6. Programació usant SQL

- Introducció
  - Introducció a la programació usant SQL
  - SQLJ versus JDBC
  - Avantatges/Inconvenients SQLJ versus JDBC
  - Desenvolupament en SQLJ versus JDBC
  - API JDBC / Drivers que la implementen
  - Tipus de drivers
- Programació amb JDBC

Introducció a la programació usant SQL

- Existeixen dos enfocaments per accedir a bases de dades SQL des de programes:
  - SQL estàtic. Les sentències SQL són fixes i les mateixes en qualsevol execució d'un programa (ex: SQLJ, SQL HOST, ...).
  - SQL dinàmic. Les sentències SQL es construeixen durant l'execució d'un programa, i poden variar d'una execució a una altra (ex: JDBC, ODBC, SQL/CLI, ...).
- Tots dos tenen avantatges i inconvenients.

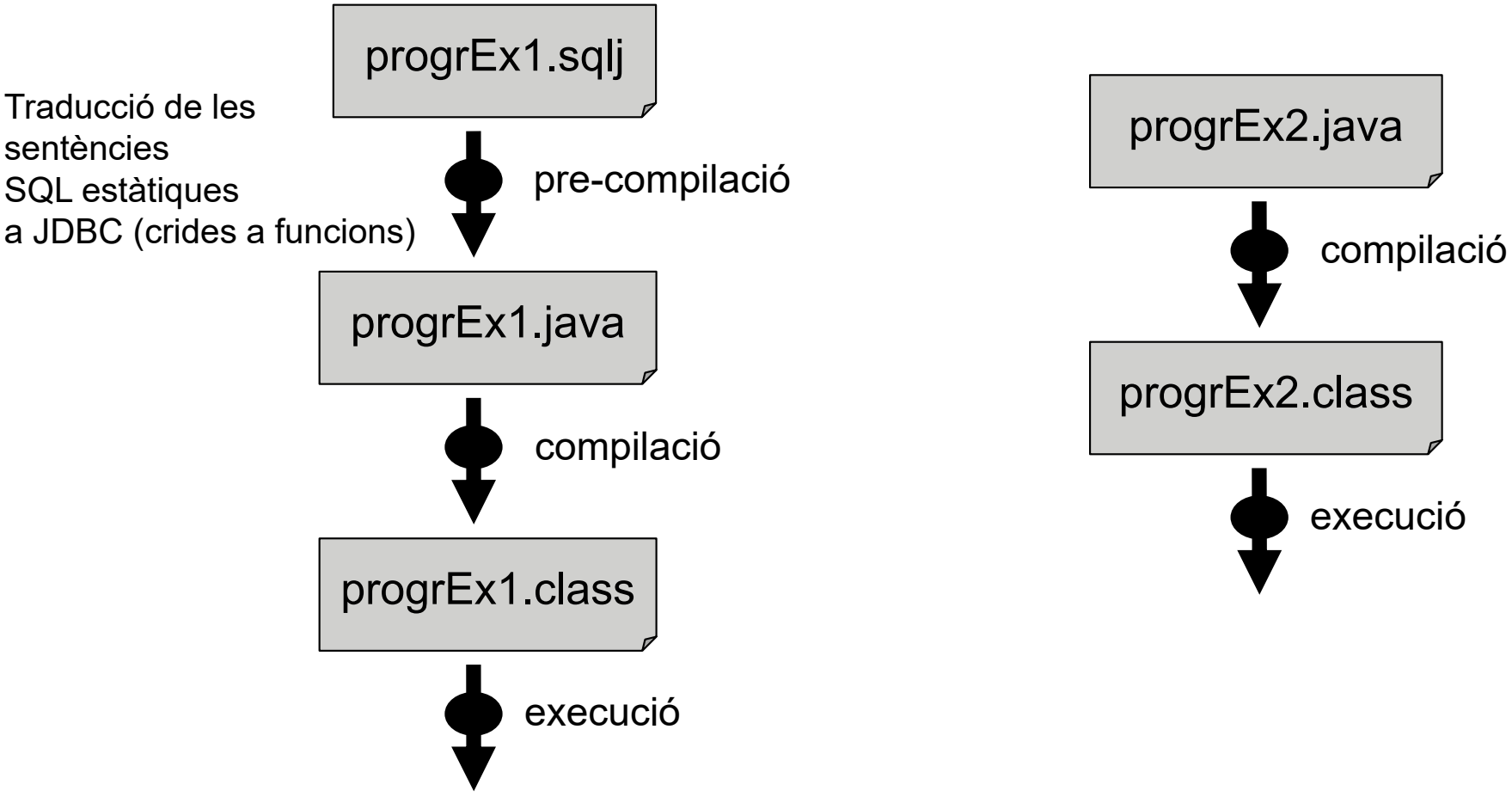
SQLJ versus JDBC

SQLJ	JDBC
Consisteix en incrustar sentències SQL estàtiques dins dels programes	Consisteix en accedir a les bases de dades mitjançant crides a funcions.
Cal una pre-compilació / traducció de les sentències SQL en sentències del llenguatge de programació	Totes les sentències, són sentències del llenguatge de programació.
Les sentències SQL es compilen durant la pre-compilació	Les sentències SQL es compilen durant l'execució
Les comprovacions amb l'esquema de la base de dades es fan durant la pre-compilació	Les comprovacions amb l'esquema de la base de dades es fan quant s'executa el programa

Avantatges/Inconvenients SQLJ versus JDBC

Avantatges/Inconvenients	SQLJ	JDBC
Portabilitat	Poc portable entre SGBD (cal recompilar)	Portable entre SGBD
Quantitat de línies de codi	Menys	Més
Possibilitat de decidir en temps d'execució quines sentències executar	No	Si
Facilitat de programació	Més	Menys
Eficiència	Més	Menys

Desenvolupament en SQLJ versus JDBC



En qualsevol dels casos per poder compilar el programa cal el driver del SGBD concret que es vol usar i que implementa l'API JDBC

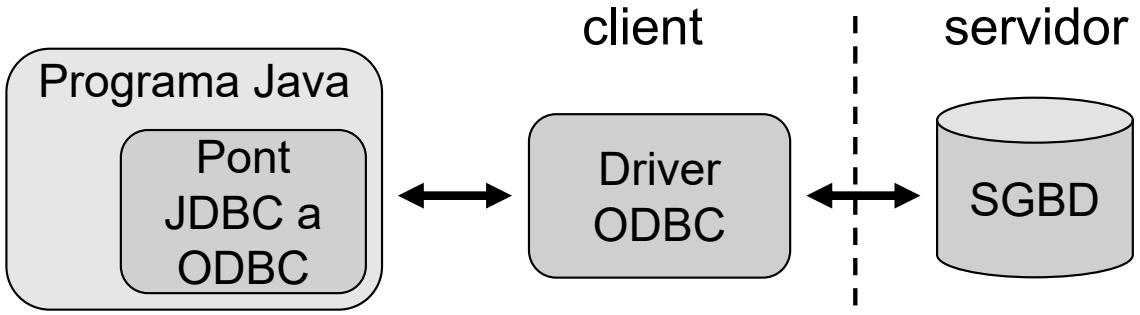
API JDBC / Drivers que la implementen

- JDBC és una API (Application Program Interface) que defineix els mètodes per la connexió i accés a bases de dades remotes des d'un programa escrit en Java.
- L'API JDBC és independent dels SGBDs dels diferents fabricants.
- Els mètodes s'implementen en drivers específics de cada SGBD.
- Un driver JDBC és un conjunt de llibreries dependents de l'SGBD que implementen els mètodes definits en l'API JDBC.

Tipus de Drivers

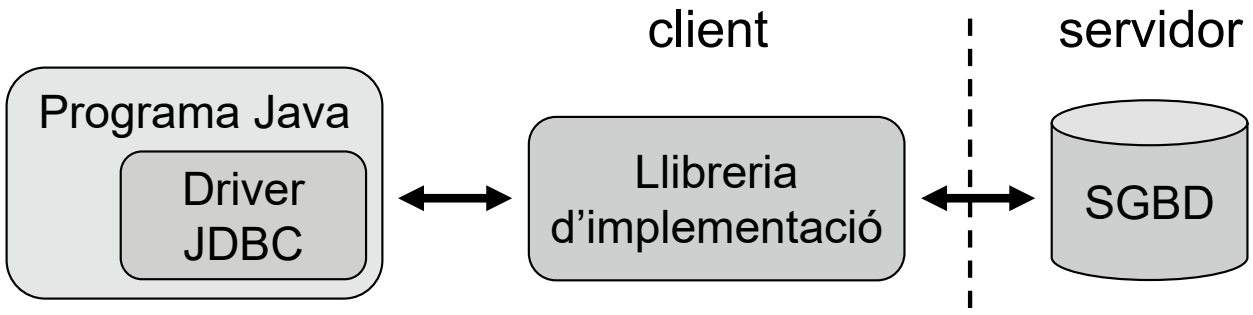
Tipus 1. Pont JDBC-ODBC

L'aplicació Java fa crides a mètodes JDBC, que són traduïdes a crides a mètodes ODBC mitjançant el pont JDBC-ODBC. El driver ODBC ha de ser específic per al SGBD concret al que volem accedir.



Tipus 2. Driver JDBC + llibreria d'implementació

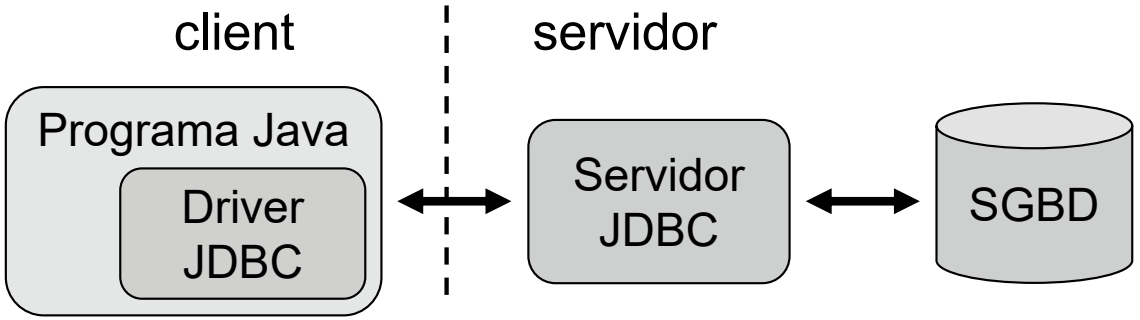
L'aplicació Java fa crides a mètodes JDBC, que són traduïdes a crides a una llibreria proporcionada pel fabricant del SGBD.



Tipus de Drivers

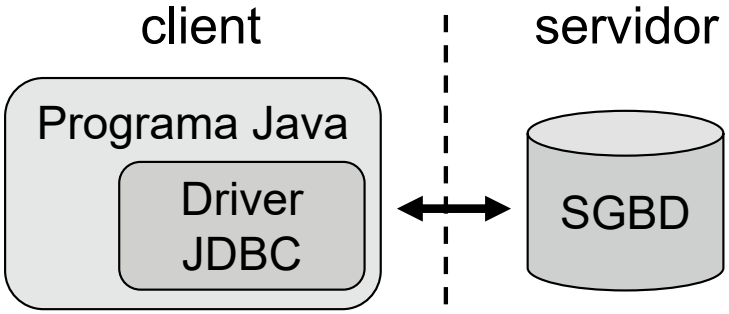
Tipus 3. Driver JDBC-protocol de xarxa

L'aplicació Java fa crides a mètodes JDBC, que el driver JDBC tradueix a un protocol genèric de xarxa que permet la comunicació amb un servidor JDBC que dóna accés al SGBD.

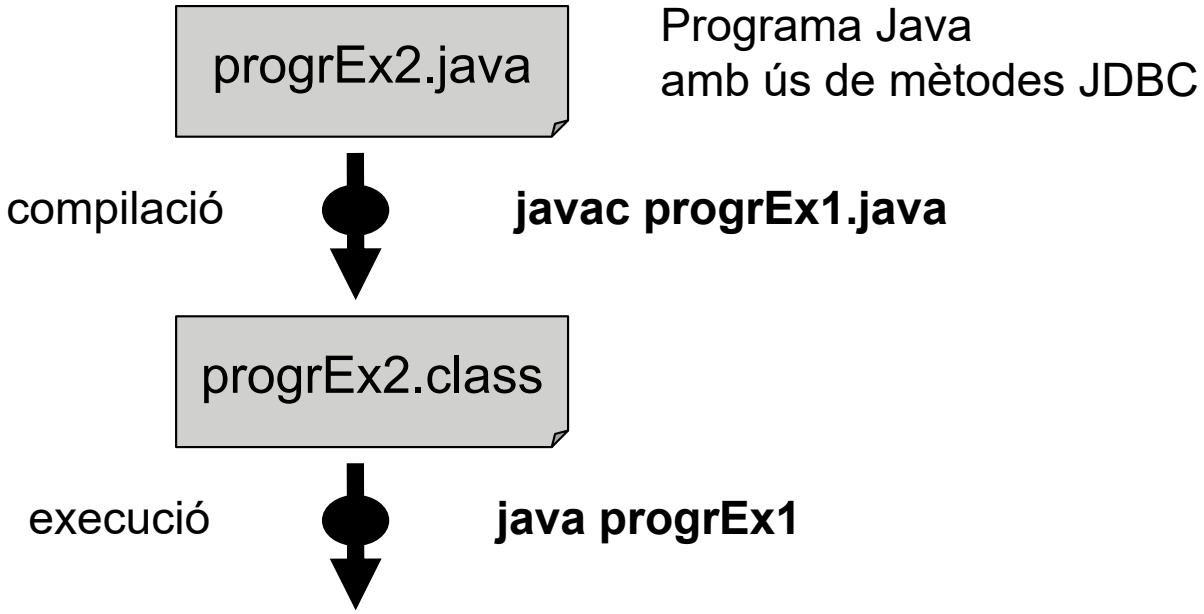


Tipus 4. Driver JDBC pur

L'aplicació Java fa crides a mètodes JDBC, que donen accés al SGBD. El driver és específic del SGBD.



Programació amb JDBC



Estructura d'un Programa amb JDBC

- Importar llibreries

```
import java.sql.*;
```

- Parts del programa:
  - Connexió amb la base de dades
  - Comunicació amb la base de dades (consultes, modificacions,..)
  - Tancament de la connexió amb la base de dades

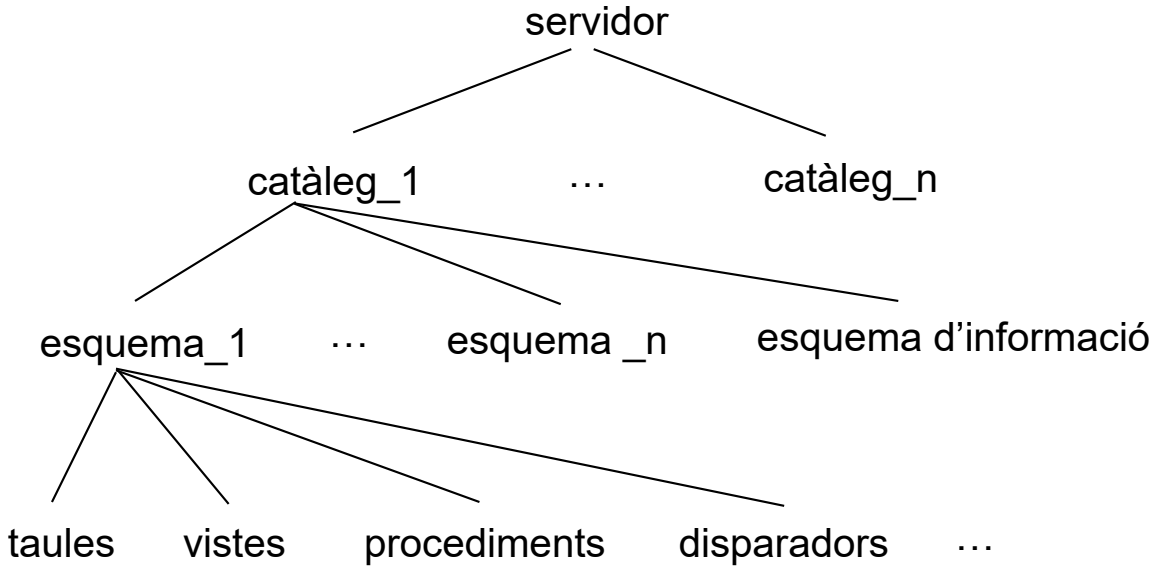
Connexió amb la Base de Dades

Cal usar el mètode *getConnection* de la classe *DriverManager*. Aquest mètode retorna un objecte de la classe *Connection* que representa la connexió amb la base de dades.

```
Connection c = DriverManager.getConnection("url", props);
```

La URL inclou:
- Adreça IP / Port del servidor
- Nom del catàleg

L'esquema al que es connecta és l'esquema per defecte.



Les propietats són entre altres el username i password de l'usuari que es vol connectar.

```
Properties props = new Properties();
props.setProperty("user", "elMeuUsername");
props.setProperty("password", "elMeuPassword");
```

Autocommit

L'autocommit s'ha d'establir amb el mètode *setAutoCommit* de la classe *Connection*.

```
c.setAutoCommit(boolean);
```

Atenció: Nosaltres sempre posarem el autocommit a false

Selecció de l'esquema

Un cop feta la connexió l'esquema que se selecciona és l'esquema per defecte.

Si es vol treballar amb un esquema que no és l'esquema per defecte, cal executar la sentència SQL corresponent de canvi d'esquema

```
set schema nomEsquema;
```

Connexió amb la Base de Dades

En el cas del SGBD PostgresSQL:

```
/* Username i password */
Properties props = new Properties();
props.setProperty("user", "elMeuUsername");
props.setProperty("password", "elMeuPassword");


/* En el cas que el servidor de PostgresSQL al que ens volem connectar
requereixi connexió SSL, cal afegir a les propietats que la connexió és SSL,
i el nom de la classe que ajudarà en aquesta connexió */
props.setProperty("ssl", "true");
props.setProperty("sslfactory","org.postgresql.ssl.NonValidatingFactory");

/* Connexió a l'esquema per defecte del catàleg nomCat */
Connection c = DriverManager.getConnection(
        "jdbc:postgresql://postgresfib.fib.upc.es:6433/nomCat",props);
c.setAutoCommit(false);


/* Canvi a un esquema diferent de l'esquema per defecte */
Statement s = c.createStatement ();
s.executeQuery ("set search_path to nomEsquema");
```

Comunicació amb la base de dades

Consisteix en accedir a les bases de dades mitjançant crides a mètodes. Les sentències SQL són en aquest cas paràmetres de tipus *String* que es passen a aquests mètodes.

Hi ha dos maneres de comunicar-se amb la base de dades: usant objectes de la classe *Statement* o bé usant objectes de la classe *PreparedStatement*.

Statement	PreparedStatement
Cada cop que s'executa una sentència es compila.	Una sentència només es compila un únic cop, i es pot executar tants cops com calgui.
Es crea amb el mètode de l'objecte *Connection* següent: `Statement createStatement()`	Es crea i es compila amb el mètode de l'objecte *Connection* següent: `PreparedStatement prepareStatement(String sql)`

Es compila i s'executa amb un dels dos mètodes de l'objecte *Statement* següents
`ResultSet executeQuery(String sql)`
`int executeUpdate(String sql)`
	S'executa amb un dels dos mètodes de l'objecte *PreparedStatement* següent:
	`ResultSet executeQuery()`
	`int executeUpdate()`

En qualsevol dels casos les consultes donen com a resultat un objecte de la classe *ResultSet* i les modificacions un enter que indica el nombre de tuples modificades.

Comunicació usant Statement: Consultes

Suposem la taula *socis*:

```
create table socis (dni char(9),
                    nom char(30) not null,
                    telefon char(9),
                    numVISA char(12),
                    ciutatResidencia char(20) default 'Barcelona',
                    primary key (dni));
```

Obtenir tots els socis que viuen a "*Barcelona*":

```
Statement s = c.createStatement ();
String cR = "Barcelona";
ResultSet r = s.executeQuery ("select dni,nom "+
                              "from socis "+
                              "where ciutatResidencia = '"+cR+"';");
// Tractar el resultat
s.close();
```

Comunicació usant Statement: Modificacions

Suposem la taula *socis*:

```
create table socis (dni char(9),
                    nom char(30) not null,
                    telefon char(9),
                    numVISA char(12),
                    ciutatResidencia char(20) default 'Barcelona',
                    primary key (dni));
```

Modificar la ciutat de residència del soci amb dni "*10*":

```
Statement s = c.createStatement ();
String dni = "10";
int numTuplesModificades = s.executeUpdate ("update socis "+
                                "set ciutatResidencia = 'Sort' "+
                                "where dni = '"+dni+"';");
if (numTuplesModificades == 0)
        System.out.println("El soci "+ dni +" no existeix");
s.close();
```

Comunicació usant PreparedStatement

Es tracta de sentències que s'executaran diverses vegades durant l'execució d'un programa. En la majoria dels casos estan parametritzades.

Els paràmetres s'especifiquen amb un signe "?" dins el *String* que forma la sentència.

Per donar valor a un paràmetre es fan servir mètodes *setXXX* que es poden invocar sobre objectes *PreparedStatement*.

```
void setXXX(int posicioParametre, XXX valor);
```

On XXX depèn del tipus del paràmetre, *posicioParametre* és el número d'ordre del paràmetre dins la sentència, i *valor* és el valor que hi volem assignar.

Concretament, amb els tipus que usem habitualment:

```
void setString(int posicioParametre, String valor);
void setInt(int posicioParametre, int valor);
void setNull(int posicioParametre, Types.Integer);
void setNull(int posicioParametre, Types.Char);
```

Comunicació usant PreparedStatement: Consultes

Suposem la taula *inscripcions*:

```
create table inscripcions (tipusActivitat char(15),
                           dataInici integer,
                           dni char(9),
                           ...
                           foreign key (dni) references socis);
```

Obtenir quantes inscripcions té cada soci de "*Barcelona*":

```
PreparedStatement ps = c.prepareStatement("select count(*) as numInscr "+
                                          "from inscripcions i "+
                                          "where i.dni = ? ;");

ResultSet rs = null;
String dni= null;
String nom= null;
while (//quedin socis a r//) {
                        dni =    // obtenir el dni del següent soci de r
                        ps.setString(1,dni);
                        rs = ps.executeQuery();
                        // Tractar el resultat
                        }
```

Comunicació usant PreparedStatement: Modificacions

Suposem la taula *tipus*:

```
create table tipus (tipus char(15),
                    numMaxInscripcions integer not null,
                    primary key (tipus));
```

Eliminar els tipus d'activitats "titelles" i "aniversari":

```
PreparedStatement ps = c.prepareStatement("delete from tipus "+
                                          "where tipus = ?;");
ps.setString(1,"titelles");
int numFilesEsborrades = ps.executeUpdate();
if (numFilesEsborrades == 0)
        System.out.println ("El tipus titelles no existeix");
ps.setString(1,"aniversari");
numFilesEsborrades = ps.executeUpdate();
if (numFilesEsborrades == 0)
        System.out.println ("El tipus aniversari no existeix");
```

Tractament de resultats: ResultSets

Un *ResultSet* es comporta com un cursor. Quan es crea el *ResultSet*, el cursor es troba en una fila imaginaria anterior a la primera.

- El mètode ***next*** - Avança el cursor a la fila següent. Retorna cert si s'ha pogut avançar, o fals si no hi ha més files.

```
boolean next()
```

- Els mètodes ***getXXX*** - Permeten recuperar els valors de la fila actual del cursor.

```
XXX getXXX(int numColumna)
```
recupera el valor de la fila situat en la posició numColumna
```
XXX getXXX(String nomColumna)
```
recupera el valor de la columna que té per nom nomColumna

	INTEGER	REAL	CHAR	....
getInt	**XX**	X	X	
getFloat	X	**XX**	X	
getString	X	X	**XX**	
....				

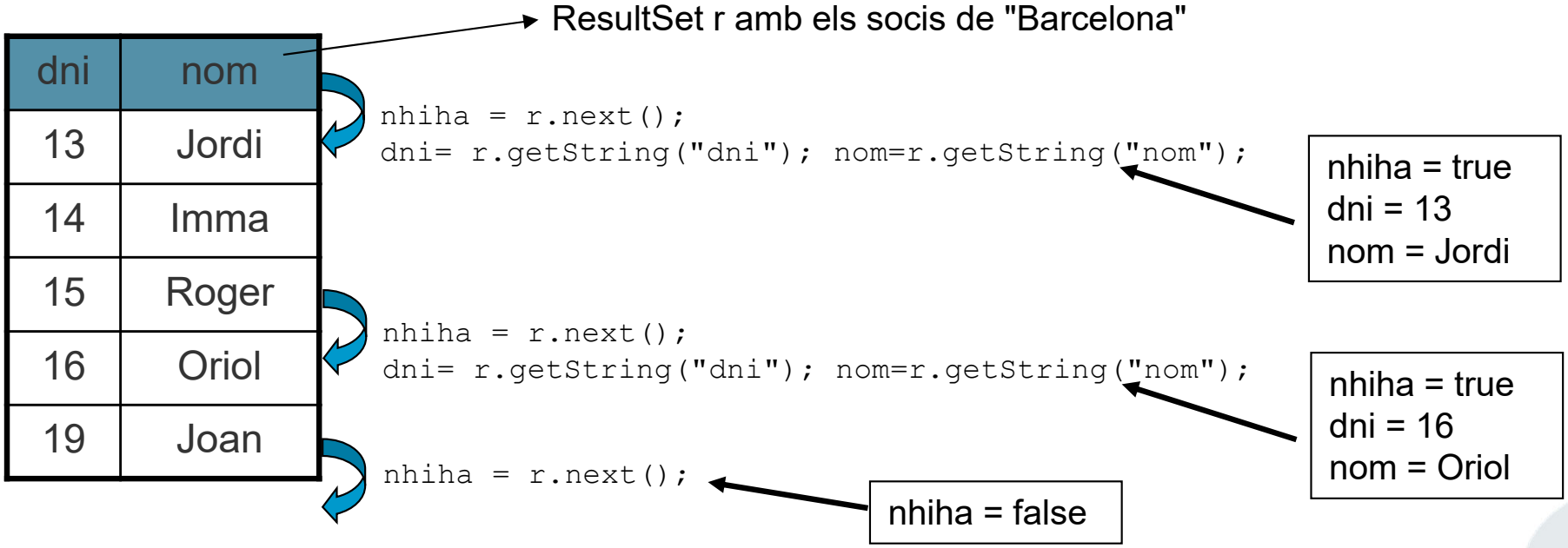
XX - el mètode és el millor per obtenir el valor
X - el mètode permet obtenir el valor

Exemple ResultSets

Suposem la consulta dels socis de "Barcelona":

```
ResultSet r = s.executeQuery ("select dni,nom "+
                              "from socis "+
                              "where ciutatResidencia = 'Barcelona';");
```

Exemple d'us de les operacions sobre un *ResultSet*:



Resultats amb valors nuls

El tractament és diferent depenent del tipus que retorni els mètodes *getXXX*:

- Comparació amb el valor ***nul*** - Si els mètodes *getXXX* retornen un tipus classe de Java, llavors la variable d'aquest tipus pren el valor *null*.

  Per exemple *getString* que retorna un valor de tipus *String*.

```
String telf = r.getString("telefon");
if (telf == null) ...;
```

- Us del mètode ***wasNull*** - Si els mètodes *getXXX* retornen un tipus simple Java, i més concretament per valors numèrics, llavors per diferenciar entre el valor 0 i el valor nul, cal invocar immediatament després el mètode *wasnull* sobre l'objecte *ResultSet* .

  Per exemple *getInt* que retorna un valor de tipus *int*.

```
int pr = r.getInt("preu");
if (r.wasNull())  ...;
```

Ús dels ResultSet: Nuls en tipus classe de Java

```
ResultSet r = s.executeQuery ("select dni, nom, numVISA "+
                                  "from socis "+
                                  "where ciutatResidencia = 'Sant Just';");
int quants = 0;
String dni;
String nom;
String nVISA;
System.out.println ("Socis de Sant Just:");
while (r.next())
    { quants = quants + 1;
    dni = r.getString("dni");
    nom = r.getString("nom");
    nVISA = r.getString("numVISA");
    System.out.print ("El soci amb dni "+dni+" i nom "+nom);
    if (nVISA == null)
          { System.out.println (" no te VISA"); }
       else { System.out.println (" te VISA "+nVISA); }
     }
r.close();
if (quants == 0)  { System.out.println ("No hi ha cap soci de Sant Just"); }
```

Ús dels ResultSet: Valors nuls en tipus Java simples

```
ResultSet r = s.executeQuery ("select tipusActivitat, dataInici, preu "+
                              "from activitats;");
int quants = 0;
String tAct= null;
int dIni= 0;
int pr= 0;
System.out.println ("Activitats sense preu assignat:");
while (r.next())
    { pr = r.getInt(3);
      if (r.wasNull()) {
    quants = quants + 1;
    tAct = r.getString(1);
    dIni = r.getInt(2);
         System.out.print ("L'activitat de tipus "+tAct+
                                  " i data d'inici "+ dIni +
                                  " no te preu assignat"); }
     }
r.close();
if (quants == 0) {
                  System.out.println ("No hi ha cap activitat sense preu");
                  }
   else {System.out.println("En total n'hi ha " + quants);}
```

Ús de Statement and PreparedStatement

- En un moment determinat només pot existir una instància de *ResultSet* per cada instància de *Statement* o *PreparedStatement*.

- És a dir, si en un cert instant necessitem accedir al resultats de dues consultes, cal que aquests resultats s'hagin obtingut en dues instàncies de *ResultSet* mitjançant dues instàncies de Statements i/o PreparedStatement.

- Quan un *Statement* o *PreparedStatement* no s'han d'usar més, es convenient de tancar-les.

```
Statement s = c.createStatement ();
PreparedStatement ps = c.prepareStatement("select i.dni count(*)"+
                                          "from inscripcions i "+
                                          "where i.dni = ? ;");
....
s.close();
ps.close();
```

Ús de ResultSet

- No es pot accedir als valors d'un *ResultSet* sense abans haver invocat el mètode *next*.

- Es convenient que quan un *ResultSet* deixa de ser necessari, es tanqui.

```
ResultSet r = s.executeQuery ("select tipusActivitat, " +
                                                "dataInici, preu " +
                                                "from activitats;");
....
r.close();
```

- Un cop es fa una altra consulta amb la instància de *Statement* usada per crear un *ResultSet*, el *ResultSet* que està obert és tancat automàticament.

Tancament de la Connexió

- Quan sigui necessari cal fer *commit* o *rollback* del que s'ha fet fins al moment. Cal usar operacions definides sobre l'objecte de la classe *Connection* que representa la connexió amb la base de dades.

```
c.commit();        /* on c és la connexió
```

o bé

```
c.rollback();      /* on c és la connexió
```

- Finalment cal fer el tancament de la connexió. Cal usar l'operació *close* definida sobre l'objecte de la classe *Connection* que representa la connexió amb la base de dades.

```
c.close();
```

Excepcions

Els errors es donen en forma d'excepcions de la classe SQLException

Classe Excepció	Pot ser produït per
SQLException	Connexió amb la base de dades Comunicació amb la base de dades Tancament de la connexió

```
public class progrEx1 {
    public static void main (String[] args) {
        try{
                // Connexió a la base de dades
                // Consultes-modificacions
                // Desconnexió
            }
        catch (SQLException se) {
            System.out.println ("Error connexió o accés a la base de dades");}
}}
```

Mètodes sobre una SQLException

Suposant que *objSE* és una instància de l'excepció *SQLException*:

Mètode	Informació que retorna
objSE.getSQLState()	"00000" - La sentència s'ha executat amb èxit. >"02XXX" - Error greu que ha impedit l'execució correcta de la sentència. "01XXX" - Situació d'avís.
objSE.getMessage()	Retorna un String que explica breument l'error que s'ha produït

Per exemple:

"23XXX" - Violació de restriccions d'integritat
"08XXX" - Excepcions relacionades amb la connexió.
"28000" - Autorització d'accés invalida.

Gestió d'errors

En el cas del SQLState en el SGBD Postgres:

```
try {
      Statement s = c.createStatement();
      int num = s.executeUpdate("insert into activitats "+
                                "values ('cine', 124, 'Dune', "+
                                "10, 17, 'Melies', 'Barcelona');");
     }
catch (SQLException se)
     {
     if (se.getSQLState().equals("23505"))
        System.out.println("Aquesta activitat ja existeix");
        else if (se.getSQLState().equals("23503"))
                System.out.println("El tipus o l'organitzador no existeixen");
            else System.out.println(se.getSQLState() +" "+ se.getMessage());
     }
```

Com saber els codis d'error (SQLState) per cada error que es pot produir?

- Provocar aquesta violació en l'editor SQL i mirar el codi d'error que ens retorna l'editor.
- Consultar la pàgina web de laboratori (codis d'error SQLState).

Resum JDBC

	Consultes	Modificacions (insert, delete, update)
No hi ha resultats	`rs.next();` `false`	`ps.executeUpdate();` `0`
Error en una sentència	`se.getSQLState();` `se.getMessage();`	`se.getSQLState();` `se.getMessage();`