

UNITAT 13

ACCÉS A BASES DE DADES

Programació
CFGS DAW

Autors:

Carlos Cacho y Raquel Torres

Revisat per:

Lionel Tarazon - lionel.tarazon@ceedcv.es

Fco. Javier Valero – franciscojavier.valero@ceedcv.es

José Manuel Martí - josemanuel.marti@ceedcv.es

2021/2022

Llicència



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) Reconeixement – No Comercial – Compartir Igual (by-nc-sa)

No es permet un ús comercial de l'obra original ni de les possibles obres derivades, la distribució de les quals s'ha de fer amb una llicència igual a la que regula l'obra original. Aquesta és una obra derivada de l'obra original de Carlos Cacho i Raquel Torres.

Nomenclatura

Al llarg d'aquest tema s'utilitzaran diferents símbols per a distingir elements importants dins del contingut. Aquests símbols són:



Important



Atenció



Interessant

ÍNDEX DE CONTINGUT

INTRODUCCIÓ	5
REPÀS DEL LENGUATGE SQL	6
Comandos	6
Clàusules	7
Operadors	7
Exemples de sentències SQL	7
JDBC	10
Funcions del JDBC	10
Drivers JDBC	11
ACCÉS BASES DE DADES DES DE NETBEANS	12
Instal·lació del JDBC Driver	12
Connexió a base de Dades mitjançant el JDBC Driver	14
Crear una taula	17
Mitjançant sentències SQL	17
Mitjançant l'assistent de creació de taules	17
Inserir dades	18
Consultar valors	18
ACCÉS A BASES DE DADES MITJANÇANT CODI JAVA	19
Afegir la llibreria JDBC al projecte	19
Carregar el Driver	19
Classe DriverManager	20
Classe Connection	21
Classe Statement	21
Classe ResultSet	22
Exemple complet	23
NAVEGABILIDAD I CONCURRÈNCIA	24
CONSULTES	25
Navegació d'un ResultSet	25
Obtenint dades del ResultSet	26
Tipus de dades i conversions	26
MODIFICACIÓ	27
INSERCIÓ	29
ESBORRAT	30
PROGRAMA GESTOR DE CLIENTS	31
A GRAÏMENTS	34

UD13. ACCÉS A BASES DE DADES

1. INTRODUCCIÓ



Una **base de dades** és una col·lecció de dades classificades i estructurades que són guardats en un o diversos fitxers, però referenciats com si d'un únic fitxer es tractara.

Per a crear i manipular bases de dades relacionals, existeixen en el mercat diversos sistemes de gestió de bases de dades (SGBD); per exemple, Access, SQL Server, Oracle i DB2. Altres SGBD de lliure distribució són MySQL/MariaDB i PostgreSQL.

Les dades d'una base de dades relacional s'emmagatzemen en taules lògicament relacionades entre si utilitzant camps clau comuns. Al seu torn, cada taula disposa les dades en files i columnes. Per exemple, pensa en una relació de clients, a les files se les denomina **tuplas** o **registres** i a les columnes **camps**.

Els usuaris d'un sistema administrador de bases de dades poden realitzar sobre una determinada base operacions com inserir, recuperar, modificar i eliminar dades, així com afegir noves taules o eliminar-les. Aquesta operacions s'expressen generalment en un llenguatge denominat **SQL**.

Abans de començar **necessitem un sistema de gestió de base de dades** instal·lat en el nostre ordinador en el qual poder tindre bases de dades a les quals connectar-nos des dels nostres programes escrits en llenguatge Java. Si ja tenim un instal·lat (per exemple MySQL, MariaDB, *PostgreSQL, etc.) podem utilitzar-lo si volem. De totes maneres en aquesta unitat us proposem utilitzar **XAMPP (Windows/Linux/iOs Apache MariaDB PHP Perl)** que incorpora un servidor web i l'eina phpMyAdmin per a treballar amb bases de dades. Ho pots descarregar des de l'adreça: <https://www.apachefriends.org> Per a la seua instal·lació tan sols has de seguir els passos de l'instal·lador.

Veureu que les versions més actuals de XAMPP han substituït MySQL per MariaDB. No passa res, tots dos són compatibles i poden utilitzar-se indistintament.

2. REPÀS DEL LENGUATGE SQL

En aquest apartat es dona un repàs als aspectes més rellevants del llenguatge SQL de manipulació de bases de dades relacionals que ja hauràs vist en el mòdul de Bases de Dades. Te vindrà bé per a repassar conceptes. Si no has cursat aquest mòdul, és molt important que lliges aquest apartat amb atenció. No és necessari tindre un coneixement avançat sobre el llenguatge SQL però sí els aspectes més bàsics que es cobreixen en aquest apartat.

SQL inclou sentències tant de creació de dades (CREATE) com de manipulació de dades (INSERT, UPDATE i DELETE) així com de consulta de dades (SELECT).

La sintaxi del llenguatge SQL està format per quatre grups sintàctics que poden combinar-se en diverses de les sentències més habituals. Aquests 4 grups són: **Comandos**, **Clàusules**, **Operadors** i **Funcions**. En aquest apartat repassarem només els tres primers.

2.1 Comandos

Existeixen tres tipus de comandos en SQL:

- Els DDL (Data Definition Language), que permeten crear, modificar i esborrar noves bases de dades, taules, camps i vistes.
- Els DML (Data Manipulation Language), que permeten introduir informació en la BD, esborrar-la i modificar-la.
- Els DQL (Data Query Language), que permeten generar consultes per a ordenar, filtrar i extraure informació de la base de dades.

Els comandos DDL són:

- *CREATE*: Crear noves taules, camps i índexs.
- *ALTER*: Modificació de taules afegint camps o modificant la definició dels camps.
- *DROP*: Instrucció per a eliminar taules, camps i índexs.

El comandos DML són:

- *INSERT*: Inserir registres a la base de dades.
- *UPDATE*: Instrucció que modifica els valors dels camps i registres especificats en els criteris.
- *DELETE*: Eliminar registres d'una taula de la base de dades.

El principal comando DQL és:

- *SELECT*: Consulta de registres de la base de dades que compleixen un criteri determinat.

2.2 Clàusules

Les clàusules són condicions de modificació, utilitzades per a definir les dades que es desitgen seleccionar o manipular:

- *FROM*: Utilitzada per a especificar la taula de la qual se seleccionaran els registres.
- *WHERE*: Clàusula per a detallar les condicions que han de reunir els registres resultants.
- *GROUP BY*: Utilitzat per a separar registres seleccionats en grups específics.
- *HAVING*: Utilitzada per a expressar la condició que ha de complir cada grup.
- *ORDER BY*: Utilitzada per a ordenar els registres seleccionats d'acord amb un criteri donat.

2.3 Operadors

Operadors lògics:

- *AND*: Avalua dues condicions i retorna el valor cert, si totes dues condicions són certes.
- *OR*: Avalua dues condicions i retorna el valor cert, si alguna de les dues condicions és certa.
- *NOT*: Negació lògica. Retorna el valor contrari a l'expressió.

Operadors de comparació:

- *< [...]* menor que [...]
- *> [...]* major que [...]
- *<> [...]* diferent a [...]
- *<= [...]* menor o igual que [...]
- *>= [...]* major o igual que [...]
- *= [...]* igual que [...]
- *BETWEEN*: Especifica un interval de valors
- *LIKE*: Compara un model
- *IN*: Operadors per a especificar registres d'una taula

2.4 Exemples de sentències SQL

Per a crear **una base de dades**: *CREATE DATABASE <nom>*

Exemple: *CREATE DATABASE tiendaonline*

Per a eliminar **una base de dades**: *DROP DATABASE <base de dades>*

Exemple: *DROP DATABASE tiendaonline*

Per a utilitzar **una base de dades**: *USE <base de dades>*

Exemple: *USE tiendaonline*

NOTA: Amb *USE* indiquem sobre quina base de dades volem treballar (podem tindre diverses).

Per a crear **una taula**: *CREATE TABLE <taula> (<columna 1> [,<columna 2>] ...)*

On <columna n> es formula segons la següent sintaxi:

<columna n> <tipus de dada> [DEFAULT <expresio>] [<const 1> [<const2>]...]

La clàusula *DEFAULT* permet especificar un valor per omissió per a la columna i, opcionalment, per a indicar la forma o característica de cada columna, es poden utilitzar les constants: *NOT NULL*, *UNIQUE* o *PRIMARY KEY*.

La clàusula *PRIMARY KEY* s'utilitza per a definir la columna com a clau principal de la taula. Això suposa que la columna no pot contindre valors nuls ni duplicats. Una taula pot contindre una sola restricció *PRIMARY KEY*.

La clàusula *UNIQUE* indica que la columna no permet valors duplicats. Una taula pot tindre diverses restriccions *UNIQUE*.

Exemple:

```
CREATE TABLE clients(  
    nom CHAR(30) NOT NULL,  
    direccio CHAR(30) NOT NULL,  
    telefon CHAR(12) PRIMARY KEY NOT NULL,  
    observacions CHAR(240)  
)
```

Per a esborrar **una taula**:

```
DROP TABLE <taula>
```

Exemple:

```
DROP TABLE clients
```

Per a inserir **registres en una taula**:

```
INSERT [INTO] <taula> [(<columna 1>[,<columna 2>]...)]  
VALUES (<expresio 1>[,<expresio 2>]...),...
```

Exemple: Inserim dos nous registres en la taula clients

```
INSERT INTO clients VALUES ('Pepito Pérez','VALÈNCIA','963003030','Ninguna'),  
('María Martínez','VALÈNCIA','961002030','Ninguna')
```

Per a modificar **registres ja existents en una taula**:

```
UPDATE <taula> SET <columna1 = (<expresio1> | NULL) [<columna2 = ...]...  
WHERE <condició de cerca >
```

Exemples: Canviem el camp 'adreça' del registre el telèfon del qual és '963003030'

```
UPDATE clients SET direccio = 'Port de Sagunt'  
WHERE telefon = '963003030'
```

Per a esborrar **registres d'una taula**:

DELETE FROM <taula> WHERE <condició de cerca>

Exemple: Esborrem els registres el camp dels quals 'telefon' és 963003030.

DELETE FROM clients WHERE telefon='963003030'

Per a realitzar **una consulta**, és a dir, obtindre registres d'una base de dades:

SELECT [ALL | DISTINCT] <llista de selecció>

FROM <taules>

WHERE <condicions de selecció>

[ORDER BY <columna1> [ASC|DESC][, <columna2>[ASC|DESC]]...]

Alguns exemples senzills:

Obtenim tots els registres de la taula 'clients':

*SELECT * FROM clients;*

Obtenim només el camp 'nomene' de tots els registres de la taula 'clients':

SELECT nom FROM clients;

Obtenim només els camps 'nom' i 'telefon' de la taula 'clients':

SELECT nom, telefon FROM clients;

Obtenim tots els registres de 'clients' ordenats per nom:

*SELECT * FROM clients ORDER BY nom;*

Obtenim només els noms i telèfons dels clients, ordenats per nom:

SELECT nom, telefon FROM clients ORDER BY nom;

Obtenim tots els registres de 'clients' on el camp 'telefon' és major que 1234

*SELECT * FROM clients WHERE telefon > '1234';*

Obtenim tots els registres de 'clients' en els quals el camp 'telefon' comença per 91

*SELECT * FROM clients WHERE telefon LIKE '91';*

Per a veure exemples més complexos i/o que impliquen dades de diverses taules podeu consultar aquest material complementari:

<https://www.cs.us.es/blogs/bd2013/files/2013/09/consultas-sql.pdf>

3. JDBC

Java pot connectar-se amb diferents SGBD i en diferents sistemes operatius. Independentment del mètode en què s'emmagatzemen les dades ha d'existir sempre un **mediador** entre l'aplicació i el sistema de base de dades i en Java aqueixa funció la realitza **JDBC**.



Per a la connexió a les bases de dades utilitzarem la API estàndard de JAVA denominada **JDBC** (Java Data Base Connection)

JDBC és un API (Application Programming Interface) inclòs dins del llenguatge Java per a l'accés a bases de dades. Consisteix en un conjunt de classes i interfícies escrites en Java que ofereixen un complet API per a la programació amb bases de dades, per tant és l'única solució 100% Java que permet l'accés a bases de dades.

JDBC és una especificació formada per una col·lecció d'interfícies i classes abstractes, que tots els fabricants de drivers han d'implementar si volen realitzar una implementació del seua driver 100% Java i compatible amb JDBC (JDBC-compliant driver). Pel fet que JDBC està escrit completament a Java també posseeix l'avantatge de ser independent de la plataforma.



No serà necessari escriure un programa per a cada tipus de base de dades, una mateixa aplicació escrita utilitzant JDBC podrà manejar bases de dades Oracle, Sybase, SQL Server, etc.

A més podrà executar-se en qualsevol sistema operatiu que posseïska una Màquina Virtual de Java, és a dir, seran aplicacions completament independents de la plataforma. Unes altres APIS que se solen utilitzar bastant per a l'accés a bases de dades són DAO (Data Access Objects) i RDO (Remote Data Objects), i ADO (ActiveX Data Objects), però el problema que ofereixen aquestes solucions és que només són per a plataformes Windows.

JDBC té les seues classes en el paquet *java.sql* i altres extensions en el paquet *javax.sql*.

3.1 Funcions del JDBC

Bàsicament el API JDBC fa possible la realització de les següents tasques:

- Establir una connexió amb una base de dades.
- Enviar sentències SQL.
- Manipular dades.
- Processar els resultats de l'execució de les sentències.

3.2 Drivers JDBC

Els drivers ens permeten connectar-nos amb una base de dades determinada. Existeixen **quatre tipus de drivers JDBC**, cada tipus presenta una filosofia de treball diferent. A continuació es passa a comentar cadascun dels drivers:

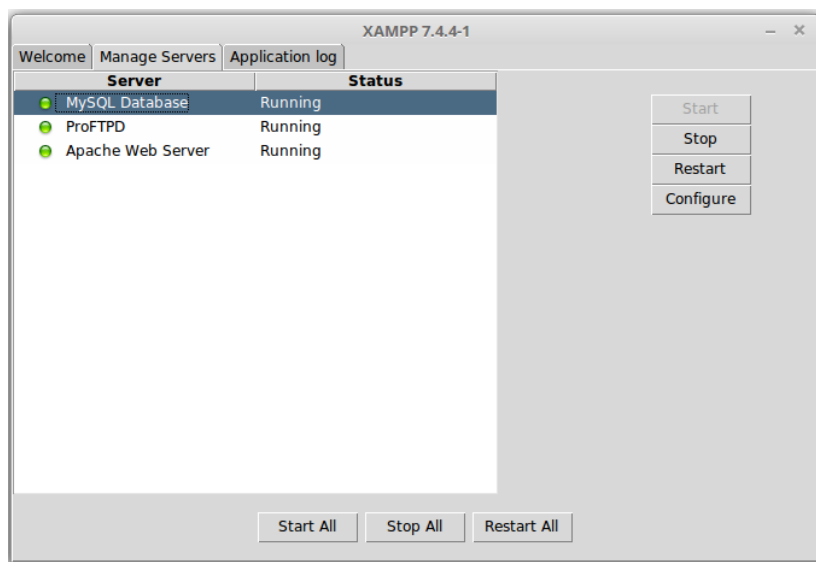
- JDBC-ODBC bridge plus ODBC driver (tipus 1): permet al programador/a accedir a fonts de dades ODBC existents mitjançant JDBC. El JDBC-ODBC Bridge (pont JDBC-ODBC) implementa operacions JDBC traduint-les a operacions ODBC, es troba dins del paquet *sun.jdbc.odbc* i conté llibreries natives per a accedir a ODBC.

Al ser usuari de ODBC, depenem de les dll de ODBC i això limita la quantitat de plataformes on es pot executar l'aplicació.

- Native-API partly-Java driver (tipus 2): són similars als drivers de tipus 1, encara que també necessiten una configuració en la màquina client. Aquest tipus de driver converteix anomenades JDBC a anomenades de Oracle, Sybase, Informix, DB2 o altres SGBD. Tampoc es poden utilitzar dins de miniaplicacions al posseir codi natiu.
- JDBC-Net pure Java driver (tipus 3): Aquests controladors estan escrits en Java i s'encarreguen de convertir les anomenades JDBC a un protocol independent de la base de dades i a més en l'aplicació servidora utilitzen les funcions natives del sistema de gestió de base de dades mitjançant l'ús d'una biblioteca JDBC en el servidor. L'avantatge d'aquesta opció és la portabilitat.
- JDBC de Java client (tipus 4): Aquests controladors estan escrits en Java i s'encarreguen de convertir les anomenades JDBC a un protocol independent de la base de dades i en l'aplicació servidora utilitzen les funcions natives del sistema de gestió de base de dades sense necessitat de biblioteques. L'avantatge d'aquesta opció és la portabilitat. Són com els drivers de tipus 3 però sense la figura de l'intermediari i tampoc requereixen cap configuració en la màquina client. Els drivers de tipus 4 es poden utilitzar per a servidors Web de grandària petita i mitjana, així com per a intranets.

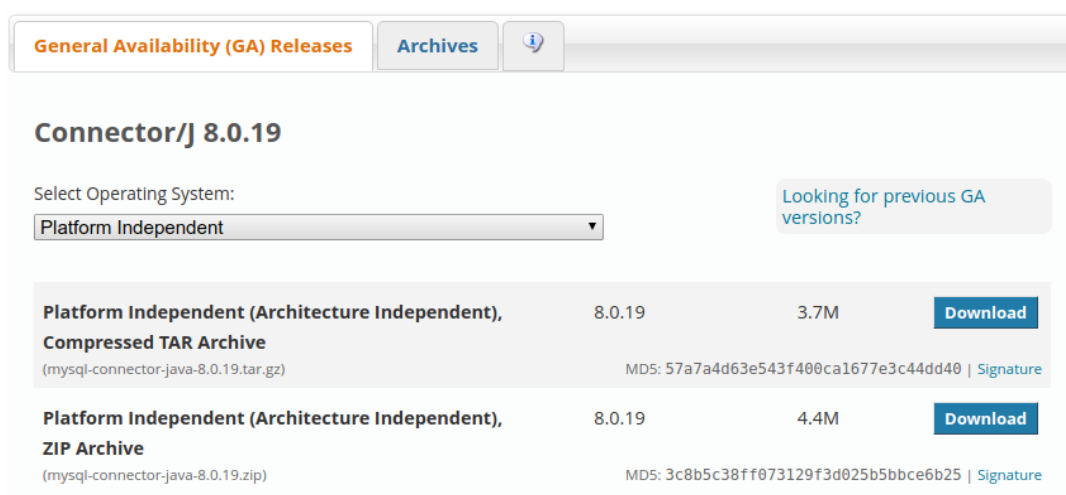
4. ACCES BASES DE DADES DES DE NETBEANS

Sempre que vulguem treballar amb el servidor i les bases de dades del mateix haurem d'iniciar l'eina XAMPP descarregada anteriorment i iniciar els serveis Apache i MySQL/MariaDB:



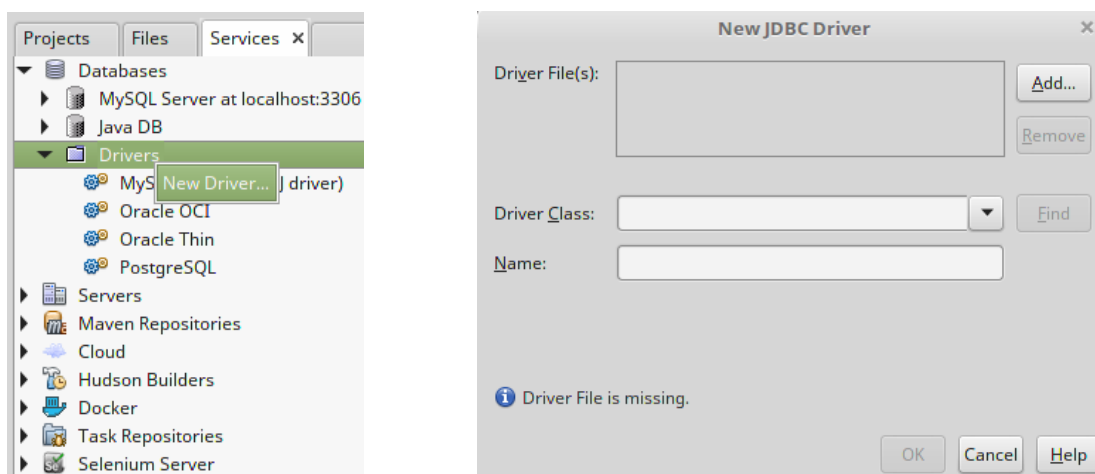
4.1 Instal·lació del JDBC Driver

Per a poder connectar-se a una base de dades MySQL/MariaDB des de l'explorador de NetBeans necessitem instal·lar el driver mysql-connector de Java. Primer cal descarregar-lo de l'adreça web <http://dev.mysql.com/downloads/connector/j/>. En l'opció de sistema operatiu devem seleccionar "Platform Independent" i després descarregar el ZIP.

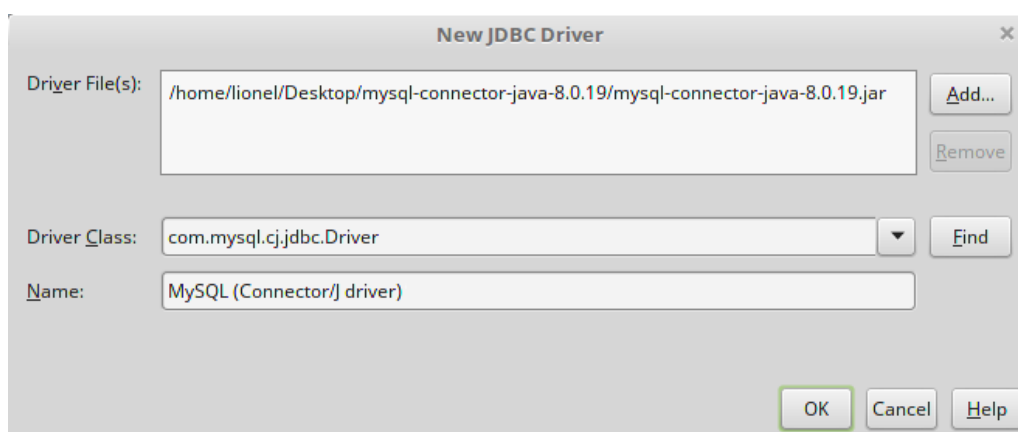


No fa falta registrar-se per a descarregar-lo, només s'ha de prémer en "No thanks, just start my download." Descomprinx el ZIP i cerca l'arxiu **mysql-connector-java-8.0.19.jar**. Tingues en compte que el número de versió pot ser diferent a la que es mostra en la imatge.

Ara **obri NetBeans**, veu al **panell Services → Databases**, fes **clic dret en l'opció Drivers** i selecciona l'opció **New Driver** del menú contextual. Es mostrarà el diàleg **New JDBC Driver**.



Fes **clic en Add...** i selecciona el fitxer `mysql-connector-java-8.0.19.jar` que has descarregat anteriorment. Una vegada seleccionat el fitxer ens ha d'emplenar la classe principal del Driver que en el nostre cas ha de ser `com.mysql.cj.jdbc.Driver` i el nom del nostre driver, per exemple **MySQL (Connector/J driver)**.



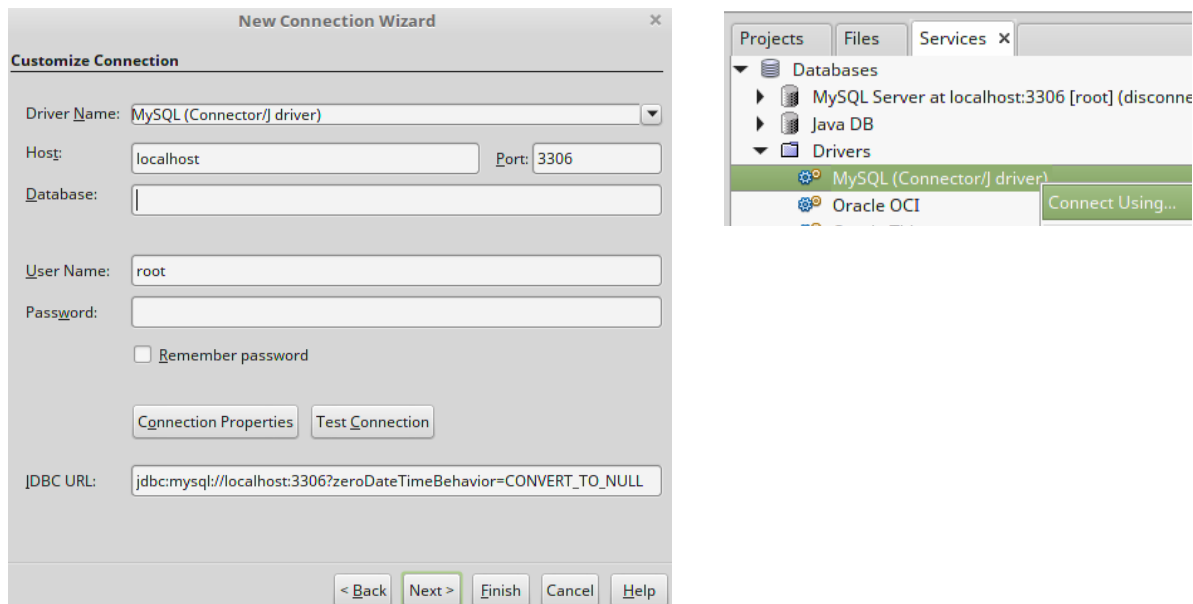
NOTA: Si estàs utilitzant l'última versió de NetBeans és possible que en Services → Databases → Drivers ja t'aparega l'opció 'MySQL (Connector/J driver)'. Malgrat això pot ser que necessites instal·lar el driver. Dona-li al driver amb clic dret → 'Connect using' i comprova si en la finestra t'apareix el missatge 'driver file is missing'. En tal cas, dona-li a 'add driver' i selecciona l'arxiu jar esmentat anteriorment.

Arribats a aquest punt ja tenim enllaçat el driver JDBC de MySQL/MariaDB des del Database Explorer. Ara el que ens queda és crear una connexió amb una base de dades MySQL/MariaDB.

4.2 Connexió a base de Dades mitjançant el JDBC Driver

Fes clic dret en 'MySQL (Connector/J driver)' i selecciona l'opció 'Connect Using...'.

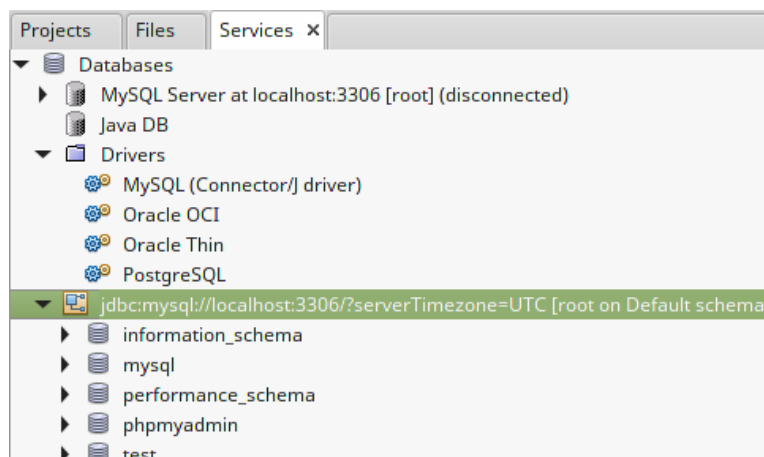
Apareixerà la finestra 'New Connection Wizard':



- **Driver Name:** El driver que volem utilitzar. En aquest cas 'MySQL (Connector/J driver)'.
- **Host:** L'adreça IP o nom de domini de la màquina on està instal·lat el servidor MySQL. En el nostre cas 'localhost' (màquina local).
- **Port:** Port de connexió. MySQL utilitza el 3306 per defecte.
- **Database:** El nom de la base de dades a la qual volem accedir. Podem deixar aquest camp buit per a poder accedir a totes les que existisquen en el servidor.
- **User Name:** Nom d'usuari. Utilitzarem 'root' (administrador).
- **Password:** Contrasenya de l'usuari. Ho deixarem en blanc ja que després d'instal·lar MySQL la contrasenya per defecte està buida. En el nostre cas per a practicar i aprendre això serà suficient, però en un entorn real tindríem que configurar una contrasenya robusta.
- **JDBC URL:** La URL de connexió amb el format jdbc:mysql://HOST:PORT. Es crea automàticament a partir de la informació anterior. És possible que incloga per defecte variables com 'zeroDateTimeBehaviour' en la imatge.

Fes clic a **Finalitzar**. Si tot va bé, en Services → Databases apareixerà una nova connexió anomenada `jdbc:mysql://localhost:3306... [root on Default schema]`

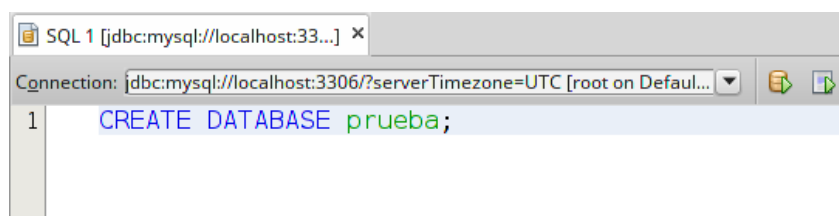
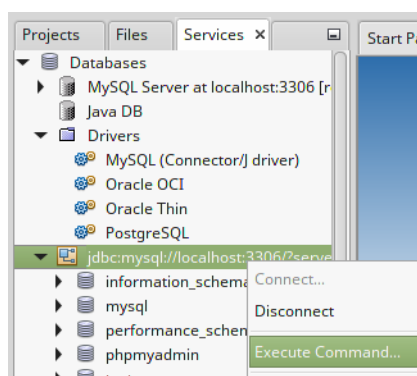
NOTA: Si falla a causa d'un error 'server time zone' pots solucionar-lo afegint a l'URL la variable `serverTimezone` amb el valor UTC: `jdbc:mysql://localhost:3306/?serverTimezone=UTC`



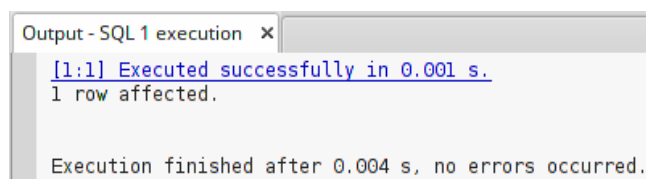
Si despleguem la connexió creada veurem que podem accedir a totes les bases de dades, taules, etc. emmagatzemades en el servidor al qual estem connectats (a cadascú li apareixeran les bases de dades que tinga en el seu sistema, poden ser diferents a les de la imatge).

Això és molt útil ja que ens permetrà accedir còmodament a la base de dades, realitzar consultes, executar comandos, etc. mentre desenvolupem un projecte amb NetBeans.

Per a executar un comando sobre la base de dades a la qual ens hem connectat només cal fer clic dret sobre la connexió i seleccionar l'opció 'Execute Command...'. S'obrirà un panell en l'editor central en el que podrem introduir sentències SQL (tantes com vulguem) i executar-les fent clic en la icona 'Run SQL':



Prova d'executar la sentència **CREATE DATABASE prueba;** com en la imatge. En el panell de baix es mostrarà un missatge amb el resultat de l'execució.



En aquest cas indica 'Executed successfully' (s'ha executat satisfactòriament). En la connexió jdbc del panell de l'esquerra apareixerà la nova base de dades 'prueba' que hem creat.

Cal tindre en compte que com hem configurat aquesta connexió sense especificar cap base de dades concreta, si volguérem executar sentències sobre una base de dades de les disponibles hauríem d'utilitzar sempre la sentència `USE nom_de_base_de_dades`. Per exemple:

```
USE tiendaonline;  
SELECT * FROM clients;
```

Si treballarem habitualment sobre la mateixa base de dades i volem evitar haver d'escriure sempre la sentència `USE` en totes les execucions de sentències SQL, podem crear una connexió configurada per a utilitzar una base de dades concreta. Per a això primer hauríem de seguir el mateix procediment explicat en aquest apartat (Clic dret en 'MySQL (Connector/J driver)' → 'Connect Using...' → Introduir les dades en el diàleg 'New Connection Wizard' i en el camp 'Database' escriure la base de dades desitjada.

Per exemple, per al cas de la base de dades prova que hem creat, la connexió seria aquesta:

The screenshot shows the 'New Connection Wizard' dialog box with the 'Customize Connection' tab selected. The fields are filled as follows: Driver Name is 'MySQL (Connector/J driver)', Host is 'localhost', Port is '3306', Database is 'prueba', User Name is 'root', and Password is empty. The 'Remember password' checkbox is unchecked. There are buttons for 'Connection Properties' and 'Test Connection'. The JDBC URL is displayed as 'jdbc:mysql://localhost:3306/prueba?serverTimezone=UTC'. A status message at the bottom says 'Connection Succeeded.' and there are navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

La URL resultant seria: **`jdbc:mysql://localhost:3306/prova?serverTimezone=UTC`**

En els següents apartats veurem alguns exemples d'execució de sentències SQL utilitzant aquesta nova connexió a la base de dades 'prova'.

4.3 Crear una taula

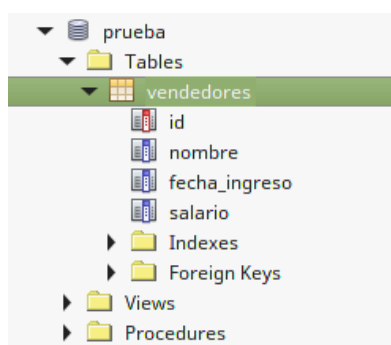
Des del IDE de NetBeans podem crear taules de dues formes: per sentències SQL o utilitzant un assistent de creació de taules.

4.3.1 Mitjançant sentències SQL

Seleccionem la connexió *jdbcm:mysql://localhost:3306/prova* creada en l'apartat anterior. Fem clic dret sobre ella → *Execute Command...* i introduïm el següent comando SQL:

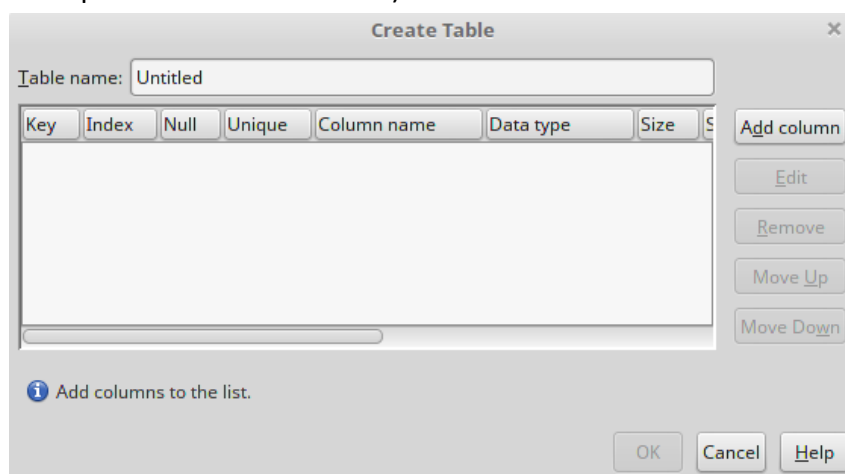
```
CREATE TABLE venedors (  
  id int NOT NULL auto_increment,  
  nom varchar(50) NOT NULL default '',  
  data_ingres date NOT NULL default '0000-00-00',  
  salari float NOT NULL default '0',  
  PRIMARY KEY (id) );
```

Si tot ha anat bé dins de 'prova → Tables' veurem la nova taula. És possible que necessitem fer clic dret → *Refresh* (sobre prova) perquè s'actualitzi la informació.



4.3.2 Mitjançant l'assistent de creació de taules

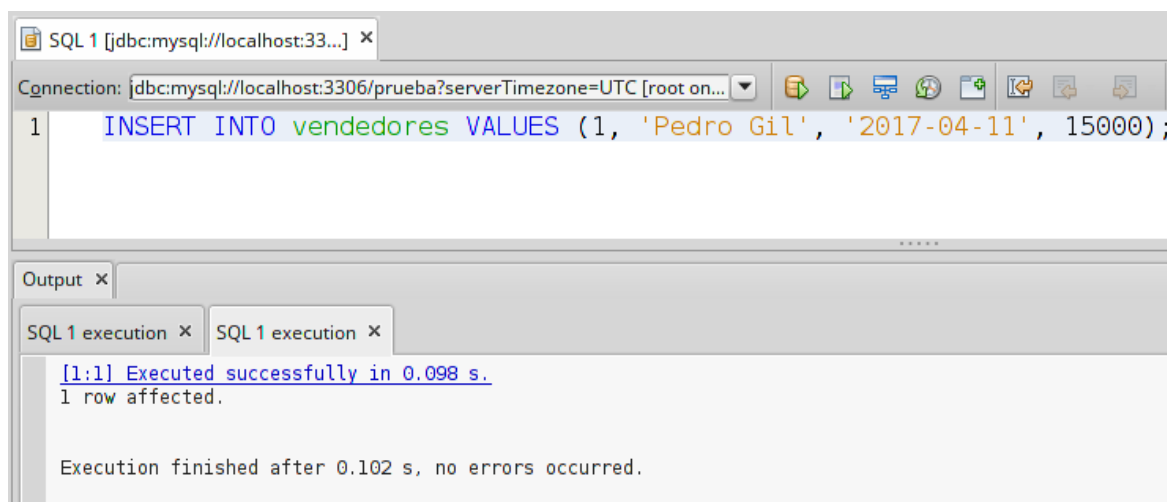
Sobre la carpeta 'Tables' de la connexió desitjada, fem clic dret i triem 'Create Table'. S'obrirà l'assistent de creació de taules. Afegim totes les columnes que necessitem mitjançant l'opció 'Add Column' (indicant les opcions de cada columna) i fem clic en 'OK'.



4.4 Inserir dades

Executar el comando INSERT INTO... sobre la connexió desitjada. Per exemple:

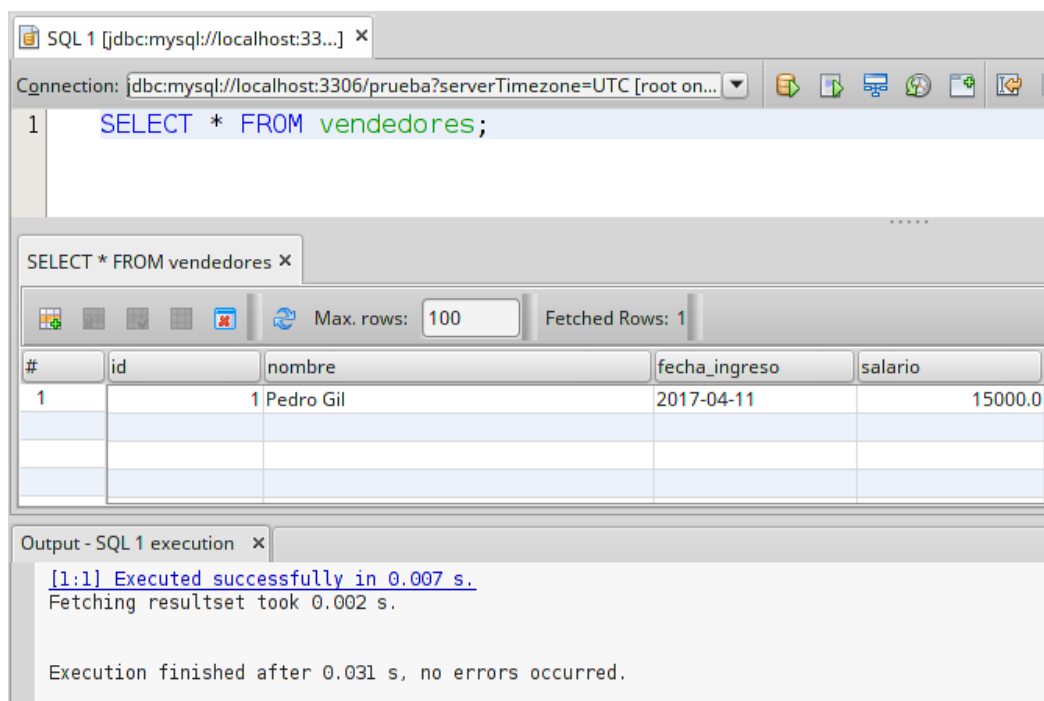
```
INSERT INTO vendedores VALUES (1, 'Pedro Gil', '2017-04-11', 15000);
```



4.5 Consultar valors

Executar el comando SELECT... sobre la connexió desitjada. Per exemple:

```
SELECT * FROM vendedores;
```

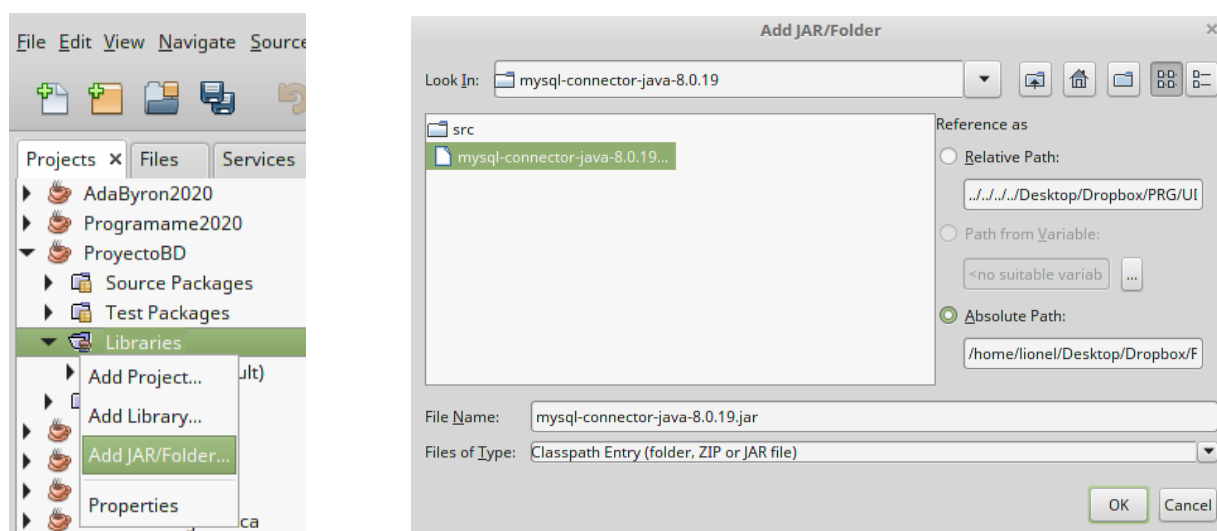


5. ACCÉS A BASES DE DADES MITJANÇANT CODI JAVA

En aquest apartat s'ofereix una introducció als aspectes fonamentals de l'accés a bases de dades mitjançant codi Java. En els següents apartats s'explicaràn alguns aspectes en major detall, sobretot els relacionats amb les classes `Statement` i `ResultSet`.

5.1 Afegir la llibreria JDBC al projecte

Per a poder utilitzar la llibreria JDBC en un projecte Java primer haurem d'afegir-la al projecte. Per a això hem de fer clic dret sobre la carpeta 'Libraries' del projecte i seleccionar 'Add JAR/Folder'. En la finestra emergent haurem de seleccionar l'arxiu del driver prèviament descarregat `mysql-connector-java-8.0.19.jar` i clic en OK.



5.2 Carregar el Driver

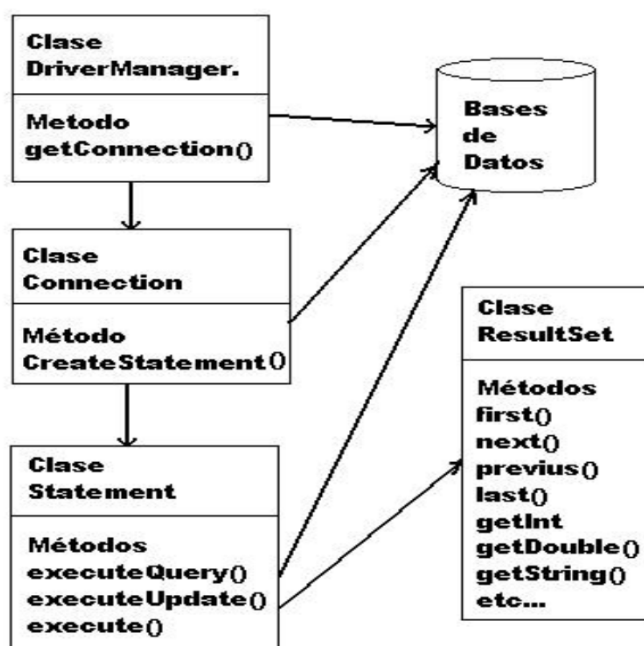
En un projecte Java que realitzi connexions a bases de dades és necessari, primer de tot, utilitzar `Class.forName(...).newInstance()` per a carregar dinàmicament el Driver que utilitzarem. Això només és necessari fer-ho una vegada en el nostre programa. Pot llançar excepcions pel que es necessari utilitzar un bloc try-catch.

```
try {  
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance();  
} catch (Exception e) {  
    // manegem l'error  
}
```

Cal tindre en compte que les classes i mètodes utilitzats per a connectar-se a una base de dades (explicats més endavant) funcionen amb tots els drivers disponibles per a Java (JDBC és només un, hi ha molts més). Açò és possible ja que l'estàndard de Java només els defineix com a interfícies (`interface`) i cada llibreria driver els implementa (defineix les classes i el seu codi). Per això és necessari utilitzar `Class.forName(...)` per a indicar-li a Java quin driver utilitzarem.

Aquest nivell de abstracció facilita el desenvolupament de projectes ja que si necessitàrem utilitzar un altre sistema de base de dades (que no fora MySQL) sols necessitaríem canviar la línia de codi que carrega el driver i poc més. Si cada sistema de base de dades necessitara que utilitzàrem diferents classes i mètodes, tot seria molt més complicat.

Les quatre classes fonamentals que tota aplicació Java necessita per a connectar-se a una base de dades i executar sentències són: **DriverManager**, **Connection**, **Statement** i **ResultSet**.



5.3 Classe DriverManager

La classe **java.sql.DriverManager** és la capa gestora del driver JDBC. S'encarrega de manejar el Driver apropiat i **permet crear connexions amb una base de dades** mitjançant el mètode estàtic **getConnection(...)** que té dues variants:

```

DriverManager.getConnection(String url)
DriverManager.getConnection(String url, String user, String password)
  
```

Aquest mètode intentarà establir una connexió amb la base de dades segons la URL indicada. Opcionalment se li pot passar l'usuari i contrasenya com a argument (també es pot indicar en la pròpia URL). Si la connexió és satisfactòria retornarà un objecte **Connection**.

Exemple de connexió a la base de dades 'prova' en localhost:

```

String url = "jdbc:mysql://localhost:3306/prova";
Connection conn = DriverManager.getConnection(url, "root", "");
  
```

Aquest mètode pot llançar dos tipus d'excepcions (que caldrà manejar amb un try-catch):

- **SQLException**: La connexió no ha pogut produir-se. Pot ser per multitud de motius com una URL mal formada, un error en la xarxa, host o port incorrecte, base de dades no existent, usuari i contrasenya no vàlids, etc.
- **SQLTimeoutException**: S'ha superat el LoginTimeout sense rebre resposta del servidor.

5.4 Classe Connection

Un objecte **java.sql.Connection** representa una sessió de connexió amb una base de dades. Una aplicació pot tindre tantes connexions com necessite, ja siga amb una o diverses bases de dades.

El mètode més rellevant és **createStatement()** que retorna un objecte **Statement** associat a aquesta connexió que permet executar sentències SQL. El mètode **createStatement()** pot llançar excepcions de tipus **SQLException**.

```
Statement stmt = conn.createStatement();
```

Quan ja no la necessitem és aconsellable **tancar la connexió amb close()** per a alliberar recursos.

```
conn.close();
```

5.5 Classe Statement

Un objecte **java.sql.Statement** permet **executar sentències SQL en la base de dades** a través de la connexió amb la qual es va crear el **Statement** (veure apartat anterior). Els tres mètodes més comuns d'execució de sentències SQL són **executeQuery(...)**, **executeUpdate(...)** i **execute(...)**. Poden llançar excepcions de tipus **SQLException** i **SQLTimeoutException**.

- **ResultSet executeQuery(String sql)**: Executa la sentència **sql** indicada (de tipus **SELECT**). Retorna un objecte **ResultSet** amb les dades proporcionades pel servidor.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM vendedores");
```

- **int executeUpdate(String sql)**: Executa la sentència **sql** indicada (de tipus **DML** com per exemple **INSERT**, **UPDATE** o **DELETE**). Retorna un nombre de registres que han sigut inserits, modificats o eliminats.

```
int nr = stmt.executeUpdate("INSERT INTO vendedores VALUES (1,  
                          'Pedro Gil', '2017-04-11', 15000);")
```

Quan ja no ho necessitem és aconsellable **tancar el statement amb close()** per a alliberar recursos.

```
stmt.close();
```

5.6 Classe ResultSet

Un objecte **java.sql.ResultSet** conté un conjunt de resultats (dades) obtinguts després d'executar una sentència SQL, normalment de tipus SELECT. És una **estructura de dades en forma de taula amb registres (files)** que podem recórrer per a accedir a la informació dels seus **campes (columnes)**.

ResultSet utilitza internament un cursor que apunta al 'registre actual' sobre el qual podem operar. Inicialment aquest cursor està situat abans de la primera fila i disposem de diversos mètodes per a desplaçar el cursor. El més comú és next():

- **boolean next():** Mou el cursor al següent registre. Retorna true si va anar possible i false en cas contrari (si ja arribem al final de la taula).

Alguns dels mètodes per a obtenir les dades del registre actual són:

String getString(String columnLabel): Retorna una dada String de la columna indicada pel seu nom. Per exemple: `rs.getString("nom")`

String getString(int columnIndex): Retorna una dada String de la columna indicada pel seu nom. La primera columna és la 1, no la zero. Per exemple: `rs.getString(2)`

Existeixen mètodes anàlegs als anteriors per a obtenir valors de tipus int, long, float, double, boolean, Date, Time, Array, etc. Poden consultar-se tots en la [documentació oficial de Java](#).

- | | |
|---|--|
| ● int getInt(String columnLabel) | int getInt(int columnIndex) |
| ● double getDouble(String columnLabel) | double getDouble(int columnIndex) |
| ● boolean getBoolean(String columnLabel) | boolean getBoolean(int columnIndex) |
| ● Date getDate(String columnLabel) | int getDate(int columnIndex) |
| ● etc. | |

Més endavant veurem com es realitza la modificació i inserció de dades.

Tots aquests mètodes poden llançar una **SQLException**.

Vegem un exemple de com recórrer un ResultSet anomenat rs i mostrar-lo per pantalla:

```
while(rs.next()) {  
    int id          = rs.getInt("id");  
    String nom      = rs.getString("nom");  
    Date data       = rs.getDate("data_ingrés");  
    float salari    = rs.getFloat("salari");  
    System.out.println(id + " " + nom + " " + data + " " + salari);  
}
```

5.7 Exemple complet

Vegem un exemple complet de connexió i accés a una base de dades utilitzant tots els elements esmentats en aquest apartat.

```
try {
    // Carreguem la classe que implementa el Driver
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance();

    // Creem una nova connexió a la base de dades 'prova'
    String url = "jdbc:mysql://localhost:3306/prova?serverTimezone=UTC";
    Connection conn = DriverManager.getConnection(url, "root", "");

    // Obtenim un Statement de la connexió
    Statement st = conn.createStatement();

    // Executem una consulta SELECT per a obtenir la taula venedors
    String sql = "SELECT * FROM venedors";
    ResultSet rs = st.executeQuery(sql);

    // Recorrem tot el ResultSet i mostrem les seues dades
    while(rs.next()) {
        int id      = rs.getInt("id");
        String nom   = rs.getString("nom");
        Date data    = rs.getDate("data_ingresse");
        float salari = rs.getFloat("salari");
        System.out.println(id + " " + nom + " " + data + " " + salari);
    }

    // Tanquem el statement i la connexió
    st.close();
    conn.close();

} catch (SQLException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
```

6. NAVEGABILIDAD I CONCURRÈNCIA

Quan invoquem a `createStatement()` **sense arguments**, com hem vist anteriorment, en executar sentències SQL obtindrem un **ResultSet per defecte en el qual el cursor només pot moure's cap avant i les dades són de només lectura**. A vegades això no és suficient i necessitem major funcionalitat.

Per això el mètode `createStatement()` està sobrecarregat (existeixen diverses versions d'aquest mètode) la qual cosa ens permet invocar-ho amb arguments en els quals podem especificar el funcionament.

- **Statement `createStatement(int resultSetType, int resultSetConcurrency)`**: Retorna un objecte Statement els objectes del qual ResultSet seran del tipus i concurrència especificats. Els valors vàlids són constants definides en ResultSet.

L'argument **resultSetType** indica el tipus de ResultSet:

ResultSet.TYPE_FORWARD_ONLY: ResultSet per defecte, forward-only i no-actualitzable.

- Només permet moviment cap avant amb `next()`.
- Les seues dades NO s'actualitzen. És a dir, no reflecteix*rá canvis produïts en la base de dades. Conté una instantània del moment en el qual es va realitzar la consulta.

ResultSet.TYPE_SCROLL_INSENSITIVE: ResultSet desplaçable i no actualitzable.

- Permetix llibertat de moviment del cursor amb altres mètodes com `first()`, `previous()`, `last()`, etc. a més de `next()`.
- Les seues dades NO s'actualitzen, com en el cas anterior.

ResultSet.TYPE_SCROLL_SENSITIVE: ResultSet desplaçable i actualitzable.

- Permet llibertat de moviments del cursor, com en el cas anterior.
- Les seues dades SÍ QUE s'actualitzen. És a dir, mentre el ResultSet estiga obert s'actualitzarà automàticament amb els canvis produïts en la base de dades. Això pot succeir fins i tot mentre s'està recorrent el ResultSet, la qual cosa pot ser convenient o contraproductiu segons el cas.

L'argument **resultSetConcurrency** indica la concurrència del ResultSet:

- **ResultSet.CONCUR_READ_ONLY**: Només lectura. És el valor per defecte.
- **ResultSet.CONCUR_UPDATABLE**: Permet modificar les dades emmagatzemades en el ResultSet per a després aplicar els canvis sobre la base de dades (més endavant es veurà com).

El ResultSet per defecte que s'obté amb `createStatement()` sense arguments és el mateix que amb `createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY)`.

7. CONSULTES

7.1 Navegació d'un ResultSet

Com ja s'ha vist, en un objecte *ResultSet* es troben els resultats de l'execució d'una sentència SQL. Per tant, un objecte *ResultSet* conté les files que satisfan les condicions d'una sentència SQL, i ofereix mètodes de navegació pels registres com `next()` que desplaça el cursor al següent registre del *ResultSet*.

A més d'aquest mètode de desplaçament bàsic, existeixen uns altres de desplaçament lliure que podrem utilitzar sempre que el *ResultSet* siga de tipus *ResultSet.TYPE_SCROLL_INSENSITIVE* o *ResultSet.TYPE_SCROLL_SENSITIVE* com s'ha dit abans.

Alguns d'aquests mètodes són:

- **void beforeFirst():** Mou el cursor abans de la primera fila.
- **boolean first():** Mou el cursor a la primera fila.
- **boolean next():** Mou el cursor a la següent fila. Permés en tots els tipus de *ResultSet*.
- **boolean previous():** Mou el cursor a la fila anterior.
- **boolean last():** Mou el cursor a l'última fila.
- **void afterLast():** Moure el cursor després de l'última fila.
- **boolean absolute(int row):** Posiciona el cursor en el número de registre indicat. Cal tindre en compte que el primer registre és el 1, no el zero. Per exemple `absolute(7)` desplaçarà el cursor al seté registre. Si valor és negatiu es posiciona en el número de registre indicat però començant a comptar des del final (l'últim és el -1). Per exemple si té 10 registres i cridem `absolute(-2)` es desplaçarà al registre núm. 9.
- **boolean relative(int registres):** Desplaça el cursor un nombre relatiu de registres, que pot ser positiu o negatiu. Per exemple si el cursor està en el registre 5 i cridem a `relative(10)` es desplaçarà al registre 15. Si després cridem a `relative(-4)` es desplaçarà al registre 11.

Els mètodes que retornen un tipus boolean retornaran 'true' si ha sigut possible moure el cursor a un registre vàlid, i 'false' en cas contrari, per exemple si no té cap registre o hem saltat a un número de registre que no existeix.

Tots aquests mètodes poden produir una excepció de tipus *SQLException*.

També existeixen altres mètodes relacionats amb la posició del cursor.

- **int getRow():** Retorna el número de registre actual. Zero si no hi ha registre actual.
- **boolean isBeforeFirst():** Retorna 'true' si el cursor està abans del primer registre.
- **boolean isFirst():** Retorna 'true' si el cursor està en el primer registre.
- **boolean isLast():** Retorna 'true' si el cursor està en l'últim registre.
- **boolean isAfterLast():** Retorna 'true' si el cursor està després de l'últim registre.

7.2 Obtenint dades del ResultSet

Els mètodes *getXXX()* ofereixen els mitjans per a recuperar els valors de les columnes (camps) de la fila (registre) actual del *ResultSet*. No és necessari que les columnes siguin obtingudes utilitzant un ordre determinat.

Per a designar una columna podem utilitzar el seu nom o bé el seu número (començant per 1).

Per exemple si la segona columna d'un objecte *ResultSet* es diu "títol" i emmagatzema dades de tipus *String*, es podrà recuperar el seu valor de les dues formes següents:

```
// rs és un objecte ResultSet
String valor = rs.getString(2);
String valor = rs.getString("títol");
```

És important tindre en compte que les columnes es numeren d'esquerra a dreta i que la primera és la número 1, no la zero. També que les columnes no són case sensitive, és a dir, no distingeixen entre majúscules i minúscules.



La informació referent a les columnes d'un *ResultSet* es pot obtenir cridant al **mètode *getMetaData()*** que retornarà un objecte *ResultSetMetaData* que contindrà el número, tipus i propietats de les columnes del *ResultSet*.

Si coneixem el nom d'una columna, però no el seu índex, el mètode *findColumn()* pot ser utilitzat per a obtenir el número de columna, passant-li com a argument un objecte *String* que siga el nom de la columna corresponent, aquest mètode ens retornarà un enter que serà l'índex corresponent a la columna.

7.3 Tipus de dades i conversions

Quan es llança un mètode *getXXX()* determinat sobre un objecte *ResultSet* per a obtenir el valor d'un camp del registre actual, el driver JDBC converteix la dada que es vol recuperar al tipus Java especificat i llavors retorna un valor Java adequat. Per exemple si utilitzem el mètode *getString()* i el tipus de la dada en la base de dades és *VARCHAR*, el driver JDBC convertirà la dada *VARCHAR* de tipus SQL a un objecte *String* de Java.

Alguna cosa semblant succeeix amb altres tipus de dades SQL com per exemple *DATE*. Podrem accedir a ell tant amb *getDate()* com amb *getString()*. La diferència és que el primer retornarà un objecte Java de tipus *Date* i el segon retornarà un *String*.

Sempre que siga possible el driver JDBC convertirà el tipus de dada emmagatzemada en la base de dades al tipus sol·licitat pel mètode *getXXX()*, però hi ha conversions que no es poden realitzar i llançaran una excepció, com per exemple si intentem fer un *getInt()* sobre un camp que no conté un valor numèric.

8. MODIFICACIÓ

Per a poder modificar les dades que conté un *ResultSet* necessitem un *ResultSet* de tipus modificable. Per a això hem d'utilitzar la constant *ResultSet.CONCUR_UPDATABLE* en cridar al mètode *createStatement()* com s'ha vist abans.

Per a modificar els valors d'un registre existent s'utilitzen una sèrie de mètodes *updateXXX()* de *ResultSet*. Les XXX indiquen el tipus de la dada i hi ha tants diferents com succeeix amb els mètodes *getXXX()* d'aquesta mateixa interfície: ***updateString()*, *updateInt()*, *updateDouble()*, *updateDate()*, etc.**

La diferència és que **els mètodes *updateXXX()* necessiten dos arguments:**

- **La columna que desitgem actualitzar** (pel seu nom o pel seu número de columna).
- **El valor que volem emmagatzemar** en aquesta columna (del tipus que siga).

Per exemple per a modificar el camp 'edat' emmagatzemant el sencer 28 caldria cridar al següent mètode, suposant que *rs* és un objecte *ResultSet*:

```
rs.updateInt("edat", 28);
```

També podria fer-se de la següent manera, suposant que la columna "edat" és la segona:

```
rs.updateInt(2, 28);
```

Els mètodes *updateXXX()* no retornen cap valor (són de tipus *void*). Si es produeix algun error es llançarà una *SQLException*.

Posteriorment cal **cridar a *updateRow()* perquè els canvis realitzats s'apliquen sobre la base de dades**. El Driver JDBC s'encarregarà d'executar les sentències SQL necessàries. Aquesta és una característica molt potent ja que ens facilita enormement la tasca de modificar les dades d'una base de dades. Aquest mètode retorna *void*.

En resum, el procés per a realitzar la modificació d'una fila d'un *ResultSet* és el següent:

1. **Desplacem el cursor al registre** que volem modificar.
2. Cridem a tots els mètodes ***updateXXX(...)*** que necessitem.
3. Cridem a *updateRow()* perquè els canvis s'apliquen a la base de dades.

És important entendre que **cal cridar a *updateRow()* abans de desplaçar el cursor**. Si desplacem el cursor abans de cridar a *updateRow()*, es perdran els canvis.

Si volem **cancel·lar les modificacions d'un registre del *ResultSet*** podem cridar a ***cancelRowUpdates()***, que cancel·la totes les modificacions realitzades sobre el registre actual.

Si ja hem anomenat a *updateRow()* el mètode *cancelRowUpdates()* no tindrà cap efecte.

El següent codi d'exemple mostra com modificar el camp 'direcció' de l'últim registre d'un `ResultSet` que conté el resultat d'un `SELECT` sobre la taula de clients. Suposarem que `conn` és un objecte `Connection` previamente creat:

```
// Creem un Statement scrollable i modificable
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                       ResultSet.CONCUR_UPDATABLE);

// Executem un SELECT i obtenim la taula clients en un ResultSet
String sql = "SELECT * FROM clients";
ResultSet rs = stmt.executeQuery(sql);

// Anem a l'últim registre, el modifiquem i actualitzem la base de dades
rs.last();
rs.updateString("direccio", "C/ Pepe Ciges, 3");
rs.updateRow();
```

9. INSERCIÓ

Per a inserir nous registres necessitarem utilitzar, almenys, aquests dos mètodes:

- **void moveToInsertRow():** Desplaça el cursor al 'registre d'inserció'. És un registre especial utilitzat per a inserir nous registres en el ResultSet. Posteriorment tindrem que cridar als mètodes updateXXX() ja coneguts per a establir els valors del registre d'inserció. Per a finalitzar cal cridar a insertRow().
- **void insertRow():** Inserida el 'registre d'inserció' en el ResultSet, passant a ser un registre normal més, i també l'insereix en la base de dades.

El següent codi inserix un nou registre en la taula 'clients'. Suposarem que `conn` és un objecte Connection previament creat:

```
// Creem un Statement scrollable i modificable
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                       ResultSet.CONCUR_UPDATABLE);

// Executem un SELECT i obtenim la taula clients en un ResultSet
String sql = "SELECT * FROM clients";
ResultSet rs = stmt.executeQuery(sql);

// Creem un nou registre i l'inserim
rs.moveToInsertRow();
rs.updateString(2, "Killy Lopez");
rs.updateString(3, "Wall Street 3674");
rs.insertRow();
```

Els camps el valor dels quals no s'haja establert amb updateXXX() tindran un valor NULL. Si en la base de dades aquest camp no està configurat per a admetre nuls es produirà una SQLException.

Després d'inserir el nostre nou registre en l'objecte ResultSet podrem tornar a l'anterior posició en la qual es trobava el cursor (abans d'invocar moveToInsertRow()) cridant al mètode moveToCurrentRow(). Aquest mètode només es pot utilitzar en combinació amb moveToInsertRow().

10. ESBORRAT

Per a eliminar un registre només cal desplaçar el cursor al registre desitjat i cridar al mètode:

- **void deleteRow()**: Elimina el registre actual del ResultSet i també de la base de dades.

El següent codi esborra el tercer registre de la taula 'clients':

```
// Creem un Statement scrollable i modificable
Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                       ResultSet.CONCUR_UPDATABLE);

// Executem un SELECT i obtenim la taula clients en un ResultSet
String sql = "SELECT * FROM clients";
ResultSet rs = stmt.executeQuery(sql);

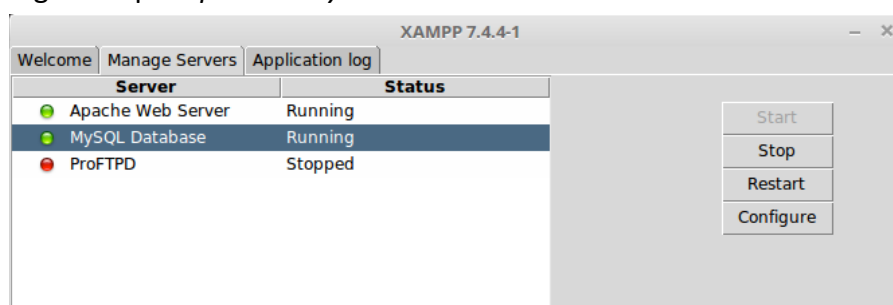
// Desplacem el cursor al tercer registre
rs.absolute(3)
rs.deleteRow();
```

11. PROGRAMA GESTOR DE CLIENTS

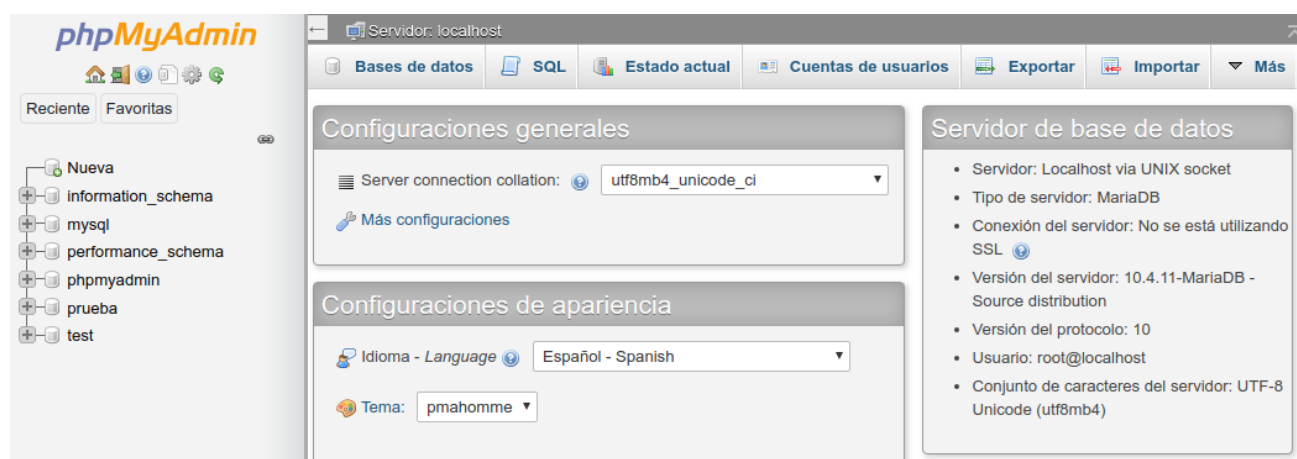
Vegem un exemple d'un programa de gestió de clients d'una botiga, amb interfície gràfica d'usuari i accés a base de dades MySQL. Aquest programa permetrà consultar i modificar la informació dels clients així com donar-los d'alta d'i de baixa. El codi del projecte i la base de dades el trobareu a l'aula virtual.

En la base de dades tindrem una sola taula amb els camps *id*, *nom* i *direcció*. Cal destacar que el camp *id* és autoincremental, és a dir que el seu valor s'incrementarà en inserir un registre en la taula, per la qual cosa en crear nous registres no és necessari donar-li un valor.

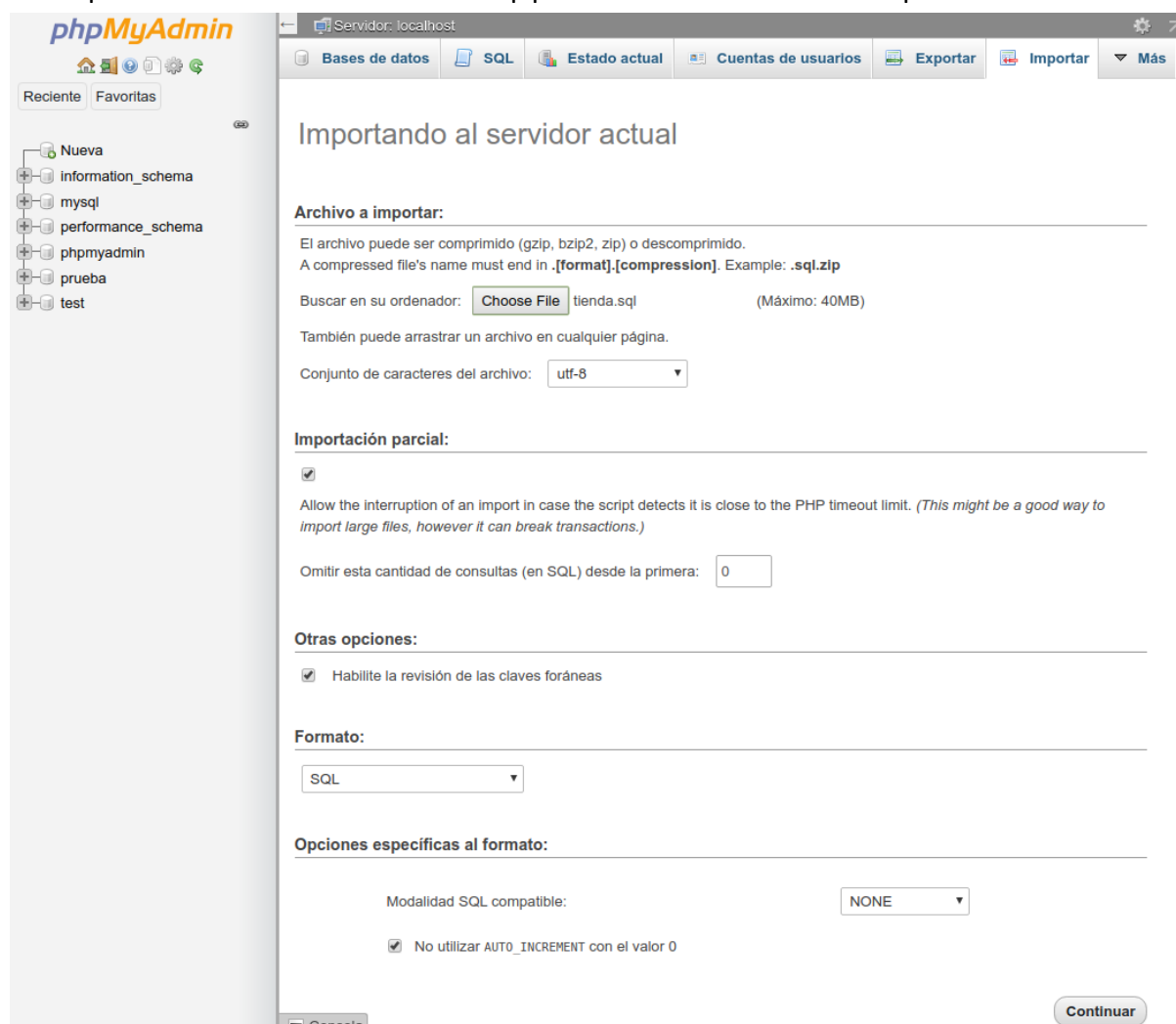
Primer necessitem importar la base de dades al nostre servidor MySQL. Obrim el panell de control XAMPP i ens assegurem que *Apache* i *MySQL* estan funcionant. En cas contrari els iniciem.



Obrim un navegador web, anem a <http://localhost> i accedim a phpMyAdmin. També podem accedir directament amb <http://localhost/phpmyadmin/>. Aquesta aplicació web allotjada en el nostre servidor Apache ens permet interactuar amb el servidor MySQL d'una forma senzilla.

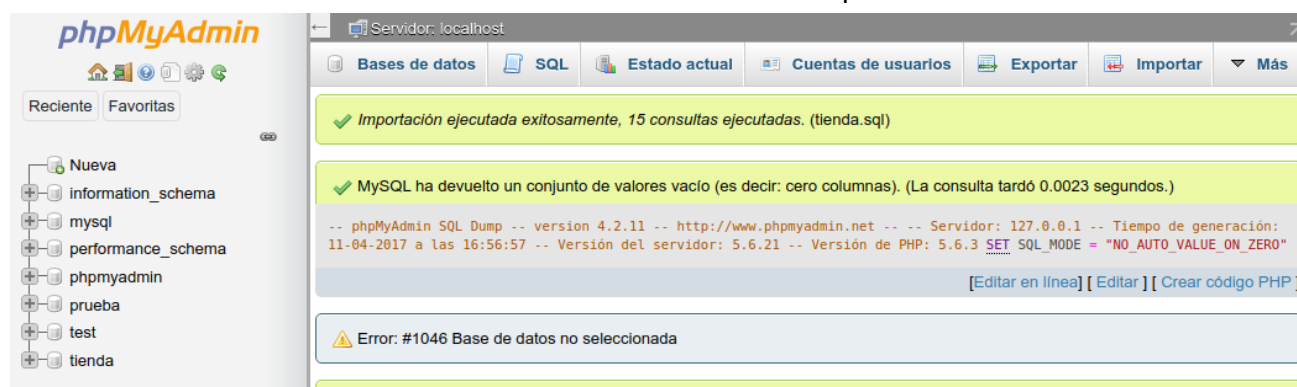


A continuació fem clic a 'Importar' i seleccionem l'arxiu 'botiga.sql' que podeu descarregar de l'aula virtual. Aquest arxiu conté les sentències sql per a crear la base de dades que necessitem.



The screenshot shows the phpMyAdmin interface for importing a file. The left sidebar shows a database tree with 'tienda' selected. The main area is titled 'Importando al servidor actual'. It includes sections for 'Archivo a importar' (with a 'Choose File' button and a file name 'tienda.sql'), 'Importación parcial' (with a checkbox for allowing interruption), 'Otras opciones' (with a checkbox for enabling foreign key checks), 'Formato' (set to 'SQL'), and 'Opciones específicas al formato' (with a checkbox for not using AUTO_INCREMENT with value 0). A 'Continuar' button is at the bottom right.

Fem clic a 'Continuar' i si tot va bé la base de dades 'tienda' s'importarà correctament.



The screenshot shows the phpMyAdmin interface after the import. The left sidebar shows the 'tienda' database selected. The main area displays a green success message: 'Importación ejecutada exitosamente, 15 consultas ejecutadas. (tienda.sql)'. Below it, another green message states: 'MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0023 segundos.)'. A console window at the bottom shows the SQL dump output, including the version and server information. A yellow error message at the bottom reads: 'Error: #1046 Base de datos no seleccionada'.

Ja tenim la base de dades en el nostre servidor MySQL.

El següent pas és desenvolupar l'aplicació. Haurem de crear un nou projecte Java i importar el JDBC connector com una llibreria JAR tal com s'ha explicat al principi d'aquesta unitat.

A continuació creem un JFrame anomenat **VentanaClientes.java** i dissenyem la interfície gràfica. Per exemple així:



El codi relacionat amb l'accés a la base de dades estarà en una classe diferent anomenada **BDClientes.java**. Aquesta classe contindrà diversos mètodes públics per a les diferents operacions que necessitarem utilitzar des de la interfície gràfica: connectar a la base de dades, consultar el llistat de clients, així com inserir, actualitzar i esborrar clients.

És a dir, la classe GestionClientes contindrà el codi de la interfície gràfica (JFrame) i utilitzarà la classe BDManager per a les operacions d'accés a la base de dades. Aquesta divisió de 'responsabilitats' facilita el desenvolupament i fa que el nostre codi siga més modular.

A l'aula virtual podeu trobar el codi a tall d'exemple.

12. A GRAÏMENTS

Anotacions actualitzades i adaptats al CEEDCV a partir de la següent documentació:

- [1] Anotacions Programació de José Antonio Díaz-Alejo. IES Camp de Morvedre.