/string-functions-3.php>

#### ■ Funciones para conversión

- <http://www.geeksengine.com/database/single-row-functions/conversion-functions.php>

#### ■ Funciones para control de flujo

- <http://www.geeksengine.com/database/single-row-functions/control-flow-functions.php>

# MySQL

## Join de tablas (I/IV)

| EMPNO | ENAME  | ... | DEPTNO |
|-------|--------|-----|--------|
| 7839  | KING   | ... | 10     |
| 7698  | BLAKE  | ... | 30     |
| ...   |        |     |        |
| 7934  | MILLER | ... | 10     |

EMP

| DEPTNO | DNAME      | LOC      |
|--------|------------|----------|
| 10     | ACCOUNTING | NEW YORK |
| 20     | RESEARCH   | DALLAS   |
| 30     | SALES      | CHICAGO  |
| 40     | OPERATIONS | BOSTON   |

DEPT

| EMPNO | DEPTNO | LOC      |
|-------|--------|----------|
| 7839  | 10     | NEW YORK |
| 7698  | 30     | CHICAGO  |
| 7782  | 10     | NEW YORK |
| 7566  | 20     | DALLAS   |
| 7654  | 30     | CHICAGO  |
| 7499  | 30     | CHICAGO  |
| ...   |        |          |

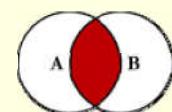
14 rows selected.

© Antonio Santos Ramos 2018

81

# MySQL

## Join de tablas (II/IV)



```
SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
FROM emp, dept
WHERE emp.deptno=dept.deptno;
```

```
SELECT e.empno, e.ename, e.deptno,
d.deptno, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno;
```

También valdría  
emp AS e, dep AS d

```
SELECT emp.empno, emp.ename, emp.deptno, dept.deptno, dept.loc
FROM emp
NATURAL JOIN dept;
```

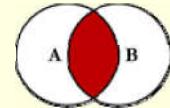
Si hay nulos no se muestran

© Antonio Santos Ramos 2018

82

# MySQL

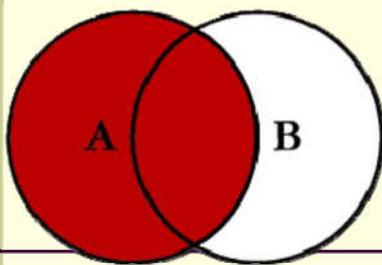
## Join de tablas (III/IV)



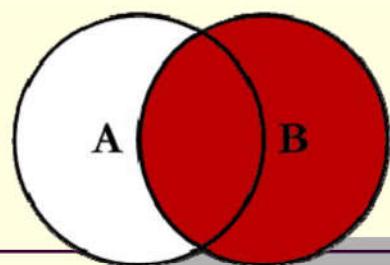
```
SELECT customer.name, item.itemid
FROM customer, ord, item
WHERE customer.custid = ord.custid AND
ord.ordid = item.ordid;
```

```
SELECT customer.name, item.itemid
FROM customer
      JOIN ord
        ON customer.custid = ord.custid)
      JOIN item i
        ON ord.ordid = item.ordid
WHERE sal>1000;
```

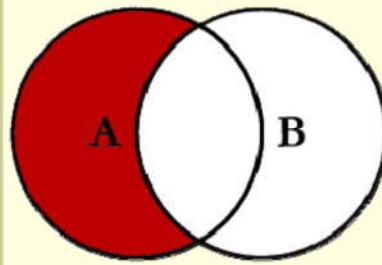
```
SELECT C.name, I.itemid
FROM customer C, ord O, item I
WHERE C.custid = O.custid AND
O.ordid = I.ordid;
```



## MySQL. Join de tablas (IV/IV)

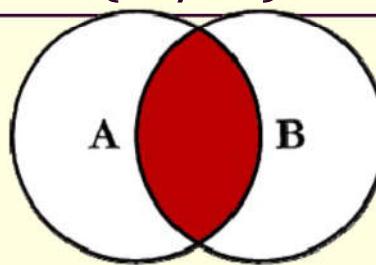


```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
```



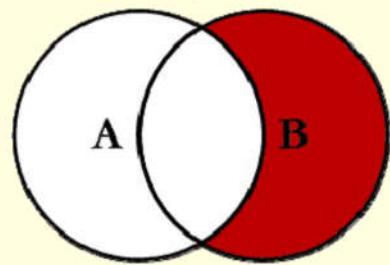
```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.Key = B.Key
```

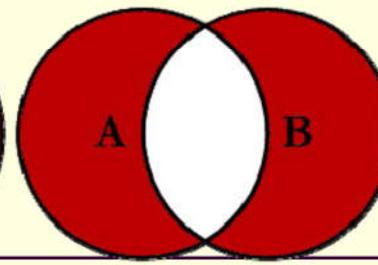
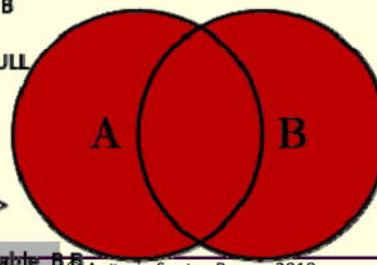


```
SELECT <select_list>
FROM Table_A A
INNER JOIN Table_B B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM Table_A A
FULL OUTER JOIN Table_B B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

# MySQL

## Modificaciones y alteraciones (I/II)

```
INSERT INTO dept (deptno, dname, loc)
VALUES      (50, 'RRHH', 'DETROIT');
1 row created.
```

```
INSERT INTO dept (deptno, dname)
VALUES      (60, 'MIS');
1 row created.
```

```
INSERT INTO dept
VALUES      (70, 'FINANCE', NULL);
1 row created.
```

```
DELETE FROM dept
WHERE        dname = 'RRHH';
1 row deleted.
```

```
UPDATE salgrade
SET LOSAL = 600.00
WHERE LOSAL = 700.00;
1 row affected.
```

# MySQL

## Modificaciones y alteraciones (II/II)



```
DELETE FROM dept;
4 rows deleted.
```

```
DROP TABLE dept30;
Table dropped.
```

```
RENAME TABLE dept TO dep;
4 rows deleted.
```

```
TRUNCATE TABLE dept;
Query OK, 0 rows affected.
```

Borrar todos los datos

```
ALTER TABLE dept30
ADD          job VARCHAR(9);
Table altered.
```

Se añade una nueva columna en último lugar

```
ALTER TABLE dept30
MODIFY       ename VARCHAR(15);
Table altered.
```

Estoy cambiando el tipo de la columna

# MySQL

## Subconsultas (I/IV)

- Si la subconsulta devuelve un resultado utiliza `=, >, <, <>, >=, <=`
- Si la subconsulta devuelve varios resultados utiliza `IN, ANY, ALL`

```
SELECT ename
FROM emp
WHERE sal >
      (SELECT sal
       FROM emp
       WHERE UPPER(ename) = "blake") ;
```

2850

```
SELECT ename, job, sal
FROM emp
WHERE sal =
      (SELECT MIN(sal)
       FROM emp) ;
```

800

# MySQL

## Subconsultas (II/IV)

```
SELECT ename, job
FROM emp
WHERE job =
      (SELECT job
       FROM emp
       WHERE empno = 7369) ;
```

CLERK

```
AND sal >
      (SELECT sal
       FROM emp
       WHERE empno = 7876) ;
```

1100

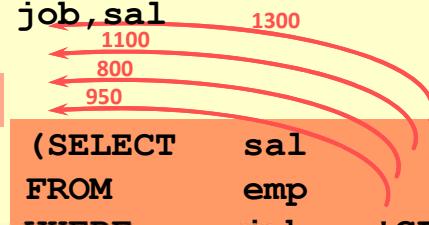
```
SELECT deptno, MIN(sal)
FROM emp
GROUP BY deptno
HAVING MIN(sal) >
      (SELECT MIN(sal)
       FROM emp
       WHERE deptno = 20) ;
```

800

# MySQL Subconsultas (III/IV)

```
SELECT ename, sal, job
FROM emp
WHERE sal IN (SELECT MIN(sal)
               FROM emp
               GROUP BY job);
```

```
SELECT empno, ename, job, sal
FROM emp
WHERE sal < ANY (SELECT sal
                  FROM emp
                  WHERE job = 'CLERK')
AND job <> 'CLERK';
```



# MySQL Subconsultas (IV/IV)

```
SELECT empno, ename, job, sal
FROM emp
WHERE sal > ALL
      4
      5
      6
      (SELECT avg(sal)
       FROM emp
       GROUP BY deptno);
```



```
SELECT a.ename, a.sal, a.deptno, b.salavg
FROM emp a, (SELECT deptno, avg(sal) salavg
              FROM emp
              GROUP BY deptno) b
WHERE a.deptno = b.deptno
AND a.sal > b.salavg;
```

# Integridad Referencial (I/IV)

- ¿Qué ocurre cuando borro una PK que es FK en otra tabla?

Una FK debe contener un valor, bien sea igual a un valor de la tabla que contiene este atributo como clave principal, o bien un valor nulo.

- Para indicar restricciones uso tablas innodb y empleo:
  - **constraint**: es opcional y permite indicar un nombre para la clave externa.
  - **foreign key**: indica los campos de esta tabla que están relacionados con campos de otra.
  - **References**: indica el nombre de la tabla relacionada y el nombre de los campos relacionados.

# Integridad Referencial (II/IV)

- Crear tablas

```
mysql> CREATE TABLE `employees` (
    `employee_id` int(11) NOT NULL AUTO_INCREMENT,
    `first_name` char(20) default NULL,
    `last_name` char(25) NOT NULL,
    `email` char(25) NOT NULL,
    `salary` int(11) default NULL,
    `hire_date` date NOT NULL,
    `job_id` char(10) NOT NULL,
    `department_id` int(11) default NULL,
    PRIMARY KEY (`employee_id`),
    UNIQUE KEY `email` (`email`));
```

Tiene que tener valor

Para índices

Se puede añadir PRIMARY KEY después de ,

Clave primaria

UNIQUE: el valor no se repite

# Integridad Referencial (III/IV)

- Al borrar registros en la tabla principal, se borran registros en la relacionada.

**on delete cascade**

- Si se borra un registro de la tabla principal, los valores de los campos relacionados se colocan a null

**on delete set null**

- No permite eliminar valores en la tabla principal si hay registros relacionados en otra tabla.

**on delete restrict**

- Al cambiar los datos de los campos en la tabla principal, se actualizan en la relacionada.

**on update cascade**

- Al cambiar valores principales, se dejan a null los relacionados.

**on update set null**

- No permite modificar valores en la tabla principal si hay registros relacionados en otra tabla.

**on update restrict**

**on delete no action**

**on update no action**

# Integridad Referencial (IV/IV)

```
CREATE TABLE IF NOT EXISTS `Alquileres`.`conductores` (
  `per_id` VARCHAR(45) NOT NULL ,
  `coche_id` VARCHAR(45) NOT NULL ,
  PRIMARY KEY (`coche_id`, `per_id`) ,
  INDEX `fk_relaccion2` (`per_id` ASC) ,
  INDEX `fk_relaccion1` (`coche_id` ASC) ,
  CONSTRAINT `fk_relaccion2`
    FOREIGN KEY (`per_id`)
    REFERENCES `Alquileres`.`empleados` (`per_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_relaccion1`
    FOREIGN KEY (`coche_id`)
    REFERENCES `Alquileres`.`coches` (`coche_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION) ENGINE = InnoDB;
```

KEY is normally a synonym for INDEX. The key attribute PRIMARY KEY can also be specified as just KEY when given in a column definition. This was implemented for compatibility with other database systems.

# Curso de Programación para Internet con Java EE (Parte III - Database)

## T04 PL/SQL

© Antonio Santos Ramos 2018

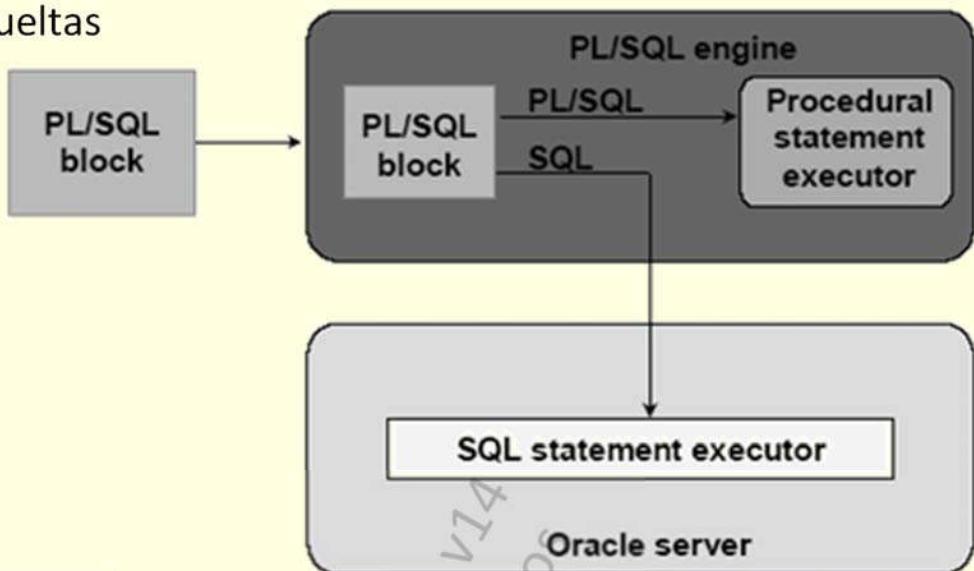
95

### ¿Qué es PL/SQL?

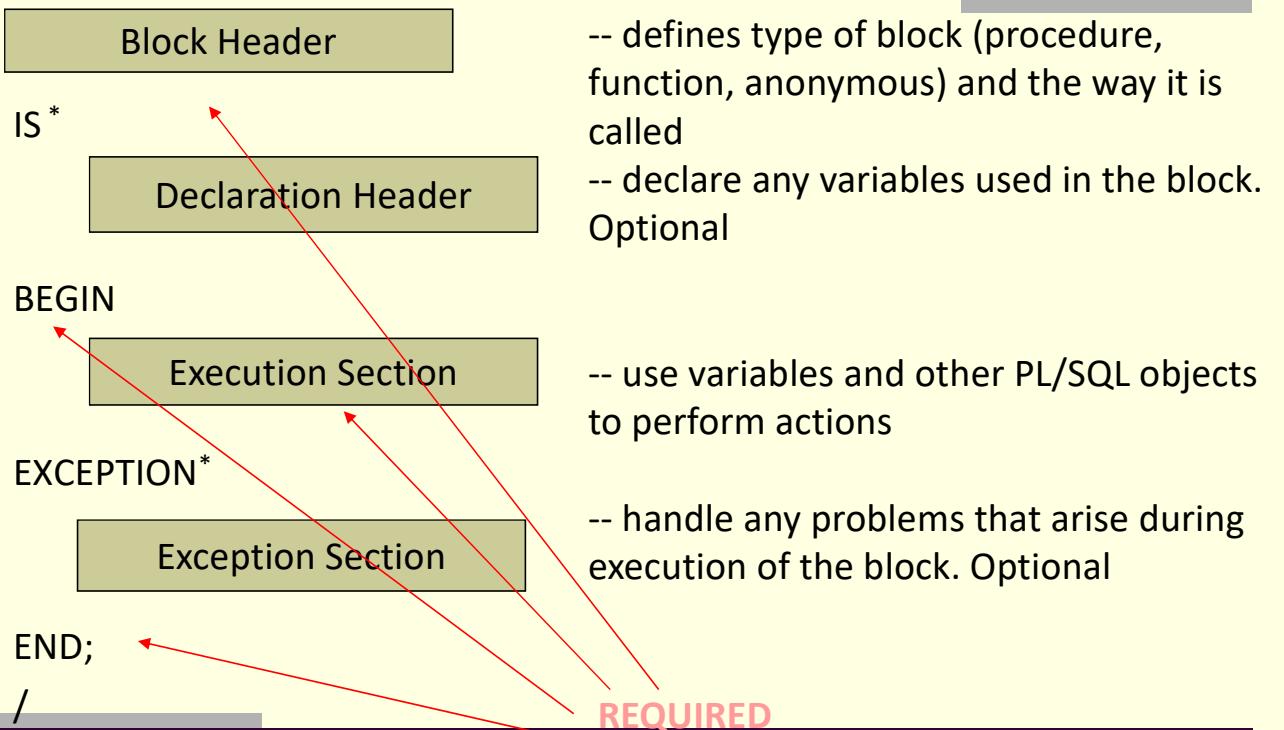
- Es un lenguaje procedural de Oracle, creado en 1992, que complementa a SQL
- SQL está embebido en PL/SQL
- Tiene conceptos:
  - PROCEDURES, FUNCTIONS, TRIGGERS, CURSORS
- Y elementos:
  - Variables, constants: para almacenar información
  - Loops
  - Conditional branching: sentencias IF
  - Functions: programas almacenados para realizar una función

# Bloque PL/SQL

- Un programa PL/SQL es un BLOQUE (Block)
  - Mejoran el rendimiento: se envían completos, no sentencias sueltas



## Estructura de Bloque



# PL/SQL...

- Is not case sensitive
- Uses the same datatypes as SQL
  - Also has boolean, record, table, ...
  - Allows reference datatypes
    - %type and %rowtype
- Allows comments
  - /\* and \*/ for multiline and -- for single line
- Uses := for assignment
- Uses ; to indicate end of line (instruction)

© Antonio Santos Ramos 2018

## PL/SQL (Example 01)

```
DECLARE
    loop_count BINARY_INTEGER := 0;
BEGIN
    LOOP
        INSERT INTO count_table VALUES (loop_count);
        DBMS_output.put_line
            ('loop_count is ' || to_char(loop_count));
        loop_count := loop_count + 1;
        EXIT WHEN loop_count = 6;
    END LOOP;
END;
/
```

© Antonio Santos Ramos 2018

# Variables and Constants

- Defined in DECLARE statement
  - This creates spaces in memory for temporary storage of data of a specific type
  - Constant values are fixed.
  - Variables can vary during execution
- Variables
  - Var\_name datatype;
  - Var\_name datatype := expression or value;
  - Var\_name datatype NOT NULL := expression or value;
- Constants
  - const\_name CONSTANT datatype := expression or value;

Are set to NULL  
by default

## Data Manipulation Language (Example 02)

```

DECLARE
    v_surname      personnel.surname%type;
    v_bonus        personnel.bonus%type;
BEGIN
    SELECT surname, bonus*1.15
    INTO v_surname, v_bonus
    FROM PERSONNEL
    WHERE SNUM = 3200;
    DBMS_OUTPUT.PUT_LINE(v_surname||' earns ' ||v_bonus);
END;
/

```

Note the single ; at end of SQL code

# Data Manipulation Language (Example 03)

```
DECLARE qty_on_hand NUMBER(6);
BEGIN
    SELECT quantity INTO qty_on_hand FROM inventory
    WHERE product = 'golf club';
    IF qty_on_hand > 0 THEN
        UPDATE inventory SET quantity = quantity - 1
        WHERE product='golf club';
        DBMS_output.put_line('in stock: '||qty_on_hand);
        INSERT INTO purchase_log
        VALUES('Golf club purchased', SYSDATE);
    ELSE
        INSERT INTO purchase_log
        VALUES('out of golf clubs', SYSDATE);
    END IF;
    COMMIT;
END;
/
```

Looks up quantity  
of golf clubs from  
inventory table and  
assigns to variable  
Checks > 0

Reduce  
quantity by 1

Record a message  
in the purchase log  
of zero stock

# LOOPS (Example 04 y 05)

```
DECLARE
    v_count number(2):=1;
BEGIN
    FOR v_count IN 1..10 LOOP
        insert into test(id_no) values(v_count);
    END LOOP;
END;
/
```

```
DECLARE
    v_count number(2):=1;
BEGIN
    WHILE v_count < 11 LOOP
        insert into test(id_no) values(v_count);
        v_count:=v_count+1;
    END LOOP;
END;
/
```

# IF...THEN...ELSIF...ELSE (Example 06)

```
IF condition THEN statement(s);  
[ELSIF condition THEN statement(s);]  
[ELSE statement(s);]  
END IF;
```

```
DECLARE  
v_count number(2):=1;  
BEGIN  
LOOP  
    IF v_count between 1 and 5 THEN  
        insert into test values (v_count, 'Group1');  
    ELSIF v_count between 6 and 10 THEN  
        insert into test values (v_count, 'Group2');  
    ELSE  
        insert into test values (v_count, 'Group3');  
    END IF;  
    EXIT WHEN v_count =15;  
    v_count:=v_count+1;  
END LOOP;  
END;  
/
```

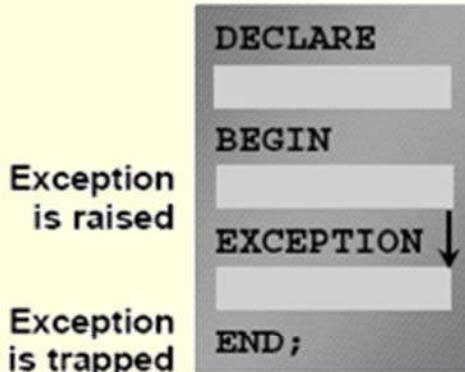
## Exceptions (I/IV)

- An exception is an identifier in PL/SQL that is raised during execution.
- How is it raised?
  - An Oracle error occurs.
  - You raise it explicitly.
- How do you handle it?
  - Trap it with a handler.
  - Propagate it to the calling environment.

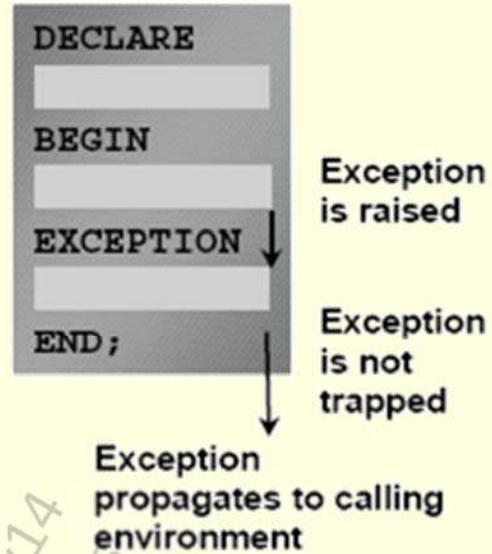
# Exceptions (II/IV)

## Handling exceptions

### Trap the exception



### Propagate the exception



© Antonio Santos Ramos 2018

# Exceptions (III/IV)

## Trapping exceptions

```
EXCEPTION
WHEN exception1 [OR exception2 . . .] THEN
    statement1; /* one or more PL/SQL or SQL statements */
    statement2;
    .
    .
    .
    [WHEN exception3 [OR exception4 . . .] THEN
        statement1;
        statement2;
        .
        .]
    [WHEN OTHERS THEN
        statement1;
        statement2;
        .
        .]
```

- Only one handler is processed
- WHEN OTHERS is unique and must be the last handler

© Antonio Santos Ramos 2018

# Exceptions (IV/IV)

## (Ejemplo 07)

```
FUNCTION build_name (name_in IN VARCHAR2, sex_in IN VARCHAR2)
RETURN VARCHAR2 IS
    unknown_sex EXCEPTION;
    name_out VARCHAR2(100);
BEGIN
    IF sex_in = 'M' THEN name_out := 'Mr. ' || name_in;
    ELSIF sex_in = 'F' THEN name_out := 'Ms. ' || name_in;
    ELSE
        RAISE unknown_sex
    END IF;
    RETURN name_out;
EXCEPTION
    WHEN unknown_sex THEN
        DBMS_OUTPUT.PUT_LINE
            ('Unable to determine gender of individual!');
END;
/
```

© Antonio Santos Ramos 2018

## Stored Procedures

- A unit of code that performs one or more tasks
- After completion, execution returns to the calling block
- To run the procedure at any time, use EXECUTE <procedure>

```
CREATE OR REPLACE PROCEDURE proc_name
IS
<declarations of variables, cursors etc> .....
BEGIN
    <executing code> .....
END proc_name;
```

```
CREATE OR REPLACE PROCEDURE sal_update
IS
BEGIN
    UPDATE personnel set salary=salary*1.1
        where div=10;
END;
```

# Functions

- Functions are similar to procedures
- They are used for calculations and returning a value

```
CREATE OR REPLACE FUNCTION
    function_name      (parameter list)
RETURN return_datatype
IS
    ... variables, cursors etc
BEGIN
    Execution code .....;
    Return expression;
END;
```

Can be:  
NUMBER  
VARCHAR2  
BOOLEAN  
etc

© Antonio Santos Ramos 2018

111

## Function (Ejemplo 08)

```
CREATE OR REPLACE FUNCTION
    get_aveSal (i_div IN NUMBER)
RETURN number
IS
    v_salary personnel.salary%type;
BEGIN
    SELECT avg(salary)
    INTO v_salary FROM Personnel
    WHERE div=i_div;
    RETURN v_salary;
END get_aveSal;
```

```
SET SERVEROUTPUT ON
DECLARE
    V_divID          personnel.div%type;
    V_divName        branch.DivName%type:= '&divName';
    V_aveSalary      personnel.salary%type;
BEGIN
    SELECT div into v_divID
    FROM branch WHERE divname=v_divName;
    V_aveSalary:=get_aveSal(v_divID);
    DBMS_OUTPUT.PUT_LINE('Division ||v_divID|| has
                         ||v_aveSalary|| average salary');
END;
```

"get the average salary for ADMIN"  
Block prompts for division name  
then passes the division number to  
the function get\_aveSal

112

# Triggers

- Triggers are stored programs
- They are automatically executed or fired when some events occur.
  - A database manipulation (DML) statement
    - (DELETE, INSERT, or UPDATE)
  - A database definition (DDL) statement
    - (CREATE, ALTER, or DROP).
  - A database operation
    - (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

## Trigger (Example 09)

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

- When a record is created in the CUSTOMERS table, the above create trigger, display\_salary\_changes will be fired