الجامعة اللبنانية الدولية

LEBANESE INTERNATIONAL UNIVERSITY

CSIT461

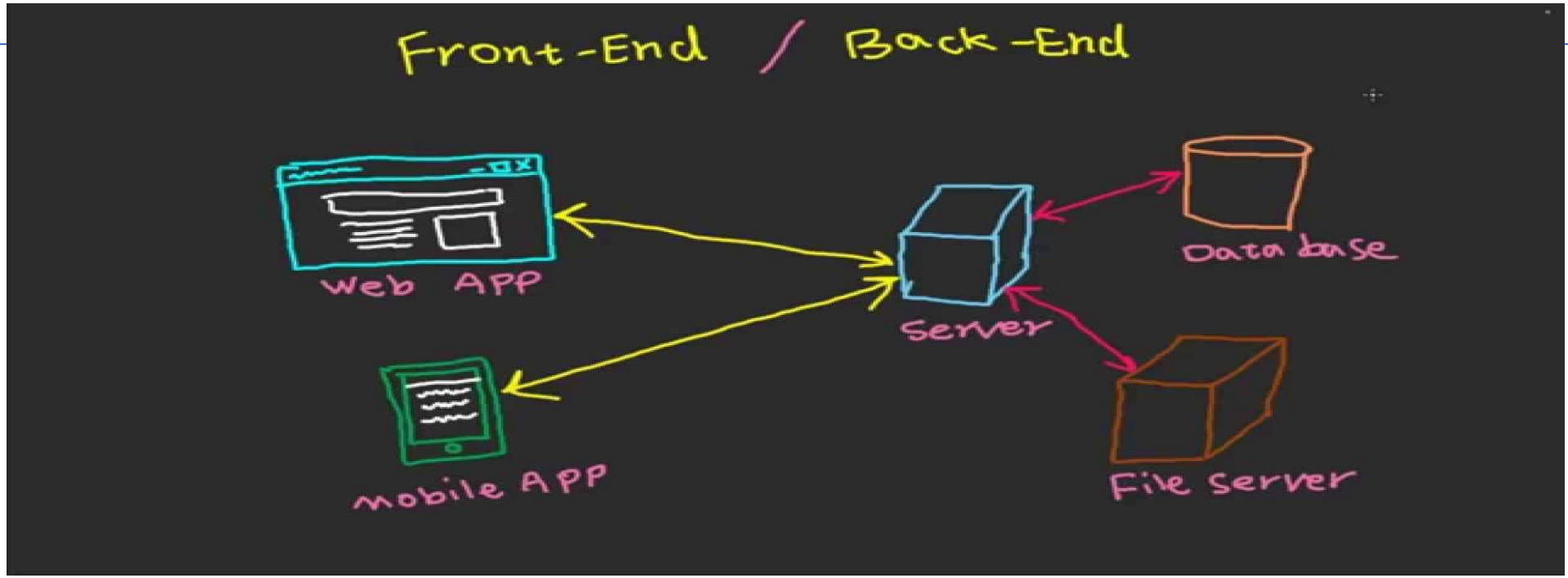
Communiquez avec la BDD en Java

EL BENANY Mohamed Mahmoud

20/12/2021

Communiquez avec une BDD MySQL

pour commencer, Nous allons, découvrir les nouveaux objets et interfaces qui doivent intervenir dans notre application afin d'établir une communication avec la base de données.



Communiquez avec une BDD MySQL en Java

Nous apprendrons :

- **à lire des données** depuis la base vers notre application,
- et **à écrire des données** depuis notre application vers la base.

Nous mettrons tout cela dans un exemple pratique
La technique employée:

1. Chargement du driver
2. Connexion à la base, création et exécution d'une requête
3. Mise en pratique

1. Chargement du driver

Nous avons, dans le chapitre précédent, récupéré le driver JDBC correspondant à MySQL, et nous l'avons ajouté au classpath du projet.
Il nous est maintenant nécessaire de procéder à ce que l'on nomme le chargement du driver depuis le code de notre application.
Voici le code minimal nécessaire

```
1. /* Chargement du driver JDBC pour ySQL */
2. try {
3.         Class.forName("com.mysql.cj.jdbc.Driver");
4. } catch ( ClassNotFoundException e )
5.  {
6. /* Gérer les éventuelles erreurs ici. */
7. }
```

1. Chargement du driver

la ligne 3 dont le rôle est justement de charger le driver.
Le nom de ce dernier,
en l'occurrence "`com.mysql.jdbc.Driver`", est fourni par son constructeur, autrement dit ici par MySQL.
Si vous utilisez un autre SGBD, vous devrez vous renseigner sur le site de son distributeur pour trouver son nom exact et ainsi pouvoir le charger depuis votre application.
Si cette ligne de code envoie une exception de type `ClassNotFoundException`, cela signifie que ***le fichier .jar*** contenant le driver JDBC pour MySQL n'a pas été correctement placé dans le *classpath*.
Vous pourrez d'ailleurs faire le test vous-mêmes lorsque nous passerons à la pratique, en retirant le driver que nous avons ajouté en tant que bibliothèque externe, et constater que cette ligne envoie bien une exception !

1. Chargement du driver

Nous allons maintenant apprendre à communiquer avec une base de données. Pour ce faire, nous devons suivre le processus suivant :

1. **Se connecter à la base ;**
2. **créer et exécuter une requête SQL ;**
3. **analyser son résultat ;**
4. **fermer la connexion.**

1. Chargement du driver

Nous allons maintenant apprendre à communiquer avec une base de données. Pour ce faire, nous devons suivre le processus suivant :

1. **Se connecter à la base ;**
2. **créer et exécuter une requête SQL ;**
3. **analyser son résultat ;**
4. **fermer la connexion.**

1. Se connecter à la base ;
2. créer et exécuter une requête SQL ;
3. analyser son résultat ;
4. fermer la connexion.

2. Connexion à la base, création et exécution d'une requête

Nous allons commencer par étudier la syntaxe et les objets à mettre en œuvre, puis nous appliquerons ensuite tout cela à travers un petit **scénario de tests**.

Connexion à la base de données

1. Identification de l'URL

Pour nous connecter à la base de données MySQL depuis notre application Java, nous avons besoin d'une URL spécifique à JDBC, qui respecte la syntaxe générale suivante :

jdbc:mysql://nomhote:port/nombdd

2. Connexion à la base, création et exécution d'une requête

jdbc:mysql://nomhote:port/nombdd

- **nomhote** : le nom de l'hôte sur lequel le serveur MySQL est installé. vous pouvez simplement spécifier **localhost** ou une adresse IP comme **127.0.0.1**.
- **port** : le port TCP/IP écouté par votre serveur MySQL. Par défaut, il s'agit du port **3306** ;
- **nombdd** : le nom de la base de données à laquelle vous souhaitez vous connecter. En l'occurrence, il s'agira pour nous de **classbd**.

Ainsi, puisque notre serveur MySQL est installé sur le même poste que notre serveur d'applications, l'URL finale sera dans notre cas :

```
jdbc:mysql://localhost:3306/classbd
```

2. Connexion à la base, création et exécution d'une requête

2. Établissement de la connexion

Après le chargement du driver, nous pouvons tenter d'établir une connexion avec notre base de données :

```
/* Connexion à la base de données */
String url = "jdbc:mysql://localhost:3306/classbd";
String user = "root";
String mdp = "";
Connection connexion = null;
try {
    connexion = DriverManager.getConnection( url, user, mdp );
    /* Ici, nous placerons nos requêtes vers la BDD */
    /* ... */
} catch ( SQLException e ) {
    /* Gérer les éventuelles erreurs ici */
}
```

2. Établissement de la connexion

L'établissement d'une connexion s'effectue à travers l'objet `DriverManager`.
Il suffit d'appeler sa méthode statique `getConnection()` pour récupérer un objet de type `Connection`.
Comme vous pouvez le voir ici, celle-ci prend en argument ***l'adresse de la base de données***, ***le nom d'utilisateur*** et **le mot de passe associé**.
L'appel à cette méthode peut retourner des erreurs de type `SQLException` :

- si une erreur **SQLException: No suitable driver** est envoyée, alors cela signifie que le driver JDBC n'a pas été chargé ou que l'URL n'a été reconnue par aucun des drivers chargés par votre application ;
- si une erreur **SQLException: Connection refused** ou **Connection timed out** ou encore **CommunicationsException: Communications link failure** est envoyée, alors cela signifie que la base de données n'est pas joignable.

Si un de ces derniers cas survient, vous trouverez ci-dessous une liste des causes possibles et les pistes de résolution associées :

2. Établissement de la connexion

Cause éventuelle	Piste de résolution
Le serveur MySQL est éteint ?	Démarrez le serveur MySQL…
Le numéro de port dans l'URL est manquant ou incorrect ?	Ouvrez le fichier de configuration my.cnf de votre serveur MySQL, et vérifiez le port qui y est spécifié.
Le nom d'hôte ou l'adresse IP dans l'URL est incorrect(e) ?	Testez la connectivité en effectuant un simple ping.
Le serveur MySQL n'accepte pas de connexions TCP/IP ?	Vérifiez que MySQL a été lancé sans l'option --skip-networking.
Il n'y a plus aucune connexion disponible sur le serveur MySQL ?	Redémarrez MySQL, et corrigez le code de votre application pour qu'il libère les connexions efficacement.
Quelque chose entre l'application Java et le serveur MySQL bloque la connexion, comme un pare-feu ou un proxy ?	Configurez votre pare-feu et/ou proxy pour qu'il(s) autorise(nt) le port écouté par votre serveur MySQL.
Le nom d'hôte dans l'URL n'est pas reconnu par votre serveur DNS local ?	Utilisez l'adresse IP dans l'URL au lieu du nom d'hôte, ou actualisez si possible votre DNS.

1. Se connecter à la base ;
2. créer et exécuter une requête SQL ;
3. analyser son résultat ;
4. fermer la connexion.

2. créer et exécuter une requête SQL ;

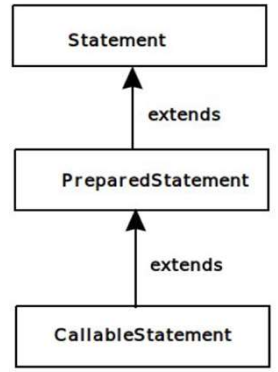
1. Création d'une requête

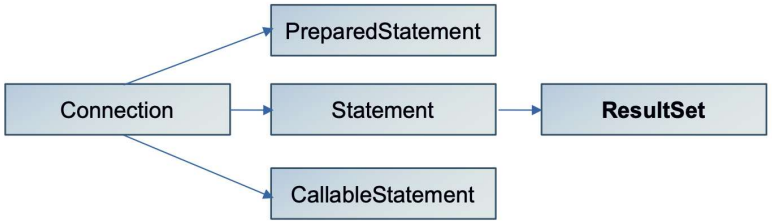
Avant de pouvoir créer des instructions SQL, vous devez tout d'abord créer un objet de type `Statement` ou `preparedStatment`.
Si vous parcourez la documentation de `Statement`, vous constaterez qu'il s'agit en réalité d'une interface dont le rôle est de permettre l'exécution de requêtes.
Pour initialiser cet objet, rien de plus simple, il suffit d'appeler la méthode `createStatement()` de l'objet `Connection` précédemment obtenu ! Donc si nous reprenons notre exemple, juste après l'établissement de la connexion dans notre bloc `try`, nous devons ajouter le code suivant :

```
try {
    connexion = DriverManager.getConnection( url, user, mdp );

    /* Ici, nous placerons nos requêtes vers la BDD */
    /* ... */
    /* Création de l'objet gérant les requêtes */
    Statement statement = connexion.createStatement();

} catch ( SQLException e ) {
    /* Gérer les éventuelles erreurs ici */
}
```





2. créer et exécuter une requête SQL ;

2. Exécution de la requête

Une fois l'objet `Statement` initialisé, il devient alors possible d'exécuter une requête. Pour ce faire, celui-ci met à votre disposition toute une série de méthodes, notamment les deux suivantes :

- `executeQuery()` : cette méthode est dédiée à la lecture de données via une requête de type `SELECT` ;
- `executeUpdate()` : cette méthode est réservée à l'exécution de requêtes ayant un effet sur la base de données (écriture ou suppression), typiquement les requêtes de type `INSERT`, `UPDATE`, `DELETE`, etc.

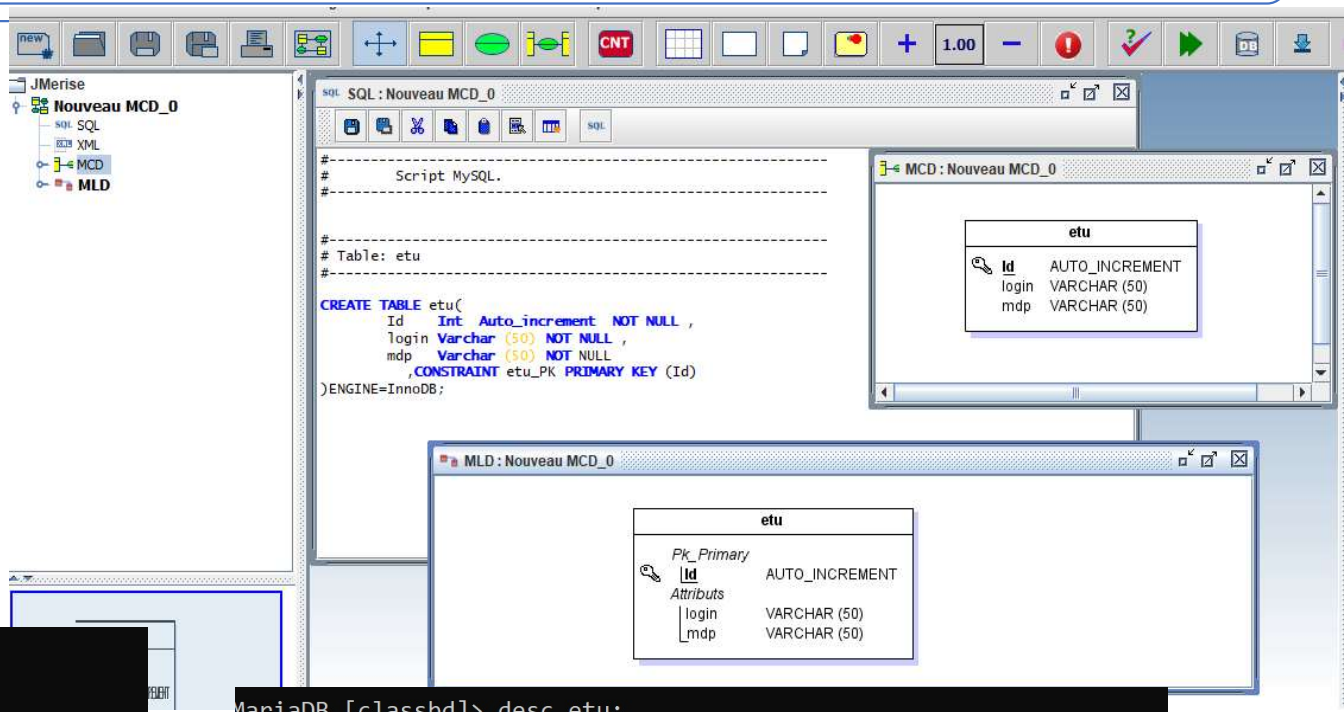
En outre, il existe des variantes de chaque méthode prenant en compte d'autres arguments, ainsi que deux autres méthodes nommées `execute()` et `executeBatch()`. Nous n'allons pas nous attarder sur les subtilités mises en jeu, je vous laisse le soin de lire la documentation de l'objet `Statement` si vous souhaitez en savoir davantage.

2. créer et exécuter une requête SQL ;

2. Exécution de la requête

On va créer la table etu dans notre base de Donnée classbd;

```
#------------------------------------------------------------
#      Script MySQL.
#------------------------------------------------------------
#------------------------------------------------------------
# Table: etu
#------------------------------------------------------------
CREATE TABLE etu(
    Id    Int  Auto_increment  NOT NULL ,
    login Varchar (50) NOT NULL ,
    mdp   Varchar (50) NOT NULL
          ,CONSTRAINT etu_PK PRIMARY KEY (Id)
)ENGINE=InnoDB;
```




```
MariaDB [(none)]> use classbd;
Database changed
MariaDB [classbd]> CREATE TABLE etu(
    ->         Id    Int  Auto_increment  NOT NULL ,
    ->         login Varchar (50) NOT NULL ,
    ->         mdp   Varchar (50) NOT NULL
    -> ,CONSTRAINT etu_PK PRIMARY KEY (Id)
    -> )ENGINE=InnoDB;
Query OK, 0 rows affected, 1 warning (0.069 sec)

MariaDB [classbd]>
```

```
MariaDB [classbd]> desc etu;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| Id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| login | varchar(50) | NO   |     | NULL    |                |
| mdp   | varchar(50) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
3 rows in set (0.022 sec)

MariaDB [classbd]>
```

2. créer et exécuter une requête SQL ;

2. Exécution de la requête

Exécution d'une requête de lecture

En parcourant la documentation de la méthode `executeQuery()`, nous apprenons qu'elle retourne un objet de type `ResultSet` contenant le résultat de la requête. Voici donc un exemple effectuant un `SELECT` sur notre table **etu** :

```
try {
    connexion = DriverManager.getConnection( url, user, mdp );
    /* Ici, nous placerons nos requêtes vers la BDD */
    /* ... */
    /* Création de l'objet gérant les requêtes */
    Statement statement = connexion.createStatement();
    /* Exécution d'une requête de lecture */
    ResultSet resultat = statement.executeQuery( "SELECT id, login, mdp  FROM etu;"
);

} catch ( SQLException e ) {
    /* Gérer les éventuelles erreurs ici */
}
```

2. créer et exécuter une requête SQL ;

2. Exécution de la requête

Exécution d'une requête d'écriture

En parcourant cette fois la documentation de la méthode `executeUpdate()`, nous apprenons qu'elle retourne un entier représentant le nombre de lignes affectées par la requête réalisée.
Si par exemple vous réalisez une insertion de données via un `INSERT`, cette méthode retournera `0` en cas d'échec et `1` en cas de succès.
Si vous réalisez une mise à jour via un `UPDATE`, cette méthode retournera le nombre de lignes mises à jour. Idem en cas d'une suppression via un `DELETE`, etc.
Je vous propose ici un exemple effectuant un `INSERT` sur notre table **etu** :

```
try {
    connexion = DriverManager.getConnection( url, user, mdp );

    /* Ici, nous placerons nos requêtes vers la BDD */
    /* ... */
    /* Création de l'objet gérant les requêtes */
    Statement statement = connexion.createStatement();
   /* Exécution d'une requête d'écriture */
    int statut = statement.executeUpdate( "INSERT INTO etu (id, login, mdp) VALUES (1, 'benany', 'ben123');" );

} catch ( SQLException e ) {
    /* Gérer les éventuelles erreurs ici */
}
```

1. Se connecter à la base ;
2. créer et exécuter une requête SQL ;
3. analyser son résultat ;
4. Libération des ressources.

3. analyser son résultat ;

3. Accès aux résultats de la requête

Retour d'une requête de lecture

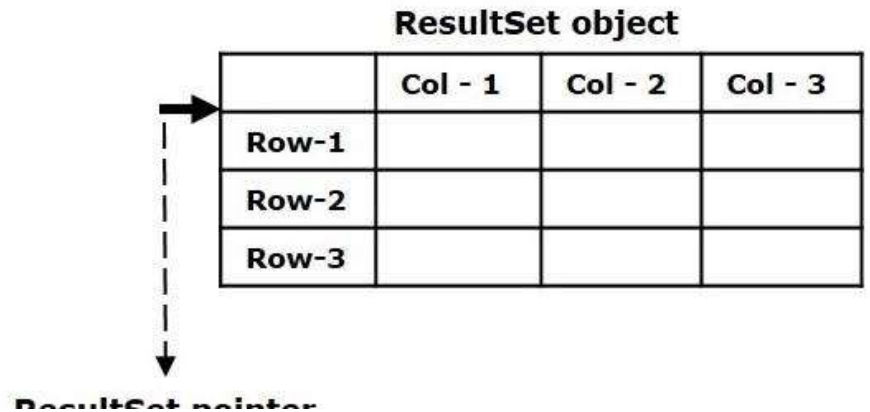
L'exécution d'une requête de lecture via la méthode `statement.executeQuery()` retourne un objet de type `ResultSet`.
Vous pouvez le voir comme un tableau, qui contient les éventuelles données retournées par la base de données sous forme de lignes.
Pour accéder à ces lignes de données, vous avez à votre disposition un curseur, que vous pouvez déplacer de ligne en ligne.
Notez bien qu'il ne s'agit pas d'un curseur au sens base de données du terme, mais bien d'un curseur propre à l'objet `ResultSet`.
Voyons cela dans un exemple, puis commentons :

```
try {
    connexion = DriverManager.getConnection( url, user, mdp );

    /* Exécution d'une requête de lecture */
    ResultSet resultat = statement.executeQuery( "SELECT id, login, mdp FROM etu;" );

    /* Récupération des données du résultat de la requête de lecture */
    while ( resultat.next() ) {
        int idEtu = resultat.getInt( "id" );
        String loginEtu = resultat.getString( "login" );
        String mdpEtu = resultat.getString( "mdp" );

        /* Traiter ici les valeurs récupérées. */
    }
} catch ( SQLException e ) {
    /* Gérer les éventuelles erreurs ici */
}
}
```



3. analyser son résultat ;

3. Accès aux résultats de la requête

Une fois le curseur positionné correctement, il ne nous reste plus qu'à récupérer les contenus des différents champs via une des nombreuses méthodes de récupération proposées par l'objet `ResultSet`.

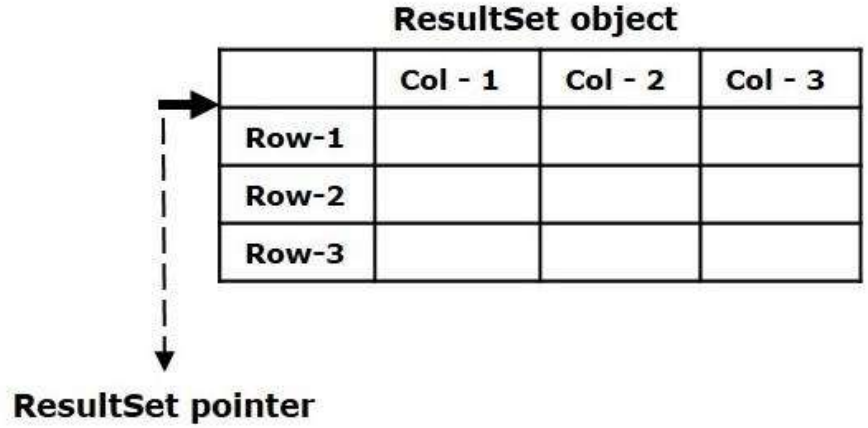
Je ne vais pas vous en faire la liste exhaustive, je vous laisse parcourir la documentation pour les découvrir en intégralité.
Sachez simplement qu'il en existe une par type de données récupérables :

- une méthode `resultat.getInt()` pour récupérer un entier ;
- une méthode `resultat.getString()` pour récupérer une chaîne de caractères ;
- une méthode `resultat.getBoolean()` pour récupérer un booléen ;

Exemple:

```
while ( resultat.next() ) {
        int idEtu = resultat.getInt( "id" );
        String loginEtu = resultat.getString( "login" );
        String mdpEtu = resultat.getString( "mdp" );

}
```



1. Se connecter à la base ;
2. créer et exécuter une requête SQL ;
3. analyser son résultat ;
4. Libération des ressources.

2. créer et exécuter une requête SQL ;

4. Libération des ressources

Tout comme il est nécessaire de fermer proprement une connexion ouverte, il est extrêmement recommandé de disposer proprement des objets `Statement` et `ResultSet` initialisés au sein d'une connexion :

```
try {
    /*
     * Ouverture de la connexion, initialisation d'un Statement, initialisation d'un ResultSet, etc.
     */
} catch ( SQLException e ) {
    /* Traiter les erreurs éventuelles ici. */
} finally {
    if ( resultat != null ) {
        try {
            /* On commence par fermer le ResultSet */
            resultat.close();
        } catch ( SQLException ignore ) {
        }
    }
    if ( statement != null ) {
        try {
            /* Puis on ferme le Statement */
            statement.close();
        } catch ( SQLException ignore ) {
        }
    }
    if ( connexion != null ) {
        try {
            /* Et enfin on ferme la connexion */
            connexion.close();
        } catch ( SQLException ignore ) {
        }
    }
```

3. Mise en pratique

Nous venons d'étudier les étapes principales de la communication avec une base de données, qui pour rappel sont :

1. la connexion à la base ;
2. la création et l'exécution d'une requête SQL ;
3. la récupération de son résultat ;
4. la fermeture des différentes ressources mises en jeu.

3. Mise en pratique

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class TestConClassbd {
	public static void main(String[] args) {
		/* Connexion à la base de données */
		String driver = "com.mysql.cj.jdbc.Driver";
		String url = "jdbc:mysql://localhost:3306/classbd";
		String user = "root";
		String mdp = "";
		Connection connexion = null;
		Statement statement = null;
		ResultSet resultat = null;
		try {
			Class.forName(driver);
			connexion = DriverManager.getConnection(url, user, mdp);
			System.out.println("Connecté");
			/* Création de l'objet gérant les requêtes */
			statement = connexion.createStatement();
			/* Exécution d'une requête de lecture */
			resultat = statement.executeQuery("SELECT id, login, mdp
			FROM etu;");
			/* Récupération des données du résultat de la requête de
			lecture */
			while (resultat.next()) {
			int idEtu = resultat.getInt("id");
			String loginEtu = resultat.getString("login");
			String mdpEtu = resultat.getString("mdp");
			System.out.print("Etu : [ID = " + idEtu + " , Login = "
			+ loginEtu + ", mdp = " + mdp);
			/* Traiter ici les valeurs récupérées. */
	}
```

```
} catch (ClassNotFoundException e) {
/* Traiter les erreurs Class.forName ici. */
} catch (SQLException e) {
/* Traiter les erreurs DriverManager ici. */
} finally {
	if (resultat != null) {
		try {
			/* On commence par fermer le ResultSet */
			resultat.close();
		} catch (SQLException ignore) {
		}
	}
	if (statement != null) {
		try {
			/* Puis on ferme le Statement */
			statement.close();
		} catch (SQLException ignore) {
		}
	}
if (connexion != null) {
	try {
		/* Et enfin on ferme la connexion */
		connexion.close();
	} catch (SQLException ignore) {
	}
}
}

}
}
```