# JDBC

Java DataBase Connectivity

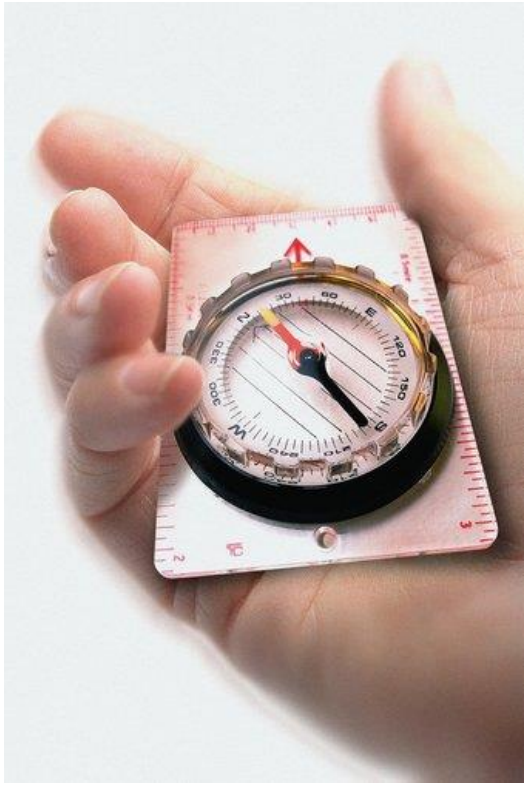
# Objectifs du cours

En complétant ce cours, vous serez en mesure de:

- Vous connecter à une base de données à partir du code Java
- Gérer la base de données avec JDBC




# Course plan



- Introduction
- DataBase Connexion
- Querying
- Gestion des transactions
- Meta data

JDBC

# INTRODUCTION



# DBMS presentation

- DataBase Management System

- Logiciel permettant de gérer des bases de données :
  - Créer/Modifier des tableaux
  - Maintenance
  - Interroger la base de données
  - Gérer les opérations
  - Assumer l'intégrité des données



# JDBC API presentation

- Java DataBase Connectivity

- API (application programming interface / interface de programmation d'application) avec des méthodes permettant de :
  - Envoyer des requêtes SQL
  - Récupérer des données
  - …

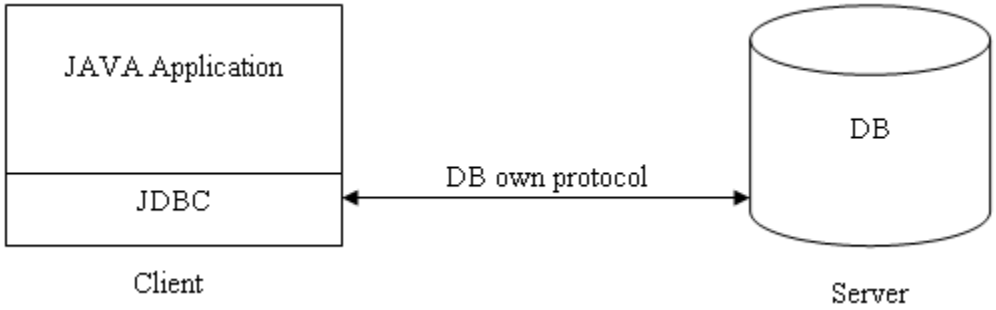


# JDBC API presentation

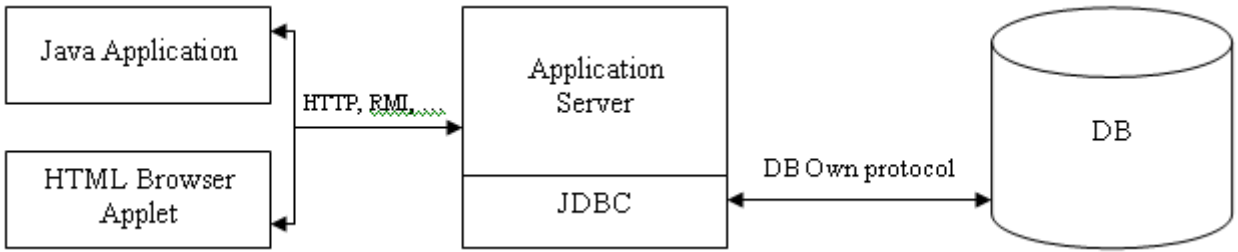
- De nombreux constructeurs fournissent leur implémentation pour leur base de données :
  - MySQL
  - Oracle
  - PostgreSQL
  - SQL Server
  - …

# Architecture Client/Server

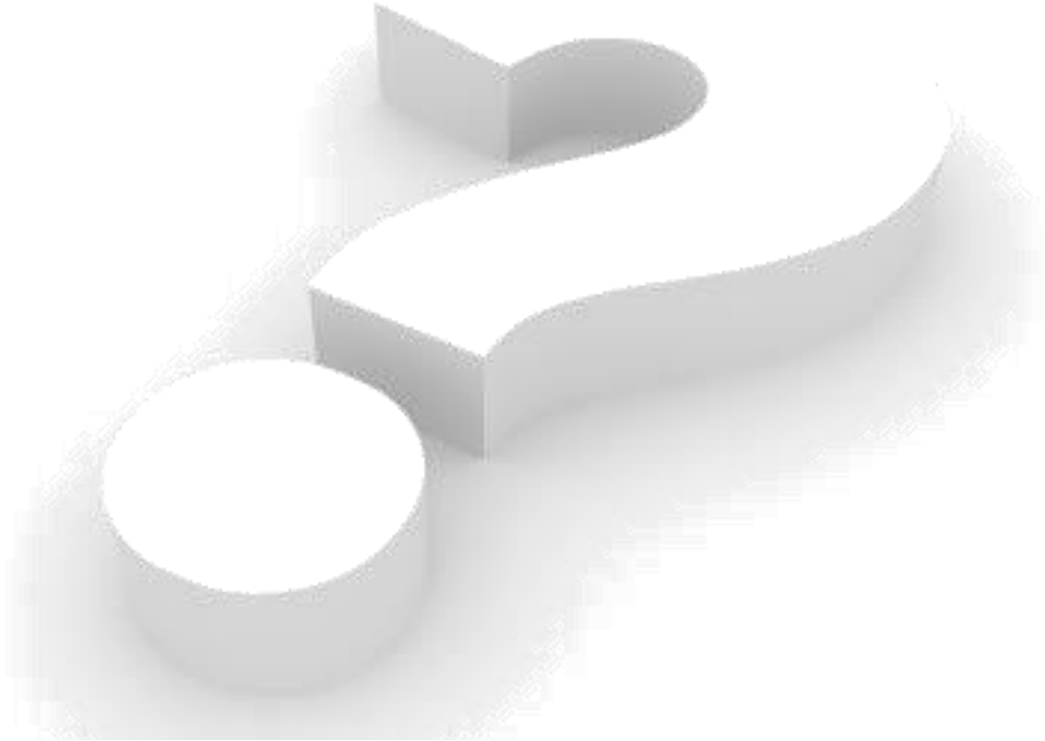
- Architecture 2/tiers :




- Architecture 3/tiers :



# Questions ?



JDBC

# DATABASE CONNEXION



# Etape par étape : charger le pilote

- La première étape consiste à charger le pilote dans la mémoire
- Le pilote est fourni par le constructeur de base de données
- Comment le charger ?

```
Class.forName(String driver)
                throws ClassNotFoundException
```

- Exemple avec MySQL :

```
try {
  Class.forName(
  com.mysql.jdbc.Driver.class.getName());
} catch(ClassNotFoundException ex) {
  System.out.println("Can't load the Driver");
}
```



# Etape par étape : Établir la connexion

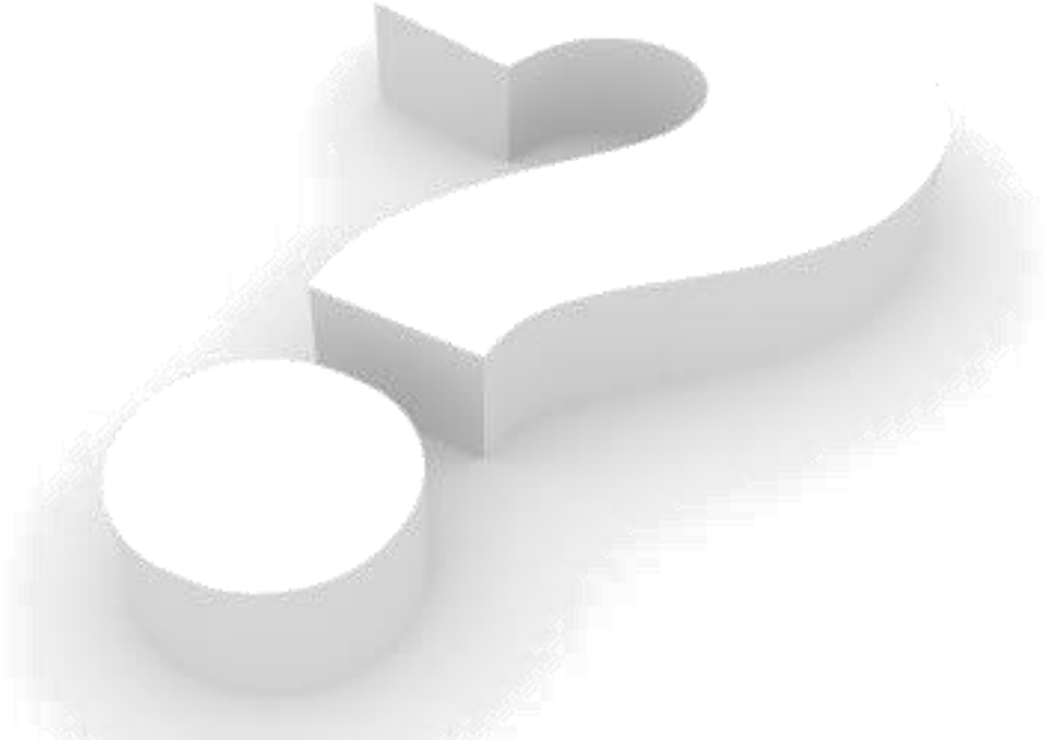
- Puis établissez la connexion avec un **DriverManager**
- Comment obtenir une connexion ?
  - Deux méthodes de **DriverManager** renvoient un java.sql.Connection :
    - getConnection(String url)
    - getConnection(String url, String login, String password)
- L'URL de MySQL ressemble à :
  - jdbc:mysql://<hostname>/<databaseName>
  - Exemple : jdbc:mysql://localhost/SunGamer



# La classe Connection

- Représente la connexion à la base de données
- Récupérer la déclaration (Statement) avec (voir plus tard)
- Méthode utile:
  - void setAutoCommit(boolean autoCommit)
  - void commit()
  - void rollback()
  - Savepoint setSavepoint()
  - Savepoint setSavepoint(String name)
  - void close()

# Questions ?



JDBC

# QUERYING (~QUESTIONNER)



# Statement (déclaration)

- Une interface
- Utilisez-le pour faire des requêtes
- Obtenez-le avec votre java.sql.Connection :
  - createStatement()
- Comment interroger la BD ?
  - ResultSet executeQuery(String query)
  - int executeUpdate(String query)

```
Statement stmt = myConnexion.createStatement();
ResutSet rs = stmt.executeQuery("SELECT * FROM dummy");
```



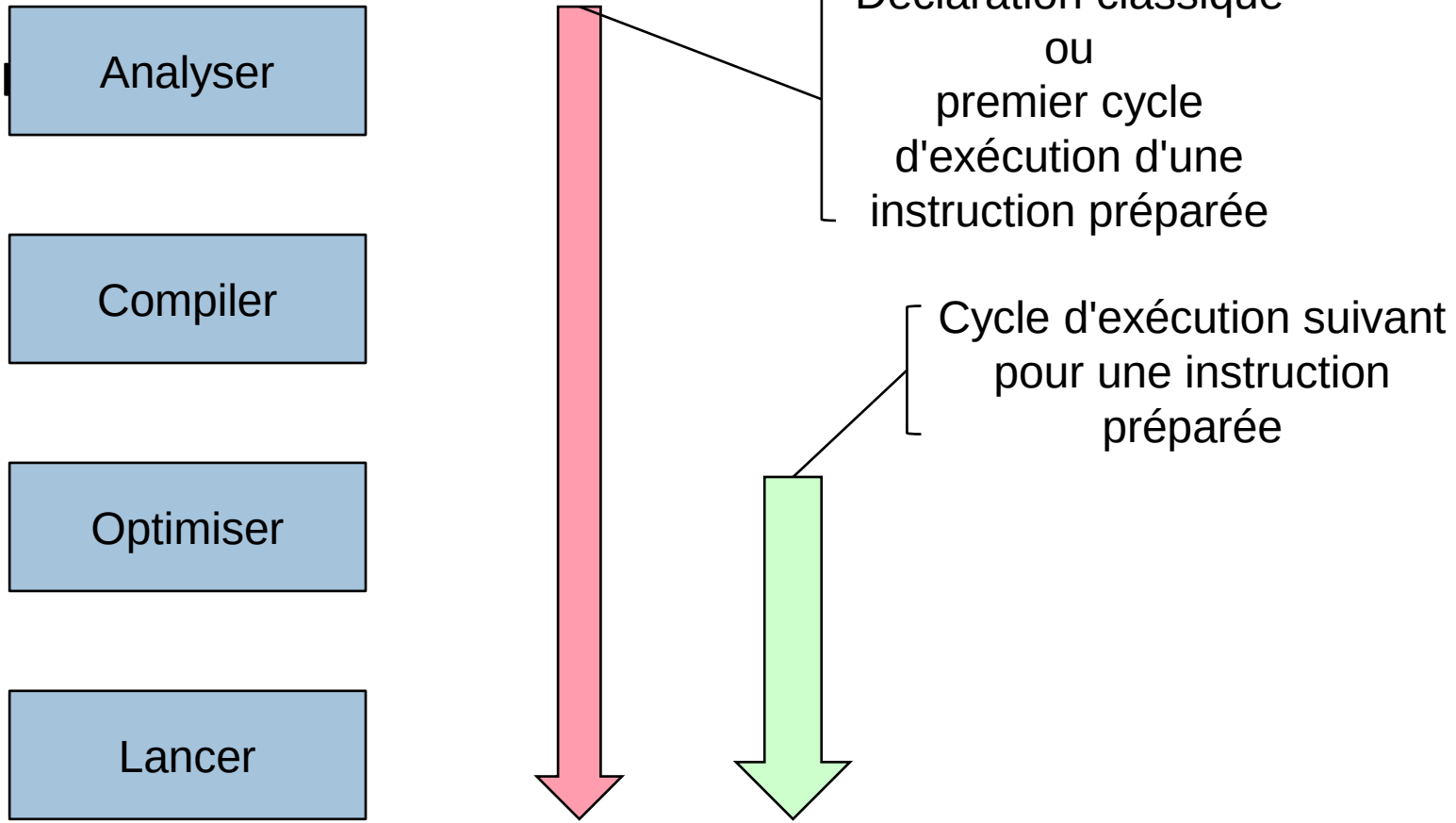
# PreparedStatement

- La plupart des bases de données les plus matures prennent en charge le concept d'instructions préparées
- Une requête préparée n'a besoin d'être analysée (ou préparée) qu'une seule fois, mais peut être exécutée plusieurs fois avec les mêmes paramètres ou des paramètres différents.
- La base de données analysera, compilera et optimisera son plan d'exécution
- Éviter de répéter le cycle analyse/compilation/optimisation
- Protège contre l'injection SQL

Cela signifie que les instructions préparées utilisent moins de ressources, s'exécutent plus rapidement et sont plus sécurisées !

# PreparedStatement

- Quer





# PreparedStatement

- Obtenez-le avec votre java.sql.Connection :
  - prepareStatement(String query)
- Beaucoup de méthodes pour définir les paramètres :
  - setInt(int index, int value)
  - …

```
PreparedStatement pstmt = myConnexion.prepareStatement(
"SELECT * FROM dummy ”
+ "WERE lastname = ? AND firstname = ?");
pstmt.setString(1, "GEORGES");
pstmt.setString(2, "Ron");
```



# ResultSet

- Contient les résultats de votre requête
- "Iterate" sur le ResultSet par les méthodes :
  - boolean previous() :
    - Renvoie vrai si le curseur a été déplacé sur le résultat précédent
  - boolean next() :
    - Renvoie vrai si le curseur a été déplacé sur le résultat suivant
- Obtenir les données d'une colonne :
  - getXXX(int columnIndex) *XXX === type*
  - getXXX(String columnName)

Querying

# ResultSet

- Example:

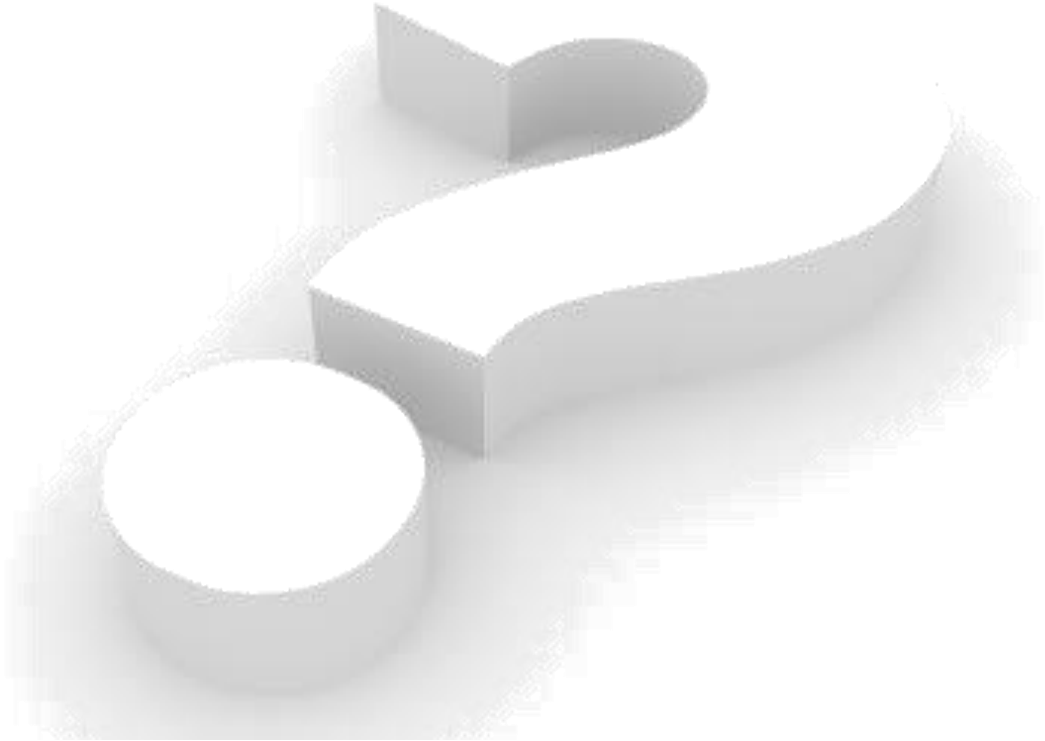
```
// ...
ResultSet rs =
    stmt.executeQuery("SELECT * FROM dummy");

while(rs.next()) {
    int id = rs.getInt(1);
    String name = rs.getString("name");
    // ...
}
// ...
```



# Type ResultSet

- L'interface ResultSet a trois variables statiques utilisable dans la méthode createStatement(…)
  - TYPE_FORWARD_ONLY :
    - Le curseur ne peut être déplacé que vers l'avant
  - TYPE_SCROLL_INSENSITIVE :
    - Le ResultSet n'est pas sensible aux changements effectués par d'autres sur les données
  - TYPE_SCROLL_SENSITIVE :
    - Le ResultSet est sensible aux changements effectués par d'autres sur les données

# Questions ?



JDBC

# GESTION DES TRANSACTIONS



# Presentation

- Une transaction est utile lorsque vous souhaitez définir un ensemble unifié de requêtes
  - S'ils réussissent tous, les modifications sont appliquées
  - En cas d'échec, aucune modification n'est appliquée

- Nous nommons *commit* l'opération qui a appliqué les modifications dans la base de données



# Étude de cas

- Par exemple, les transactions sont très importantes dans les applications bancaires :
  - Imaginez un virement bancaire
  - L'opération se déroule en deux temps :
    - Retirer de l'argent d'un compte
    - Ajouter à un autre
- Imaginez que chaque étape est une requête de base de données
- Que se passe-t-il si la seconde requête échoue ? Où est l'argent ?



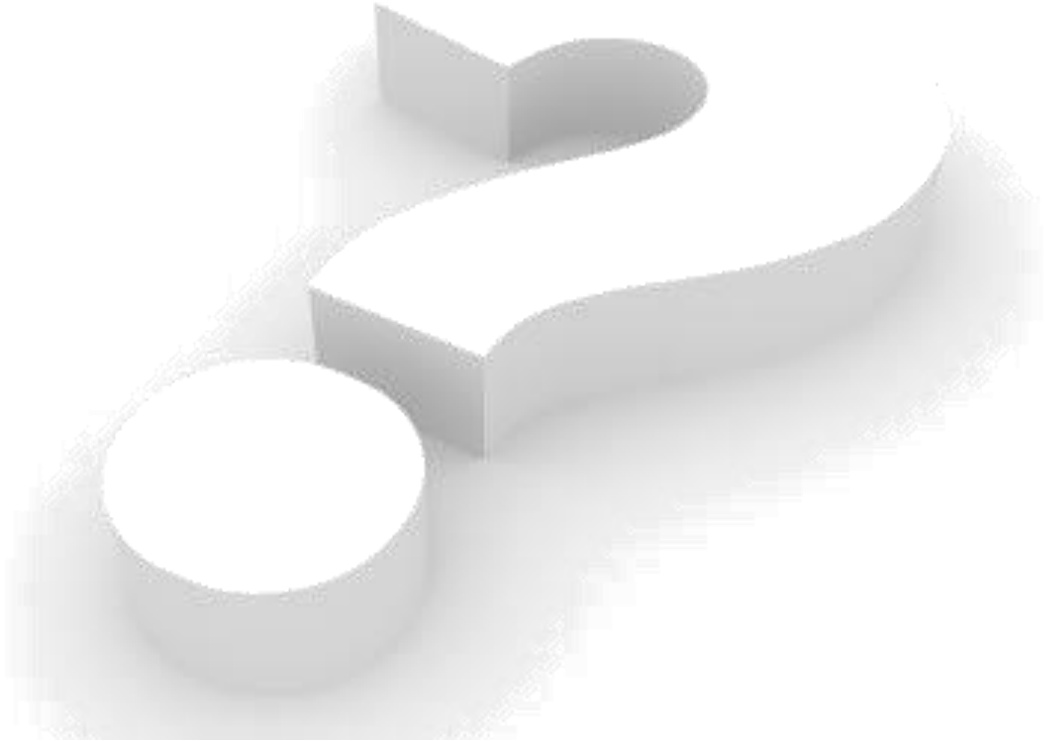
# Méthodes fournies

- Les instances de Connection fournissent trois méthodes pour gérer les transactions:
  - void setAutoCommit ( boolean ) :
    - Définit le mode de validation automatique de cette connexion sur l'état donné. Si une Connexion est en auto-commit. Par défaut, les nouvelles Connexions sont en mode auto-commit
  - void commit ( ) :
    - Rend permanentes toutes les modifications apportées depuis le commit/rollback précédent
  - void rollback ( ) :
    - Annule toutes les modifications apportées de la transaction en cours

# Exemple

```
Connexion Connexion =
DriverManager.getConnexion(URL,USER,PASSWORD);

Connexion.setAutoCommit(false);
try {
  String sql1 =
"INSERT INTO author (id, firstname, lastname)"
+ " VALUES (1, 'Clark', 'Kent' )";
  Connexion.createStatement().executeUpdate(sql1);
  String sql2 =
"INSERT INTO article(id, title, body, author_id)"
+ " VALUES(1, 'Plop', '...', 1)";
  Connexion.createStatement().executeUpdate(sql2);

  Connexion.commit();
} catch (Exception e) {
  Connexion.rollback();
}
```

# Questions ?



JDBC

# META DATA



# DatabaseMetaData

- Obtenez-les avec un java.sql.Connexion :
  - DatabaseMetaData getMetaData()

- Vous pouvez obtenir toutes les informations sur la base de données :
  - Les catalogues
  - Les schémas
  - La version
  - …

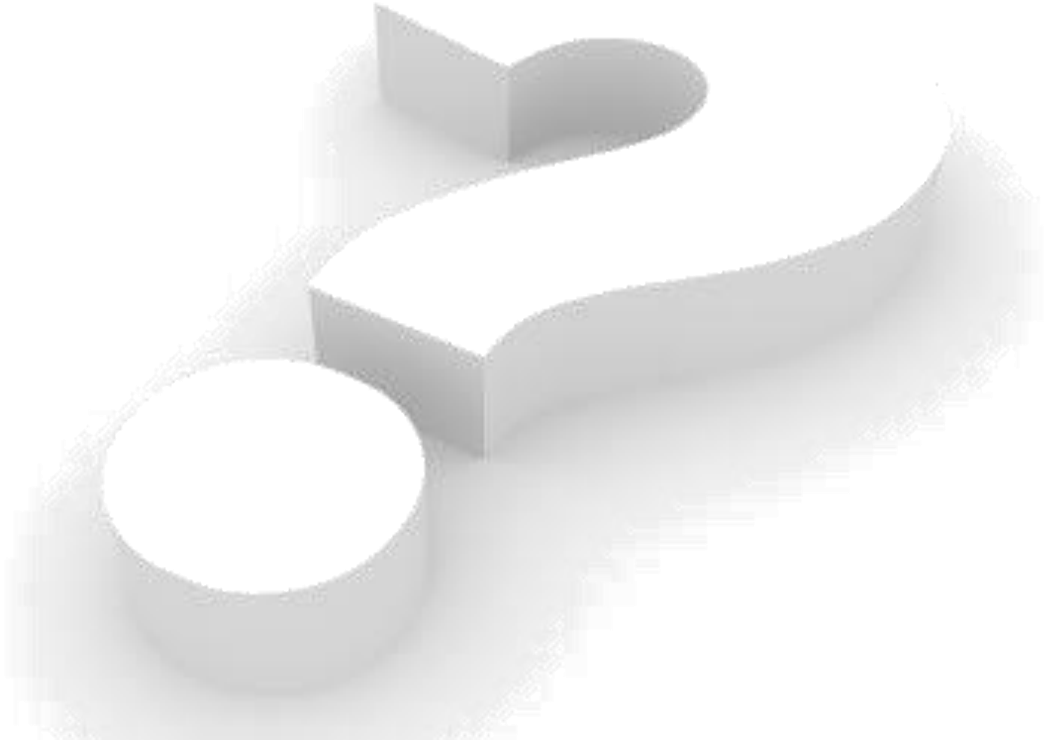


# ResultSetMetaData

- Obtenez-les avec un ResultSet :
  - ResultSetMetaData getMetaData()

- Vous pouvez obtenir toutes les informations sur les colonnes
  - Le nom
  - Le type
  - Le nombre de colonne
  - …

# Questions ?





# Exercise (1/8)

- Nous allons développer un nouveau Projet :
  - Un logiciel de gestion pour une agence de voyage
  - Ce projet s'appelle JavaParadise!

- Pour l'instant, nous allons développer une version CLI en utilisant une base de données MySQL
- Plus tard, nous développerons une version graphique

# Exercise (2/8)

- La version CLI donne quelque chose comme ça :

```
Welcome aboard !

What do you want to do ?
1 - Add a place
2 - Find a place
3 - Edit a place
4 - Remove a place
5 - Add a trip
6 - Find a trip
7 - Remove a trip
8 - Quit

1
Name: Tokyo
Place added with the ID=1.
```



# Exercise (3/8)

- Votre application doit être composée d'au moins deux interfaces :
  - PlaceDao, définissant les méthodes abstraites suivantes :
    - Long createPlace(Place p)
    - Place findPlaceById(Long id)
    - boolean updatePlace(Place p)
    - boolean removePlace(Place p)
  - TripDao, définissant les méthodes abstraites suivantes :
    - Long createTrip(Trip t)
    - Trip findTripById(Long id)
    - boolean removeTrip(Trip t)



# Exercise (4/8)

- Votre application doit être composée d'au moins cinq classes :
  - Place : un JavaBean composé des champs suivants :
    - Long id : L'identifiant unique du lieu
    - String name : Le nom du lieu
  - Trip : un JavaBean composé des champs suivants :
    - Long id : L'identifiant unique du voyage
    - Place departure : Le lieu de départ du voyage
    - Place destination : Le lieu de destination du voyage
    - BigDecimal price : Le prix du voyage



# Exercise (5/8)

- Votre application doit être composée d'au moins cinq classes :
  - JdbcPlaceDao : une implémentation de l'interface PlaceDao en utilisant JDBC
  - JdbcTripDao : une implémentation de l'interface TripDao en utilisant JDBC
  - Launcher : la classe contenant la méthode principale de l'application
- En option, vous pouvez créer une classe Factory pour vos DAO

# Exercise (6/8)

- JDBC

```
What do you want to do ?
1 - Add a place
2 - Find a place
3 - Edit a place
4 - Remove a place
5 - Add a trip
6 - Find a trip
7 - Remove a trip
8 - Quit

1
Name: Paris
Place added with the ID=2.
```

# Exercise (7/8)

- JDB

```
What do you want to do ?
1 - Add a place
2 - Find a place
3 - Edit a place
4 - Remove a place
5 - Add a trip
6 - Find a trip
7 - Remove a trip
8 - Quit

5
Departure: Please enter the id of the place : 2
Place : Paris
Destination: Please enter the id of the place : 1
Place : Tokyo
Price: 999
Trip added with the ID=2.
```

# Exercise (8/8)

- JDB

```
What do you want to do ?
1 - Add a place
2 - Find a place
3 - Edit a place
4 - Remove a place
5 - Add a trip
6 - Find a trip
7 - Remove a trip
8 - Quit

4
Please enter the id of the place : 1
Place : Tokyo
All trips with this Place will be removed with it.
 Are you sure you want to remove it ? [yes/no]
yes
Place removed !
```

# Fin

*Merci de votre attention*