

Application web et base MySQL

Tout ce dont vous avez besoin pour ce module, dans un seul document. Cochez au fur et à mesure.

U6-S2-M6

module

Élève

document

TechnoVert

scénario 2

B validé · 3 h 45

format et durée

Ce qu'il faut savoir avant de commencer	4
C'est quoi une requête SQL ?	4
C'est quoi une clé étrangère ?	4
C'est quoi un tuple en Python ?	4
Comment marche une page Flask ?	5
Comment lancer MySQL sur votre poste ?	5
Vérifiez-vous	5
Brancher l'application sur la base	6
Capsule 1 sur 5 Parler à la base depuis Python	6
Capsule 2 sur 5 L'injection SQL et la requête paramétrée	8
Capsule 3 sur 5 Une couche d'accès aux données	11
Capsule 4 sur 5 Les identifiants hors du code	13
Capsule 5 sur 5 Un jeu de données qu'on peut rejouer	14
Ce qu'on retient	15
Brancher l'application sur la base	16
L'essentiel	16
Les mots à connaître	16
La méthode : écrire une fonction de la couche d'accès	17
Un exemple concret	17
Les pièges	18
Injection, correction, restauration	19
PARTIE A — Brancher l'application	20
PARTIE B — Isoler et sortir les secrets	26
PARTIE C — Injecter puis corriger	28
PARTIE D — Exporter et restaurer	32
POUR ALLER PLUS LOIN	35
CE QU'IL FAUT RETENIR	36
Un tableau de bord qui tient	37
COMMENT VOUS SEREZ NOTÉ	37
CE QU'ON VOUS DEMANDE	39

PAR OÙ COMMENCER	39
CE QUI DOIT MARCHER À LA FIN	40
Tous les documents du module	41
A1 — L'application de départ et son mode d'emploi	41
A2 — Le jeu de données de la base	44
A3 — Le compte MySQL de l'application	48
A4 — Le fichier d'environnement à remplir	48
A5 — Extrait de la documentation du connecteur MySQL	49
A6 — Le script de vérification de la base	50
A7 — Le tableau de bord de départ et son mode d'emploi	52

Ce qu'il faut savoir avant de commencer

Ce qu'il faut savoir avant de commencer. Lisez-le si vous avez un doute. Environ 10 minutes.

C'est quoi une requête SQL ?

Une base de données range les données dans des tables. Chaque table a des colonnes et des lignes, comme un tableur.

On parle à la base en SQL. `SELECT` lit des lignes, `WHERE` les filtre, `INSERT` en ajoute.

```
SELECT nom, statut FROM chantier WHERE statut = 'ouvert';  
INSERT INTO client (nom, ville) VALUES ('Mairie de Lusse', 'Lusse');
```

En SQL, un texte s'écrit entre apostrophes : `'ouvert'`. Un nombre s'écrit sans apostrophes.

C'est quoi une clé étrangère ?

Une colonne qui contient le numéro d'une ligne d'une autre table. Dans la table `chantier`, la colonne `id_client` contient le numéro du client.

De la même façon, dans la table `intervention`, la colonne `id_chantier` contient le numéro du chantier. C'est elle qui relie une intervention à son chantier.

Pour afficher le nom du client avec le chantier, on relie les deux tables avec `JOIN` :

```
SELECT chantier.nom, client.nom FROM chantier JOIN client ON client.id =  
chantier.id_client;
```

C'est quoi un tuple en Python ?

Un tuple, c'est des valeurs rangées entre parenthèses : `(3, "atelier")`. On s'en sert pour passer les valeurs à une requête.

Pour une seule valeur, on écrit une virgule à la fin : `(nom,)`. Sans elle, ce n'est pas un tuple.

Comment marche une page Flask ?

Flask est une boîte à outils Python pour faire un site web. Une **route** relie une adresse à une fonction Python, appelée **vue**.

La vue prépare les données, puis remplit un **gabarit** : une page HTML avec des trous. On lance le site avec `flask --app app run`.

Comment lancer MySQL sur votre poste ?

Sur votre poste, MySQL tourne grâce à XAMPP. On le pilote depuis le panneau de contrôle de XAMPP :

- **Start**, en face de « MySQL », lance la base. Le nom « MySQL » passe au vert.
- **Stop** l'arrête.

On parle ensuite à la base dans un terminal, avec la commande `mysql -u root`.

Vérifiez-vous

-
- ☐ 1. Écrivez la requête qui lit le nom de tous les chantiers clôturés.
-
- ☐ 2. Dans la table `intervention`, quelle colonne relie une intervention à son chantier ?
-
- ☐ 3. MySQL ne répond plus. Où vérifiez-vous qu'il est lancé ?
-

Réponses

1. `SELECT nom FROM chantier WHERE statut = 'cloture';`
2. `id_chantier`, qui contient le numéro du chantier.
3. Dans le panneau de contrôle de XAMPP : « MySQL » doit être en vert. Sinon, cliquez sur « Start ».

Brancher l'application sur la base

COURS-U6-S2-06

L'application de suivi des chantiers de TechnoVert marche. Les techniciens saisissent leurs interventions chaque soir. Mais les données sont dans un simple fichier, sur le serveur web.

Hélène Vasseur, la dirigeante, veut qu'elles passent dans la base MySQL de l'entreprise. Elle a une question : « Et si quelqu'un tape n'importe quoi dans une page, il peut abîmer la base ? »

La réponse est oui, si l'application est mal écrite. À la fin de la séance, vous saurez brancher l'application sur la base sans lui ouvrir cette porte.

LE PLAN DE LA SÉANCE

- 1 Parler à la base depuis Python.
- 2 L'injection SQL et la requête paramétrée.
- 3 Une couche d'accès aux données.
- 4 Les identifiants hors du code.
- 5 Un jeu de données qu'on peut rejouer.

LA QUESTION DE DÉPART

Sur un site, vous tapez une apostrophe dans un champ de recherche. La page affiche une erreur de base de données. Comment un seul caractère peut-il faire planter le site ?

CAPSULE 1 SUR 5

Parler à la base depuis Python

Commençons par le branchement lui-même. Comment un programme Python envoie-t-il une requête SQL à MySQL ?

Il passe par un **CONNECTEUR** : une bibliothèque Python qui sait parler à MySQL. Ici, c'est `mysql.connector`. Le connecteur ouvre une connexion, puis on lui confie un **CURSEUR** : l'objet qui envoie les requêtes et ramène les lignes.

Pensez à un appel téléphonique. On compose le numéro, on parle, on écoute la réponse, on raccroche.

Le connecteur, c'est le téléphone. Le curseur, c'est la conversation.

- 1 **Ouvrir** la connexion : l'adresse du serveur, le compte, le mot de passe, la base.
- 2 **Créer** un curseur.
- 3 **Exécuter** la requête avec `execute()`.
- 4 **Lire** les lignes avec `fetchall()`. Pour une écriture, **valider** avec `commit()`.
- 5 **Fermer** la connexion avec `close()`.

```
import mysql.connector

connexion = mysql.connector.connect(host="127.0.0.1", user="tv_appli",
                                   password="...", database="technovert")
curseur = connexion.cursor(dictionary=True)
curseur.execute("SELECT id, nom, statut FROM chantier")
chantiers = curseur.fetchall()      # une liste de dictionnaires
connexion.close()
```

Avec `dictionary=True`, chaque ligne arrive sous forme de dictionnaire : `{"id": 3, "nom": "Éclairage des allées", ...}`. C'est la forme qu'attendent les gabarits.

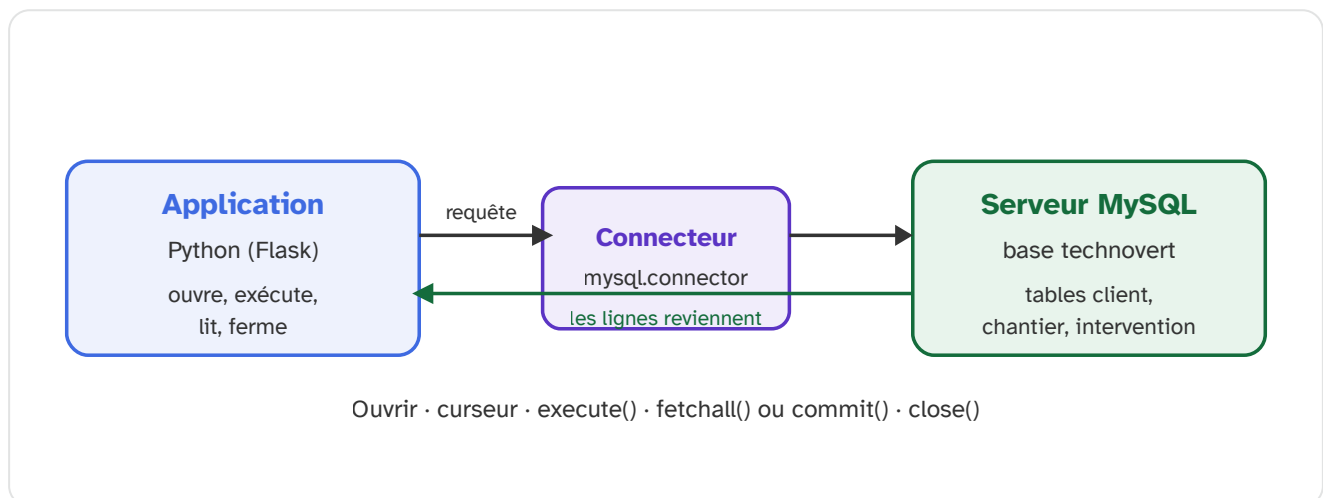


Schéma · L'application passe par le connecteur pour parler au serveur MySQL, puis ferme la connexion

ON LE FAIT ENSEMBLE — AJOUTER UNE INTERVENTION

1. Quelle est la première chose à faire ?
2. Quelle requête le curseur exécute-t-il ?
3. Faut-il `fetchall()` ?
4. Que faut-il ajouter avant de fermer ?

Exemple

Pourquoi

✓	Ouvrir, exécuter, lire, fermer, dans la même fonction	La connexion ne reste pas ouverte pour rien.
✗	Un <code>INSERT</code> sans <code>commit()</code>	Aucune erreur ne s'affiche, mais la ligne n'est jamais enregistrée.
✗	Oublier <code>close()</code> dans une page très visitée	Les connexions s'accumulent. MySQL finit par refuser les nouvelles.

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Quel objet envoie la requête et ramène les lignes ?
2. Un `UPDATE` qui clôture un chantier marche, mais le changement disparaît. Qu'est-ce qui manque ?

CAPSULE 2 SUR 5

L'injection SQL et la requête paramétrée

On sait maintenant envoyer une requête. Souvent, elle contient une valeur tapée par l'utilisateur : un service, un nom. Comment l'y mettre ?

La mauvaise façon, c'est de coller la saisie dans le texte de la requête. L'utilisateur peut alors taper un morceau de SQL. C'est l'**INJECTION SQL** : la saisie devient une partie de l'ordre donné à la base.

C'est comme un bon de commande où le client remplit la case « quantité ». S'il écrit « 2, et offrez-moi le reste du magasin », et que personne ne relit, le magasinier exécute tout.

- 1 Le code **colle** la saisie dans la requête, entre apostrophes.
- 2 L'utilisateur tape une **apostrophe** : elle ferme le texte trop tôt.
- 3 La suite de la saisie devient **du SQL**, que la base exécute.
- 4 La base fait ce que l'utilisateur a écrit, pas ce que le développeur voulait.

```
Code :      "... WHERE technicien = '" + login + "' AND service = '" + service +
            "'"

Saisie :    x' OR '1'='1

Requête :   ... WHERE technicien = 'ngarcia' AND service = 'x' OR '1'='1'
```

'1'='1' est toujours vrai. La base renvoie les interventions de **tous** les techniciens.

La parade s'appelle la **REQUÊTE PARAMÉTRÉE**. On écrit `%s` à la place de chaque valeur.

On donne les valeurs à part, dans un tuple. Le connecteur les envoie comme des données, jamais comme du SQL.

```
curseur.execute("SELECT ... WHERE technicien = %s AND service = %s", (login,
service))
```

Avec la même saisie, la base cherche un service qui s'appelle vraiment `x' OR '1'='1'`. Elle n'en trouve aucun.

Saisie collée : elle devient du SQL

```
"... WHERE service = '" + service + "'"
```

```
service = x' OR '1'='1
```



```
... WHERE service = 'x' OR '1'='1'
```

'1'='1' est toujours vrai → la base renvoie tout

Saisie en paramètre : elle reste une donnée

```
execute("... WHERE service = %s", (service,))
```

```
service = x' OR '1'='1
```

→ cherché tel quel, aucune ligne

Schéma • La saisie collée devient du SQL ; la saisie passée en paramètre reste une donnée

ON LE FAIT ENSEMBLE — LA RECHERCHE D'UN CHANTIER PAR NOM

1. Le code écrit `"... WHERE nom = '" + nom + "'"`. Est-il vulnérable ?
2. Par quoi remplace-t-on `'" + nom + "'"` ?
3. Comment donne-t-on la valeur ?
4. Pourquoi une virgule dans `(nom,)` ?

Exemple

Pourquoi



```
execute("... WHERE id = %s",  
        (numero,))
```

La valeur part à part : elle ne peut pas changer la requête.



```
execute(f"... WHERE service =  
        '{service}')
```

Une f-string colle la saisie, exactement comme `+`.



```
"... WHERE service = '%s'" avec des  
apostrophes
```

Le connecteur ajoute les siennes. Le texte obtenu est faux.

INSTANT HISTOIRE — 2015 : TALKTALK ET SES VIEILLES PAGES

En 2015, TalkTalk est un grand opérateur de téléphone et d'Internet au Royaume-Uni.

Des pirates, dont des adolescents, trouvent d'anciennes pages de son site qui collent la saisie dans leurs requêtes.

Par injection SQL, ils volent les données d'environ 157 000 clients. L'entreprise reçoit une amende record de 400 000 livres.

Qu'est-ce qui aurait évité ça ?

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Réécrivez `"SELECT * FROM client WHERE ville = '" + ville + "'"` en requête paramétrée.
2. Un collègue dit : « Mon menu déroulant ne propose que trois services, pas de risque. » A-t-il raison ?

Une règle de droit, enfin. Tester une injection sur un site qui n'est pas le vôtre, sans autorisation écrite, est un délit. On ne s'entraîne que sur ses propres machines.

CAPSULE 3 SUR 5

Une couche d'accès aux données

La requête est sûre. Reste à savoir où ranger tout ce SQL. Dans chaque vue, au milieu du code de la page ?

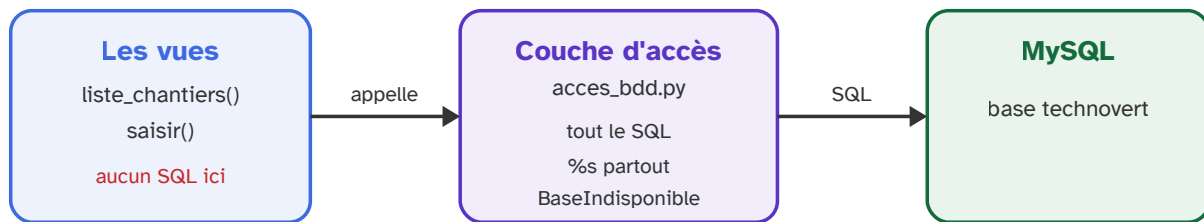
Non. On le range dans une **COUCHE D'ACCÈS AUX DONNÉES** : un module Python à part, qui contient toutes les requêtes.

Les vues appellent ses fonctions, comme `lire_chantiers()`. Elles ne voient jamais de SQL.

Au restaurant, le serveur ne va pas en cuisine. Il passe la commande au passe-plat, et le plat revient. Si la cuisine change de four, la salle ne voit aucune différence.

- 1 Toutes les requêtes vont dans **un seul fichier**, par exemple `acces_bdd.py`.
- 2 Chaque besoin devient **une fonction** au nom clair : `lire_chantier(numero)`.
- 3 Les vues **appellent** ces fonctions et reçoivent des listes ou des dictionnaires.
- 4 La couche **traduit les pannes** : si MySQL ne répond pas, elle lève une erreur à elle, `BaseIndisponible`.
- 5 L'application **répond proprement** à cette erreur : une page « Service indisponible », code 503.

Le code 503 veut dire : le serveur marche, mais il attend un autre service qui ne répond pas. L'utilisateur voit un message clair, pas une trace.



Pour changer de base, on touche la couche seule. Les vues ne bougent pas.

Schéma • Les vues appellent la couche d'accès ; seule la couche parle à MySQL

ON LE FAIT ENSEMBLE — LA BASE S'ARRÊTE PENDANT LA NUIT

1. Où l'erreur de connexion apparaît-elle en premier ?
2. Que fait `ouvrir()` de cette erreur ?
3. Que voit le technicien qui ouvre la liste ?
4. Où le technicien de maintenance lit-il la vraie cause ?

Exemple

Pourquoi

- | | |
|---|--|
| ✓ On passe du fichier à MySQL en changeant une ligne d'import | Les vues n'ont pas bougé : c'est la preuve que la couche est bien isolée. |
| ✗ Un <code>SELECT</code> écrit dans la vue <code>liste_chantiers()</code> | Le SQL se disperse. Pour vérifier les injections, il faut relire tout le site. |
| ✗ La trace MySQL affichée à l'utilisateur quand la base tombe | Elle montre le nom du serveur et du compte. Et l'utilisateur ne sait pas quoi faire. |

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. Vous cherchez toutes les requêtes de l'application pour les vérifier. Où regardez-vous ?
2. Que veut dire le code 503 ?

Les identifiants hors du code

La couche d'accès se connecte avec un compte et un mot de passe. Où les écrire ?

Surtout pas dans le code. Le code se copie, s'envoie, se range dans un dépôt partagé. Le mot de passe partirait avec.

On le met dans une **VARIABLE D'ENVIRONNEMENT** : une valeur que le système donne au programme quand il le lance.

C'est comme le code du portail d'une résidence. On ne le grave pas sur le portail. Chaque habitant le connaît, et on le change sans changer le portail.

- 1 On écrit les identifiants dans un **fichier d'environnement**, hors du dossier du code.
- 2 On le **protège** : on le range dans son dossier personnel, que les autres comptes de la machine ne lisent pas. Sur un serveur Linux, on ajoute `chmod 600`.
- 3 L'application le **charge** au démarrage, avec `load_dotenv()`.
- 4 Le code les lit avec `os.environ`.

```
Fichier technover.env (dossier perso)  Code de acces_bdd.py
TV_BDD_UTILISATEUR=tv_appli             user=os.environ["TV_BDD_UTILISATEUR"]
TV_BDD_MOT_DE_PASSE=...                 password=os.environ["TV_BDD_MOT_DE_PASSE"]
```

Autre gain : le même code tourne sur votre poste de test et sur le serveur de production. Seul le fichier d'environnement change.

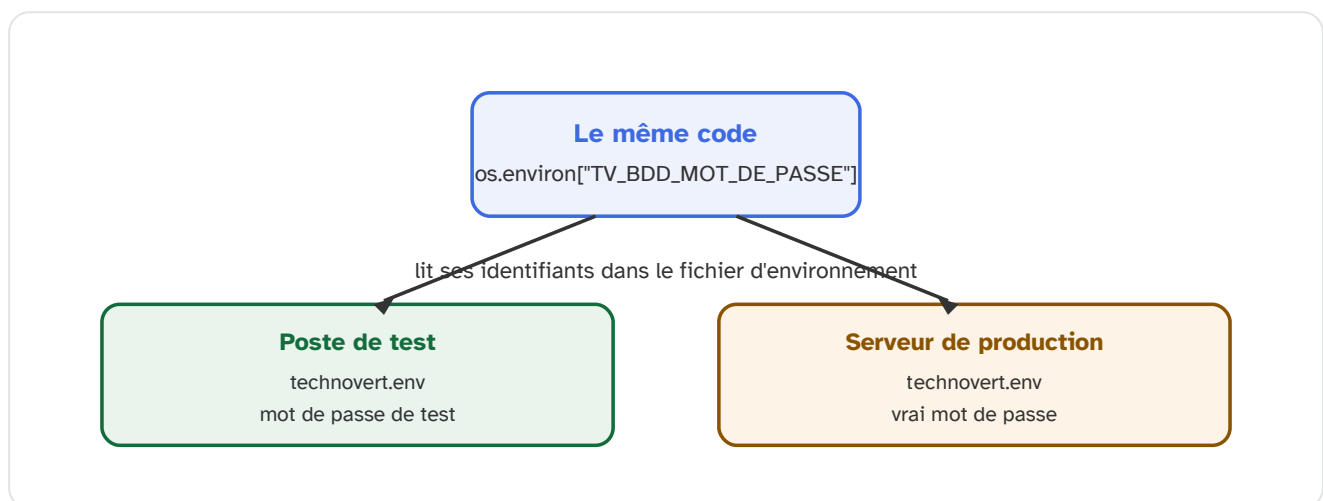


Schéma • Le code reste le même ; seul le fichier d'environnement change entre le test et la production

ON LE FAIT ENSEMBLE — LA CLÉ SECRÈTE DE LA SESSION

1. Le code contient `app.secret_key = "cle-de-labo"` . Est-ce un secret ?
2. Où la mettre ?
3. Qu'écrit-on dans le code ?

Exemple

Pourquoi

- | | | |
|---|---|--|
| ✓ | Un fichier d'exemple livré avec le code, avec <code>a-remplacer</code> comme valeur | Il montre les noms attendus, sans aucun vrai secret. |
| ✗ | Le fichier d'environnement rangé dans le dossier du code | Il part dans le ZIP ou dans le dépôt, avec le code. |
| ✗ | Le mot de passe écrit en commentaire, « pour mémoire » | Un commentaire se lit aussi bien qu'une ligne de code. |

CHECKPOINT

À l'oral, 3 minutes. Chacun note sa réponse sur sa feuille, puis on corrige.

1. On passe de votre poste de test au serveur de production. Que modifie-t-on ?
2. Sur le serveur Linux de production, que fait `chmod 600` sur le fichier d'environnement ?

CAPSULE 5 SUR 5

Un jeu de données qu'on peut rejouer

Dernier outil, pour tester sans crainte. Pendant un test, on ajoute des lignes, on en casse. Comment revenir à un état connu ?

On garde un **jeu de données de test** : un script SQL qui efface la base et la recrée, toujours à l'identique. On le rejoue avant chaque série de tests. Un script de vérification compte ensuite les lignes et teste les cas connus.

```
mysql -u root -e "source technover.sql"    # la base revient à l'état connu
python verifier_base.py                    # chaque ligne doit dire OK
```

Le même couple sert à prouver une restauration. On restaure la base sur une machine neuve, puis on lance la vérification. Si tout est OK, la restauration a marché.

Ce qu'on retient

SI VOUS NE DEVEZ RETENIR QUE TROIS CHOSES

1. Une valeur saisie ne se colle jamais dans une requête : on écrit `%s` et on passe la valeur à part.
2. Tout le SQL vit dans la couche d'accès. Les vues appellent des fonctions, et une base arrêtée donne une page 503 propre.
3. Les identifiants vivent dans un fichier d'environnement protégé, jamais dans le code.

Revenons à Hélène Vasseur. Oui, quelqu'un peut taper n'importe quoi dans une page.

Avec des requêtes paramétrées, sa saisie reste une simple donnée, et la base ne l'exécute pas. Et le mot de passe de la base n'est écrit nulle part dans le code.

Brancher l'application sur la base

COURS-U6-S2-06

L'essentiel

- **Connecteur** : `mysql.connector` relie Python à MySQL. Ouvrir, curseur, `execute()`, `fetchall()` ou `commit()`, `close()`.
- **Injection SQL** : une saisie collée dans la requête devient du SQL.
- **Requête paramétrée** : `%s` dans la requête, les valeurs à part dans un tuple.
- **Couche d'accès** : tout le SQL dans un seul module. Les vues appellent ses fonctions.
- **Base indisponible** : la couche lève `BaseIndisponible`, l'application répond 503 avec une page claire.
- **Identifiants** : dans un fichier d'environnement protégé, lus avec `os.environ`.
- **Jeu de test** : un script SQL rejouable, puis un script de vérification.

Les mots à connaître

Mot	En clair
Connecteur	La bibliothèque qui permet à Python de parler à MySQL.
Curseur	L'objet qui envoie une requête et ramène les lignes.
Injection SQL	Une saisie qui change la requête envoyée à la base.
Requête paramétrée	Une requête avec <code>%s</code> , dont les valeurs sont envoyées à part.
Couche d'accès	Le module qui contient toutes les requêtes de l'application.
Variable d'environnement	Une valeur donnée au programme par le système, hors du code.
Bibliothèque, module	Un fichier de code déjà écrit qu'on réutilise. Le connecteur est une bibliothèque ; la couche d'accès est un module.

Mot	En clair
<code>127.0.0.1</code>	L'adresse de la machine elle-même. La base tourne sur la même machine que l'application.
<code>chmod 600</code>	Sur un serveur Linux, rend un fichier lisible par son seul propriétaire. On l'utilise pour le fichier d'environnement.
f-string	Un texte Python écrit <code>f"...{x}..."</code> , où <code>{x}</code> est remplacé par la valeur. À ne jamais utiliser pour une requête SQL.

La méthode : écrire une fonction de la couche d'accès

1. **Ouvrez** la connexion avec `ouvrir()`.
2. **Écrivez** la requête avec un `%s` par valeur, sans apostrophes autour.
3. **Exécutez** avec les valeurs dans un tuple : `(valeur,)`.
4. **Lisez** avec `fetchall()`, ou **validez** avec `commit()`.
5. **Fermez** la connexion, puis renvoyez le résultat.

Un exemple concret

Lire les interventions d'un service, sans risque d'injection.

```
def interventions_du_service(service):
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute("SELECT * FROM intervention WHERE service = %s", (service,))
    lignes = curseur.fetchall()
    connexion.close()
    return lignes
```

Saisie	Ce que la base reçoit	Résultat
<code>atelier</code>	le service <code>atelier</code>	les interventions de l'atelier
<code>x' OR '1'='1</code>	un service nommé <code>x' OR '1'='1</code>	aucune ligne

Conclusion : la saisie piégée est traitée comme une simple donnée.

Les pièges

- **✗ Une f-string dans `execute()`** → toujours `%s` et un tuple.
- **✗ `'%s'` entre apostrophes** → `%s` seul.
- **✗ Oublier `commit()`** après une écriture → l'écriture est perdue.
- **✗ Le mot de passe dans le code** → fichier d'environnement, rangé dans votre dossier personnel.

Injection, correction, restauration

LABO-U6-S2-06

Contexte — Vous reprenez l'application de suivi des chantiers de TechnoVert. Elle marche, mais elle range ses données dans un fichier. Hélène Vasseur veut qu'elles passent dans la base MySQL de l'entreprise.

Un collègue a repris une vieille page écrite par l'ancien prestataire. Elle colle la saisie dans sa requête. Vous allez le prouver, puis corriger.

RÈGLE 1A DE LA SÉANCE

Scénario 2 : l'IA peut vous expliquer une notion ou un message d'erreur. Elle n'écrit pas votre code.

Si vous lui posez une question, dites-la au formateur au point de contrôle suivant.

À la fin de ce labo, vous saurez brancher une application sur MySQL, isoler l'accès aux données, corriger une injection SQL et prouver une restauration.

Comment ça se valide — à chaque point de contrôle, vous montrez votre résultat au formateur. Il valide, puis vous passez à la suite.

Les mots de la séance

Mot	Ce que ça veut dire
Connecteur	La bibliothèque <code>mysql.connector</code> qui relie Python à MySQL.
Curseur	L'objet qui envoie une requête et ramène les lignes.
Injection SQL	Une saisie qui change la requête envoyée à la base.
Requête paramétrée	Une requête avec <code>%s</code> , dont les valeurs sont passées à part.
Couche d'accès	Le module <code>acces_bdd.py</code> qui contient toutes les requêtes.
Restaurer	Recréer la base à l'identique à partir d'un fichier d'export.

Ce dont vous avez besoin

- Votre poste Windows, avec XAMPP, Python, VS Code et un navigateur.
- [Annexe A1](#) : l'application de départ et son mode d'emploi. Le dossier se télécharge dans [A1_technovert_web.zip](#).
- [Annexe A2](#) : le jeu de données de la base, fichier [A2_technovert.sql](#).
- [Annexe A3](#) : le compte MySQL de l'application, fichier [A3_compte_appli.sql](#).
- [Annexe A4](#) : le fichier d'environnement à remplir.
- [Annexe A5](#) : un extrait de la documentation du connecteur MySQL.
- [Annexe A6](#) : le script de vérification, fichier [A6_verifier_base.py](#).

En binôme — le pilote tape, le lecteur lit les consignes et les annexes à voix haute, et vérifie chaque résultat. On inverse à la partie C.

Le plan du labo

Partie	Étapes	Ce que vous obtenez	Temps
A — Brancher l'application	1 à 6	l'application qui lit la base MySQL	35 min
B — Isoler et sortir les secrets	7 à 11	une couche d'accès propre, sans mot de passe dans le code	25 min
C — Injecter puis corriger	12 à 17	une injection prouvée, puis rendue inoffensive	35 min
D — Exporter et restaurer	18 à 22	une base restaurée sur une pile vierge, vérifiée	25 min

PARTIE A — Brancher l'application

Étapes 1 à 6, environ 35 min. À la fin, l'application affiche les chantiers lus dans MySQL.

Besoin d'un rappel ? Revoir le cours : [« Parler à la base depuis Python »](#).

01 Installer l'application et lancer MySQL

- ☐ Téléchargez `A1_technovert_web.zip` depuis l'[annexe A1](#). Faites un clic droit dessus, « Extraire tout », puis avec « Parcourir », choisissez votre dossier « Documents ».
- ☐ Ouvrez le dossier `Documents\technovert-web` dans VS Code. Ouvrez un terminal : menu « Terminal », puis « Nouveau terminal ».
- ☐ Ouvrez le panneau de contrôle de XAMPP. Cliquez sur « Start », en face de « MySQL ».

Vous devez voir : « MySQL » en vert dans le panneau de XAMPP. Dans VS Code, le dossier contient `app.py`, `acces_bdd.py` et les fichiers [A2_technovert.sql](#), [A3_compte_appli.sql](#), [A6_verifier_base.py](#).

02 Créer la base et son compte

- ☐ Dans le terminal, vérifiez que la commande `mysql` répond :

```
mysql --version
```

Si vous voyez « mysql n'est pas reconnu » : suivez l'[annexe A1](#), « Rendre la commande mysql disponible », puis rouvrez VS Code.

- ☐ Chargez le jeu de données, puis le compte de l'application :

```
mysql -u root -e "source A2_technovert.sql"
mysql -u root -e "source A3_compte_appli.sql"
```

☐ Vérifiez que la base existe :

```
mysql -u root -e "SELECT COUNT(*) FROM technovert.chantier;"
```

Vous devez voir : aucun message pour les deux premières commandes, puis un petit tableau avec le nombre 6.

03 Écrire la connexion et la lecture des chantiers

La couche d'accès est le fichier `acces_bdd.py`. Ses premières lignes, déjà écrites, chargent vos identifiants avec `load_dotenv`. Deux fonctions sont à écrire.

☐ Dans `acces_bdd.py`, remplacez tout le corps de `ouvrir()`, `pass` compris, par :

```
try:
    return mysql.connector.connect(
        host=os.environ["TV_BDD_HOTE"],
        user=os.environ["TV_BDD_UTILISATEUR"],
        password=os.environ["TV_BDD_MOT_DE_PASSE"],
        database=os.environ["TV_BDD_BASE"],
        connection_timeout=3)
except mysql.connector.Error as erreur:
    raise BaseIndisponible(str(erreur)) from erreur
```

☐ Remplacez tout le corps de `lire_chantiers()`, `return []` compris, par :

```
connexion = ouvrir()
curseur = connexion.cursor(dictionary=True)
curseur.execute(COLONNES_CHANTIER + "ORDER BY ch.date_debut")
lignes = curseur.fetchall()
connexion.close()
return lignes
```

- ☐ Lisez la ligne `COLONNES_CHANTIER`, écrite en haut du fichier. Notez la table reliée à `chantier` par le `JOIN` : _____

Le `+` colle ici deux textes fixes, écrits par vous. Ce n'est pas une saisie d'utilisateur : il n'y a donc aucun risque d'injection.

Vous devez avoir : un fichier enregistré avec Ctrl+S, et le nom d'une table relié par le `JOIN`. On teste le tout à l'étape 5.

Pourquoi ? Pourquoi le `SELECT` relie-t-il deux tables au lieu de lire la seule table `chantier` ?

Votre réponse :

04 Remplir le fichier d'environnement

Le fichier d'environnement se range dans votre dossier personnel, jamais dans le dossier du code. L'application le lit seule au démarrage.

- ☐ Copiez le modèle de l'[annexe A4](#) dans votre dossier personnel :

```
Copy-Item technovert.env.exemple $HOME\technovert.env
```

- ☐ Ouvrez-le avec `notepad $HOME\technovert.env`. Mettez le mot de passe du compte `tv_appli` de l'[annexe A3](#), et une longue valeur au hasard pour `TV_CLE_SECRETE`. Enregistrez.

- ☐ Vérifiez son contenu :

```
Get-Content $HOME\technovert.env
```

Vous devez voir : quatre lignes de commentaire, puis vos cinq lignes `TV_...`, sans aucun `a-` remplacer. Les accents des commentaires peuvent mal s'afficher : ce n'est pas grave.

05 Basculer l'application sur la base

L'application importe encore l'ancienne couche, celle du fichier. On la fait pointer sur la base.

- ☐ Ouvrez `app.py`. Repérez la ligne `import donnees`.

- ☐ Remplacez-la par :

```
import acces_bdd as donnees
```

- ☐ Remplacez la ligne de la clé secrète par :

```
app.secret_key = os.environ["TV_CLE_SECRETE"]
```

- ☐ Dans `app.py`, juste au-dessus de `from flask import ...`, ajoutez la ligne `import os`. Enregistrez, puis lancez le serveur :

```
python -m flask --app app run --debug
```

- ☐ Connectez-vous sur `/connexion` avec le compte `ngarcia`, mot de passe `Chantier2026`. Les autres logins sont dans l'[annexe A2](#), table `compte`. Le mot de passe de test est le même pour tous. Puis ouvrez `http://127.0.0.1:5000/chantiers`.

Vous devez voir : les six chantiers, comme avant. Cette fois, ils viennent de MySQL.

Si vous voyez la page « Service momentanément indisponible » : MySQL est arrêté dans XAMPP, ou le mot de passe de `technovert.env` est faux. Corrigez, puis rechargez la page.

06 Vérifier que les données viennent de la base

- ☐ Ouvrez un second terminal avec le bouton « + » du panneau Terminal. Le premier fait tourner le serveur.
- ☐ Changez un nom de chantier directement dans la base :

```
mysql -u root -e "UPDATE technovert.chantier SET nom='Essai base' WHERE id=1;"
```

- ☐ Rechargez `/chantiers` dans le navigateur.

Vous devez voir : le premier chantier s'appelle maintenant « Essai base ». L'application lit bien la base en direct.

POINT DE CONTRÔLE A

- ☐ Montrez au formateur la liste avec « Essai base », et votre réponse à l'étape 3.
- ☐ Le formateur a validé. Passez à la partie B.

PARTIE B — Isoler et sortir les secrets

Étapes 7 à 11, environ 25 min. À la fin, tout le SQL est dans la couche d'accès, et aucun mot de passe n'est dans le code.

Besoin d'un rappel ? Revoir le cours : « [Une couche d'accès aux données](#) ».

07 Vérifier que les vues ne contiennent pas de SQL

- ☐ Cherchez le mot `SELECT` dans le fichier des vues :

```
Select-String -Pattern "SELECT", "INSERT", "mysql" -Path app.py
```

Vous devez voir : aucune ligne. Tout le SQL est dans `acces_bdd.py`.

Pourquoi ? Pourquoi ranger toutes les requêtes dans un seul fichier, au lieu d'en mettre dans chaque vue ?

Votre réponse :

08 Provoquer une panne de base

☐ Laissez le serveur tourner. Dans le panneau de XAMPP, cliquez sur « Stop », en face de « MySQL ».

☐ Rechargez `/chantiers` dans le navigateur.

Vous devez voir : la page « Service momentanément indisponible », pas une trace d'erreur.

09 Lire la vraie cause dans le journal

☐ Regardez le terminal du serveur.

Vous devez voir : une ligne qui contient `ERROR in app: Base indisponible`, suivie de la cause. La ligne juste en dessous se termine par `503 -`.

☐ Dans le panneau de XAMPP, cliquez sur « Start » en face de « MySQL ». Rechargez la page.

Vous devez voir : la liste des chantiers revient.

10 Trouver dans la doc

☐ Ouvrez l'[annexe A5](#).

☐

Trouvez comment passer une valeur à une requête sans la coller dans le texte. Notez le symbole employé dans la requête : _____

☐ Notez sous quelle forme les valeurs sont passées à `execute()` : _____

Vous devez avoir : un symbole, et le mot « tuple ». Ils servent en partie C.

11 Vérifier qu'aucun secret n'est dans le code

☐ Cherchez un mot de passe en clair dans les fichiers du code :

```
Select-String -Pattern "Labo-Appli", "password=" -Path app.py, acces_bdd.py
```

Vous devez voir : la seule ligne trouvée est `password=os.environ[...]`. Aucun mot de passe en clair.

POINT DE CONTRÔLE B

☐ Montrez au formateur la page 503, la ligne du journal, et vos réponses aux étapes 7 et 10.

☐ Le formateur a validé. Passez à la partie C.

PARTIE C — Injecter puis corriger

Étapes 12 à 17, environ 35 min. À la fin, une injection est prouvée, puis rendue inoffensive.

Besoin d'un rappel ? Revoir le cours : [« L'injection SQL et la requête paramétrée »](#).

Changez de rôle : le lecteur devient pilote.

Cadre légal. Vous testez une injection sur votre propre machine, dans un but d'apprentissage. Le faire sur un site qui n'est pas le vôtre, sans autorisation écrite, est un délit.

12 Lire la page vulnérable

Un collègue a repris la page « Mes interventions » de l'ancien prestataire. Sa fonction est `interventions_du_technicien`, en bas de `acces_bdd.py`.

☐ Ouvrez `acces_bdd.py` et lisez cette fonction.

☐ Notez comment le login et le service entrent dans la requête :

Vous devez avoir : les valeurs collées dans le texte avec des `+` et des apostrophes.

13 Voir la page normale

☐ Si le serveur est arrêté, relancez-le dans le premier terminal :

```
python -m flask --app app run --debug
```

☐ Connectez-vous avec le compte `ngarcia`, mot de passe `Chantier2026`.

☐ Ouvrez `/mes-interventions?service=atelier`.

Vous devez voir : la phrase `4 intervention(s) pour le service « atelier ».`, et un tableau de 4 lignes, toutes de `ngarcia`.

14 Prouver l'injection

- ☐ Dans la barre d'adresse, remplacez le service par une injection. Tapez cette adresse, puis validez :

```
http://127.0.0.1:5000/mes-interventions?service=x' OR '1'='1
```

Vous devez voir : un nombre d'interventions bien plus grand que 4, et des interventions d'autres techniciens. Vous voyez des données qui ne sont pas les vôtres.

- ☐ Notez le nombre d'interventions affiché : _____
- ☐ Faites tout de suite une capture d'écran de cette page. C'est votre preuve « avant correction » : après l'étape 15, vous ne pourrez plus la refaire.

Pourquoi ? Pourquoi la base a-t-elle renvoyé les interventions de tout le monde ?

Votre réponse :

15 Corriger avec une requête paramétrée

- ☐ Dans `interventions_du_technicien`, remplacez les lignes de `requete = (` jusqu'à `curseur.execute(requete)` par :

```
requete = ("SELECT i.date_intervention, c.nom AS chantier, i.technicien,  
i.service, "  
            "i.duree_heures FROM intervention i JOIN chantier c ON c.id =  
i.id_chantier "  
            "WHERE i.technicien = %s AND i.service = %s "  
            "ORDER BY i.date_intervention")  
curseur.execute(requete, (login, service))
```

☐ Enregistrez le fichier.

Vous devez voir : le fichier s'enregistre sans erreur. On teste l'effet à l'étape 16.

16 Vérifier que l'attaque échoue

☐ Rechargez la page normale `/mes-interventions?service=atelier`.

Vous devez voir : les 4 interventions de `ngarcia`, comme avant. La correction n'a rien cassé.

☐ Rechargez l'adresse d'injection de l'étape 14.

Vous devez voir : « 0 intervention(s) ». La base a cherché un service qui s'appelle vraiment `x' OR '1'='1`. Il n'existe pas.

17 Garder la preuve

Vous avez déjà la capture « avant correction », faite à l'étape 14.

☐ Faites maintenant une capture de la même adresse d'injection après correction : « 0 intervention(s) ».

Vous devez avoir : deux captures, l'une de l'étape 14, l'autre d'ici. Vous les montrez au point de contrôle C.

POINT DE CONTRÔLE C

- ☐ Montrez au formateur les deux preuves et votre réponse à l'étape 14.
- ☐ Le formateur a validé. Passez à la partie D.

PARTIE D — Exporter et restaurer

Étapes 18 à 22, environ 25 min. À la fin, la base est restaurée sur une pile vierge, et vérifiée.

Besoin d'un rappel ? Revoir le cours : [« Un jeu de données qu'on peut rejouer »](#).

Le jeu de test de l'[annexe A2](#) recrée une base neuve, toujours la même. Un export, lui, sauvegarde la **vraie** base, avec les saisies des utilisateurs. C'est cet export qu'on restaure quand une machine tombe en panne. On le prouve ici sur une base neuve.

18 Exporter la base

- ☐ Exportez toute la base dans un fichier, dans le dossier de l'application :

```
mysqldump -u root technover - --result-file=export_technover.sql
```

- ☐ Cherchez dans le fichier les lignes qui créent les tables et qui ajoutent les données :

```
Select-String -Pattern "CREATE TABLE", "INSERT INTO" -Path export_technover.sql
```

Vous devez voir : des lignes `CREATE TABLE` pour les quatre tables, et des lignes `INSERT INTO`. L'export contient la structure et les données.

19 Simuler une pile vierge

Une pile vierge, c'est une base qui n'existe pas encore. On efface la base pour le simuler.

-
- ☐ Supprimez la base :
-

```
mysql -u root -e "DROP DATABASE technover;"
```

-
- ☐ Rechargez `/chantiers` dans le navigateur.
-

Vous devez voir : la page « Service momentanément indisponible ». La base n'existe plus.

20 Restaurer

-
- ☐ Recréez la base vide, puis rechargez l'export dedans :
-

```
mysql -u root -e "CREATE DATABASE technover;"  
mysql -u root technover -e "source export_technover.sql"
```

-
- ☐ Rechargez `/chantiers`.
-

Vous devez voir : les six chantiers reviennent.

21 Vérifier la restauration avec le script

Une page qui s'affiche ne prouve pas que tout est là. Le script de l'[annexe A6](#) compte les lignes et teste les cas connus.

-
- ☐ Dans le second terminal, lancez le script de vérification. Il lit lui aussi `technovert.env` :

```
python A6_verifier_base.py
```

Vous devez voir : chaque ligne commence par OK, et la dernière ligne dit « 13 vérification(s) sur 13 réussie(s). ».

Si vous voyez un ÉCHEC : la restauration est incomplète, ou le jeu de test n'est pas le bon. Reprenez à l'étape 20.

22 Trouver dans la doc et conclure

-
- ☐ Ouvrez l'[annexe A6](#). Trouvez la dernière vérification du script, celle qui teste une injection.

- ☐ Notez ce qu'elle envoie comme service, et le résultat attendu :

Vous devez avoir : un service piégé, avec des apostrophes, et zéro ligne attendue.

Pourquoi ? Pourquoi vérifier une injection dans un script de recette, en plus de l'avoir corrigée ?

Votre réponse :

POINT DE CONTRÔLE D

☐ Montrez au formateur le script qui affiche 13 sur 13, et vos réponses aux étapes 22.

☐ Le formateur a validé. Le labo est fini.

POUR ALLER PLUS LOIN

10 à 15 min, si vous avez fini avant la fin de la séance.

Une IA a proposé cette fonction pour chercher un client par sa ville. Elle marche, mais elle contient trois défauts.

```
def clients_par_ville(ville):
    connexion = mysql.connector.connect(host="127.0.0.1", user="root",
                                       password="root",
                                       database="technovert")
    curseur = connexion.cursor(dictionary=True)
    curseur.execute(f"SELECT * FROM client WHERE ville = '{ville}'")
    return curseur.fetchall()
```

☐ Trouvez les trois défauts avant la mise en production. Pour chacun, notez la ligne et le danger.

Ligne	Le défaut	Le danger

CE QU'IL FAUT RETENIR

1. Une valeur saisie ne se colle jamais dans une requête : on écrit `%s` et on passe la valeur à part.
2. Tout le SQL vit dans la couche d'accès. Une base arrêtée donne une page 503 propre, jamais une trace.
3. Une restauration ne se croit pas : elle se prouve avec un script de vérification.

Un tableau de bord qui tient

MISE EN SITUATION-U6-S2-M6

Contexte — Karim Benali veut un tableau de bord pour suivre les interventions de l'atelier. Il sera exposé aux utilisateurs de TechnoVert.

Hélène Vasseur a une exigence : « Je ne veux pas qu'un petit malin casse la base en tapant dans une case. » Le formateur jouera cet utilisateur malveillant.

RÈGLE IA DE LA SÉANCE

Scénario 2 : l'IA peut vous expliquer une notion ou un message d'erreur. Elle n'écrit ni votre requête ni votre vue.

Si vous avez posé une question à une IA, dites-le pendant l'entretien. C'est pris en compte, pas reproché.

COMMENT VOUS SEREZ NOTÉ

Il n'y a aucun fichier à rendre. Vous êtes noté **à l'oral**, sur votre poste, pendant un entretien de 5 minutes avec le formateur. Il vient vous voir dès que votre tableau marche.

Pendant l'entretien, le formateur :

1. vous demande de montrer votre tableau, sans filtre puis avec un filtre ;
2. tape lui-même une saisie piégée dans l'adresse de votre page ;
3. vous pose des questions sur votre code ;
4. vous demande une petite modification, à faire devant lui en 2 minutes ;
5. regarde votre façon de travailler et d'expliquer.

Il suit cette grille. Lisez-la avant de commencer : c'est exactement ce qui sera noté.

Critère	Insuffisant (0 ou 1)	Fragile (2)	Acquis (3)	Maîtrisé (4)
Démonstration · le tableau lit la base	La page ne s'affiche pas, ou plante sur un cas normal.	La page s'affiche, mais le filtre ou le nom du chantier manque.	Toutes les interventions, avec leur chantier, et le filtre par service marche.	Tout marche, et le total d'heures correspond aux lignes montrées.
Résistance · la saisie piégée reste sans effet	La saisie affiche des données en plus, ou une trace d'erreur.	La saisie est bloquée, mais seulement par le menu ou par un test fragile.	La requête est paramétrée : la saisie ne change rien.	La requête est paramétrée, et vous le prouvez vous-même devant le formateur.
Questions techniques · vous expliquez votre code	Vous ne savez pas dire ce que fait votre requête.	Vous l'expliquez avec de l'aide.	Vous expliquez la requête, le <code>JOIN</code> et le rôle de <code>%s</code> .	Vous expliquez aussi, avec vos mots, pourquoi l'attaque échoue.
Travail personnel · la modification en direct	Vous ne trouvez pas où modifier.	Vous trouvez l'endroit, mais la modification n'aboutit pas.	La modification marche en 2 minutes.	Elle marche, et vous la testez vous-même sans qu'on vous le demande.
Professionnalisme · votre façon de travailler	Réponses floues, dossier en désordre, rien n'est dit de ce qui ne marche pas.	Vocabulaire approximatif, explications à reprendre.	Mots du métier, explications claires. Vous dites ce qui ne marche pas et l'aide reçue.	En plus, vous signalez un risque qui reste, ou une amélioration possible.

La note est la somme des cinq lignes, sur 20.

Ce dont vous disposez

- Votre poste Windows, avec XAMPP, Python, VS Code et un navigateur.
- [Annexe A7](#) : le tableau de bord de départ et son mode d'emploi. Le dossier se télécharge dans [A7_tableau_de_bord.zip](#). Il contient aussi les scripts de la base.
- [Annexe A2](#) : le jeu de données de la base.
- [Annexe A3](#) : le compte MySQL de l'application.
- [Annexe A4](#) : le fichier d'environnement, à ranger dans votre dossier personnel.

- [Annexe A5](#) : un extrait de la documentation du connecteur MySQL. Il rappelle la requête paramétrée avec `%s`.

Pour lire le service choisi dans l'adresse (`/tableau?service=atelier`), la vue utilise `request.args.get("service")`. Il renvoie `None` si l'adresse ne contient pas de service, et un texte vide `""` si on choisit « Tous les services » dans le menu. Écrivez `request.args.get("service") or None` : les deux cas donnent alors `None`.

CE QU'ON VOUS DEMANDE

Le tableau de bord a une seule page, `/tableau`. Elle affiche toutes les interventions, avec le nom de leur chantier. Un menu filtre les interventions par service.

La fonction `interventions_par_service` de la couche d'accès est à écrire. La vue qui l'appelle est à compléter. La saisie du service ne doit jamais pouvoir changer la requête envoyée à la base.

Pendant l'entretien, le formateur tapera lui-même un service piégé dans votre page. Votre tableau doit rester sain : aucune donnée qui ne devrait pas s'afficher, aucune trace d'erreur.

PAR OÙ COMMENCER

45 minutes en tout : environ 40 minutes de travail, et l'entretien de 5 minutes dès que votre tableau marche.

1. Préparez votre poste avec l'[annexe A7](#) : dossier extrait, MySQL lancé, base chargée, `technovert.env` rempli. Lancez le tableau de bord et ouvrez `/tableau`. Environ 5 min.
2. Écrivez `interventions_par_service` d'abord **sans filtre** : la requête relie l'intervention à son chantier avec `JOIN`, et renvoie tout. Environ 10 min.
3. Ajoutez le **filtre** : quand un service est donné, ajoutez `WHERE i.service = %s` et passez le service en paramètre. Environ 10 min.
4. Complétez la vue `tableau()` : elle lit le service avec `request.args.get`, appelle la couche, et calcule le total des heures. Le gabarit attend aussi `message` : c'est un texte à afficher quand le service demandé n'est pas dans `SERVICES`, sinon `None`. Environ 10 min.
5. Testez avec un service normal, puis avec une injection de votre choix. Environ 5 min.

CE QUI DOIT MARCHER À LA FIN

- ☐ Sans filtre, le tableau affiche les 12 interventions, avec le nom de chaque chantier.
 - ☐ Le filtre « atelier » n'affiche que les interventions de l'atelier.
 - ☐ Le total des heures affiché correspond aux lignes montrées.
 - ☐ Une saisie qui contient une apostrophe ne renvoie aucune ligne et ne provoque aucune trace.
 - ☐ Vous savez expliquer chaque ligne de votre requête et de votre vue.
-

Bloqué ? Dites-le au formateur pendant l'entretien. Savoir expliquer où l'on bloque fait partie du professionnalisme.

Tous les documents du module

A1 — L'application de départ et son mode d'emploi

↓ Télécharger [A1_technovert_web.zip](#)

Source : application écrite pour le module. Le mini-site vient du travail sur le développement web ; il est fourni ici en entier, prêt à brancher sur la base. Fichier [A1_technovert_web.zip](#) à télécharger ci-dessus. Utilisée dans : LABO parties A à D.

Ce que contient le dossier

technovert-web/	
app.py	les routes et les vues : vous y changez une ligne
d'import	
acces_bdd.py	la couche d'accès à la base : c'est ici que vous
écrivez	
donnees.py	l'ancienne couche, sur fichier : gardée pour comparer,
non modifiée	
chantiers.json	les anciennes données sur fichier : plus utilisées
après la bascule	
templates/	les gabarits HTML : fournis
static/style.css	la feuille de style : fournie
technovert.env.exemple	le modèle de fichier d'environnement (annexe A4)
A2_technovert.sql	le jeu de données de la base (annexe A2)
A3_compte_appli.sql	le compte MySQL de l'application (annexe A3)
A6_verifier_base.py	le script de vérification (annexe A6)

Installer le dossier sur votre poste

1. Téléchargez [A1_technovert_web.zip](#) depuis cette annexe.
2. Faites un clic droit sur le fichier, puis « Extraire tout ». Avec « Parcourir », choisissez le dossier « Documents ».
3. Ouvrez le dossier `Documents\technovert-web` dans VS Code, puis un terminal : menu « Terminal », « Nouveau terminal ».

Pour repartir de zéro, supprimez le dossier, extrayez de nouveau le zip, et rejouez

[A2_technovert.sql](#) .

Lancer MySQL et l'application

1. Dans le panneau de contrôle de XAMPP, cliquez sur « Start » en face de « MySQL ». Le nom passe au vert.
2. Dans le terminal de VS Code, tapez `python -m flask --app app run --debug`. Si `python` n'est pas reconnu, tapez `py`.
3. Ouvrez `http://127.0.0.1:5000/chantiers` dans le navigateur. On arrête le serveur avec Ctrl+C.

Rendre la commande mysql disponible

La commande `mysql` est dans le dossier `C:\xampp\mysql\bin`. Pour qu'elle marche dans tous les terminaux, tapez ces deux lignes **une seule fois**, dans PowerShell, puis fermez et rouvrez VS Code :

```
$u = [Environment]::GetEnvironmentVariable("Path", "User")  
[Environment]::SetEnvironmentVariable("Path", "$u;C:\xampp\mysql\bin", "User")
```

Test : `mysql --version` affiche une version de MariaDB.

Le fichier acces_bdd.py de départ

Les premières lignes chargent vos identifiants avec `load_dotenv`. Les fonctions `ouvrir()` et `lire_chantiers()` sont à écrire. La fonction `interventions_du_technicien`, tout en bas, est volontairement vulnérable : vous la corrigez en partie C.

```

"""Couche d'accès aux données de TechnoVert : tout le SQL de l'application est ici.

Les vues de app.py n'écrivent jamais de SQL. Elles appellent ces fonctions.
"""

import os
from pathlib import Path

import mysql.connector
from dotenv import load_dotenv
from werkzeug.security import check_password_hash

# Les identifiants sont dans votre dossier personnel, hors du code.
load_dotenv(Path.home() / "technovert.env")

SERVICES = ["atelier", "bureau etudes", "commercial"]

COLONNES_CHANTIER = ("SELECT ch.id, ch.id_client, cl.nom AS client, ch.nom,
ch.adresse, "
                    "ch.date_debut, ch.date_fin_prevue, ch.statut "
                    "FROM chantier ch JOIN client cl ON cl.id = ch.id_client ")

class BaseIndisponible(Exception):
    """La base ne répond pas."""

def ouvrir():
    # À ÉCRIRE (partie A)
    pass

def lire_chantiers():
    # À ÉCRIRE (partie A)
    return []

def lire_chantier(id_chantier):
    """Un chantier, ou None s'il n'existe pas."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute(COLONNES_CHANTIER + "WHERE ch.id = %s", (id_chantier,))
    ligne = curseur.fetchone()
    connexion.close()
    return ligne

def ajouter_intervention(id_chantier, technicien, service, date_intervention,
                        duree_heures, description, blocage=None):
    """Ajoute une intervention et renvoie son numéro."""
    connexion = ouvrir()
    curseur = connexion.cursor()
    curseur.execute(

```

```

        "INSERT INTO intervention (id_chantier, technicien, service,
date_intervention, "
        "duree_heures, description, cloturee, blocage) "
        "VALUES (%s, %s, %s, %s, %s, %s, FALSE, %s)",
        (id_chantier, technicien, service, date_intervention, duree_heures,
description,
        blocage or None))
    connexion.commit()
    numero = curseur.lastrowid
    connexion.close()
    return numero

def verifier_compte(login, mot_de_passe):
    """True si le login existe et si le mot de passe est le bon."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    curseur.execute("SELECT empreinte FROM compte WHERE login = %s", (login,))
    ligne = curseur.fetchone()
    connexion.close()
    return ligne is not None and check_password_hash(ligne["empreinte"],
mot_de_passe)

# Écrite par l'ancien prestataire. Elle marche, mais elle est dangereuse.
# Ne la corrigez pas avant l'étape 15.
def interventions_du_technicien(login, service):
    """Les interventions d'un technicien pour un service."""
    connexion = ouvrir()
    curseur = connexion.cursor(dictionary=True)
    requete = ("SELECT i.date_intervention, c.nom AS chantier, i.technicien,
i.service, "
               "i.duree_heures FROM intervention i JOIN chantier c ON c.id =
i.id_chantier "
               "WHERE i.technicien = '" + login + "' AND i.service = '" +
service + "' "
               "ORDER BY i.date_intervention")
    curseur.execute(requete)
    lignes = curseur.fetchall()
    connexion.close()
    return lignes

```

A2 — Le jeu de données de la base

↓ Télécharger [A2_technovert.sql](#)

Source : structure d'après le document de scénario, § 5, « Schéma relationnel de référence ». Les lignes sont écrites pour le module. Fichier [A2_technovert.sql](#) à télécharger ci-dessus,

aussi présent dans les dossiers de départ. Utilisée dans : LABO étapes 2, 5 et 21, MISE EN SITUATION.

Ce script efface la base `technovert`, la recrée et la remplit. On le rejoue pour revenir à un état connu. Dans le dossier du fichier :

```
mysql -u root -e "source A2_technovert.sql"
```

Le mot de passe de test de tous les comptes est `Chantier2026`. La base ne range que l'empreinte du mot de passe, jamais le mot de passe lui-même.

```

-- Base TechnoVert : structure et jeu de données de test.
-- Rejouable : chaque lancement efface la base et la recrée à l'identique.
-- Lancement, dans le dossier du fichier : mysql -u root -e "source
A2_technovert.sql"

SET NAMES utf8mb4;

DROP DATABASE IF EXISTS technovert;
CREATE DATABASE technovert CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE technovert;

CREATE TABLE client (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    ville VARCHAR(60),
    telephone VARCHAR(20),
    email VARCHAR(100)
);

CREATE TABLE chantier (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_client INT NOT NULL,
    nom VARCHAR(100),
    adresse VARCHAR(150),
    date_debut DATE,
    date_fin_prevue DATE,
    statut ENUM('ouvert', 'cloture') NOT NULL DEFAULT 'ouvert',
    FOREIGN KEY (id_client) REFERENCES client(id)
);

CREATE TABLE intervention (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_chantier INT NOT NULL,
    technicien VARCHAR(30) NOT NULL,
    service ENUM('atelier', 'bureau etudes', 'commercial') NOT NULL,
    date_intervention DATE,
    duree_heures DECIMAL(4,1),
    description VARCHAR(255),
    cloturee BOOLEAN NOT NULL DEFAULT FALSE,
    blocage VARCHAR(100) NULL,
    FOREIGN KEY (id_chantier) REFERENCES chantier(id)
);

CREATE TABLE compte (
    login VARCHAR(30) PRIMARY KEY,
    empreinte VARCHAR(255) NOT NULL
);

INSERT INTO client (id, nom, ville, telephone, email) VALUES
    (1, 'Commune de Vernay', 'Vernay', '01 99 00 10 01',
'services.techniques@vernay.example'),
    (2, 'Golf des Trois Chênes', 'Montval', '01 99 00 10 02', 'accueil@golf-trois-

```

```

chenes.example'),
  (3, 'Copropriété Les Tilleuls', 'Vernay', '01 99 00 10 03', 'syndic@les-
tilleuls.example'),
  (4, 'Commune de Bellerive', 'Bellerive', '01 99 00 10 04',
'mairie@bellerive.example');

INSERT INTO chantier (id, id_client, nom, adresse, date_debut, date_fin_prevue,
statut) VALUES
  (1, 1, 'Arrosage du parc de la mairie', 'Parc de la mairie, Vernay',
'2026-03-02', '2026-04-30', 'cloture'),
  (2, 2, 'Arrosage des greens 1 à 9', 'Route de Montval, Montval', '2026-04-06',
'2026-06-26', 'ouvert'),
  (3, 3, 'Éclairage des allées', '12 rue des Tilleuls, Vernay', '2026-05-11',
'2026-06-12', 'ouvert'),
  (4, 4, 'Éclairage du stade', 'Stade municipal, Bellerive', '2026-06-01',
'2026-09-30', 'ouvert'),
  (5, 2, 'Éclairage du parking', 'Route de Montval, Montval', '2026-09-07',
'2026-11-27', 'ouvert'),
  (6, 1, 'Arrosage des ronds-points', 'Avenue de la Gare, Vernay', '2026-09-14',
'2026-12-18', 'ouvert');

INSERT INTO intervention (id, id_chantier, technicien, service, date_intervention,
duree_heures, description, cloturee, blocage) VALUES
  (1, 1, 'ngarcia', 'atelier', '2026-03-03', 6.0, 'Pose des programmeurs', TRUE,
NULL),
  (2, 1, 'ngarcia', 'atelier', '2026-03-10', 7.5, 'Raccordement des vannes', TRUE,
NULL),
  (3, 1, 'yropars', 'commercial', '2026-04-28', 1.0, 'Visite de réception avec le
client', TRUE, NULL),
  (4, 2, 'lmarchetti', 'bureau etudes', '2026-04-02', 3.0, 'Relevé des pentes des
greens', TRUE, NULL),
  (5, 2, 'lmoreau', 'atelier', '2026-04-07', 8.0, 'Tranchées et pose des tuyaux',
TRUE, NULL),
  (6, 2, 'lmoreau', 'atelier', '2026-05-19', 4.0, 'Réglage des arroseurs', FALSE,
'acces refuse'),
  (7, 3, 'ngarcia', 'atelier', '2026-05-12', 5.5, 'Pose des bornes lumineuses',
FALSE, NULL),
  (8, 3, 'yropars', 'commercial', '2026-05-20', 0.5, 'Point d'avancement avec le
syndic', FALSE, 'client absent'),
  (9, 4, 'lmarchetti', 'bureau etudes', '2026-06-02', 2.5, 'Calcul de l'éclairage
du terrain', TRUE, NULL),
  (10, 4, 'lmoreau', 'atelier', '2026-06-15', 8.0, 'Montage des mâts', FALSE,
NULL),
  (11, 5, 'yropars', 'commercial', '2026-09-08', 1.0, 'Signature du devis sur
place', FALSE, NULL),
  (12, 6, 'ngarcia', 'atelier', '2026-09-15', 6.0, 'Pose des goutteurs', FALSE,
NULL);

-- Mot de passe de test de tous les comptes : Chantier2026. Seule l'empreinte est
rangée.
INSERT INTO compte (login, empreinte) VALUES
  ('kbenali',

```

```
'pbkdf2:sha256:1000000$0UkaJXqPPzn8eBRD$777cc1cadfd0aaaf8922f784fb4fd7df9518e94b9b4fa894552b7bf6
('yropars',
'pbkdf2:sha256:1000000$6ZEROCdDqgIhstCD$9ea8f945a738982e98d73214d52eb78212e1247ffb3d549c726d702e9
('lmarchetti',
'pbkdf2:sha256:1000000$DONiyJYIQZcnyZ3M$af59cc4fdb251362d0a509b3456488116c429628be176867cf7b04fa
('ngarcia',
'pbkdf2:sha256:1000000$26To0JU2Eavjsw6a$67c3369e9af86e1869331e65c6dd39d04c09dd174fd77a3b07e08fc
('lmoreau',
'pbkdf2:sha256:1000000$JhIzqxAL0VVgGcJ$92cc58a9dc617191d58a4c623d66b276cf2e67e6b395d9049ec975d1c
```

A3 — Le compte MySQL de l'application

↓ Télécharger [A3_compte_appli.sql](#)

Source : compte écrit pour le module. Fichier [A3_compte_appli.sql](#) à télécharger ci-dessus, aussi présent dans les dossiers de départ. Utilisée dans : LABO étapes 2 et 4, MISE EN SITUATION.

L'application ne se connecte jamais avec le compte administrateur. Elle utilise un compte à droits réduits : lire, ajouter, modifier. Il ne peut ni supprimer de table ni changer la structure.

```
mysql -u root -e "source A3_compte_appli.sql"
```

```
-- Compte MySQL de l'application TechnoVert, pour le labo seulement.
-- Lancement, dans le dossier du fichier : mysql -u root -e "source
A3_compte_appli.sql"
-- En production, le mot de passe est choisi à l'installation.
-- Il n'est écrit que dans le fichier d'environnement du serveur, jamais dans le
code.

-- Deux lignes par compte : MySQL distingue « localhost » et « 127.0.0.1 ».
CREATE USER IF NOT EXISTS 'tv_appli'@'localhost' IDENTIFIED BY 'Labo-Appli-2026';
CREATE USER IF NOT EXISTS 'tv_appli'@'127.0.0.1' IDENTIFIED BY 'Labo-Appli-2026';
GRANT SELECT, INSERT, UPDATE ON technover.* TO 'tv_appli'@'localhost';
GRANT SELECT, INSERT, UPDATE ON technover.* TO 'tv_appli'@'127.0.0.1';
```

A4 — Le fichier d'environnement à remplir

Source : modèle écrit pour le module, fourni dans les dossiers de départ sous le nom `technover.env.exemple`. Utilisée dans : LABO étape 4, MISE EN SITUATION.

Ce fichier contient les identifiants de la base et la clé secrète de l'application. Il ne va jamais dans le dossier du code : il se range dans votre dossier personnel, `C:\Users\` suivi de votre nom. Les autres comptes de la machine ne le lisent pas.

L'application le lit seule au démarrage, avec la ligne `load_dotenv(Path.home() / "technovert.env")` de `acces_bdd.py`. Rien à charger à la main, même dans un nouveau terminal.

Dans le dossier de l'application :

```
Copy-Item technovert.env.exemple $HOME\technovert.env
notepad $HOME\technovert.env
```

Le modèle à recopier :

```
# Fichier d'environnement de l'application : modèle à recopier.
# Il se range dans votre dossier personnel, jamais dans le dossier du code.
# Copie, dans PowerShell : Copy-Item technovert.env.exemple $HOME\technovert.env
# Ouverture pour le remplir : notepad $HOME\technovert.env
TV_BDD_HOTE=127.0.0.1
TV_BDD_UTILISATEUR=tv_appli
TV_BDD_MOT_DE_PASSE=a-replacer
TV_BDD_BASE=technovert
TV_CLE_SECRETE=a-replacer-par-une-longue-valeur-au-hasard
```

- `TV_BDD_MOT_DE_PASSE` : le mot de passe du compte `tv_appli` de l'[annexe A3](#).
- `TV_CLE_SECRETE` : une longue valeur au hasard. Elle signe le cookie de session.

A5 — Extrait de la documentation du connecteur MySQL

Source : documentation de `mysql-connector-python`, pages « Connecting » et « `Cursor.execute` ». Extrait traduit et raccourci pour le module. Utilisée dans : LABO étapes 10 et 15, MISE EN SITUATION.

Ouvrir une connexion. `mysql.connector.connect()` reçoit `host`, `user`, `password`, `database`. Il lève `mysql.connector.Error` si la connexion échoue. L'option `connection_timeout` fixe le temps d'attente, en secondes.

Le curseur. `connexion.cursor(dictionary=True)` crée un curseur qui renvoie chaque ligne sous forme de dictionnaire, avec les noms de colonnes comme clés.

Exécuter une requête paramétrée. On écrit `%s` à chaque endroit où une valeur entre. On passe les valeurs à part, dans un tuple, en second argument.

```
curseur.execute("SELECT * FROM chantier WHERE statut = %s", (statut,))
```

- Le connecteur remplace chaque `%s` par la valeur, en la traitant comme une donnée, jamais comme du SQL.
- Le `%s` ne prend pas d'apostrophes autour, même pour un texte. Le connecteur les ajoute.
- Pour une seule valeur, le tuple s'écrit `(valeur,)`, avec une virgule.

Lire et écrire.

- `curseur.fetchall()` renvoie toutes les lignes d'un `SELECT`, dans une liste.
- `curseur.fetchone()` renvoie une seule ligne, ou `None`.
- Après un `INSERT`, un `UPDATE` ou un `DELETE`, il faut `connexion.commit()` pour garder le changement.
- `curseur.lastrowid` donne le numéro de la dernière ligne ajoutée.

Fermer. `connexion.close()` ferme la connexion. On la ferme dès que la requête est finie.

A6 — Le script de vérification de la base

↓ Télécharger [A6_verifier_base.py](#)

Source : script écrit pour le module. Fichier [A6_verifier_base.py](#) à télécharger ci-dessus, aussi présent dans le dossier de départ du labo. Utilisée dans : LABO étapes 21 et 22.

Ce script se branche sur la base par la couche d'accès. Il compte les lignes, teste les cas connus, et vérifie qu'une injection reste sans effet. Il affiche OK ou ÉCHEC sur chaque ligne.

```
python A6_verifier_base.py
```

Il lit lui aussi `technovert.env`. On le lance après avoir rejoué le jeu de données de l'[annexe A2](#).

""Vérifie la base TechnoVert et la couche d'accès, avec le jeu de données de test.

À lancer dans le dossier de l'application :

```
python A6_verifier_base.py
```

Les identifiants sont lus dans technovert.env, rangé dans votre dossier personnel.

Chaque ligne affiche OK ou ÉCHEC. Le jeu de test doit avoir été rejoué juste avant.

```
"""
```

```
import sys
```

```
import acces_bdd
```

```
resultats = []
```

```
def verifier(nom, condition):
    resultats.append(condition)
    print(("OK      " if condition else "ÉCHEC  ") + nom)
```

```
def compter(table):
    connexion = acces_bdd.ouvrir()
    curseur = connexion.cursor()
    curseur.execute("SELECT COUNT(*) FROM " + table)    # nom de table fixe, jamais
saisi
    nombre = curseur.fetchone()[0]
    connexion.close()
    return nombre
```

```
try:
    acces_bdd.ouvrir().close()
except acces_bdd.BaseIndisponible as erreur:
    print("ÉCHEC  connexion à la base :", erreur)
    print("Vérifiez que MySQL est lancé dans le panneau XAMPP, et le contenu de
technovert.env.")
    sys.exit(1)
except KeyError as erreur:
    print("ÉCHEC  variable d'environnement absente :", erreur)
    print("Vérifiez que technovert.env existe dans votre dossier personnel, et
qu'il est complet.")
    sys.exit(1)
```

```
verifier("connexion à la base", True)
verifier("4 clients", compter("client") == 4)
verifier("6 chantiers", compter("chantier") == 6)
verifier("12 interventions", compter("intervention") == 12)
verifier("5 comptes", compter("compte") == 5)

chantiers = acces_bdd.lire_chantiers()
```

```

verifier("lire_chantiers() renvoie 6 chantiers", len(chantiers) == 6)
verifier("chaque chantier a le nom de son client",
        all(c.get("client") for c in chantiers))
premier = acces_bdd.lire_chantier(1)
verifier("le chantier 1 est clôturé", premier is not None and premier["statut"] ==
"cloture")
verifier("le chantier 99 n'existe pas", acces_bdd.lire_chantier(99) is None)
verifier("le compte kbenali accepte son mot de passe",
        acces_bdd.verifier_compte("kbenali", "Chantier2026"))
verifier("le compte kbenali refuse un mauvais mot de passe",
        not acces_bdd.verifier_compte("kbenali", "faux"))
verifier("ngarcia a 4 interventions pour l'atelier",
        len(acces_bdd.interventions_du_technicien("ngarcia", "atelier")) == 4)

try:
    lignes = acces_bdd.interventions_du_technicien("ngarcia", "x' OR '1'='1")
    verifier("une saisie avec des apostrophes ne renvoie aucune ligne", len(lignes)
== 0)
except Exception as erreur:
    verifier("une saisie avec des apostrophes ne provoque pas d'erreur SQL (" +
        type(erreur).__name__ + ")", False)

print()
print(f"{sum(resultats)} vérification(s) sur {len(resultats)} réussie(s).")
sys.exit(0 if all(resultats) else 1)

```

A7 — Le tableau de bord de départ et son mode d'emploi

↓ Télécharger [A7_tableau_de_bord.zip](#)

Source : application écrite pour le module. Fichier [A7_tableau_de_bord.zip](#) à télécharger ci-dessus. Utilisée dans : MISE EN SITUATION.

Ce que contient le dossier

```

tableau-de-bord/
  app.py                la vue tableau() à compléter
  acces_bdd.py          la fonction interventions_par_service() à écrire
  templates/            les gabarits, dont tableau.html : fournis
  static/style.css      la feuille de style : fournie
  technovert.env.exemple le modèle de fichier d'environnement (annexe A4)
  A2_technovert.sql     le jeu de données de la base (annexe A2)
  A3_compte_appli.sql   le compte MySQL de l'application (annexe A3)

```

Préparer le poste

1. Extrayez `A7_tableau_de_bord.zip` dans « Documents » : « Extraire tout », puis « Parcourir ». Ouvrez le dossier `Documents\tableau-de-bord` dans VS Code, avec un terminal.
2. Lancez MySQL dans XAMPP.
3. Rechargez la base, même si elle existe déjà depuis le labo : `mysql -u root -e "source A2_technovert.sql"`, puis `mysql -u root -e "source A3_compte_appli.sql"`.
4. Si `technovert.env` n'est pas déjà dans votre dossier personnel, suivez l'[annexe A4](#).
5. Lancez le tableau de bord : `python -m flask --app app run --debug`, puis ouvrez `http://127.0.0.1:5000/tableau`.

La fonction à écrire

`interventions_par_service(service=None)` renvoie les interventions, avec le nom de leur chantier. Si on donne un service, seulement celles de ce service. Si on ne donne rien, toutes.

Chaque ligne renvoyée doit avoir ces clés, car le gabarit les lit : `date_intervention`, `chantier`, `technicien`, `service`, `duree_heures`, `blocage`. Le nom du chantier doit s'appeler `chantier` : écrivez `c.nom AS chantier` dans la requête. Une clé mal nommée ne donne aucun message d'erreur : la colonne reste vide.

La vue à compléter

`tableau()` lit le service dans l'adresse, appelle la couche, puis passe au gabarit la liste, le total des heures et le service choisi. Le gabarit `tableau.html` attend : `services`, `service`, `lignes`, `total`, `message`.