

1.3.7. Ressource R3.07 : SQL dans un langage de programmation

Compétence ciblée :

- Concevoir, gérer, administrer et exploiter les données de l'entreprise et mettre à disposition toutes les informations pour un bon pilotage de l'entreprise

SAÉ au sein de laquelle la ressource peut être mobilisée et combinée :

- SAÉ 3.Integ.01 | Gestion de projet et développement logiciel

Descriptif :

L'objectif de cette ressource consiste à étudier les différents aspects de l'intégration du langage SQL dans les langages de programmation. Cette ressource permet de comprendre tous les usages standards de la base de données hors aspect interactif en mode applicatif comme les applications web ou les applications mobiles, batch et procédure stockées.

Savoirs de référence étudiés

- SQL intégré dans un langage de programmation (par ex. : PL/SQL, JDBC, PDO, JPA, SPRING...)
- Procédures, Curseurs, Triggers, exception
- Transactions et gestion de la concurrence d'accès
- Persistance des données
- Index et optimisation

Prolongements suggérés

- Les différents savoirs de référence pourront être approfondis

Apprentissages critiques ciblés :

- AC24.01 | Optimiser les modèles de données de l'entreprise
- AC24.02 | Assurer la sécurité des données (intégrité et confidentialité)
- AC24.03 | Organiser la restitution de données à travers la programmation et la visualisation

Mots clés :

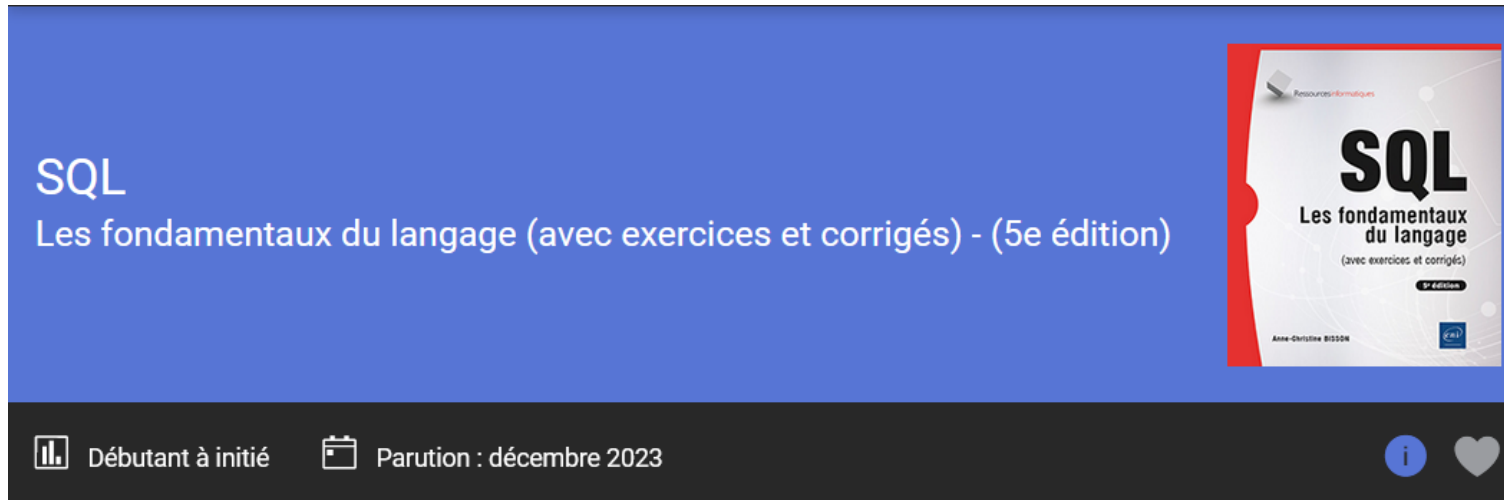
SQL – SQL intégré – trigger – procédures stockées – optimisation de requêtes

Volume horaire :

Volume horaire défini nationalement : 24 heures dont 14 heures de TP

Documents

- Cours de bases de données : <http://sql.bdpedia.fr/>
- Cours du CNAM : <http://orm.bdpedia.fr/>
- Introduction au langage SQL : <https://enseignement.alexandre-mesle.com/sql/>
- Bibliothèque numérique ENI : <https://bases-doc.univ-lorraine.fr/login?qurl=https%3A//www.eni-training.com/cs/univ-lorraine/>



Outils

- NotebookLM : <https://notebooklm.google/>
- ChatGPT avec la BDD : <https://www.askyourdatabase.com/>
- Introduction au langage SQL : <https://enseignement.alexandre-mesle.com/sql/>

BDD et représentation graphique

- drawDB : <https://drawdb.vercel.app/editor> (on-line)
- MySQL Workbench : <https://www.mysql.com/products/workbench/> (off-line)
- Oracle Database Free : <https://www.oracle.com/database/free/get-started/#quick-start> (professionnel)
- JMerise : <https://www.jfreesoft.com/JMerise/> (amateur)

Eclipse ou IntelliJ

- Langage : Java 21 LTS
- Gestion de projets : Maven ou Gradle
- Spring Boot
- Connecteurs JDBC

BDD

- MySQL
- Outil : MySQL Workbench

IHM

- JavaFX
- Hilla/React
- Vaadin

Liens

- <https://start.vaadin.com/app/p>
- <http://sql.bdpedia.fr/intro.html>
- <http://sql.bdpedia.fr/genindex.html>
- <http://sys.bdpedia.fr/>
- <http://b3d.bdpedia.fr/>
- <http://orm.bdpedia.fr/>
- <http://orm.bdpedia.fr/index.html>
- <http://sql.bdpedia.fr/conception.html#chap-ea> (conception d'une BDD)
- <https://deptfod.cnam.fr/bd/tp/>
- <https://deptfod.cnam.fr/bd/tp/datasets/>

MYSQL Workbench

- Manuel de référence : <https://dev.mysql.com/doc/workbench/en/>
- <https://fsmrel.developpez.com/basesrelationnelles/workbench/>
- https://ocw.cs.pub.ro/courses/_media/ewis/laboratoare/workbench.pdf

- Alternative : <https://dbeaver.io/>
- Autre alternative : <https://dbschema.com/index.html>

MySQL dans Windows

```
C:\Program Files\MySQL\MySQL Workbench 8.0 CE>mysql --user=root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 17
Server version: 8.0.42 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.


Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.


Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

MySQL Connections

Local instance MySQL80

 root

 localhost:3306

Procédures et déclencheurs

- <https://start.vaadin.com/app/p>
- <http://sql.bdpedia.fr/procedures.html#>
- <http://sql.bdpedia.fr/etudecas.html>
- <https://deptfod.cnam.fr/bd/tp/transactions/>

Avantages

- **Simplification :**

un même code qui doit souvent être effectuée peut être enregistré afin d'être appelé rapidement.

- **Amélioration des performances :**

les opérations peuvent être exécutées du côté du serveur de base de données et envoyées directement prêtes à l'emploi par la solution informatique qui va utiliser ces données. Par ailleurs, cela va réduire les échanges entre le client et le serveur.

- **Sécurisation :**

des applications peuvent avoir accès uniquement aux procédures stockées, sans avoir accès aux données des tables directement, et/ou s'assurer que l'accès aux données soit toujours effectué de la même manière.

→ Mais l'écriture varie légèrement selon les BDD (MySQL, PostgreSQL, ...)

Procédures stockées

Création d'une table EMPLOYEE :

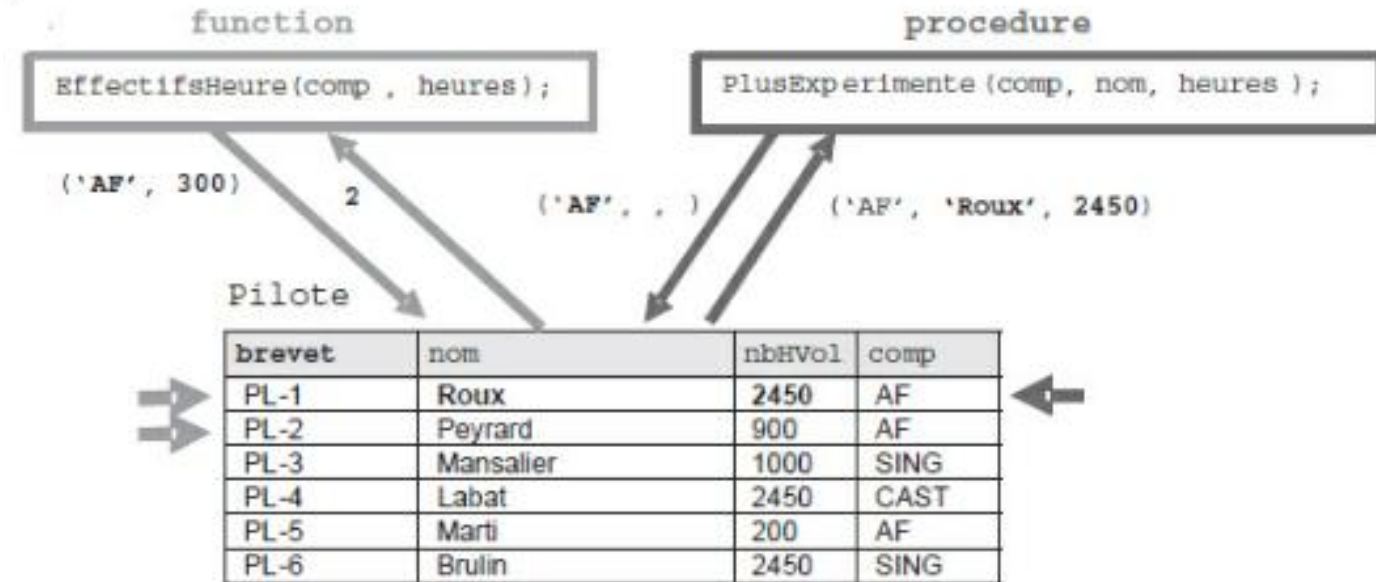
```
CREATE TABLE EMPLOYEE (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL  
);
```

Objectif :

Gérer cette table à partir d'une architecture **REST** et des **endpoints** (points de terminaison).

BDD MySQL

```
CREATE { PROCEDURE | FUNCTION }  
    nomSousProgramme [(...)] [RETURNS typeMySQL]  
BEGIN  
    [DECLARE déclaration]; ...  
    instructions MySQL;  
  
    BEGIN  
        [DECLARE déclaration]; ...  
        ...  
        instructions MySQL;  
    END;  
...  
END;
```



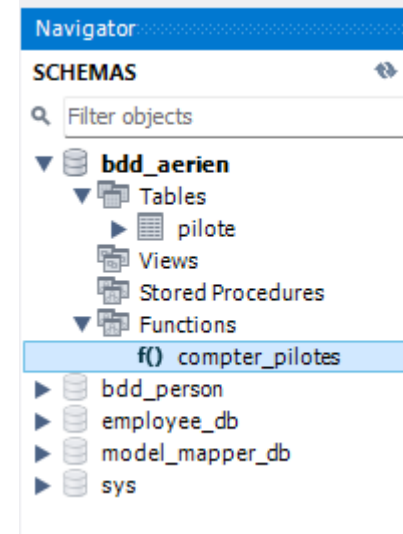
```
INSERT INTO Pilote (brevet, nom, nbHVol, comp)  
VALUES  
('PL-1', 'Roux', '2450', 'AF'),  
('PL-2', 'Peyrard', '800', 'AF'),  
('PL-3', 'Mansalier', '1000', 'SING'),  
('PL-4', 'Labat', '2450', 'CAST'),  
('PL-5', 'Marti', '200', 'AF'),  
('PL-6', 'Brulin', '2450', 'SING');
```

Result Grid					Filter Rows:	Edit:
	brevet	nom	nbHVol	comp		
	PL-1	Roux	2450.00	AF		
	PL-2	Peyrard	800.00	AF		
	PL-3	Mansalier	1000.00	SING		
	PL-4	Labat	2450.00	CAST		
	PL-5	Marti	200.00	AF		
	PL-6	Brulin	2450.00	SING		
»*	NULL	NULL	NULL	NULL		

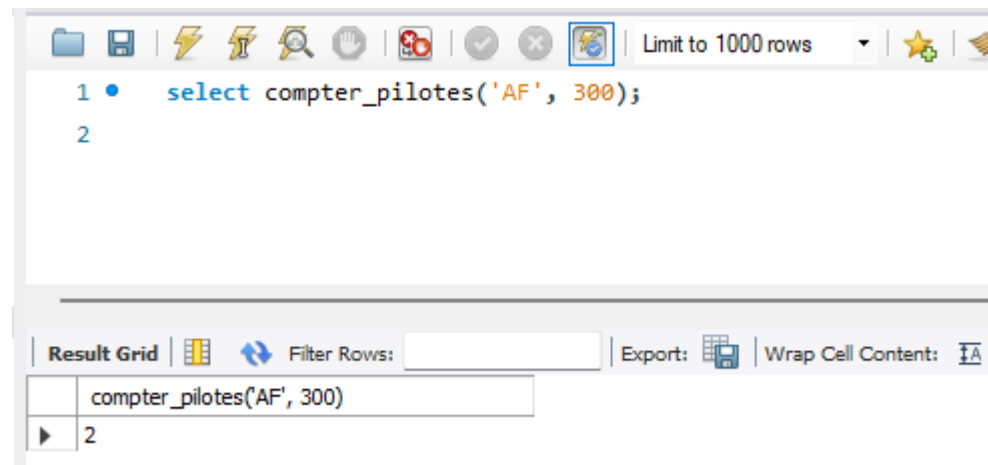
Fonctions

Fonction pour retourner le nombre de pilotes d'une compagnie donnée (1^{er} param.) qui ont plus d'heures de vols que la valeur du 2^{ème} param.

```
1 • CREATE DEFINER='root'@'localhost' FUNCTION `compter_pilotes`(  
2     pcomp varchar(6),  
3     pheuresVol DECIMAL(10,2)  
4 ) RETURNS smallint  
5     READS SQL DATA  
6 BEGIN  
7     DECLARE resultat smallint;  
8  
9     IF pcomp IS NULL THEN  
10         SELECT COUNT(*) INTO resultat  
11         FROM Pilote  
12         WHERE nbHVol > pheuresVol;  
13     ELSE  
14         SELECT COUNT(*) INTO resultat  
15         FROM Pilote  
16         WHERE nbHVol > pheuresVol  
17             AND comp = pcomp;  
18     END IF;  
19  
20     RETURN resultat;  
21 END
```



```
mysql> select bdd_aerien.compter_pilotes('AF',300);  
+-----+  
| bdd_aerien.compter_pilotes('AF',300) |  
+-----+  
|                                     2 |  
+-----+  
1 row in set (0.00 sec)
```



Procédures stockées

Procédure pour retourner tous les pilotes d'une compagnie donnée :

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `lister_tous_pilotes_compagnie` (IN pcomp VARCHAR(20))
2 BEGIN
3     SELECT nom AS Nom, nbHVol AS HeuresVol
4     FROM Pilote
5     WHERE comp = pcomp;
6 END
```

Limit to 1000 rows

```
1 • call bdd_aerien.lister_tous_pilotes_compagnie('AF');
2
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Nom	HeuresVol		
►	Roux	2450.00		
	Peyrard	800.00		
	Marti	200.00		

```
mysql> call bdd_aerien.lister_tous_pilotes_compagnie('AF');
+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Roux  | 2450.00   |
| Peyrard | 800.00    |
| Marti | 200.00    |
+-----+-----+
3 rows in set (0.00 sec)



Query OK, 0 rows affected (0.01 sec)
```

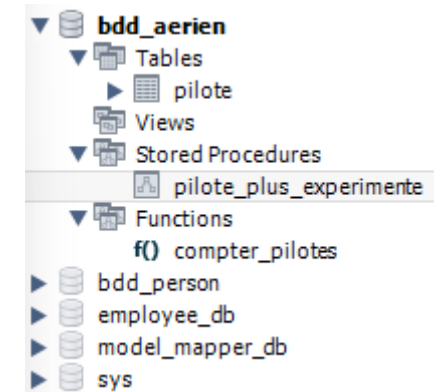
Procédures stockées

Procédure pour retourner le pilote d'une compagnie donnée qui a le plus d'heures de vols :
« le pilote le plus expérimenté ».

```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `pilote_plus_experimente`(IN pcomp VARCHAR(6))
2 BEGIN
3     IF pcomp IS NULL OR pcomp = '' THEN
4         SELECT *
5         FROM Pilote
6         ORDER BY nbHVol DESC
7         LIMIT 1;
8     ELSE
9         SELECT *
10        FROM Pilote
11        WHERE comp = pcomp
12        ORDER BY nbHVol DESC
13        LIMIT 1;
14    END IF;
15 END
```

```
1 • call bdd_aerien.pilote_plus_experimente('AF');
2
```

Result Grid				
Filter Rows: <input type="text"/>				
Export:  Wrap Cell Content: 				
	brevet	nom	nbHVol	comp
▶	PL-1	Roux	2450.00	AF



```
mysql> call bdd_aerien.pilote_plus_experimente('AF');
+-----+-----+-----+-----+
| brevet | nom  | nbHVol | comp |
+-----+-----+-----+-----+
| PL-1   | Roux | 2450.00 | AF   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

Procédures stockées

```
-- Procédure pour ajouter un employé (SQL)
DELIMITER //

CREATE PROCEDURE AddEmployee(
    IN p_first_name VARCHAR(50),
    IN p_last_name VARCHAR(50)
)
BEGIN
    INSERT INTO EMPLOYEE (FIRST_NAME, LAST_NAME)
    VALUES (p_first_name, p_last_name);
END;
//
DELIMITER ;

-- Utilisation:
CALL AddEmployee('John', 'Doe');
```

```
-- Procédure pour ajouter un employé (H2)
CREATE ALIAS AddEmployee AS $$
import java.sql.*;

@CODE
void addEmployee(Connection conn, String firstName, String
lastName) throws SQLException {
    try (PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO EMPLOYEE (FIRST_NAME, LAST_NAME)
VALUES (?, ?)") {
        stmt.setString(1, firstName);
        stmt.setString(2, lastName);
        stmt.executeUpdate();
    }
}
$$;
```

H2 ne supporte pas DELIMITER et utilise Java (avec ALIAS) pour simuler des procédures.

Définition

Un curseur est un mécanisme SQL qui permet de parcourir le résultat d'une requête ligne par ligne.

C'est une structure de données temporaire qui stocke le résultat d'une requête (mise en cache côté serveur).

Le programme peut alors itérer sur le curseur pour traiter chaque ligne individuellement renvoyée par un SELECT.

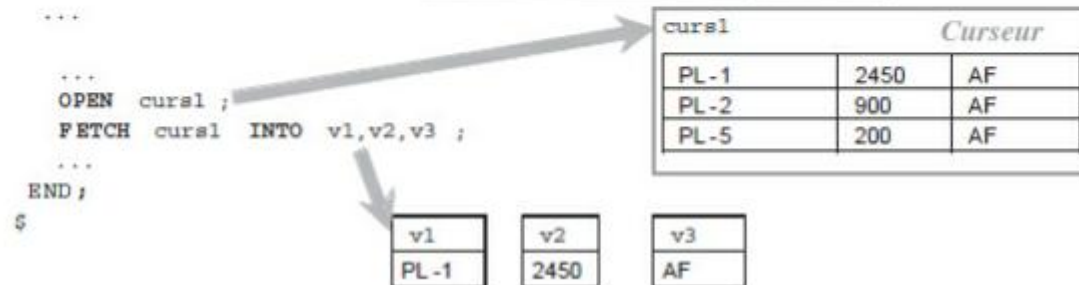
Utile lorsque le jeu de résultats est volumineux, car cela évite de charger l'intégralité du jeu de résultats en mémoire.

Principe d'un curseur

Tout curseur MySQL dispose des propriétés suivantes :

- lecture seule : aucune modification dans la base n'est possible à travers ce dernier ;
- non navigable : une fois ouvert, le curseur est parcouru du début à la fin sans pouvoir revenir à l'enregistrement précédent ;
- insensible (asensitive) : toute mise à jour opérée dans la base n'est pas répercutée dans le curseur une fois ouvert.

```
CREATE PROCEDURE nomSousProgramme [(...)]  
BEGIN  
  DECLARE v1...;  
  DECLARE v2...;  
  DECLARE v3...;  
  DECLARE CURSOR curs1 FOR SELECT brevet,nbhVol, comp  
                             FROM Pilote WHERE comp = 'AF' ;  
  ...  
  OPEN curs1 ;  
  FETCH curs1 INTO v1,v2,v3 ;  
  ...  
END ;  
$
```



Principe d'un curseur

CURSOR FOR requête;	Déclaration du curseur.
OPEN nomCurseur;	Ouverture du curseur (chargement des lignes).
FETCH nomCurseur INTO listeVariables;	Positionnement sur la ligne suivante et chargement de l'enregistrement courant dans une ou plusieurs variables.
CLOSE nomCurseur	Ferme le curseur.

Avantages

Traitement ligne par ligne : utile lorsque vous devez effectuer des opérations complexes sur chaque ligne, comme des calculs ou des mises à jour.

Gestion de la mémoire : Les curseurs peuvent être utilisés pour gérer efficacement la mémoire lors du traitement de grands ensembles de résultats. Au lieu de charger l'ensemble du résultat en mémoire, le curseur ne charge qu'une seule ligne à la fois.

Isolation des données : Les curseurs peuvent fournir un certain niveau d'isolation des données. Cela signifie que les modifications apportées à la base de données par d'autres utilisateurs ou programmes n'affecteront pas le résultat du curseur.

Déclarations

`CURSOR nom_curseur IS requête_sql;`

- `nom_curseur`: Le nom que vous donnez au curseur.
- `requête_sql`: La requête SQL qui définit le résultat du curseur.

Exemple :

`CURSOR curseur_voyageurs IS`

`SELECT * FROM Voyageur WHERE lieu = 'Corse';`

Ce curseur sélectionne toutes les lignes de la table `Voyageur` où la colonne `lieu` est égale à `'Corse'`.

Curseurs paramétrés

Vous pouvez également déclarer des curseurs avec des paramètres. Cela permet de créer des curseurs plus flexibles qui peuvent être réutilisés avec différentes valeurs de paramètres.

Exemple :

```
CURSOR curseur_voyageurs (p_lieu VARCHAR2) IS  
SELECT * FROM Voyageur WHERE lieu = p_lieu;
```

Ce curseur sélectionne toutes les lignes de la table Voyageur où la colonne lieu est égale à la valeur du paramètre p_lieu.

Exécution d'un curseur

L'exécution d'un curseur se fait en trois phases :

- 1. Ouverture du curseur:** La requête SQL associée au curseur est exécutée.
- 2. Parcours du résultat :** Le curseur est parcouru ligne par ligne. Pour chaque ligne, les valeurs des colonnes sont récupérées et peuvent être traitées par le programme.
- 3. Fermeture du curseur :** Le curseur est fermé, libérant les ressources associées

Exemple (Affiche en sortie console la concaténation des champs nom et prénom trouvés dans v_voyageur)

```
-- Ouverture du curseur
OPEN curseur_voyageurs;
-- Boucle de parcours du curseur
LOOP
    FETCH curseur_voyageurs INTO v_voyageur;

    EXIT WHEN curseur_voyageurs%NOTFOUND;

    -- Traitement de la ligne
    DBMS_OUTPUT.PUT_LINE(v_voyageur.nom || ' ' || v_voyageur.prenom);
END LOOP;

-- Fermeture du curseur
CLOSE curseur_voyageurs;
```

Exemple (met à jour la colonne code de chaque ligne pour la mettre en majuscules)

```
-- Déclaration du curseur
CURSOR curseur_genre IS
SELECT * FROM Genre;

-- Boucle de parcours du curseur
FOR v_genre IN curseur_genre LOOP
    UPDATE Genre SET code = UPPER(code) WHERE CURRENT OF curseur_genre;
END LOOP;
```

Modifie les lignes parcourues par un curseur en utilisant la clause **CURRENT OF** dans une instruction **UPDATE** ou **DELETE**.

CURSEUR : EXEMPLE COMPLET

CURSOR MonCurseur IS

SELECT * FROM Voyageur WHERE lieu = 'Corse';

-- Déclaration d'une variable pour stocker le n-uplet courant

v_voyageur Voyageur%ROWTYPE;

BEGIN

-- Ouverture du curseur

OPEN MonCurseur;

-- Parcours du curseur

LOOP

FETCH MonCurseur INTO v_voyageur;

EXIT WHEN MonCurseur%NOTFOUND;

-- Traitement du n-uplet courant

DBMS_OUTPUT.PUT_LINE(v_voyageur.nom);

END LOOP;

-- Fermeture du curseur

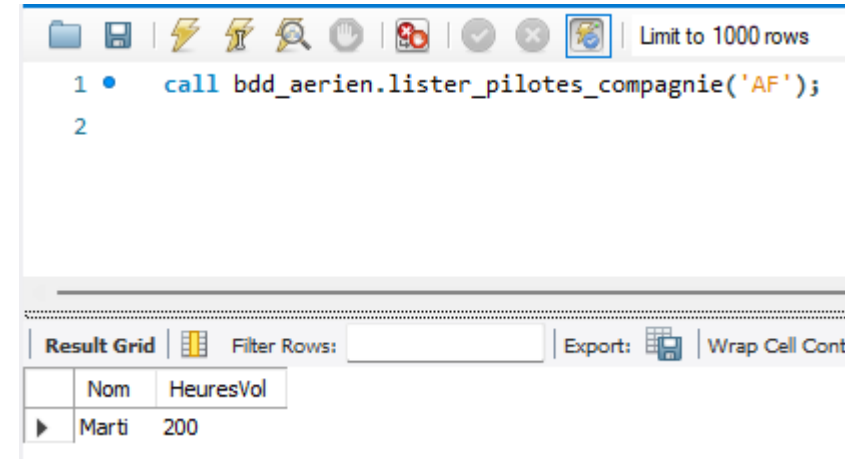
CLOSE MonCurseur;

END;

Curseurs (procédures stockées)

Retourne chaque pilote d'une compagnie donnée.

```
1 • CREATE DEFINER=`root`@`localhost` PROCEDURE `lister_pilotes_compagnie`(IN pcomp VARCHAR(20))
2 BEGIN
3     DECLARE fini INT DEFAULT 0;
4     DECLARE v_nom VARCHAR(50);
5     DECLARE v_nbHVol INT;
6
7     -- Curseur limité à la compagnie donnée
8     DECLARE curPilote CURSOR FOR
9         SELECT nom, nbHVol
10        FROM Pilote
11        WHERE comp = pcomp;
12
13     -- Gestion fin de données
14     DECLARE CONTINUE HANDLER FOR NOT FOUND SET fini = 1;
15
16     OPEN curPilote;
17
18     boucle: LOOP
19         FETCH curPilote INTO v_nom, v_nbHVol;
20         IF fini = 1 THEN
21             LEAVE boucle;
22         END IF;
23
24         -- Traitement : on renvoie les infos du pilote
25         SELECT v_nom AS Nom, v_nbHVol AS HeuresVol;
26     END LOOP;
27
28     CLOSE curPilote;
29 END
```

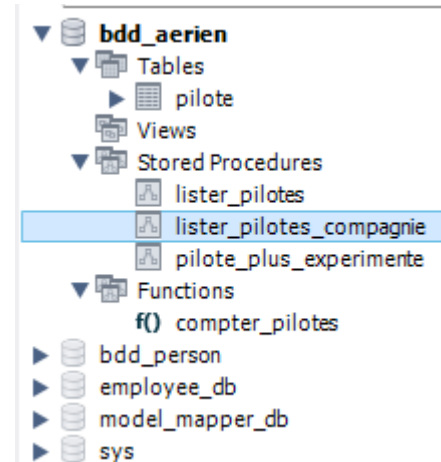


Limit to 1000 rows

```
1 • call bdd_aerien.lister_pilotes_compagnie('AF');
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

	Nom	HeuresVol
▶	Marti	200



```
mysql> call bdd_aerien.lister_pilotes_compagnie('AF');
+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Roux  |      2450 |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Peyrard |      800 |
+-----+-----+
1 row in set (0.00 sec)

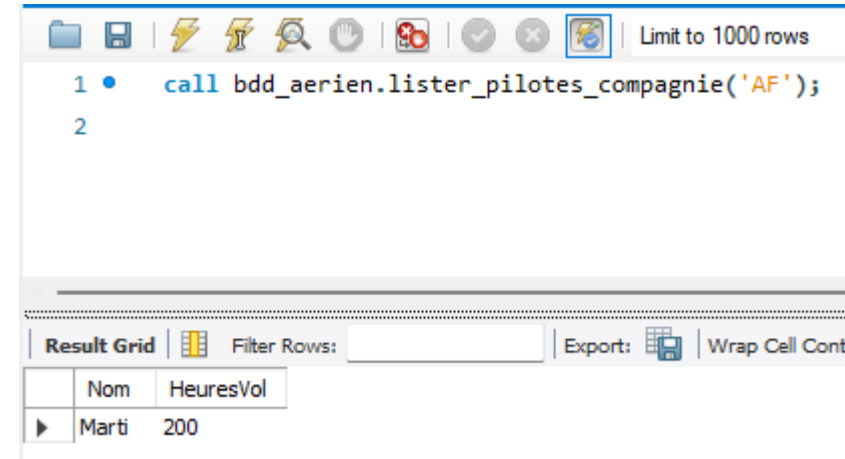
+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Marti |      200 |
+-----+-----+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Curseurs (procédures stockées)

Retourne chaque pilote d'une compagnie donnée.

```
1 • CREATE DEFINER=`root`@`localhost` PROCEDURE `lister_pilotes_compagnie`(IN pcomp VARCHAR(20))
2 BEGIN
3     DECLARE fini INT DEFAULT 0;
4     DECLARE v_nom VARCHAR(50);
5     DECLARE v_nbHVol INT;
6
7     -- Curseur limité à la compagnie donnée
8     DECLARE curPilote CURSOR FOR
9         SELECT nom, nbHVol
10        FROM Pilote
11        WHERE comp = pcomp;
12
13     -- Gestion fin de données
14     DECLARE CONTINUE HANDLER FOR NOT FOUND SET fini = 1;
15
16     OPEN curPilote;
17
18     boucle: LOOP
19         FETCH curPilote INTO v_nom, v_nbHVol;
20         IF fini = 1 THEN
21             LEAVE boucle;
22         END IF;
23
24         -- Traitement : on renvoie les infos du pilote
25         SELECT v_nom AS Nom, v_nbHVol AS HeuresVol;
26     END LOOP;
27
28     CLOSE curPilote;
29 END
```

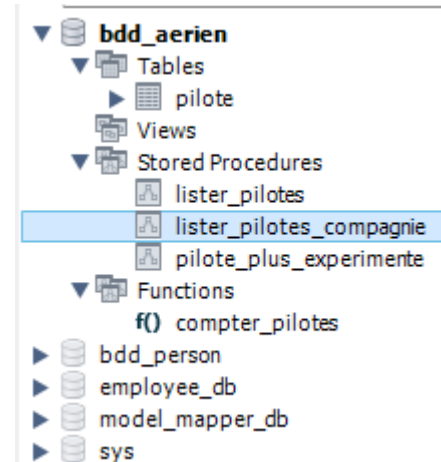


Limit to 1000 rows

```
1 • call bdd_aerien.lister_pilotes_compagnie('AF');
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Cont

	Nom	HeuresVol
▶	Marti	200



```
mysql> call bdd_aerien.lister_pilotes_compagnie('AF');
+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Roux  |      2450 |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Peyrard |      800 |
+-----+-----+
1 row in set (0.00 sec)

+-----+-----+
| Nom   | HeuresVol |
+-----+-----+
| Marti |      200 |
+-----+-----+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Curseurs avec exception

Retourne chaque pilote d'une compagnie donnée.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lister_pilotes_compagnie_exception`(IN pcomp
VARCHAR(6))
BEGIN
    DECLARE fini INT DEFAULT 0;
    DECLARE v_nom VARCHAR(50);
    DECLARE v_nbHVol DECIMAL(10,2);
    DECLARE v_cnt INT;

    -- Déclaration du curseur AVANT les handlers
    DECLARE curPilote CURSOR FOR
        SELECT nom, nbHVol
        FROM Pilote
        WHERE comp = pcomp;

    -- Handler pour NOT FOUND (doit être déclaré APRÈS le curseur)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fini = 1;

    -- Handler d'erreur générique (déclaré en dernier)
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        DROP TEMPORARY TABLE IF EXISTS tmp_pilotes;
        CREATE TEMPORARY TABLE tmp_pilotes (Message VARCHAR(255));
        INSERT INTO tmp_pilotes VALUES ('Erreur lors de l'exécution de la procédure.');
```

SELECT * FROM tmp_pilotes;

```
END;

-- Vérifier l'existence de la compagnie
SELECT COUNT(*) INTO v_cnt
FROM Pilote
WHERE comp = pcomp;

IF v_cnt = 0 THEN
    SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Aucun pilote trouvé pour cette compagnie.';
END IF;
```

```
-- Créer la table temporaire pour les résultats
DROP TEMPORARY TABLE IF EXISTS tmp_pilotes;
CREATE TEMPORARY TABLE tmp_pilotes (
    Nom VARCHAR(50),
    nbHVol DECIMAL(10,2)
);

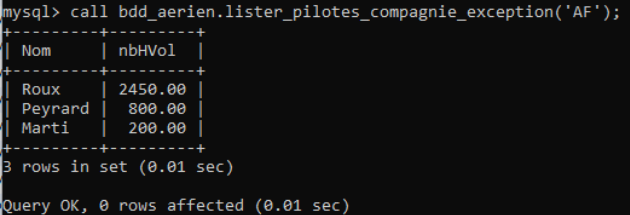
-- Ouvrir le curseur et traiter les données
OPEN curPilote;

boucle: LOOP
    FETCH curPilote INTO v_nom, v_nbHVol;
    IF fini = 1 THEN
        LEAVE boucle;
    END IF;
    INSERT INTO tmp_pilotes VALUES (v_nom, v_nbHVol);
END LOOP;

CLOSE curPilote;

-- Retourner les résultats
SELECT * FROM tmp_pilotes;

END
```



```
mysql> call bdd_aerien.lister_pilotes_compagnie_exception('AF');
+-----+-----+
| Nom   | nbHVol |
+-----+-----+
| Roux  | 2450.00 |
| Peyrard | 800.00 |
| Marti | 200.00 |
+-----+-----+
3 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

CONTINUE HANDLER FOR NOT FOUND :

Continue l'exécution quand le curseur n'a plus de lignes

EXIT HANDLER FOR SQLEXCEPTION :

Arrête tout et retourne un message d'erreur en cas de pb SQL

Retourne chaque pilote d'une compagnie donnée.

1. Ordre des déclarations :

En MySQL, l'ordre des déclarations dans une procédure stockée doit respecter cette séquence :

- Variables (DECLARE variable)
- Curseurs (DECLARE cursor)
- Handlers (DECLARE handler)

2. Position du curseur : Le curseur curPilote a été déplacé avant les handlers, car il doit être déclaré avant le handler NOT FOUND.

3. Échappement de l'apostrophe : Dans le message d'erreur, l'exécution a été corrigé en l'exécution pour échapper l'apostrophe.

4. Structure plus claire : J'ai ajouté des commentaires et une meilleure organisation du code pour faciliter la lecture.

Points importants à retenir :

- L'ordre des déclarations est crucial en MySQL et ne peut pas être modifié
- Les handlers doivent toujours être déclarés en dernier
- Les curseurs doivent être déclarés avant les handlers qui les concernent
- Le handler EXIT interrompra l'exécution en cas d'erreur SQL et retournera un message d'erreur personnalisé

Définition d'un Handler

Un handler (gestionnaire) en MySQL est un mécanisme qui permet de gérer automatiquement les erreurs et les conditions exceptionnelles qui peuvent survenir lors de l'exécution d'une procédure stockée.

Les handlers sont essentiels pour créer des procédures robustes qui gèrent proprement les erreurs (au lieu de planter silencieusement).

Types d'action (handler_action)

1. **CONTINUE** : Continue l'exécution après avoir traité l'erreur
2. **EXIT** : Arrête l'exécution de la procédure/fonction
3. **UNDO** : Annule les modifications (non supporté actuellement)

Types d'action courantes

- **NOT FOUND** : Aucune ligne trouvée (curseur vide, SELECT sans résultat)
- **SQLEXCEPTION** : Toute erreur générique
- **SQLWARNING** : Avertissements SQL
- **SQLSTATE** : Ex. '23000' pour une violation de contrainte
- **Code d'erreur spécifique** : Ex. 1062 pour une violation de clé unique

CURSEUR dans une procédure stockée pour MySQL

```
DELIMITER //
```

```
CREATE PROCEDURE PrintVoyageurs()
```

```
BEGIN
```

```
-- Declare variables for cursor
```

```
DECLARE v_nom VARCHAR(100);
```

```
DECLARE v_prenom VARCHAR(100);
```

```
DECLARE done BOOLEAN DEFAULT FALSE;
```

```
-- Declare the cursor
```

```
DECLARE curseur_voyageurs CURSOR FOR
```

```
    SELECT nom, prenom FROM Voyageurs;
```

```
-- Declare the handler for end of cursor
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET
```

```
done = TRUE;
```

```
-- Open the cursor
```

```
OPEN curseur_voyageurs;
```

```
-- Loop through each row
```

```
read_loop: LOOP
```

```
    FETCH curseur_voyageurs INTO v_nom, v_prenom;
```

```
    IF done THEN
```

```
        LEAVE read_loop;
```

```
    END IF;
```

```
-- Output the traveler's name (using SELECT for simplicity)
```

```
    SELECT CONCAT(v_nom, ' ', v_prenom) AS FullName;
```

```
END LOOP;
```

```
-- Close the cursor
```

```
CLOSE curseur_voyageurs;
```

```
END;
```

```
//
```

```
DELIMITER ;
```

Trigger (déclencheur)

- Un trigger, ou déclencheur, est une procédure stockée qui s'exécute automatiquement lors de certains événements sur une table d'une base de données.
- Il est généralement utilisé pour maintenir l'intégrité et la cohérence des données, ou pour effectuer des actions en réponse à des modifications de la base de données.
- Les triggers sont basés sur le modèle ECA (**Événement-Condition-Action**) :
 - **Événement** : Un trigger est déclenché par un événement spécifique, tel que l'insertion, la suppression ou la modification d'une ligne dans une table.
 - **Condition** : Une condition peut être spécifiée pour déterminer si l'action du trigger doit être exécutée. Si la condition n'est pas satisfaite, l'exécution du trigger s'arrête.
 - **Action** : Si la condition est satisfaite, l'action du trigger est exécutée. L'action peut consister en un ensemble d'opérations sur la base de données, effectuées à l'aide d'un langage procédural tel que PL/SQL.

Utilités des triggers

- **Gestion des redondances** : les triggers peuvent maintenir automatiquement des valeurs calculées ou agrégées dans d'autres tables.
- **Enregistrement automatique d'événements** : ils peuvent enregistrer des actions importantes, telles que les suppressions de données, à des fins d'audit.
- **Gestion de contraintes complexes** : ils peuvent imposer des règles métier complexes qui ne peuvent pas être exprimées par des contraintes standard.
- **Gestion de contraintes liées à l'environnement** : ils peuvent appliquer des restrictions basées sur l'heure, l'utilisateur ou d'autres facteurs contextuels.

Inconvénients des triggers

- **Exécution cachée :**

Les triggers s'exécutent en arrière-plan, ce qui peut rendre difficile la détection des problèmes.

- **Risque de cycles infinis :**

Des triggers mal conçus peuvent déclencher une cascade d'actions, créant une boucle infinie et bloquant le système.

- **Difficulté de débogage :**

La nature cachée des triggers peut rendre leur débogage difficile.

Exemple de TRIGGER pour MySQL

```
1 • CREATE DEFINER='root'@'localhost'
2   TRIGGER exemple_trigger
3   BEFORE INSERT ON Pilote
4   FOR EACH ROW
5   BEGIN
6       -- Exécuté AUTOMATIQUEMENT avant chaque INSERT
7       -- Ne retourne PAS de valeur
8       -- Ne peut PAS être appelé manuellement
9
10      IF NEW.nbHVol < 0 THEN
11          SIGNAL SQLSTATE '45000'
12          SET MESSAGE_TEXT = 'Heures de vol négatives interdites';
13      END IF;
14
15      SET NEW.nom = UPPER(NEW.nom);
16  END
```

CREATE

```
[DEFINER = { utilisateur | CURRENT_USER }]
TRIGGER nomDéclencheur
{ BEFORE | AFTER } { DELETE | INSERT | UPDATE }
ON nomTable
FOR EACH ROW
{ instruction; /
[etiquette:] BEGIN
    instructions;
END [etiquette]; }
```

Exemple de TRIGGER pour MySQL

```
DELIMITER //
```

```
CREATE TRIGGER nom_du_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nom_de_la_table
FOR EACH ROW
BEGIN
    -- Corps du trigger
    -- Vous pouvez utiliser OLD et NEW pour accéder aux valeurs
    -- OLD.colonne : ancienne valeur (pour UPDATE et DELETE)
    -- NEW.colonne : nouvelle valeur (pour INSERT et UPDATE)

    -- Exemple de logique :
    IF NEW.colonne > 100 THEN
        SET NEW.colonne = 100;
    END IF;
END;//

DELIMITER ;
```

Exemple 1 de TRIGGER pour MySQL

```
DELIMITER ;  
CREATE TRIGGER CumulCapacite  
AFTER UPDATE ON Salle  
FOR EACH ROW  
WHEN (new.capacite != old.capacite)  
BEGIN  
    UPDATE Cinema SET capacite = capacite - :old.capacite + :new.capacite  
    WHERE nom = :new.nomCinema;  
END;  
DELIMITER ;
```

Exemple 2 de TRIGGER pour MySQL

```
DELIMITER //
```

```
CREATE TRIGGER after_voyageur_insert
AFTER INSERT ON voyageur
FOR EACH ROW
BEGIN
    -- Mettons à jour une table liée, par exemple une table statistiques
    UPDATE statistiques_voyageurs
    SET nombre_total = nombre_total + 1,
        derniere_inscription = NOW()
    WHERE annee = YEAR(NOW());

    -- Si vous voulez aussi mettre à jour le statut du voyageur
    UPDATE voyageur
    SET statut = 'Nouveau membre'
    WHERE id = NEW.id;
END; //
```

```
DELIMITER ;
```

Commandes utiles :

```
-- Lister tous les triggers
SHOW TRIGGERS;

-- Voir le code d'un trigger
SHOW CREATE TRIGGER nom_trigger;

-- Supprimer un trigger
DROP TRIGGER IF EXISTS nom_trigger;

-- Informations détaillées sur les triggers
SELECT * FROM information_schema.TRIGGERS
WHERE EVENT_OBJECT_SCHEMA = 'votre_db';
```

Points critiques :

À faire

- Tester exhaustivement avant déploiement
- Documenter le but de chaque trigger
- Utiliser des noms explicites
- Gérer les erreurs avec SIGNAL

À éviter

- Triggers qui modifient la même table (boucles infinies)
- Logique trop complexe (impact performance)
- Cascades de triggers difficiles à déboguer
- Oublier la gestion d'erreurs

Maven

- Intégré à IntelliJ (ou Eclipse), **Maven** est un composant permettant de créer des projets Java
- Open source et soutenu par la fondation Apache : <http://maven.apache.org/>
- Facilite et automatise certaines tâches de la gestion d'un projet Java, notamment :
 - d'automatiser certaines tâches : compilation, tests unitaires et déploiement des applications qui composent le projet ;
 - de gérer des dépendances vis à vis des bibliothèques nécessaires au projet ;
 - de générer des documentations concernant le projet.
- Pour gérer les dépendances du projet vis à vis de bibliothèques, **Maven** utilise un ou plusieurs dépôts (appelés *repositories*) qui peuvent être :
 - locaux (.maven/repository)
 - distants (<http://www.ibiblio.org/maven> par défaut)

Source : https://fr.wikibooks.org/wiki/D%C3%A9veloppeur_en_Java/Introduction_%C3%A0_Apache_Maven

Arborescence d'un projet Maven

mon_projet

un dossier qui va contenir toutes les données du projet

mon_projet/src

ce dossier va contenir tous les fichiers sources créés par les développeurs pour le projet

mon_projet/src/main

mon_projet/src/main/java

contient les codes sources Java, rangés selon le hiérarchie des packages.

mon_projet/src/main/resources

les ressources nécessaires au code Java. Fichiers de configuration, icônes, fichiers de données...

mon_projet/src/test

contient toutes les données nécessaire pour tester l'application

mon_projet/src/test/java

contient les codes sources des tests

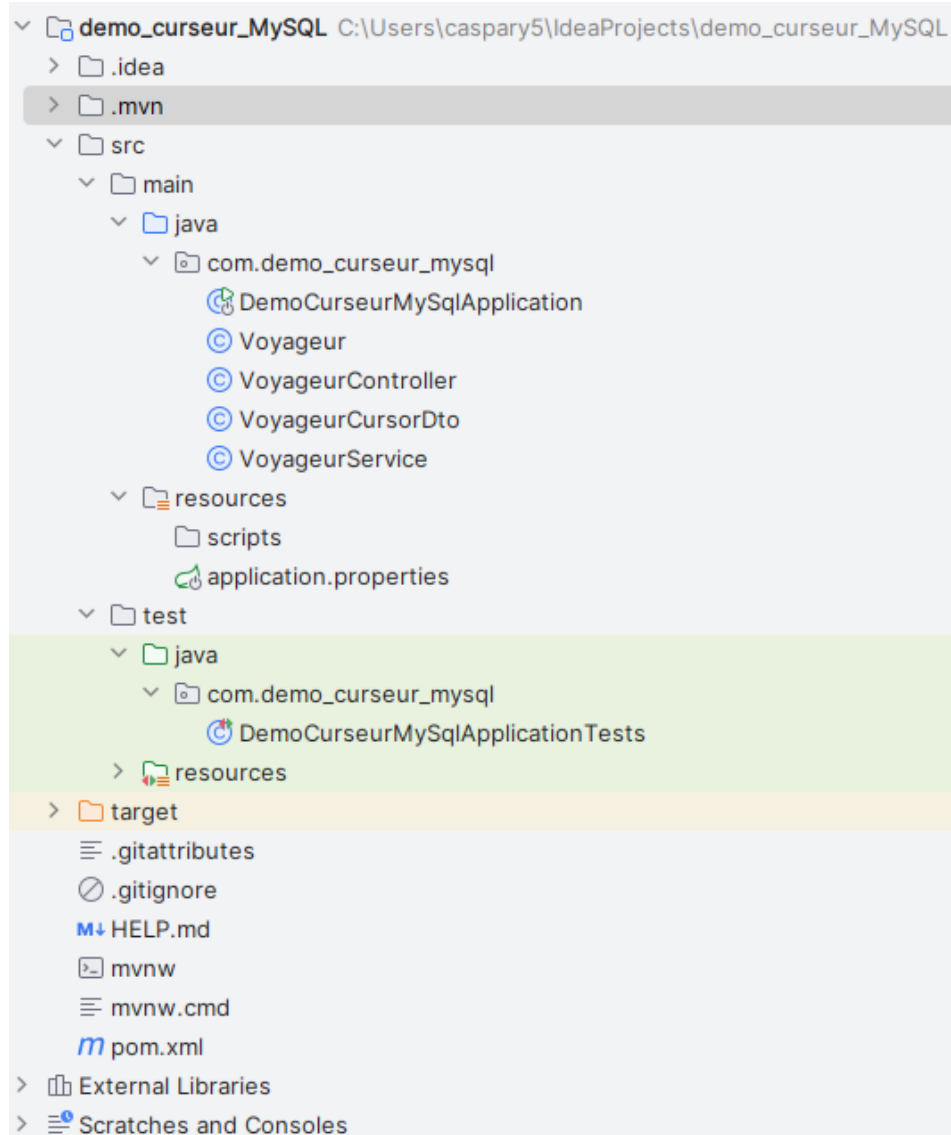
mon_projet/src/test/resources

les ressources qui sont utilisées dans les tests

mon_projet/pom.xml

un fichier XML qui décrit le projet pour permettre à Maven d'interagir avec lui.

Arborescence d'un projet Maven



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.4.0</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.example</groupId>
  <artifactId>demo_procedure_mysql</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo_procedure_mysql</name>
  <description>demo_procedure_mysql</description>

  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>
  </dependencies>
  ....
</dependencies>
```

- **JPA (Java Persistence API)**

- JPA est présenté comme une solution standard pour la persistance des objets en Java, particulièrement adapté aux serveurs d'applications Java EE, mais aussi utilisable dans des applications Java SE.
- L'utilisation de JPA nécessite un fournisseur de persistance (ex : Hibernate, EclipseLink).
- JPA (Java Persistence API) simplifie la persistance objet-relationnelle en fournissant **un framework de mapping objet/relationnel**. Ce framework permet de gérer la correspondance entre les données d'un modèle objet et celles d'une base de données relationnelle de manière plus simple qu'en le faisant manuellement.

• JPA (Java Persistence API) :

Annotations pour mapper les objets Java aux tables de BDD

Hibernate : ORM, implémentation JPA robuste

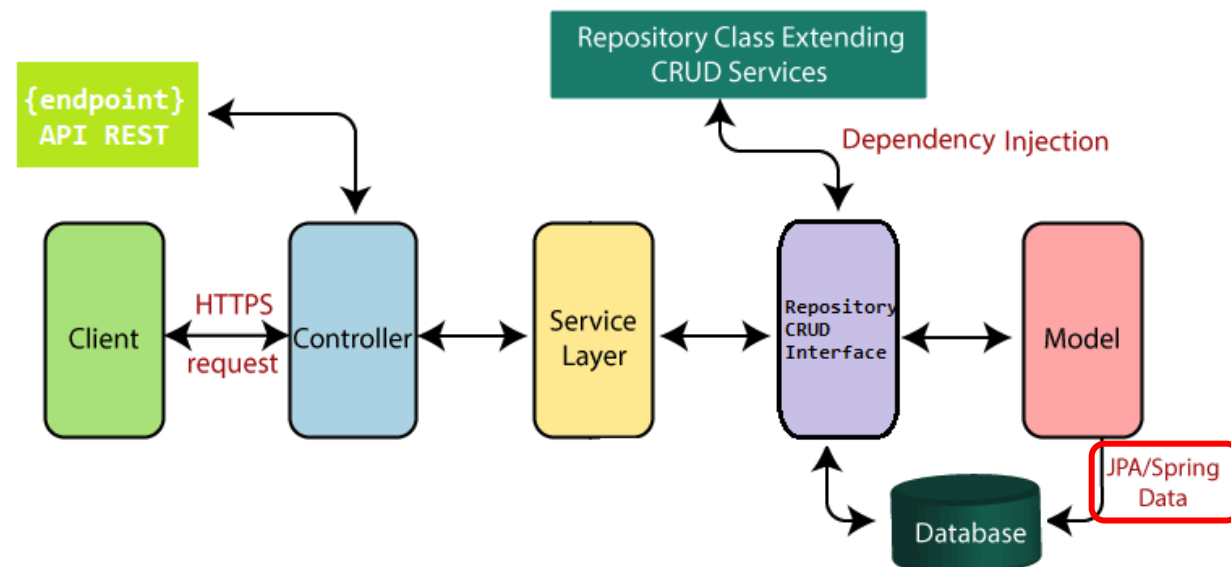
Spring Data : simplifie l'accès aux données

JPA : haut niveau d'abstraction

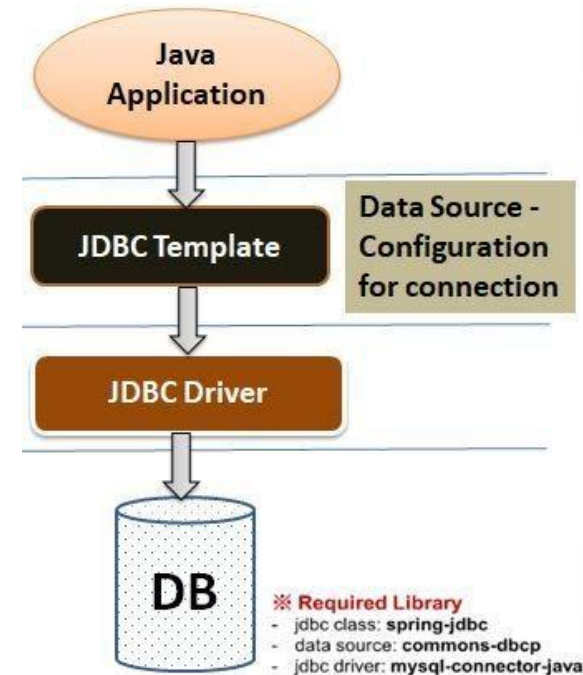
JDBC : direct et bas niveau d'abstraction (plus de code)

Spring Data JPA/Hibernate : fournit une interface unifiée

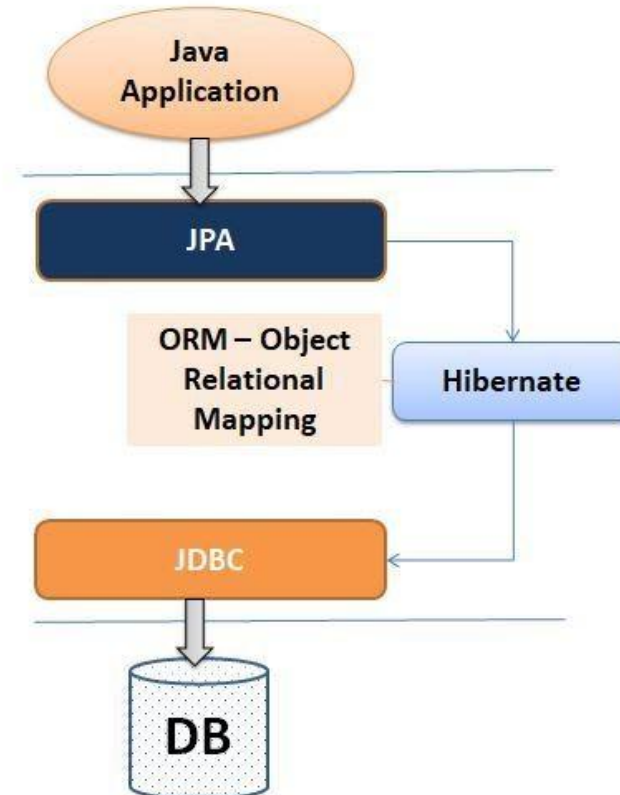
Spring Boot flow architecture



JDBC Implementation:



JPA Implementation:



• Principaux aspects de JPA

■ Mapping automatique :

JPA prend en charge la traduction des classes en relations, des objets en tuples et des données membres en attributs, ce qui réduit la complexité du mapping manuel.

■ Gestion des associations :

JPA facilite la gestion des associations entre objets (1:1, 1:N, M:N), qui peuvent être représentées par des variables d'instance référençant l'objet associé ou des collections. Dans le modèle relationnel, ces associations sont traduites en clés étrangères ou en tables d'association.

■ Gestion de l'héritage :

JPA permet de gérer l'héritage dans le modèle objet et de le faire correspondre au modèle relationnel, ce qui est un problème complexe à résoudre manuellement.

Pour JPA, une **entité** est une classe dont les instances peuvent être rendues persistantes.

```
public class Employe {  
  
    private int id;  
    private String nom;  
    private long salaire;  
    private Departement departement;  
  
    public Employe() {}  
  
    public Employe(int id) { this.id = id; }  
  
    public int getId() { return this.id; }  
  
    private void setId(int id) { this.id = id; }  
  
    public String getNom() { return this.nom; }  
  
    public void setNom(String nom) { this.nom = nom; }  
  
    public int getSalaire() { return this.salaire; }  
  
    public void setSalaire(long salaire) { this.salaire = salaire; }  
  
    public Departement getDepartement() { return this.departement; }  
  
    public void setDepartement(Departement dept) { this.departement = dept; }  
}
```

Pour transformer Employe en entité il suffit de lui rajouter les annotations suivantes :

@Entity

```
public class Employe {  
  
    @Id private int id;  
    private String nom;  
    private long salaire;  
    @ManyToOne private Departement departement;  
  
    public Employe() {}  
  
    public Employe(int id) { this.id = id; }  
  
    public int getId() { return this.id; }  
  
    private void setId(int id) { this.id = id; }  
  
    public String getNom() { return this.nom; }  
  
    public void setNom(String nom) { this.nom = nom; }  
  
    public int getSalaire() { return this.salaire; }  
  
    public void setSalaire(long salaire) { this.salaire = salaire; }  
  
    public Departement getDepartement() { return this.departement; }  
  
    public void setDepartement(Departement dept) { this.departement = dept; }  
  
}
```

• Commentaires : Objet \leftrightarrow Relationnel

- Par convention cette entité sera associée à la relation EMPLOYE.
- les propriétés seront stockées dans des attributs de même nom avec le type SQL adapté.
- L'annotation `@Entity` permet d'indiquer à JPA que la classe pourra être rendue persistante.
- L'annotation `@Id` précise la donnée membre qui sera utilisée comme identifiant (clef primaire).
- L'annotation `@ManyToOne` précise que la donnée membre departement est une association hiérarchique de type N:1.
- Pour sauvegarder l'état d'une instance de Employe, il faudra faire un appel à l'API JPA.

• Annotations JPA

@Entity :

Indique à JPA qu'une classe peut être rendue persistante. **Elle transforme une classe Java en une entité JPA**, ce qui signifie que les instances de cette classe peuvent être stockées et récupérées d'une base de données. Par convention, cette entité sera associée à une relation du même nom.

@Id :

Spécifie la donnée membre qui sera utilisée comme identifiant, c'est-à-dire **la clé primaire de l'entité**. Cette clé primaire permet d'identifier de manière unique chaque instance de l'entité dans la base de données.

@ManyToOne:

Est utilisée pour préciser qu'une donnée membre est **une association hiérarchique de type N:1**.

→ Il existe d'autres annotations (voir : *Liste des annotations JPA.html*).

→ Sans ces annotations, JPA ne peut pas gérer la persistance de la classe.

• Principaux aspects de JPA

■ Identification des objets :

JPA gère l'identification des objets, qui est basée sur l'adresse mémoire dans le monde objet, alors que dans le monde relationnel, l'identification est basée sur les valeurs des attributs et les clés primaires. JPA utilise généralement **un identifiant numérique unique pour chaque objet**, qui est stocké dans une donnée membre de chaque classe.

■ Persistance des objets :

JPA utilise l'annotation **@Entity** pour indiquer qu'une classe peut être rendue persistante. L'annotation **@Id** précise la donnée membre qui sera utilisée comme clé primaire.

■ Facilité d'utilisation de l'API :

Les opérations principales de l'API sont contenues dans un petit nombre de classes faciles à apprendre et à comprendre. **La persistance est basée uniquement sur des POJO** (Plain Old Java Objects), il n'est pas nécessaire d'étendre ou d'implémenter quoi que ce soit pour gérer la persistance.

- **Principaux aspects de JPA**

- **Requêtes JPQL :**

JPA offre un langage de requête orienté objet appelé JPQL, qui permet d'interroger la base de données d'une manière plus intuitive que le SQL.

- **Gestion du cycle de vie des objets :**

JPA permet de gérer le cycle de vie des objets, y compris la persistance, la récupération, la mise à jour et la suppression. Les modifications sur une entité gérée sont directement répercutées dans la base de données.

- **Unit of Work :**

JPA permet de gérer un "cache" des objets déjà chargés afin d'éviter les problèmes de duplication et d'incohérence des données.

- ✓ JPA simplifie la persistance objet-relationnelle en automatisant des tâches complexes et permet aux développeurs de se concentrer sur la logique métier.
- ✓ JPA est non-intrusif car les objets métiers n'ont pas besoin d'être modifiés pour devenir persistant.

- **Outils pour mieux gérer les projets**

- **Maven**

pour gérer le cycle de construction d'un projet (compilation, gestion des dépendances, packaging automatique,...)

- **JUnit**

pour écrire le code testant tous les modules que l'on développe pour éviter d'introduire de nouveaux bugs lors de la modification du code.

- **DBUnit**

qui est un framework de tests unitaires pour tester le code de la couche de persistance (extension de **JUnit** destinée aux tests d'applications qui utilisent une BDD relationnelle). Voir le projet Spring Boot : **dbunit-test**.

- **Git**

pour gérer facilement les évolutions de notre base de code lorsque l'on travaille à plusieurs.