
Mini-Projet Individuel : GameTracker

Pipeline de données de scores de jeux vidéo
Automatisation et Tests en Programmation

BUT Science des Données – 2025-2026

Durée : 6h

Objectifs du projet

- Construire un projet complet **from scratch** mobilisant les compétences des séances 1 à 5
- Conteneuriser une application multi-services avec Docker Compose
- Développer un pipeline ETL en Python avec connexion MySQL
- Automatiser des tâches avec des scripts Bash
- Documenter et versionner son travail avec Git

Prérequis

- Cours et TD des séances 1 à 5 (Docker, Bash, Docker Compose, Python/MySQL, Git)
- Docker et Docker Compose installés et fonctionnels
- Un compte GitHub

Attention

Ce projet est **individuel**. Vous pouvez consulter vos notes de TD et vous inspirer du projet fil rouge, mais le code doit être **écrit par vos soins et adapté** au contexte de ce projet. L'objectif est de vérifier que vous maîtrisez les compétences de manière autonome.

Contexte du projet

Vous travaillez pour **GameTracker**, une startup qui analyse les performances des joueurs de jeux vidéo. On vous fournit deux fichiers CSV contenant des données brutes :

- `Players.csv` – Profils des joueurs (25 entrées)
- `Scores.csv` – Scores et sessions de jeu (40 entrées)

Problème : Ces données sont **sales**. Votre mission est de construire un pipeline automatisé qui nettoie ces données, les charge dans une base MySQL, et génère un rapport de synthèse. Le tout doit être conteneurisé, automatisé et versionné.

Aperçu des données

Players.csv :

Colonne	Type attendu en BDD	Description
player_id	INT (PK)	Identifiant unique du joueur
username	VARCHAR(100)	Pseudo du joueur
email	VARCHAR(200)	Adresse email
registration_date	DATE	Date d'inscription
country	VARCHAR(100)	Pays du joueur
level	INT	Niveau actuel

Scores.csv :

Colonne	Type attendu en BDD	Description
score_id	VARCHAR(20) (PK)	Identifiant du score
player_id	INT (FK)	Référence vers le joueur
game	VARCHAR(100)	Nom du jeu
score	INT	Score obtenu
duration_minutes	INT	Durée de la session
played_at	DATETIME	Date et heure de la session
platform	VARCHAR(50)	Plateforme (PC, Console, Mobile)

Problèmes de qualité présents dans les données

Votre pipeline doit détecter et traiter les 7 problèmes suivants :

1. **Doublons** : certains joueurs et scores apparaissent plusieurs fois
2. **Emails invalides** : certains emails ne contiennent pas de @
3. **Dates incohérentes** : formats variés (2023-06-15, 15/03/2023, date_inconnue, 30-02-2024)
4. **Espaces parasites** : espaces en début/fin de certains username
5. **Scores négatifs** : certains scores sont négatifs (aberrants)
6. **Valeurs manquantes** : certains champs email ou score sont vides
7. **Références orphelines** : certains scores réfèrent à un player_id qui n'existe pas dans Players.csv

Structure attendue du projet

```
gametracker/
  docker-compose.yml
  Dockerfile
  requirements.txt
  .gitignore
  README.md
  data/
    raw/
      Players.csv
      Scores.csv
  scripts/
    init-db.sql
    wait-for-db.sh
    run_pipeline.sh
  src/
```

```
__init__.py
config.py
database.py
extract.py
transform.py
load.py
report.py
main.py
output/
    (rapports générés ici)
```

Barème

Barème		
Critère	Détail	Points
Docker Compose	Le projet se lance avec <code>docker compose up --build</code>	/3
Base de données	Schéma SQL correct, tables créées automatiquement	/2
Pipeline ETL	Extract, Transform et Load fonctionnels	/5
Nettoyage des données	Les 7 problèmes de qualité sont gérés	/3
Scripts Bash	Automatisation complète du pipeline	/2
Rapport de synthèse	Génération automatique de statistiques	/2
Git	Historique propre, commits réguliers et explicites	/1
README	Documentation claire et complète	/1
Qualité du code	Code lisible, bien structuré, modulaire	/1
Total		/20

Étape 1 : Initialisation du projet et Docker Compose*env. 1h*

1. Créez l'arborescence complète du projet `gametracker/` et initialisez un dépôt Git.
2. Créez un `.gitignore` adapté (ignorant `__pycache__`, `*.pyc`, `.env`, `output/`).
3. Créez un `Dockerfile` pour l'application Python :
 - Image de base : `python:3.11-slim`
 - Installation du client MySQL et des dépendances Python
 - Copie du code et scripts exécutables
4. Créez un `requirements.txt` contenant `mysql-connector-python` et `pandas`.
5. Créez un `docker-compose.yml` avec deux services :
 - `db` : MySQL 8.0 avec `healthcheck` et volume persistant
 - `app` : votre application, dépendante de `db` (`service_healthy`), avec les volumes nécessaires pour monter `data/`, `src/`, `scripts/` et `output/`
6. Vérifiez que les deux services démarrent et que vous pouvez entrer dans le conteneur `app`.

Livrable attendu

Premier commit contenant : structure du projet, `Dockerfile`, `docker-compose.yml`, `requirements.txt`, `.gitignore`, fichiers CSV.

Étape 2 : Base de données et connexion Python*env. 1h*

1. Créez `scripts/init-db.sql` : le schéma de la base avec les tables `players` et `scores`. La table `scores` doit avoir une clé étrangère vers `players`. Utilisez `CREATE TABLE IF NOT EXISTS`.
2. Créez `scripts/wait-for-db.sh` : un script qui boucle en tentant une connexion MySQL (max 30 tentatives, 2s d'intervalle) en utilisant les variables d'environnement.
3. Créez `src/config.py` : une classe `Config` qui lit les paramètres de connexion depuis les variables d'environnement, avec des valeurs par défaut.
4. Créez `src/database.py` avec :
 - `get_connection()` – retourne une connexion MySQL
 - `get_connection_with_retry()` – retente en cas d'échec
 - `database_connection()` – context manager gérant commit, rollback et fermeture
5. Testez : lancez le script d'attente, initialisez les tables avec le script SQL, puis vérifiez la connexion Python.

Livrable attendu

Commit avec : `init-db.sql`, `wait-for-db.sh`, `config.py`, `database.py`, `__init__.py`.

Étape 3 : Pipeline ETL*env. 1h30*

C'est le cœur du projet.

3.1 – Extract (src/extract.py)

Écrivez une fonction `extract(filepath)` qui lit un CSV avec pandas et retourne un `DataFrame`. Elle doit vérifier l'existence du fichier et afficher le nombre de lignes extraites.

3.2 – Transform (src/transform.py)

Écrivez deux fonctions de transformation.

`transform_players(df)` doit :

1. Supprimer les doublons sur `player_id`
2. Nettoyer les espaces des `username` (`strip`)
3. Convertir les dates d'inscription (`pd.to_datetime, errors='coerce'`)
4. Remplacer les emails invalides (sans @) par `None`

`transform_scores(df, valid_player_ids)` doit :

1. Supprimer les doublons sur `score_id`
2. Convertir les dates et les scores en types numériques appropriés
3. Supprimer les lignes avec un score négatif ou nul
4. Supprimer les scores dont le `player_id` n'est pas dans `valid_player_ids`

Attention

Contrairement au projet fil rouge, `transform_scores` prend un deuxième paramètre : la liste des `player_id` valides (issus du `DataFrame` `players` nettoyé). Cela permet de filtrer les références orphelines **avant** le chargement en base.

3.3 – Load (src/load.py)

Écrivez `load_players(df, conn)` et `load_scores(df, conn)` qui insèrent les données en base avec `ON DUPLICATE KEY UPDATE`.

Conseil

Gérez correctement les valeurs `NaN/NaT` de pandas en les convertissant en `None` pour MySQL.

3.4 – Vérification

Testez chaque module individuellement depuis le conteneur avant de passer à la suite.

Livrable attendu

Commit avec : `extract.py`, `transform.py`, `load.py`.

Étape 4 : Rapport de synthèse et automatisation

env. 1h

4.1 – Module de rapport (src/report.py)

Créez une fonction `generate_report()` qui interroge la base de données et génère un fichier `output/rapport.txt`. Ce module est **nouveau** par rapport au projet fil rouge.

Le rapport doit contenir :

1. **Statistiques générales** : nombre de joueurs, nombre de scores, nombre de jeux différents
2. **Top 5 des meilleurs scores** avec le pseudo du joueur et le nom du jeu
3. **Score moyen par jeu**
4. **Répartition des joueurs par pays**
5. **Répartition des sessions par plateforme**

Conseil

Utilisez des requêtes SQL avec `JOIN`, `GROUP BY`, `ORDER BY`, `COUNT` et `AVG`. Pensez à utiliser votre context manager `database_connection()` et à écrire les résultats dans le fichier avec `open()` en mode écriture.

Exemple de format de sortie attendu :

```
=====
      GAMETRACKER - Rapport de synthese
      Genere le : 2024-02-22 15:30:00
=====

--- Statistiques generales ---
Nombre de joueurs : 22
Nombre de scores  : 35
Nombre de jeux    : 3

--- Top 5 des meilleurs scores ---
  1. DarkPhoenix      | SpaceInvaders | 22500
  2. DragonSlayer99   | SpaceInvaders | 21000
  3. QuantumLeap      | SpaceInvaders | 20200
  4. FrostBite        | SpaceInvaders | 19400
  5. CyberNinja       | SpaceInvaders | 18500

--- Score moyen par jeu ---
SpaceInvaders : 15850.5
TetrisUltra   : 7980.3
MarioKart     : 8540.0

--- Joueurs par pays ---
France : 14
Belgique : 4
Suisse : 3
Canada : 3

--- Sessions par plateforme ---
PC : 20
Console : 12
Mobile : 5
```

=====

Note : les chiffres ci-dessus sont indicatifs. Vos résultats dépendront de votre nettoyage.

4.2 – Script d’automatisation (scripts/run_pipeline.sh)

Créez un script Bash qui enchaîne automatiquement les 4 actions suivantes :

1. Attente de la base de données (via `wait-for-db.sh`)
2. Initialisation des tables (via le script SQL)
3. Exécution du pipeline ETL Python
4. Génération du rapport

Le script doit afficher la progression et s’arrêter à la première erreur.

Livrable attendu

Commit avec : `report.py`, `run_pipeline.sh`.

//

Étape 5 : Orchestration

5.1 – Point d’entrée (src/main.py)

Créez `src/main.py` qui orchestre le pipeline complet : ouverture de connexion, Extract, Transform, Load pour les deux jeux de données.

Attention

Les joueurs doivent être chargés **avant** les scores (contrainte de clé étrangère). La liste des `player_id` valides doit être extraite du DataFrame `players` nettoyé et passée à `transform_scores`.

5.2 – Documentation (README.md)

Rédigez un `README.md` contenant :

- Nom du projet et description courte
- Prérequis techniques
- Instructions de lancement
- Structure du projet
- Description des problèmes de qualité traités

5.3 – Test complet

Vérifiez que le pipeline fonctionne **de zéro** : détruisez les volumes, relancez les services, exécutez le script, et vérifiez que :

- Le rapport est généré dans `output/rapport.txt`
- La base ne contient ni doublons, ni scores négatifs, ni références orphelines

Livrable attendu

Commit final avec : `main.py`, `README.md`. Le projet complet doit être fonctionnel.

Étape 6 : Livraison*env. 30 min*

1. Vérifiez votre historique Git : vous devez avoir **au minimum 5 commits** correspondant aux étapes du projet.
2. Poussez votre projet sur GitHub.
3. Transmettez le lien du dépôt.

Attention

Le correcteur clonera votre dépôt et exécutera uniquement ces deux commandes pour évaluer votre travail :

```
docker compose up -d --build
docker compose exec app ./scripts/run_pipeline.sh
```

Assurez-vous que **tout fonctionne** avec ces seules commandes.

Récapitulatif des compétences mobilisées

Compétence	Séance d'origine	Utilisation dans le projet
Docker	Séance 1	Dockerfile de l'application
Bash	Séance 2	Scripts d'automatisation
Docker Compose	Séance 3	Orchestration MySQL + App
Python + MySQL	Séance 4	Pipeline ETL complet
Git / GitHub	Séance 5	Versionnement et livraison

Conseil**Conseils pour réussir :**

- **Commitez souvent** : un commit par étape au minimum
- **Testez au fur et à mesure** : ne passez pas à l'étape suivante sans avoir vérifié la précédente
- **Lisez les messages d'erreur** : ils vous indiquent précisément le problème
- **Consultez vos notes de TD** : les patterns sont les mêmes, seul le contexte change
- **Restez simple** : un code clair et fonctionnel vaut mieux qu'un code complexe et buggé