



Les concepts objet en Java

Benoît Roche
@rocheb83

Penser objet et maîtriser son code ...

1




PRÉSENTATION


Benoît Roche / @rocheb83

2

2



PRÉSENTATION



Objet du cours :

Ce cours a pour objectif d'apprendre les bases de l'accès aux données en JAVA à l'aide de l'interface de programmation JDBC


Pré requis :

Connaissance du langage Java, de la POO et de l'IDE Eclipse.


Benoît Roche / @rocheb83

3

3



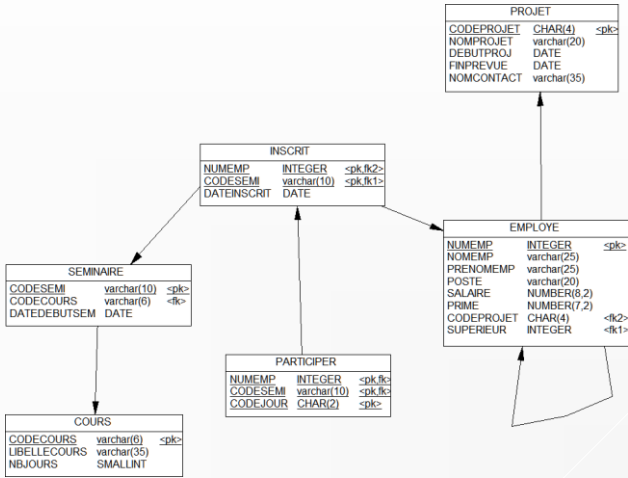
PRÉSENTATION



La base exemple :

On reste sur la BD du cours SQL

Environnements : Oracle et Mysql





```
graph TD
    SEMINAIRE --> INSCRIT
    INSCRIT --> EMPLOYE
    INSCRIT --> PARTICIPER
    SEMINAIRE --> COURS
    EMPLOYE --> PROJET
```

Detailed description of the ER diagram: The diagram shows six tables. SEMINAIRE (CODESEMI varchar(10) pk, CODECOURS varchar(6) fk, DATEDEBUTSEM DATE) is linked to INSCRIT (NUMEMP INTEGER pk/fk2, CODESEMI varchar(10) pk/fk1, DATEINSCRIT DATE) and COURS (CODECOURS varchar(6) pk, LIBELLECOURS varchar(35) pk, NBJOURS SMALLINT). INSCRIT is linked to EMPLOYE (NUMEMP INTEGER pk, PRENOMEMP varchar(25), POSTE varchar(20), SALAIRE NUMBER(8,2), PRIME NUMBER(7,2), CODEPROJET CHAR(4) fk2, SUPERIEUR INTEGER fk1) and PARTICIPER (NUMEMP INTEGER pk/fk1, CODESEMI varchar(10) pk/fk3, CODEJOUR CHAR(2) pk). EMPLOYE is linked to PROJET (CODEPROJET CHAR(4) pk, NOMPROJET varchar(20), DEBUTPROJ DATE, FINPREVUE DATE, NOMCONTACT varchar(35)).

Benoît Roche / @rocheb83

4

4





GÉNÉRALITÉS


Benoît Roche / @rocheb83

5

5



PRÉSENTATION



Comment fonctionne l'accès aux données entre une application et une Base de Données ?

Les données gérées par les applications sont stockées en mémoire et sont donc **volatiles**, on dit **non persistantes**.



Quasiment toutes les applications ont donc besoin des bases de données pour :

- ✓ Récupérer des données métier persistées
- ✓ Persister des données métier modifiées par l'application
- ✓ Persister des données métier créées par l'application


Benoît Roche / @rocheb83

6

6

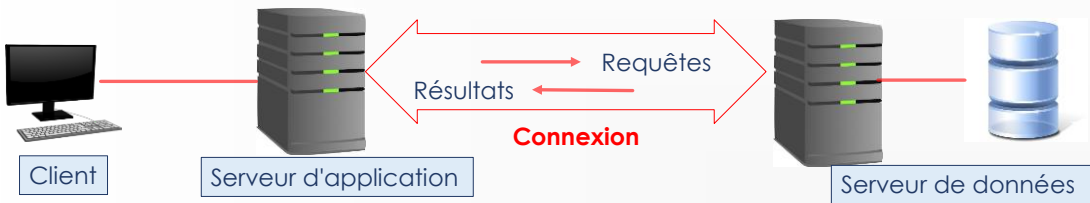


PRÉSENTATION




Comment fonctionne l'accès aux données entre une application et une Base de Données ?

Toutes les applications fonctionnent sur le même principe :



```
graph LR; Client[Client] -- "Requêtes" --> AppServer[Serveur d'application]; AppServer -- "Résultats" --> Client; AppServer -- "Requêtes" --> DataServer[Serveur de données]; DataServer -- "Résultats" --> AppServer; AppServer ---|Connexion| DataServer;
```





L'application est **cliente** de la BD
la BD est **serveur** de l'application
Ils communiquent à travers une **CONNEXION**

Benoît Roche / @rocheb83


7

7



PRÉSENTATION

JDBC ?



JDBC = **J**ava **D**ata**B**ase **C**onnectivity


API développée par java qui gère :

- ✓ la connexion à une base de données,
- ✓ l'émission de requêtes et de commandes
- ✓ la gestion des jeux de résultats obtenus


Benoît Roche / @rocheb83

8

8



PRÉSENTATION




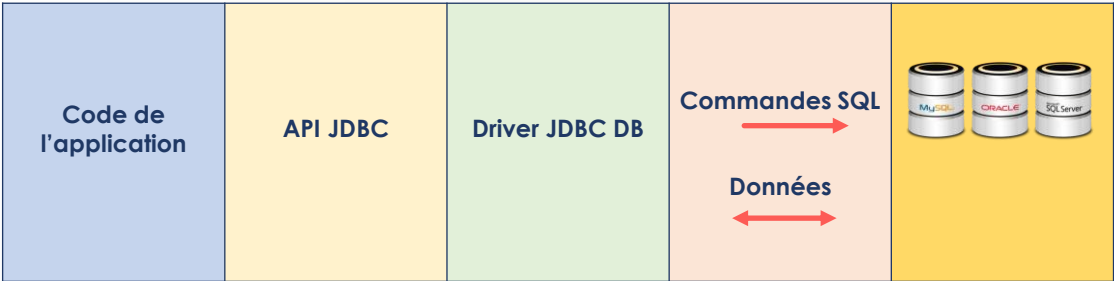
JDBC ?


Schéma de l'architecture JDBC dans la couche de persistance JAVA :




```
graph LR; A[Code de l'application] --> B[API JDBC]; B --> C[Driver JDBC DB]; C --> D[Commandes SQL]; D --> E[(MySQL, ORACLE, SQL Server)]; E --> D; D <--> F[Données];
```


Benoît Roche / @rocheb83


9

9



PRÉSENTATION




JDBC ?


L'API JDBC prend en charge la communication entre :

- ✓ Le code applicatif
- ✓ Et le driver JDBC. Ce driver est **spécifique** au SGBD cible de l'application

JDBC est donc **l'API commune** avec laquelle le code d'application interagit et qui communiquera avec le driver spécifique à la base de données qui prendra en charge les demandes et les résultats obtenus





Il faudra donc télécharger et installer dans l'application le driver spécifique à la BD :

Oracle : [Voir ici](#)
Mysql : [Voir ici](#)

Benoît Roche / @rocheb83

10

10



LES TYPES DE PILOTES JDBC

Benoît Roche / @rocheb83

11

11



LES TYPES DE PILOTES JDBC


JDBC propose quatre types de pilotes :

- ✓ Pilotes de type 1 : JDBC-ODBC
- ✓ Pilotes de type 2 : Java Driver – Native API
- ✓ Pilotes de type 3 : JDBC Net-Pure Java Driver
- ✓ Pilotes de type 4 : Native protocol pure Java Driver


Benoît Roche / @rocheb83

12

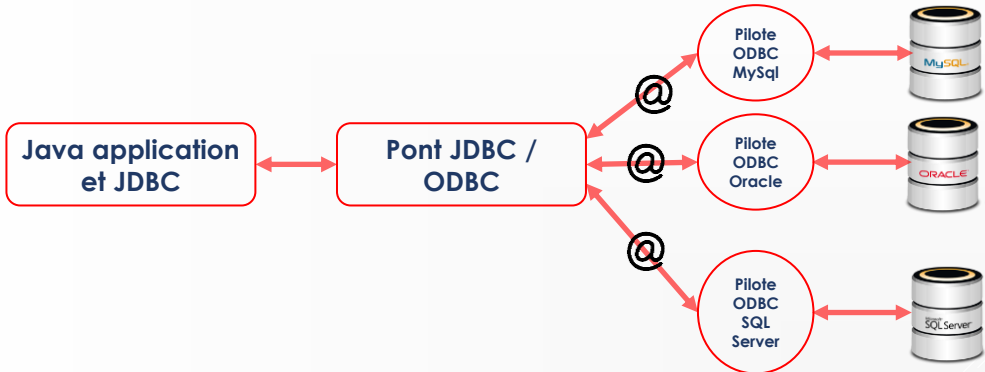
12



LES TYPES DE PILOTES JDBC




Pilotes de type 1 : JDBC-ODBC




Benoît Roche / @rocheb83
13

13



LES TYPES DE PILOTES JDBC



Pilotes de type 1 : JDBC-ODBC

Ces drivers JDBC accèdent à une base de données par l'intermédiaire de la technologie ODBC (Open DataBase Connectivity) développée en C/C++ par et pour Microsoft

Avantage :


- ✓ Connexion possible à TOUS les SGBD, notamment si le driver natif n'existe pas

Inconvénients :


- ✓ Installation nécessaire du pilote ODBC sur le poste de travail
- ✓ Application moins portable dépendante de l'OS
- ✓ Pas adapté aux Applets java

Benoît Roche / @rocheb83
14

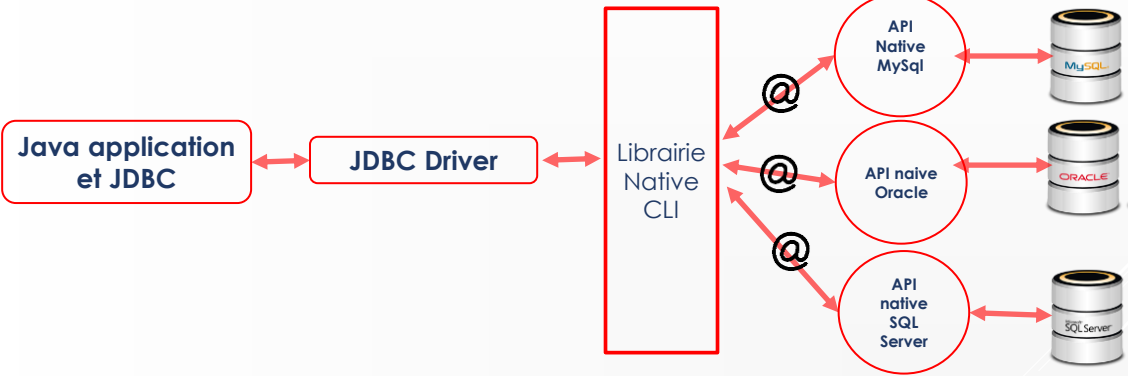
14




LES TYPES DE PILOTES JDBC




Pilotes de type 2 : Java Driver – Native API



15



LES TYPES DE PILOTES JDBC



Pilotes de type 2 : Java Driver – Native API

Ce sont des pilotes qui convertissent les appels JDBC en appels de méthodes natives (Call Level Interface)
de l'API du fournisseur de la base de données (JDBC → API Oracle, JDBC → API Informix ...).


Avantage :

- ✓ Meilleures performance que le type 1


Inconvénients :

- ✓ Installation nécessaire du pilote natif sur le poste de travail
- ✓ Pas adapté aux Applets java

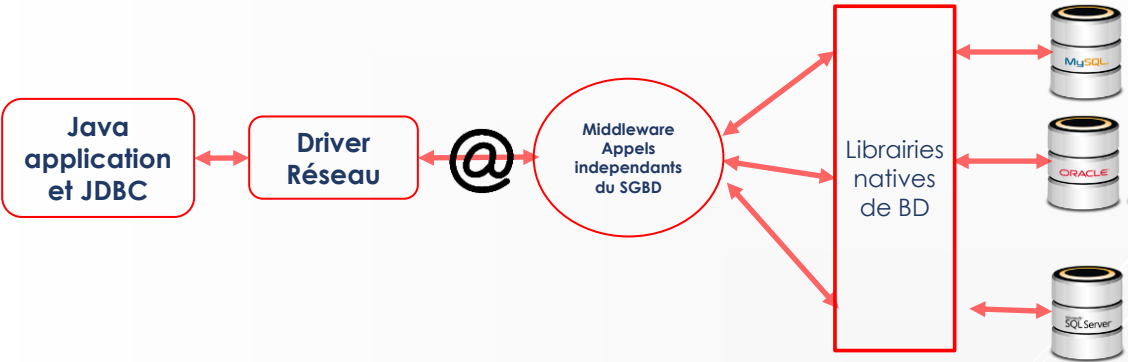
16




LES TYPES DE PILOTES JDBC




Pilotes de type 3 : JDBC Net-Pure Java Driver



17



LES TYPES DE PILOTES JDBC



Pilotes de type 3 : JDBC Net-Pure Java Driver

Le driver réseau convertit les appels JDBC utilise des sockets pour appeler une application intermédiaire sur le serveur qui traduit les requetes du client en une API specifique au pilote voulu


Avantage :

- ✓ Aucune installation sur le poste client
- ✓ Pas de pilotes spécifiques au SGBD / découplage applicaton- SGBD


Inconvénients :

- ✓ Couche réseau sur le poste client.

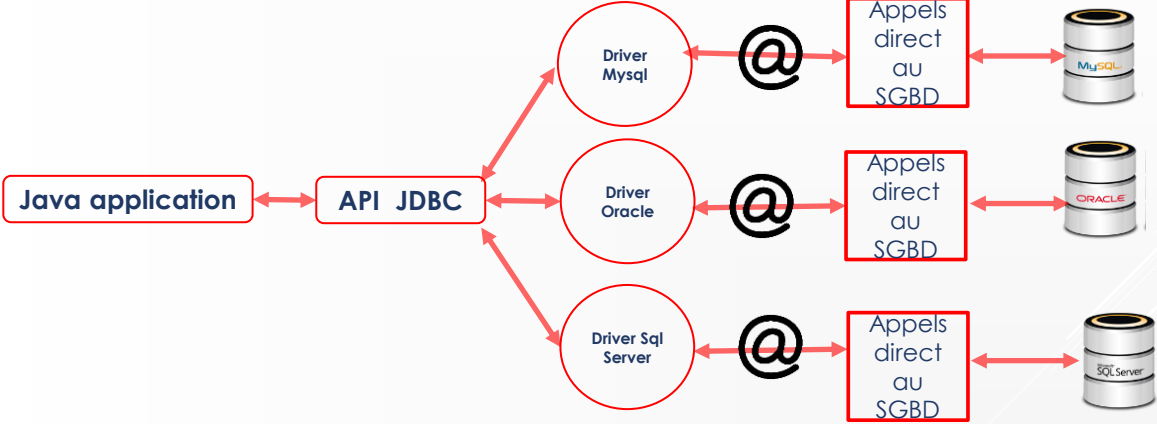
18




LES TYPES DE PILOTES JDBC




Pilotes de type 4 : Native protocol pure Java Driver



19



LES TYPES DE PILOTES JDBC



Pilotes de type 4 : Native protocol pure Java Driver

Ce sont des pilotes écrits en java qui permettent une communication directe à travers le réseau avec le SGBD. Ils convertissent les appels JDBC directement en un protocole réseau exploité par le SGBD.



Avantage :

- ✓ Uniformité de la plateforme, toute la chaine applicative est écrite en Java
- ✓ performances

Inconvénients :

- ✓ Nécessité d'embarquer dans l'application un pilote dépendant du SGBD

20



LES TYPES DE PILOTES JDBC

Choix du type de driver :


Les drivers des catégories 1 et 2 sont à privilégier :

- ✓ lorsque le driver purement Java n'est pas encore disponible ;
- ✓ pour des accès à des bases de données de différents types.

Les drivers des catégories 3 et 4 sont à privilégier

- ✓ lorsque l'installation des postes clients n'est pas possible.

Les solutions 1 et 3 doivent être préférées pour des accès à des bases de données de différents types alors que les solutions 2 et 4 sont plutôt adaptées à des accès visant un seul type de BDD.



Nous allons travailler avec des bases de données **mySQL** et on utilisera les pilotes de type 4.

Benoît Roche / @rocheb8321


21




LES COMPOSANTS DE L'API JDBC

Benoît Roche / @rocheb8322

22



LES COMPOSANTS DE L'API JDBC



Les composants

- ✓ **L'API JDBC** : les classes sont contenues dans le package `java.sql`
- ✓ **Le driver** : spécifique au SGBD, à télécharger et installer dans l'application
- ✓ Drivers Oracle : [ici](#) Drivers Mysql : [ici](#)
- ✓ **Le DriverManager** : c'est la classe **Class** et sa méthode **forName** permettant le chargement dynamique du pilote depuis le code de l'application
- ✓ **La classe Connection** : il s'agit de la classe qui va permettre de créer l'objet qui créera le "tunnel " entre l'application et la BD pour faire passer les requêtes et les résultats
- ✓ **La classe Statement** : permet de créer un l'objet qui portera la requête SQL
- ✓ **La classe ResultSet** : qui permettra de récupérer et d'exploiter le résultat d'une requête

Benoît Roche / @rocheb83

23

23



LES COMPOSANTS DE L'API JDBC



Les composants



Le téléchargement et l'installation des pilotes seront vus dans la partie TD.



MySQL Community Downloads
Connector/J
General Availability (GA) Releases
Archives
Connector/J 8.0.19
Select Operating System:
Platform Independent
Looking for previous GA version?
Platform Independent (Architecture Independent), Compressed TAR Archive
mysql-connector-java-8.0.19.tar.gz
MD5: 5767a6d3c5d9f8bca3372c344d0d8 | Signature
Platform Independent (Architecture Independent), ZIP Archive
mysql-connector-java-8.0.19.zip
MD5: 3d85c39f8732d93982558ee4b251 | Signature
We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.



Database / Technical Details / JDBC and UCP Downloads page
Oracle Database 19c and 18c JDBC driver
Oracle Database 19c (19.3) drivers - NEW !!
Oracle Database 18c (18.3) drivers
Online JDBC Javadoc
Online UCP Javadoc
Older Releases - 12.2,12.1, and 11.2 JDBC drivers
Oracle Database 12c Release 2 (12.2.0.3) drivers
Oracle Database 12c Release 1 (12.1.0.2) drivers
Oracle Database 12c Release 1 (12.1.0.1) drivers
Oracle Database 11g Release 2 (11.2.0.4) drivers
Get the Oracle JDBC drivers from the Oracle Maven Repository

Benoît Roche / @rocheb83

24

24



LES COMPOSANTS DE L'API JDBC



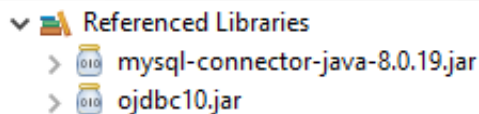
Chargement du pilote : la méthode `Class.forName()`

La connexion à une BD nécessite au préalable le chargement du pilote correspondant au SGBD

Les classes des pilotes de la BD implémentent obligatoirement l'interface `Driver`, définie dans l'API JDBC, et spécifiant les services de base devant être fournis par le pilote.

La classe **Class** et sa méthode **forName** permettent le chargement dynamique du pilote depuis le code

Dans le projet, on peut voir les drivers disponibles :



Benoît Roche / @rocheb83

25

25



LES COMPOSANTS DE L'API JDBC



Chargement du pilote : la méthode `Class.forName()`

Cette instruction charge le pilote et crée une instance de cette classe.

L'application reste indépendante du SGBD si on stocke le nom du driver dans un fichier de paramètres

Exemples de chargement de pilotes :

```
public static void chargeDriver() throws ClassNotFoundException {
    Class.forName("oracle.jdbc.OracleDriver");
}
```

```
public static void chargeDriver() throws ClassNotFoundException {
    Class.forName("com.mysql.cj.jdbc.Driver");
}
```



Dans un bloc
Try.. Catch
bien entendu

Benoît Roche / @rocheb83

26

26

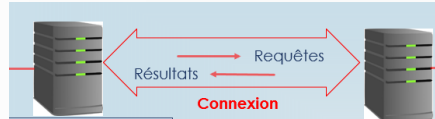


LES COMPOSANTS DE L'API JDBC



Ouverture d'une connexion : la classe DriverManager

- ✓ La classe **DriverManager** est responsable de la communication avec le pilote chargé. La méthode
- ✓ Le méthode **getConnection()** permet d'établir une connexion avec la base de données.



Les connexions sont des ressources coûteuses, il faut donc penser

- ✓ à les fermer
- ✓ à les laisser le moins longtemps possible ouvertes

On les inclura dans un bloc try-catch.

Benoît Roche / @rocheb83

27

27



LES COMPOSANTS DE L'API JDBC



Ouverture d'une connexion : la classe DriverManager

Les informations à fournir sont toujours les mêmes quels que soient les langages :

- L'URL de la base de données (syntaxe dépendant du pilote) , composée :
 - ✓ De l'adresse du serveur,
 - ✓ Du port d'écoute,
 - ✓ Du nom de l'instance ou du schéma à accéder
- Des informations d'authentification :
 - ✓ Un nom d'utilisateur
 - ✓ un mot de passe

Benoît Roche / @rocheb83

28

28



LES COMPOSANTS DE L'API JDBC



Ouverture d'une connexion : la classe DriverManager

La méthode `getConnection` est surchargée. Voir [ici](#)

```
public static Connection getConnection(String url, Properties info) throws SQLException  
public static Connection getConnection(String url, String user, String password) throws SQLException  
public static Connection getConnection(String url) throws SQLException
```

Exemples :

```
String url = "jdbc:oracle:thin:@freesio.lyc-bonaparte.fr:21521:slam";  
String nom= "userjdbc";  
String pwd= "userjdbc";  
try {  
    Class.forName("oracle.jdbc.OracleDriver");  
    Connection connectionOracle = DriverManager.getConnection(url,nom, pwd);
```

```
String url = "jdbc:mysql://localhost/dbcours";  
String nom = "userjdbc";  
String pwd = "userjdbc";  
try {  
    Class.forName("com.mysql.cj.jdbc.Driver");  
    Connection connectionMySQLe = DriverManager.getConnection(url, nom, pwd);
```

Benoît Roche / @rocheb83

29

29



LES COMPOSANTS DE L'API JDBC



Création d'une requête SQL : classe Statement

La création d'une requête se fait à l'aide de la méthode **`createStatement()`** de la classe **Connection**


Cette méthode va créer un objet de classe Statement qui va permettre d'exécuter des requêtes de trois types dont :

- ✓ La consultation : **`executeQuery()`**
- ✓ La modification des données : **`executeUpdate()`**
- ✓ L'exécution de procédures stockées : **`execute()`**


Benoît Roche / @rocheb83

30

30



LES COMPOSANTS DE L'API JDBC




Création d'une requête SQL : classe Statement

La création d'une requête se fait à l'aide de la méthode **createStatement()** de la classe **Connection**

Cette méthode va créer un objet de classe Statement qui va permettre d'exécuter des requêtes de trois types dont :

- ✓ La consultation : **executeQuery(String requeteSQL)**
- ✓ La modification des données : **executeUpdate(String requeteSQL)**
- ✓ L'exécution de procédures stockées : **execute(String nomProcedureStockee)**




Les résultats d'une requête de sélection sont récupérés dans un objet de la classe **ResultSet**.


Benoît Roche / @rocheb83

31

31



LES COMPOSANTS DE L'API JDBC



Création d'une requête SQL : classe Statement

Exemple d'une requête select sur une base Oracle :

```
41 try {
42     String sql= "select * from cours";
43     Class.forName("oracle.jdbc.OracleDriver");
44     Connection connectionOracle = DriverManager.getConnection(url,nom, pwd);
45     Statement statement = connectionOracle.createStatement();
46     ResultSet result= statement.executeQuery(sql);
47
48     connectionOracle.close();
49 }
```


Le point d'arrêt montre que 6 lignes ont été récupérées

▼ result	ForwardOnlyResultSet (id=26)
acProxy	null
closed	false
> connection	T4CConnection (id=1009)
currentRow	-1
fetchedRowCount	6


Benoît Roche / @rocheb83

32

32



LES COMPOSANTS DE L'API JDBC



Création d'une requête SQL : classe Statement


Exemple d'une requête insert sur la table cours sur une base mysql :

```
30 public static void requeteUpdate() throws ClassNotFoundException {
31     String url = "jdbc:mysql://localhost/dbcours";
32     String nom = "userjdbc";
33     String pwd = "userjdbc";
34     try {
35         String sql= "insert into cours (codecours, libellecours, nbjours) values('BR099', 'Java JDBC', 3)";
36         Class.forName("com.mysql.cj.jdbc.Driver");
37         Connection connectionMySQL = DriverManager.getConnection(url, nom, pwd);
38         System.out.println("connecté Mysql");
39         Statement statement = connectionMySQL.createStatement();
40         int nb= statement.executeUpdate(sql);
41         connectionMySQL.close();
42     } catch (Exception e) {
43         e.printStackTrace();
44     }
45 }
```


Le point d'arrêt montre qu'un seul enregistrement a été impacté par la requête

statement	StatementImpl (id=44)
nb	1

code	libelle	nbjours
BR070	Administration d'une BD	4
BR099	Java JDBC	3



LES COMPOSANTS DE L'API JDBC



Création d'une requête SQL : classe Statement

Exemple d'exécution d'une procédure stockée sur un serveur Oracle :

```
try {
    String sql= "CALL exempleinsjdbc()";
    Class.forName("oracle.jdbc.OracleDriver");
    Connection connectionOracle = DriverManager.getConnection(url,nom, pwd);
    Statement statement = connectionOracle.createStatement();
    boolean result= statement.execute(sql);
}
```


Le point d'arrêt montre qu'aucune ligne n'a été récupérée. Mais le cours a bien été créé.

statement	OracleStatementWrapper (id=...)
result	false


ID	MOMENT
1	107/03/20 15:16:12,000000000

[Procédure plsql](#)

Benoît Roche / @rocheb83

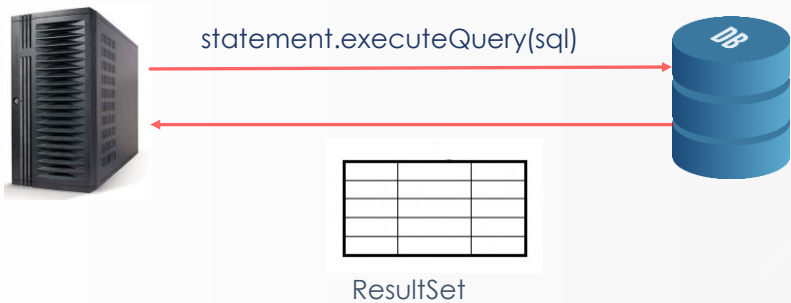


LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select

Les résultats d'une requête de sélection sont récupérés dans un objet de type `ResultSet`.
Le résultat d'une requête SQL peut être vu comme un tableau de résultats dont chaque colonne est un champ du select et chaque ligne un enregistrement récupéré.




The diagram illustrates the process of executing a query. A server icon on the left sends a query `statement.executeQuery(sql)` to a database icon on the right. The database returns a `ResultSet`, represented as a table with 4 columns and 4 rows.

ResultSet


Benoît Roche / @rocheb83

35

35



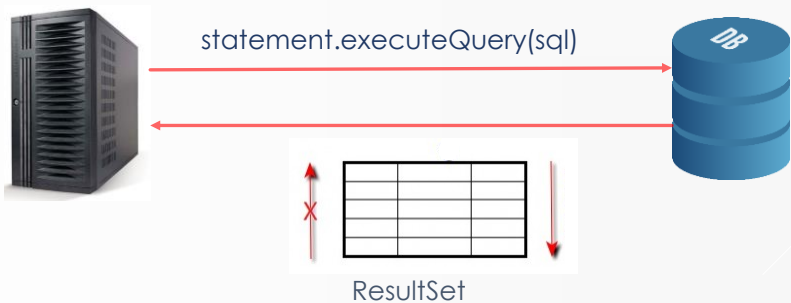
LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select

Il existe plusieurs façons d'exploiter les données d'un objet `ResultSet`.
Le mode par défaut est **Forward only** :

- ✓ on ne peut qu'avancer séquentiellement
- ✓ on ne peut pas modifier les données du `ResultSet`




The diagram illustrates the process of executing a query. A server icon on the left sends a query `statement.executeQuery(sql)` to a database icon on the right. The database returns a `ResultSet`, represented as a table with 4 columns and 4 rows. Red arrows indicate sequential navigation: a vertical arrow on the left with an 'X' at the top, and a vertical arrow on the right pointing downwards.

ResultSet


Benoît Roche / @rocheb83

36

36



LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select


La méthode **next()** :

- ✓ permet de se déplacer à la ligne suivants
- ✓ Renvoie true si une ligne a été effectivement lue

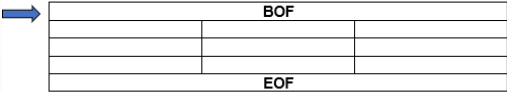
Il est nécessaire d'exécuter la méthode **next()** une première fois pour exploiter la première ligne

Algorithme de lecture d'un ResultSet par défaut :

```
ResultSet result = statement.executeQuery(sql);
while (result.next()) {
    //exploiter la ligne
}
result.close();
```




Ne pas oublier de fermer le resultSet




Benoît Roche / @rocheb83

37

37



LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select

Exploitation d'une ligne du ResultSet : méthode **getXxx()** de la classe ResultSet


Xxx représente le type de la colonne afin de mapper le type du SQGD sur un type Java

La méthode getXxx est surchargée :

- ✓ **getXxx(int i)** : accès à la valeur d'indice i de la ligne courante de l'objet resultSet courant. *1ere colonne = indice 1*
- ✓ **get Xxx(String col)** : accès à la colonne de nom col dans la ligne courante du ResultSet

Les 2 instructions récupéreront la même valeur
Select codecours,...


```
while (result.next()) {
    String codeCours1 = result.getString(1);
    String codeCours2 = result.getString("CODECOURS");
}
result.close();
```




Benoît Roche / @rocheb83

38

38



LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select


Correspondance des type Xxx :

Type SQL	Description	Type Java conseillé	Méthode de ResultSet
BIT	1 bit	Boolean	getBoolean()
TINYINT	Entier 8 bits	Byte	getByte()
SMALLINT	Entier 16 bits	Short	getShort()
INTEGER	Entier 32 bits	Int	getInt()
BIGINT	Entier 64 bits	Long	getLong()
REAL	Flottant 32 bits	Float	getFloat()
DOUBLE	Flottant 64 bits	Double	getDouble()
CHAR(n) VARCHAR(n)	Chaîne de n caractères Chaîne variable de n caractères	String	getString()
DATE	Date	java.sql.Date	getDate()
TIME	Heure	java.sql.Time	getTime()
TIMESTAMP	Date et heure	java.sql.Timestamp	getTimestamp()
DECIMAL(c,d)	Nombre décimal de c chiffres dont d décimales	java.math.BigDecimal	getBigDecimal()


Benoît Roche / @rocheb83

39

39



LES COMPOSANTS DE L'API JDBC



Exploitation du résultat d'une requête select

Il existe 2 autres types de ResultSet

TYPE_SCROLL_INSENSITIVE :

- ✓ On peut se déplacer comme on veut à l'intérieur du ResultSet
- ✓ Les données du ResultSet ne sont pas mises à jour si les données de la base sont modifiées pendant l'exploitation du ResultSet


TYPE_SCROLL_SENSITIVE :

- On peut se déplacer comme on veut à l'intérieur du ResultSet
- Les données du ResultSet **sont** mises à jour si les données de la base sont modifiées pendant l'exploitation du ResultSet


Benoît Roche / @rocheb83

40

40



LES COMPOSANTS DE L'API JDBC




Les méthodes de la classe ResultSet pour les types TYPE_SCROLL_INSENSITIVE et TYPE_SCROLL_SENSITIVE

previous()	Déplace le curseur sur la ligne précédente. Si le curseur se trouve en dehors du tableau, la méthode retourne false.
first()	Positionne le curseur sur la première ligne
last()	Positionner le curseur sur la dernière ligne du tableau
absolute(nombre)	Déplacer le curseur sur la ligne indiquée par nombre.
relative(nombre)	Déplacer le curseur de « nombre » lignes vers le haut si nombre est négatif, vers le bas si nombre est positif.
BeforeFirst()	Place le curseur avant le premier enregistrement
afterLast()	Place le curseur après le dernier enregistrement


Benoît Roche / @rocheb83

41

41



LES COMPOSANTS DE L'API JDBC



Exemple :

```
try {
    String sql = "select * from employe";
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection connectionMysql = DriverManager.getConnection(url, nom, pwd);
    Statement statement = connectionMysql.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
    ResultSet result = statement.executeQuery(sql);
    if(result.first()) {
        System.out.println(result.getString("nomemp"));
    }
    if(result.absolute(5)) {
        System.out.println(result.getString("nomemp"));
    }
    if(result.relative(2)) {
        System.out.println(result.getString("nomemp"));
    }
}
```

<terminated> Programme (12) [

DUPONT

MARTIN

GIMOND

Benoît Roche / @rocheb83

42

42



LES COMPOSANTS DE L'API JDBC



Mise à jour des données à travers un ResultSet :

Il est possible de mettre à jour les données directement à travers un ResultSet si :

- ✓ Le ResultSet n'est pas en read only,
- ✓ On le déclare en **CONCUR_UPDATABLE** (deuxième paramètre de la méthode create statement)

Benoît Roche / @rocheb83

43

43



LES COMPOSANTS DE L'API JDBC



Mise à jour des données à travers un ResultSet :

```
try {
    String sql = "select numemp, nomemp, prenomemp, salaire from employe";
    Class.forName("oracle.jdbc.OracleDriver");
    Connection connectionOracle = DriverManager.getConnection(url, nom, pwd);
    Statement statement = connectionOracle.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
    ResultSet result = statement.executeQuery(sql);

    while(result.next()) {
        if(result.getDouble("salaire") < 10000) {
            double newSalaire=result.getDouble("salaire");
            newSalaire*=1.09;
            result.updateDouble("salaire", newSalaire);
            result.updateRow();
        }
    }
    connectionOracle.close();
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```



N'accepte pas, sous Oracle, les requête de type :
select * ...



On peut aussi ajouter ou supprimer des lignes dans un objet Resultset et mettre à jour la BD

Benoît Roche / @rocheb83

44

44



LES COMPOSANTS DE L'API JDBC



Les Requêtes préparées

- Elles sont obligatoires
 - ✓ dans le cas de requêtes paramétrées. Cela évite de concaténer la requête SQL avec les paramètres et **évite les injections SQL** 😬
- Elles sont vivement conseillées
 - ✓ Pour gagner du temps quand on doit exécuter plusieurs fois la même requête avec des paramètres différents. La requête sera pré-compilée. Le plan d'exécution sera établi une seule fois

Benoît Roche / @rocheb83

45

45



LES COMPOSANTS DE L'API JDBC



Les Requêtes préparées

Séquence d'exécution d'une requête préparée :

- ✓ **Préparer** la requête : méthode **prepareStatement()** de la classe **Connection** : cette méthode retourne un objet de la classe **PreparedStatement**
- ✓ **Affecter** et valoriser les paramètres : méthodes **setXxx** de la classe **PreparedStatement**
- ✓ **Exécuter** la requête pré-compilée à l'aide d'une méthode **executeQuery**, **executeUpdate** ou **execute**.

Benoît Roche / @rocheb83

46

46



LES COMPOSANTS DE L'API JDBC



Les Requêtes préparées

- ✓ **Préparer** la requête : méthode **prepareStatement()** de la classe **Connection** : cette méthode retourne un objet de la classe **PreparedStatement**

Les paramètres dans la requête sont indiqués avec un ?

Ils seront valorisés par leur ordre d'apparition dans la requête

```
Connection connectionOracle = DriverManager.getConnection(url, nom, psw);
String sql = "select numemp, nomemp, prenomemp, salaire from employe "
    + "where salaire between ? AND ?";
PreparedStatement statement = connectionOracle.prepareStatement(sql);
```

Benoît Roche / @rocheb83

47

47



LES COMPOSANTS DE L'API JDBC



Les Requêtes préparées

- ✓ **Affecter** et valoriser les paramètres : méthodes setXxx de la classe **PreparedStatement**

On utilise les méthodes setXxx en fonction du type de paramètre ;

```
PreparedStatement statement = c
statement.setDouble(1,10000);
statement.setDouble(2,11000);
```

Benoît Roche / @rocheb83

48

48



LES COMPOSANTS DE L'API JDBC



Les Requêtes préparées

- ✓ **Exécuter** la requête pré-compilée à l'aide d'une méthode `executeQuery`, `executeUpdate` ou `execute`.

```
statement.setDouble(2, 11000);
ResultSet result= statement.executeQuery();
while(result.next()) {
```

- ✓ Ou `executeUpdate` :

```
String sql = "insert into cours(codecours, libellecours, nbjours) values(?, ?, ?)";
PreparedStatement statement = connectionMySQL.prepareStatement(sql);
statement.setString(1, "BR099");
statement.setString(2, "Java JDBC avancé");
statement.setInt(3, 4);
int nb= statement.executeUpdate();
System.out.println("Nombre de lignes mises à jour : " + nb);
```

Nombre de lignes mises à jour : 1

Benoît Roche / @rocheb83

49

49



LES COMPOSANTS DE L'API JDBC



Cas des identifiants en auto-incrément

On va distinguer deux cas :

Oracle / Postgres qui proposent une gestion normalisée SQL des champs auto-incrémentés par l'utilisation de séquences

```
String sql = "insert into categorie(id, libelle) values(seq_categorie.nextval, ?)";
```

Mysql qui propose les champs auto-incrément pour les champs clé primaire.

```
String sql = "insert into categorie(libelle) values(?)";
```



Rien d'exceptionnel...
Sauf si l'on doit récupérer les clés générées pour s'en servir dans la suite de l'application

Benoît Roche / @rocheb83

50

50



LES COMPOSANTS DE L'API JDBC



Récupération de l'id auto-généré **ORACLE**

Il se peut que l'on doive récupérer l'id généré pour la suite du traitement :

Sous Oracle il y a deux possibilités :

- Soit on **refait une requête** après l'exécution de l'ordre insert pour récupérer la dernière clé distribuée

- ✓ Inconvénients :

On effectue 2 requêtes donc 2 allers/retours sur le serveur de bases de données

On prend le risque de ne pas retourner la bonne valeur de clé si il y a eu une autre insertion d'une autre connexion entre les 2 requêtes

- Soit on utilise la **classe CallableStatement** pour exécuter la requête dans un script

- Avantages

Les inconvénients de la précédente

Benoît Roche / @rocheb83

51

51



LES COMPOSANTS DE L'API JDBC



Cas des identifiants en auto-incrément **ORACLE**

- Soit on refait une requête après l'exécution de l'ordre insert pour récupérer la dernière clé distribuée

```
String sql = "insert into categorie(id, libelle) values(seq_categorie.nextval, ?)";
```

```
public void insertCategorie(String libelle) throws SQLException {
    String sql = "begin insert into categorie(id, libelle) values(seq_categorie.nextval, ?) returning id into ?; end;";
    try {
        CallableStatement statement = this.connection.prepareCall(sql);
        statement.setString(1, libelle);
        statement.registerOutParameter(2, OracleTypes.NUMBER);
        statement.execute();
        System.out.println("clé générée (Oracle) : " + statement.getInt(2));
    }
}
```

Benoît Roche / @rocheb83

52

52



LES COMPOSANTS DE L'API JDBC



Cas des identifiants en auto-incrément **ORACLE**

Soit on utilise la classe CallableStatement pour exécuter la requête dans un script

```
String sql = "begin insert into categorie(id, libelle) values(seq_categorie.nextval, ?) "
            + "returning id into ?; end;";
```

Script plsql

```
String sql = "begin insert into categorie(id, libelle) values(seq_categorie.nextval, ?) "
            + "returning id into ?; end;";
try {
    CallableStatement statement = this.connection.prepareCall(sql);
    statement.setString(1, libelle);
    statement.registerOutParameter(2, OracleTypes.NUMBER);
    statement.execute();
    System.out.println("clé générée (Oracle) : "+ statement.getInt(2));
}
```

Benoît Roche / @rocheb83

53

53



LES COMPOSANTS DE L'API JDBC



Récupération de l'id auto-généré **MySQL**

Il se peut que l'on doive récupérer l'id généré pour la suite du traitement :

Sous Mysql il faut utiliser l'option **Statement.RETURN_GENERATED_KEYS** au moment de la préparation de la requête :

```
PreparedStatement statement = this.connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
```

Ensuite on récupère la ou les clés générées dans un objet de la classe ResultSet :

```
ResultSet rs = statement.getGeneratedKeys();
```

Il suffit après de parcourir ce resultSet

Benoît Roche / @rocheb83

54

54



LES COMPOSANTS DE L'API JDBC



Récupération de l'id auto-généré

Il se peut que l'on doive récupérer l'id généré pour la suite du traitement :

Exemple

```
String sql = "insert into categorie(libelle) values(?)";
try {
    PreparedStatement statement = this.connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
    statement.setString(1, libelle);
    int nb = statement.executeUpdate();
    ResultSet rs = statement.getGeneratedKeys();
    if (rs.next()) {
        System.out.println("clé générée (Mysql) : " + rs.getInt(1));
    }
}
catch (SQLException ex) {
    System.out.println(ex.getMessage());
}
```

Benoît Roche / @rocheb83

55

55





LES TRANSACTIONS


Benoît Roche / @rocheb83

56

56



LES TRANSACTIONS

Java
JDBC

Rappel

Une transaction est définie comme une **unité logique de traitements (ensemble d'ordres *insert, update, delete*)** qui, appliquée à un état cohérent de la base de données, restitue un nouvel état cohérent, en principe modifié de la base.

Une transaction ne peut être QUE

- ✓ validée (**COMMIT**) : tous les ordres de mise à jour sont complètement exécutés,
- ✓ invalidé (**ROLLBACK**) : tous les ordres de mise à jour depuis le début de la transaction sont annulés

Benoît Roche / @rocheb83

57

57



LES TRANSACTIONS

Java
JDBC

Rappel

Structure du programme :

```
try {  
    // Plusieurs ordre de mise à jour  
    // insert, update delete  
  
    commit;  
} catch (Exception e) {  
  
    rollback  
    //...  
}
```

Benoît Roche / @rocheb83

58

58



LES TRANSACTIONS

**Principe en JDBC :**

Par défaut, JDBC travaille en mode **autoCommit ON**. C'est-à-dire en mode non transactionnel. Chaque ordre de mise à jour est automatiquement validé après exécution.

Une transaction se fait dans une connexion. On utilisera donc les méthodes de la classe Connection :

- ✓ **setAutoCommit(false)** : pour débiter une transaction
- ✓ **commit** : pour valider les ordres de la transaction
- ✓ **rollback** : pour annuler les ordres de la transaction déjà effectués

Benoît Roche / @rocheb83

59

59



LES TRANSACTIONS



**Exemple :**

```
try {  
    // début transaction  
    connectionOracle.setAutoCommit(false);  
    Statement statement= connectionOracle.createStatement();  
    statement.executeUpdate("update cours set nbjours= 4 where codecours='BR035'");  
    statement.executeUpdate("insert into cours (codecours, libellecours, nbjours) values "  
        + "('BR077', 'Cours JDBC', 3)");  
    statement.executeUpdate("update employe set salaire= salaire*1.03");  
    connectionOracle.commit();  
}  
catch(SQLException ex) {  
    System.out.println(ex.getMessage());  
    connectionOracle.rollback();  
}  
finally {  
    connectionOracle.close();  
}
```

Benoît Roche / @rocheb83

60

60





GESTION DES ERREURS

Benoît Roche / @rocheb83

61

61



LES COMPOSANTS DE L'API JDBC

Gestion des erreurs

JDBC permet de connaître les avertissements et les exceptions générées par la base de données lors de l'exécution d'une requête.

La classe **`SQLException`** étend la classe générale **`java.lang.Exception`** pour fournir des informations supplémentaires sur les erreurs des bases de données.



Les informations fournies sont :

- ✓ la chaîne **`SQLState`** décrivant l'état SQL de l'erreur obtenue avec la méthode **`getSQLState()`** ;
- ✓ le code d'erreur **`ErrorCode`** propre au fournisseur du pilote de la base de données : **`getErrorCode()`** ;
- ✓ le message qui contient la description de l'erreur : **`getMessage()`**.

Benoît Roche / @rocheb83

62

62



LES COMPOSANTS DE L'API JDBC

Gestion des erreurs



Il est possible de visualiser toute la chaîne d'exceptions qui ont conduit à la première annoncée avec la méthode **getNextException()**. La méthode renvoie null une fois la dernière exception renvoyée.


JDBC fournit une extension de la classe `SQLException` appelée **SQLWarning** qui gère les erreurs non bloquantes. Ces avertissements peuvent être consultés en appelant la méthode **getWarnings()**.

Benoît Roche / @rocheb83

63

63







TP 5 : EMPLOYES


Benoît Roche / @rocheb83

64

64







Fin du cours, merci

Questions/Réponses

Benoît Roche
@rocheb83

Benoît Roche / @rocheb83

65