Livrables pour le TP - Cycle de vie de la donnée : de la source au Dashboard - 1ère Version du TP

Formateur : Mourad Elchyakhi

Membres du groupe :

- Aymeric BOISGONTIER
- Dunvael LE ROUX

Projet

Objectifs

Simuler un projet data complet, de la découverte de la donnée brute à la création d'un Dashboard décisionnel, en intégrant les bonnes pratiques de modélisation (Médaillon) et de sécurité.

Outils

- OpenMetadata (image Docker complète seulement)
- PostgreSQL
- Metabase

Scénario

Vous êtes Data Engineer/Analyst chez "VéloCity", une entreprise de location de vélos en libre-service. La direction Marketing souhaite un Dashboard pour suivre l'activité quotidienne : nombre de locations, durée moyenne des trajets, les vélos les plus utilisés, habitude par ville, âge des consommateurs, type d'abonnement pris, ... etc.

Lien GitHub docker : https://github.com/mouradelchyakhi/enseignement_epsi/tree/main/tp_docker_light

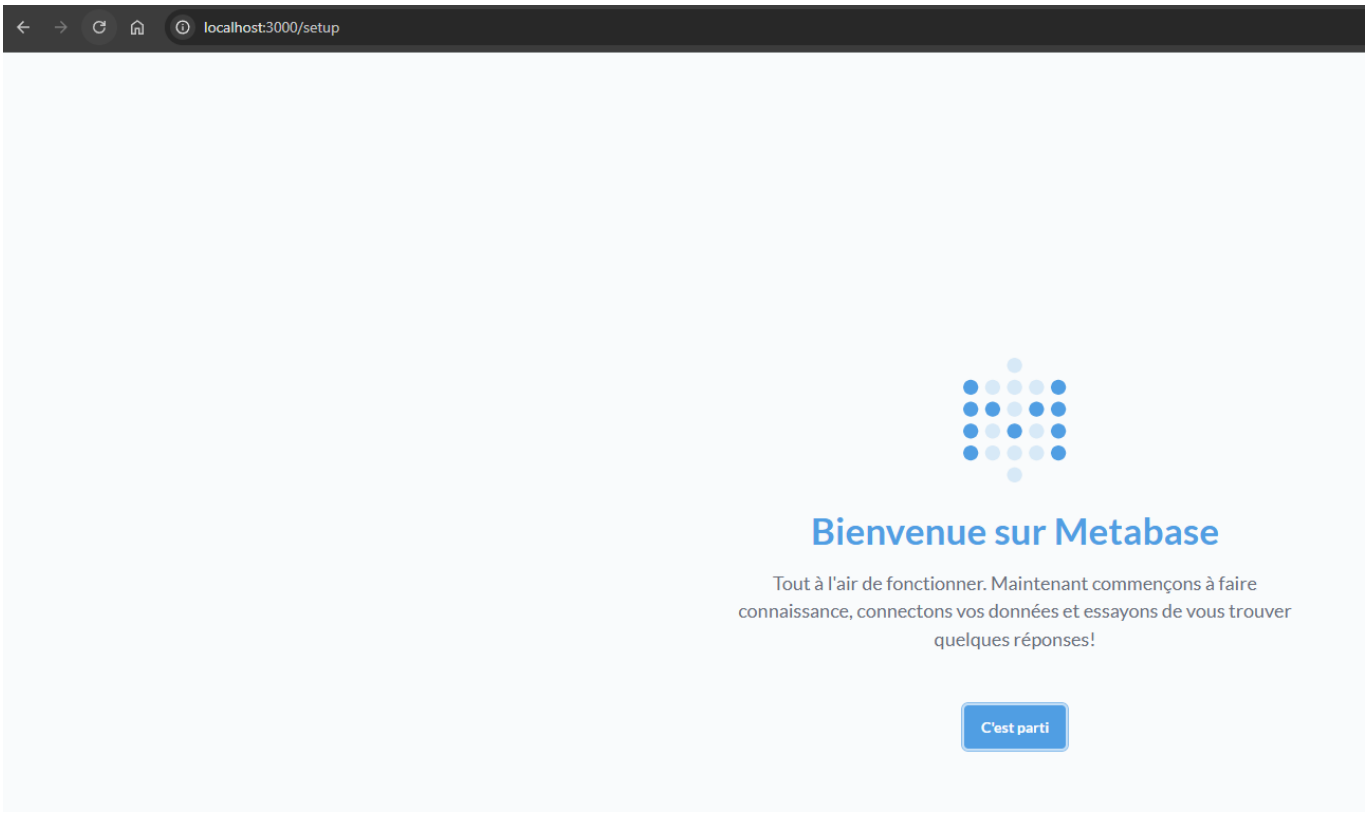
Étapes du projet

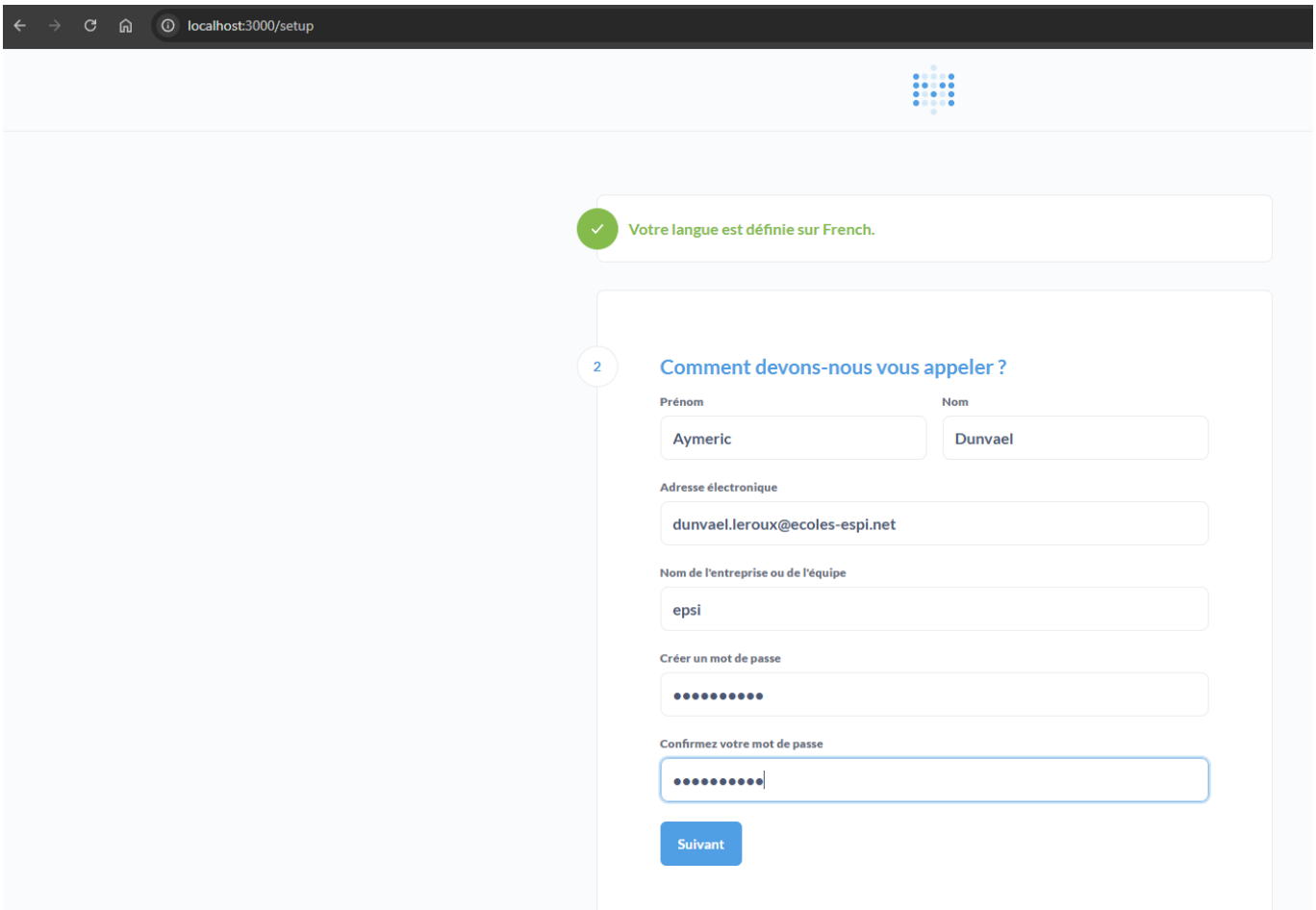
Partie 1 : Découverte et Compréhension (OpenMetadata ou fichier yaml)

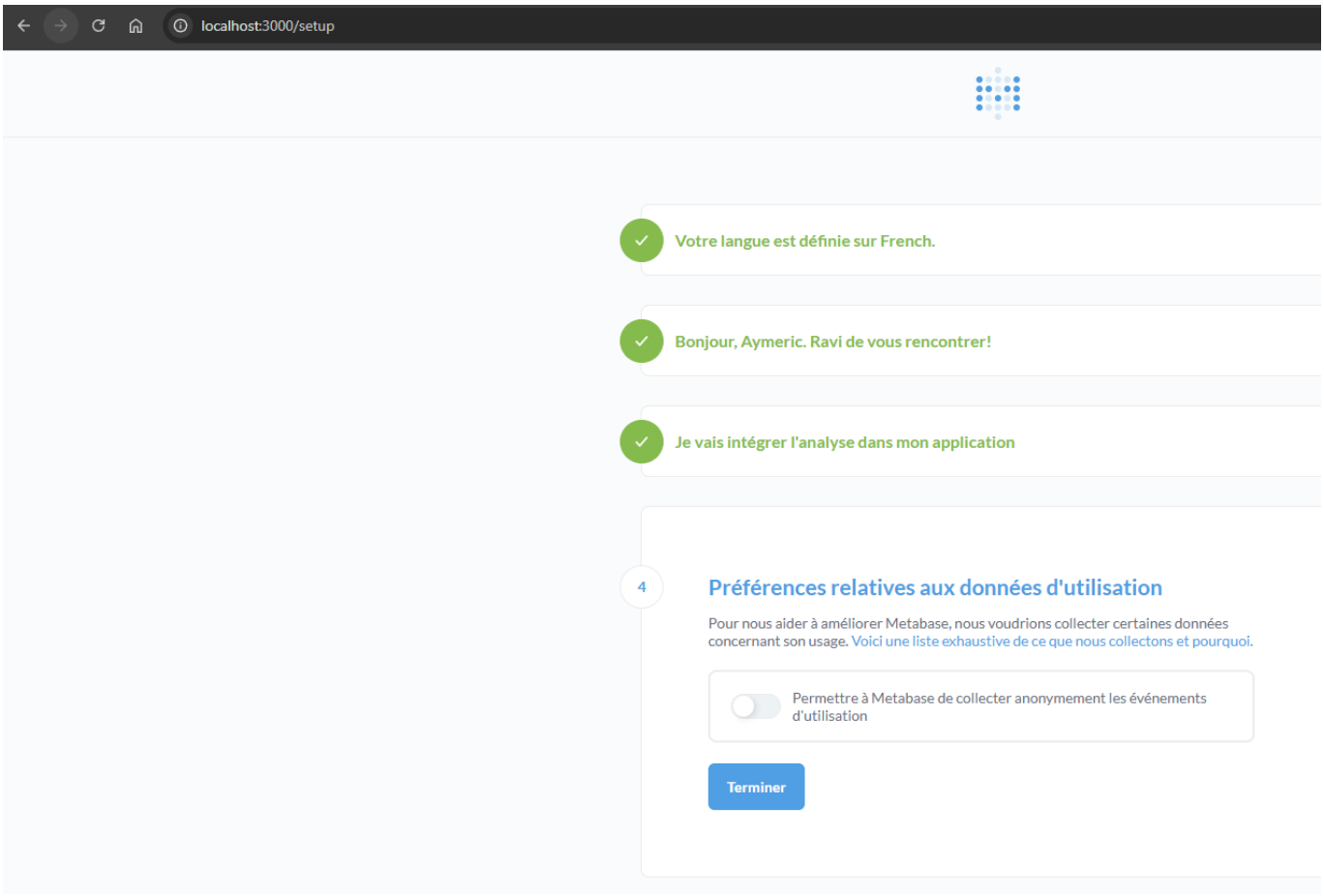
Objectif : Identifier les données sources pertinentes pour répondre aux besoins métiers.

1. Connexion à OpenMetadata

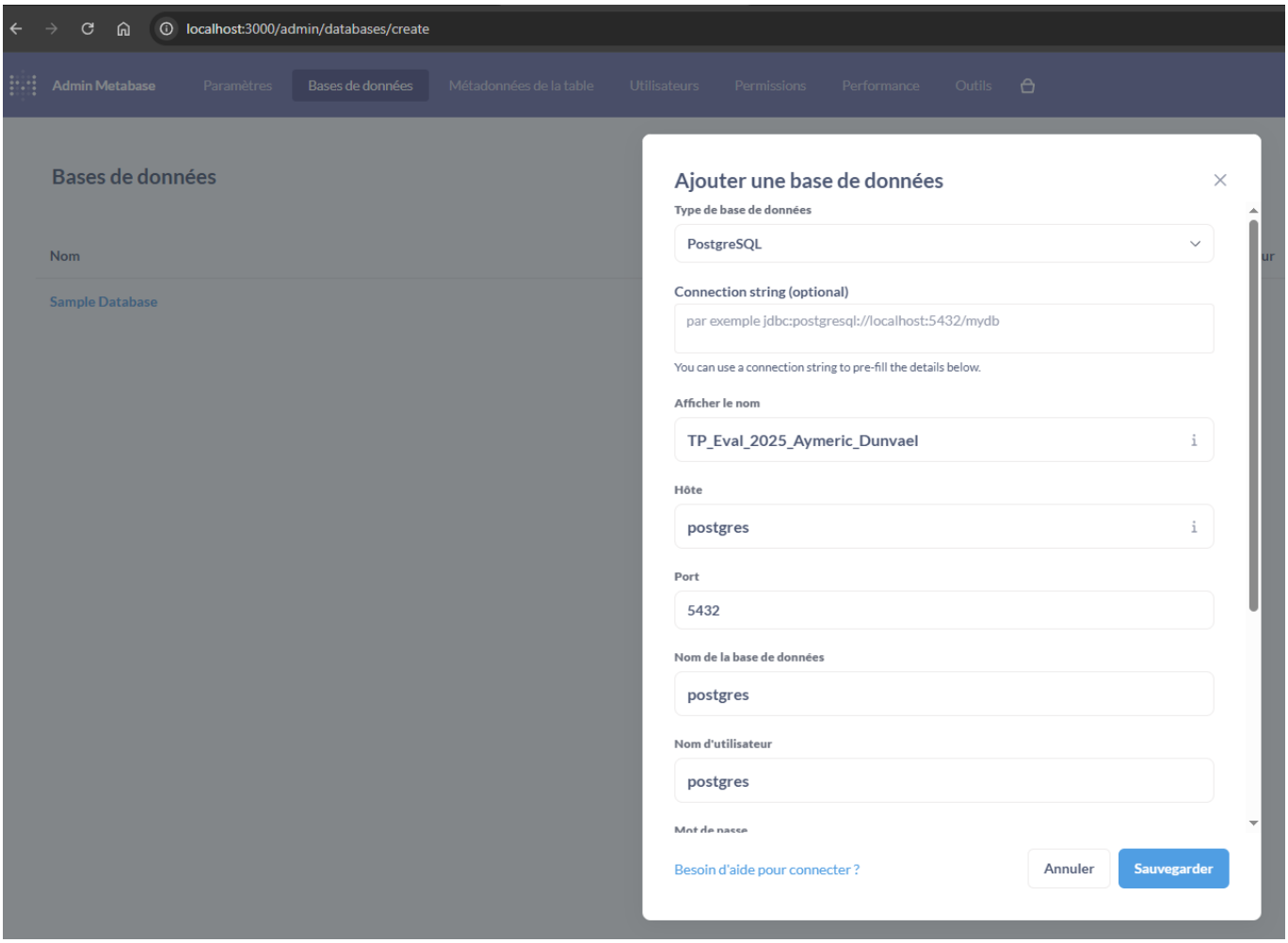
Nous nous sommes connectés à l'instance OpenMetadata de "VéloCity"

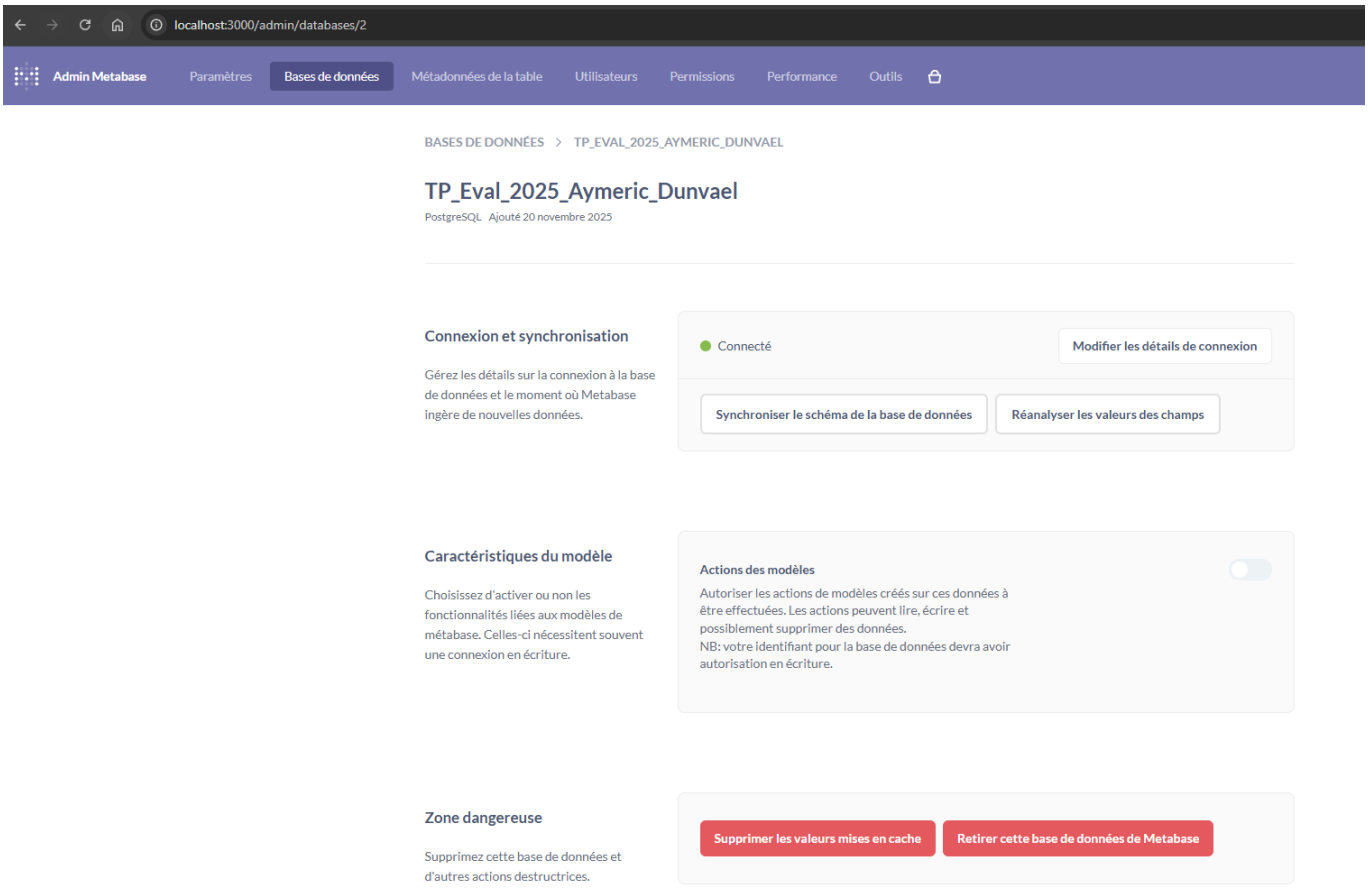

localhost:3000/setup
Bienvenue sur Metabase
Tout à l'air de fonctionner. Maintenant commençons à faire connaissance, connectons vos données et essayons de vous trouver quelques réponses!
C'est parti


localhost:3000/setup
Votre langue est définie sur French.
2
Comment devons-nous vous appeler ?
Prénom
Aymeric
Nom
Dunvael
Adresse électronique
dunvael.leroux@ecoles-espi.net
Nom de l'entreprise ou de l'équipe
epsi
Créer un mot de passe
Confirmez votre mot de passe
Suivant



Nous avons ensuite ajouté la base de données :





2. Exploration d'OpenMetadata

- Nous avons navigué dans le catalogue et identifié les tables qui semblent pertinentes pour ce TP.
- Nous avons utilisé la recherche et les "Tags" (ex: "Source", "PII") pour trouver les bonnes tables.

3. Analyse des tables sur OpenMetadata

- Nous avons analysé (quand disponible) les schémas, documentations, profils de données et propriétaires.

4. Tables et faits pertinents

Après réflexion et analyse, nous sommes partis sur ces douze tables de données qui permettent de réaliser des dashboards complets et croisés :

	table_name name
1	bike_maintenance_logs
2	bikes
3	bikes_rentals
4	bikes_station
5	cities
6	daily_activity_summary_old
7	marketing_campaigns
8	rental_archives_2022
9	subscriptions
10	user_accounts
11	user_session_logs
12	weather_forecast_hourly

Avant de justifier les choix de tables et de déterminer s'il s'agit de tables de faits ou de dimensions, nous avons défini et synthétisé les caractéristiques d'une table de fait et d'une table de dimensions dans un tableau :

Aspect	Table de faits	Table de dimensions
Contenu	Mesures, chiffres	Descriptions, attributs
Rôle	Analyse quantitative	Contexte et qualification
Type	Numérique	Textuel / catégoriel
Volume	Très élevé	Moyen / faible
Fréquence d'ajout de lignes/données	Très élevée	Peu élevée
Exemple	Montant des ventes	Produit, magasin, client

Il est possible d'avoir une table de faits et de dimensions associés.

Justification des tables et attribution Fait/Dimension :

Tables	Données sélectionnés	justification	Faits	Dimensions
Bikes rentals	Nombre de locations / Start T - End T	Permet de déterminer les vélos les plus utilisés et leur temps d'utilisation et donc les préférences et besoins des utilisateurs sur les types de vélos	☑	
Bikes Station	Station ID, Station Name, Capacity	Permet de connaître les stations de vélos ainsi que leur capacité afin de déterminer quelles stations ont le plus de "succès", possibilité de lier les données avec les types de vélos et d'environnement (régions/villes...)	☑	
Bikes	Bike ID (type) et Status	Permet de cibler les vélos utilisés ou non dans les stations, possibilité de lier les données avec les stations mais aussi les locations de vélos (notamment selon leur type)	☑	
Cities	City ID (city name), Regions	Permet de cibler les villes et régions où est présent VeloCity, possibilité de lier les données avec les stations, types de vélos, locations de vélos et par exemple de cibler les villes/régions les plus rentables et celles qui nécessitent plus de publicité		☑
Daily activity summary old	Total rentals	Permet de calculer le taux de locations de vélos quotidiennes et de réaliser des comparatifs en liant les données avec les données météorologiques ou encore les campagnes marketing ou les types de vélos disponibles dans chaque station par exemple	☑	
Marketings campaigns	Start date, End date	Permet de définir l'impact de la publicité sur le taux de location en liant les données avec le nombre de locations de vélos quotidiennes, leur début et fin de location par exemple	☑	
Rental archives 2022	Start T, End T, Bike ID	Permet d'observer l'historique des ventes d'une année à une autre et donc de déterminer l'évolution du marché dans les années à venir, possibilité de lier les données avec les campagnes marketing pour déterminer l'impact de la publicité sur les utilisateurs	☑	

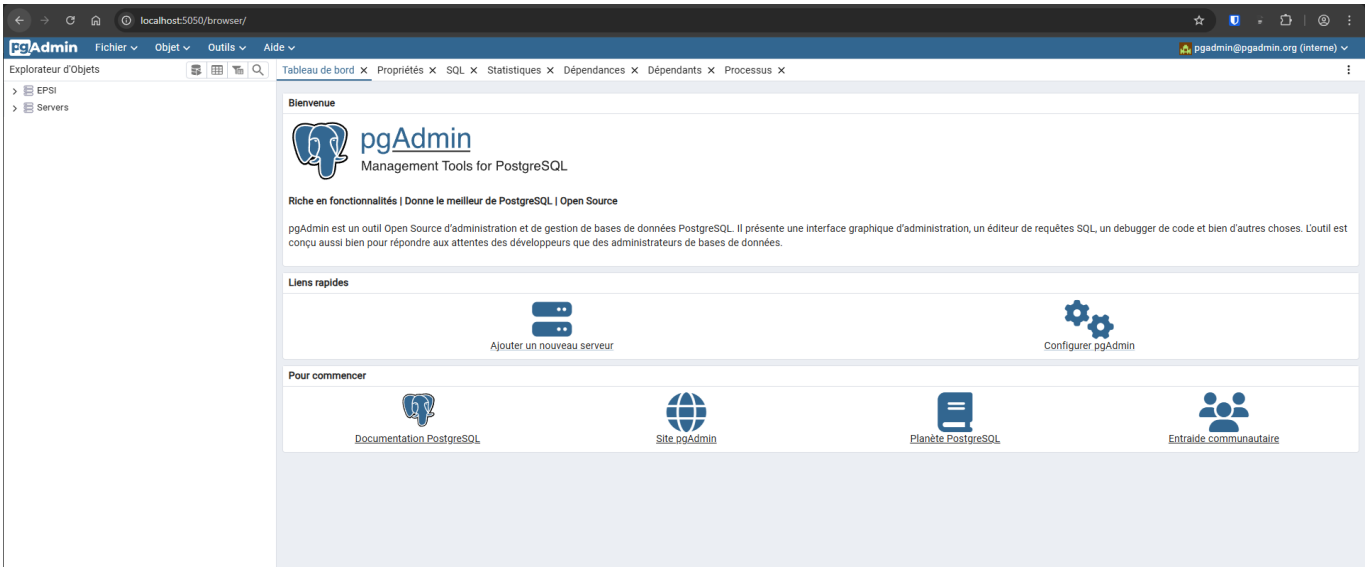
Tables	Données sélectionnés	justification	Faits	Dimensions
Subscriptions	Sub type, Sub ID	Permet de déterminer quels abonnements ont le plus de succès auprès des utilisateurs, possibilité de lier les données avec les villes/régions, avec les campagnes marketing ou encore le type d'utilisateur (différents profils utilisateurs pour différents besoins)	☑	
User accounts	Birthdate, Sub ID	Permet de connaître le type de clientèle de VéloCity et de d'établir des profils utilisateurs selon les souscriptions et âge (clientèle plus agée, jeune, retraités ou employés...), possibilité de lier les données avec le type de souscription, le type de location, ... et de définir les préférences des clients par exemple	☑	
User session logs	Device type	Permet de déterminer le type de connexion pour les locations (téléphone, web, ...) afin de comprendre les préférences d'interface des utilisateurs pour louer un vélo	☑	
Bike maintenance logs	Bike ID et Issue description	Permet de comprendre quel type de vélos est le plus âbimé, possibilité de lier les données avec les status de vélos par exemple		☑
Weather forecast hourly	Temperature Celsius, Precipitations Mm	Permet de déterminer le temps et l'impact des conditions météorologiques sur la location de vélos, possibilité de lier avec les données de locatins quotidienne, du temps de location d'un vélo, des villes et régions par exemple	☑	

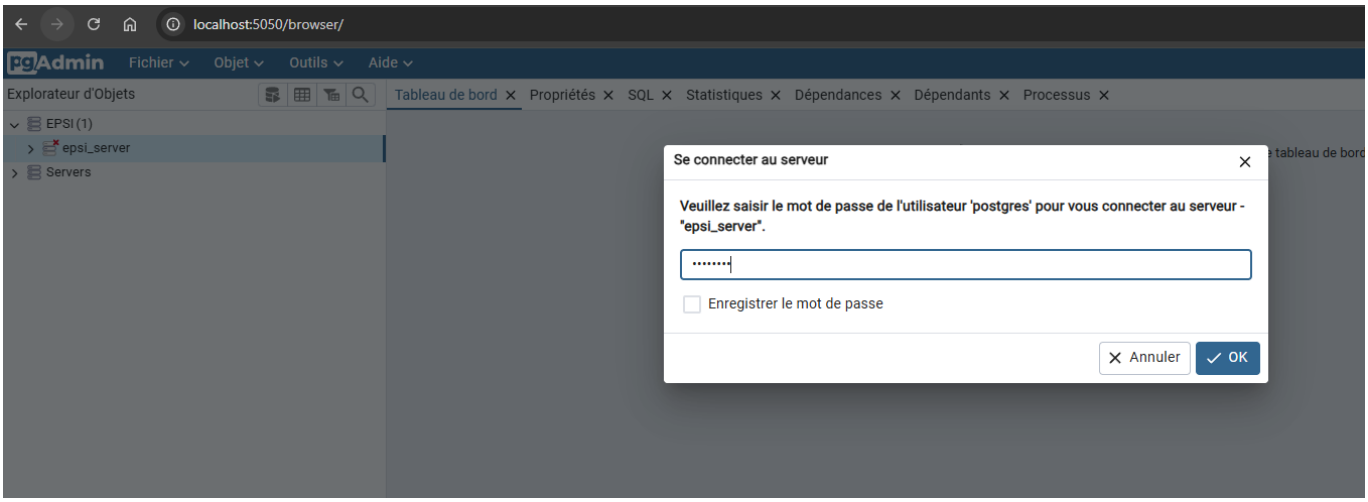
Partie 2 : Modélisation et Transformation (PostgreSQL)

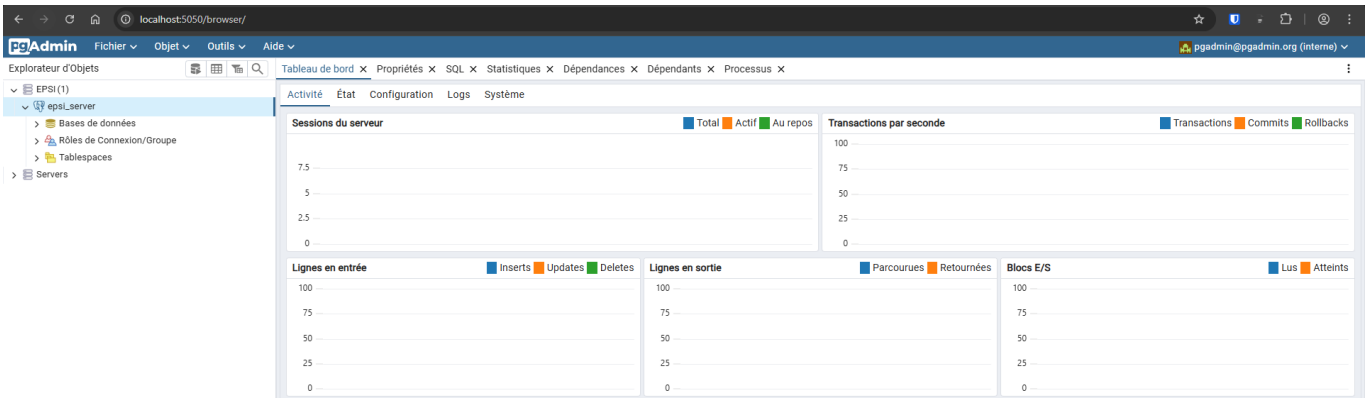
1. Connection à la base PostGreSQL

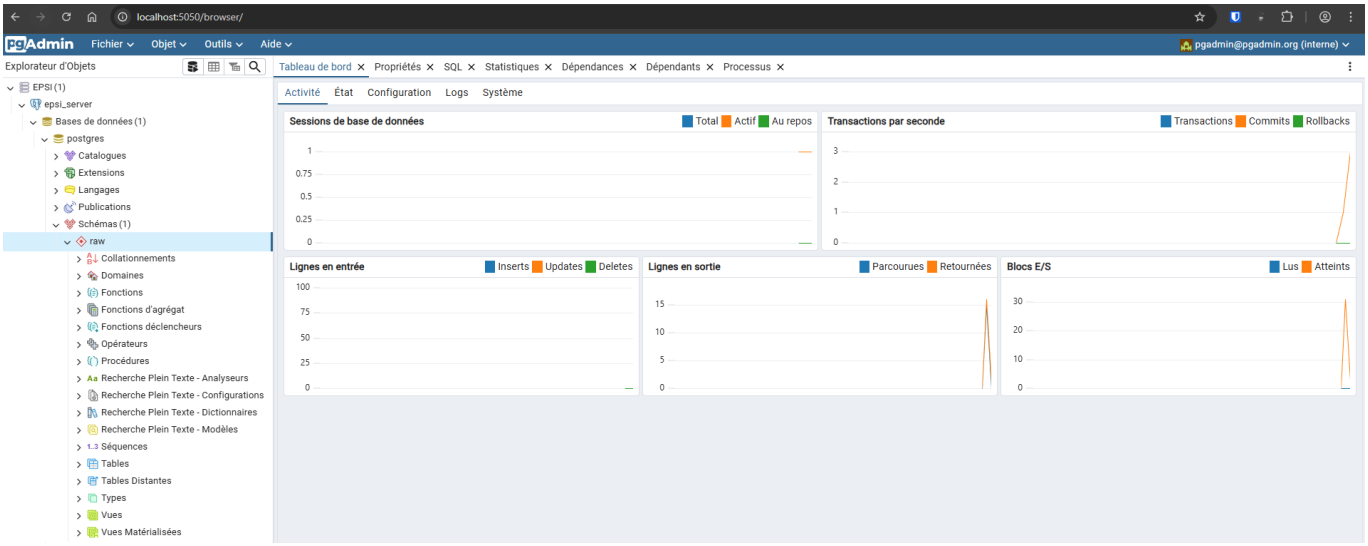
Nous nous sommes connectés à la base PostgreSQL avec pgAdmin.


localhost:5050/login?next=/
pgAdmin
Login
pgadmin@pgadmin.org
Forgotten your password?
French
Login


localhost:5050/browser/
pgAdmin
Fichier
Objet
Outils
Aide
pgadmin@pgadmin.org (interne)
Explorateur d'Objets
EPSI
Servers
Tableau de bord
Propriétés
SQL
Statistiques
Dépendances
Dépendants
Processus
Bienvenue
pgAdmin
Management Tools for PostgreSQL
Riche en fonctionnalités | Donne le meilleur de PostgreSQL | Open Source
pgAdmin est un outil Open Source d'administration et de gestion de bases de données PostgreSQL. Il présente une interface graphique d'administration, un éditeur de requêtes SQL, un debugger de code et bien d'autres choses. L'outil est conçu aussi bien pour répondre aux attentes des développeurs que des administrateurs de bases de données.
Liens rapides
Ajouter un nouveau serveur
Configurer pgAdmin
Pour commencer
Documentation PostgreSQL
Site pgAdmin
Planète PostgreSQL
Entraide communautaire

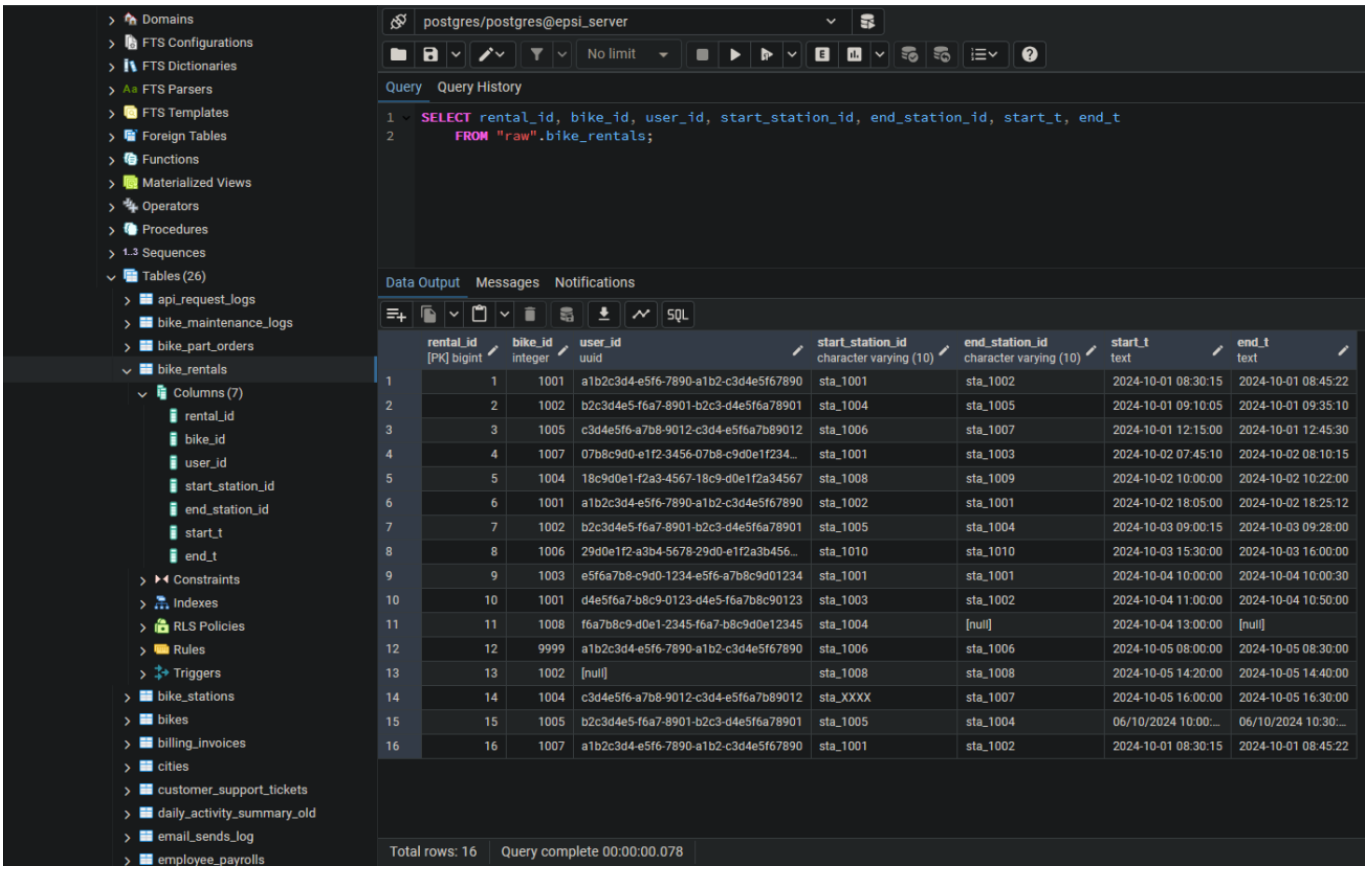

localhost:5050/browser/
pgAdmin
Fichier
Objet
Outils
Aide
Explorateur d'Objets
EPSI (1)
epsi_server
Servers
Tableau de bord
Propriétés
SQL
Statistiques
Dépendances
Dépendants
Processus
Se connecter au serveur
Veuillez saisir le mot de passe de l'utilisateur 'postgres' pour vous connecter au serveur - "epsi_server".
Enregistrer le mot de passe
Annuler
OK





Nous trouvons les mêmes informations, la différence réside dans l'interface utilisateur (visuel et présentation).

Exemple :



tp_data / Bike Rentals

Rental ID	Bike ID	User ID	Start Station ID	End Station ID	Start T	End T
1	1,001	a1b2c3d4-e5f6-7890-a1b2-c3d4e5f67890	sta_1001	sta_1002	2024-10-01 08:30:15	2024-10-01 08:45:22
2	1,002	b2c3d4e5-f6a7-8901-b2c3-d4e5f6a78901	sta_1004	sta_1005	2024-10-01 09:10:05	2024-10-01 09:35:10
3	1,005	c3d4e5f6-a7b8-9012-c3d4-e5f6a7b89012	sta_1006	sta_1007	2024-10-01 12:15:00	2024-10-01 12:45:30
4	1,007	07b8c9d0-e1f2-3456-07b8-c9d0e1f23456	sta_1001	sta_1003	2024-10-02 07:45:10	2024-10-02 08:10:15
5	1,004	18c9d0e1-f2a3-4567-18c9-d0e1f2a34567	sta_1008	sta_1009	2024-10-02 10:00:00	2024-10-02 10:22:00
6	1,001	a1b2c3d4-e5f6-7890-a1b2-c3d4e5f67890	sta_1002	sta_1001	2024-10-02 18:05:00	2024-10-02 18:25:12
7	1,002	b2c3d4e5-f6a7-8901-b2c3-d4e5f6a78901	sta_1005	sta_1004	2024-10-03 09:00:15	2024-10-03 09:28:00
8	1,006	29d0e1f2-a3b4-5678-29d0-e1f2a3b45678	sta_1010	sta_1010	2024-10-03 15:30:00	2024-10-03 16:00:00
9	1,003	e5f6a7b8-c9d0-1234-e5f6-a7b8c9d01234	sta_1001	sta_1001	2024-10-04 10:00:00	2024-10-04 10:00:30
10	1,001	d4e5f6a7-b8c9-0123-d4e5-f6a7b8c90123	sta_1003	sta_1002	2024-10-04 11:00:00	2024-10-04 10:50:00
11	1,008	f6a7b8c9-d0e1-2345-f6a7-b8c9d0e12345	sta_1004		2024-10-04 13:00:00	
12	9,999	a1b2c3d4-e5f6-7890-a1b2-c3d4e5f67890	sta_1006	sta_1006	2024-10-05 08:00:00	2024-10-05 08:30:00
13	1,002		sta_1008	sta_1008	2024-10-05 14:20:00	2024-10-05 14:40:00
14	1,004	c3d4e5f6-a7b8-9012-c3d4-e5f6a7b89012	sta_XXXX	sta_1007	2024-10-05 16:00:00	2024-10-05 16:30:00
15	1,005	b2c3d4e5-f6a7-8901-b2c3-d4e5f6a78901	sta_1005	sta_1004	06/10/2024 10:00:00	06/10/2024 10:30:00
16	1,007	a1b2c3d4-e5f6-7890-a1b2-c3d4e5f67890	sta_1001	sta_1002	2024-10-01 08:30:15	2024-10-01 08:45:22

Nous avons relevés des anomalies potentielles (doublons, manque de données ou "null" ou "inconnu", problème de nommage de colonne, IDs non standard, type hétérogène, dates au format texte/timestamp mixte...).

Exemple :

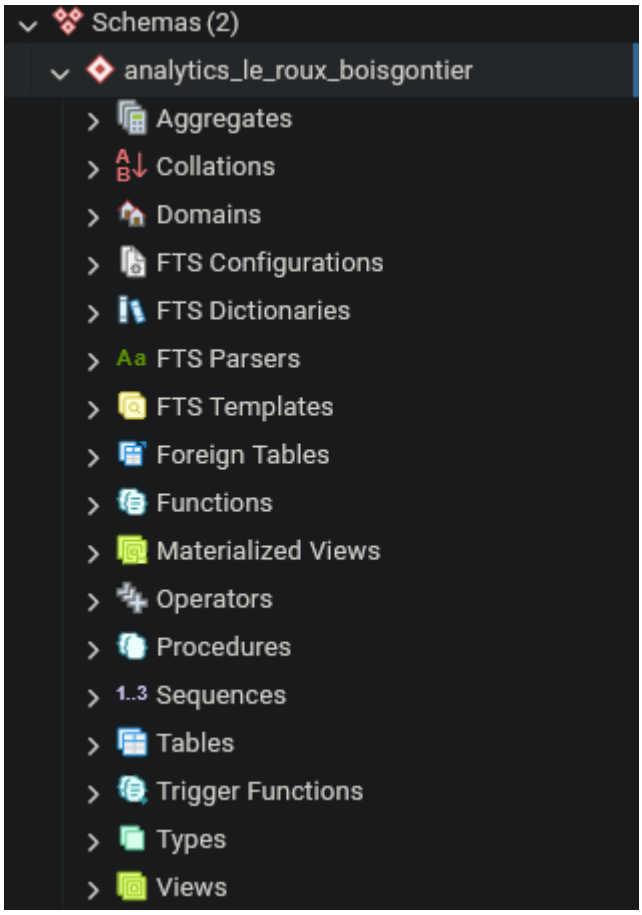
	station_id [PK] character varying (10)	station_name character varying (255)	latitude text	longitude text	capacity integer	city_id integer
1	sta_1001	Hôtel de Ville Paris	48.8566	2.3522	30	1
2	sta_1002	Louvre-Rivoli	48.8606	2.3400	25	1
3	sta_1003	Gare de Lyon	48.8444	2.3744	40	1
4	sta_1004	Place Bellecour	45.7578	4.8320	35	2
5	sta_1005	Vieux Lyon	45.7618	4.8256	20	2
6	sta_1006	Vieux-Port	43.2965	5.3700	50	3
7	sta_1007	Gare St-Charles	43.3030	5.3807	30	3
8	sta_1008	Gare Lille Flandres	50.6366	3.0694	45	4
9	sta_1009	[null]	50.6292	3.0573	25	4
10	sta_1010	Capitole	43.6045	1.4442	30	5
11	sta_1011	Place Bellecour	45.7578	4.8320	35	2
12	sta_1012	Notre Dame	48.8529	2.3500	20	1
13	sta_1013	Station Invalide	coord_lat	coord_lon	15	1
14	sta_1014	Station Orpheline	48.8500	2.3500	10	99

Puis nous avons réalisé un tableau reccueillant les différentes anomalies pour les tables que nous avons sélectionnées :

Table	Anomalies Principale	Correction à appliquer
bike_maintenance_logs	Dates au format texte/timestamp mixte	Cast en DATE standard
bikes	"Types hétérogènes (""E-bike"" vs ""Electrique"")"	Standardisation via CASE WHEN
bikes_rentals	Trajets < 2 min et IDs non standards	Filtre durée & Renommage colonnes
bikes_station	"Lat/Lon en texte, Ville 99, Doublons"	Nettoyage Regex & suppression orphelins
cities	Régions vides	"Remplacement par ""Region Inconnue"""
daily_activity	Colonne date mal nommée	Renommage
marketing_campaigns	Budgets vides	Remplacement par 0
rental_archives	Trajets < 2 min et IDs non standards	Filtre durée & Renommage colonnes
subscriptions	Types d'abo vides	"Valeur par défaut ""Standard"""
user_accounts	Dates FR/EN mélangées	Parsing intelligent avec Regex
user_session_logs	Appareils inconnus	"Remplacement par ""unknown"""
weather_forecast_hourly	Précipitations NULL	Remplacement par 0

Nous avons ensuite créé un nouveau schéma pour les transformations avec la commande SQL suivante :

```
CREATE SCHEMA IF NOT EXISTS analytics_le_roux_boisgontier;
```



2. Couche Silver (raffinage)

Afin de nettoyer, typer et standardiser les données brutes pour qu'elles soient exploitables, nous avons créé une table nettoyée avec conversions de types, corrections de valeurs manquantes ou aberrantes, et ajouts de calculs métiers (durée, statuts, etc.).

Cf. le fichier script `SQL_unique.sql` qui contient la création des schémas, tables, transformations et commandes GRANT et REVOKE pour les accès.

Nous avons requêté en amont PostgreSQL afin de lister les tables présentes pour notre script SQL :

```
SELECT table_schema, table_name
FROM information_schema.tables
WHERE table_schema NOT IN ('information_schema', 'pg_catalog')
ORDER BY table_schema, table_name;
```

3. Couche Gold (Agrégation métier)

Afin de créer une table prête à l'emploi pour le Dashboard et répondant au besoin métier, nous avons créé une table avec les métriques clefs agrégées au bon niveau (jour, ville, type vélo etc.) : totalrentals, averagedurationminutes, uniqueusers, etc :

```
CREATE TABLE analytics_le_roux_boisgontier.gold_daily_activity AS
SELECT
    -- 1. Dimensions (Axes d'analyse)
```

```
    DATE(r.start_time) AS rental_date,       -- Granularité : Jour
    c.city_name,                             -- Granularité : Ville
    s.station_name,                          -- Granularité : Station
    b.bike_type,                             -- Granularité : Type de vélo
    sub.sub_type AS subscription_type,       -- Granularité : Abonnement

    -- 2. Métriques (KPIs)
    COUNT(r.id) AS total_rentals,                       -- Nombre total de
locations [cite: 44]
    ROUND(AVG(r.duration_minutes)::numeric, 2) AS average_duration_minutes, --
Durée moyenne [cite: 45]
    COUNT(DISTINCT r.user_id) AS unique_users           -- Utilisateurs uniques
[cite: 45]

FROM
    -- Table de faits (Silver)
    analytics_le_roux_boisgontier.bikes_rentals r

    -- Jointures vers les dimensions (Silver)
    JOIN analytics_le_roux_boisgontier.bikes b ON r.bike_id = b.bike_id
    JOIN analytics_le_roux_boisgontier.bikes_station s ON r.station_start_id =
s.station_id
    JOIN analytics_le_roux_boisgontier.cities c ON s.city_id = c.city_id
    JOIN analytics_le_roux_boisgontier.user_accounts u ON r.user_id = u.user_id
    JOIN analytics_le_roux_boisgontier.subscriptions sub ON u.sub_id = sub.sub_id

GROUP BY
    DATE(r.start_time),
    c.city_name,
    s.station_name,
    b.bike_type,
    sub.sub_type;

-- Vérification rapide du résultat
SELECT * FROM analytics_le_roux_boisgontier.gold_daily_activity LIMIT 10;
```

Résultat:

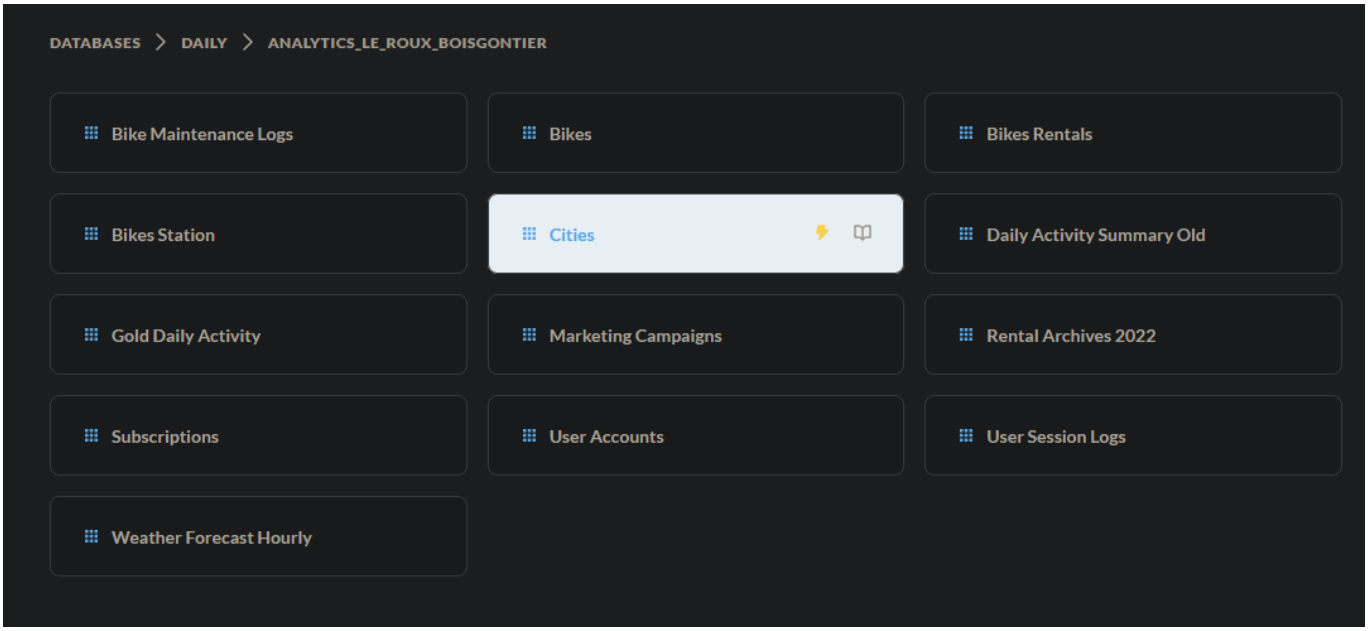
	rental_date date	city_name character varying (100)	station_name character varying (255)	bike_type text	subscription_type character varying	total_rentals bigint	average_duration_minutes numeric	unique_users bigint
1	2024-06-10	Lyon	Vieux Lyon	Electrique	Annuel	1	30.00	1
2	2024-10-01	Marseille	Vieux-Port	Electrique	Étudiant	1	30.50	1
3	2024-10-01	Paris	Hôtel de Ville Paris	Mecanique	Mensuel	2	15.12	1
4	2024-10-02	Lille	Gare Lille Flandres	Mecanique	Étudiant	1	22.00	1
5	2024-10-02	Paris	Hôtel de Ville Paris	Mecanique	Annuel	1	25.08	1
6	2024-10-02	Paris	Louvre-Rivoli	Mecanique	Mensuel	1	20.20	1
7	2024-10-03	Lyon	Vieux Lyon	Electrique	Annuel	1	27.75	1
8	2024-10-03	Toulouse	Capitole	Electrique	Mensuel	1	30.00	1

Partie 3 : Visualisation (Metabase)

1. Connexion à Metabase & 2. Source de données

Afin de créer un Dashboard simple pour le métier Marketing :

- Nous nous sommes connectés à Metabase.
- Nous avons ajouté la base PostgreSQL comme source.
- Nous avons ajouté la table *analytics_le_roux_boisgontier.gold_daily_activity* comme dataset dans Metadata.

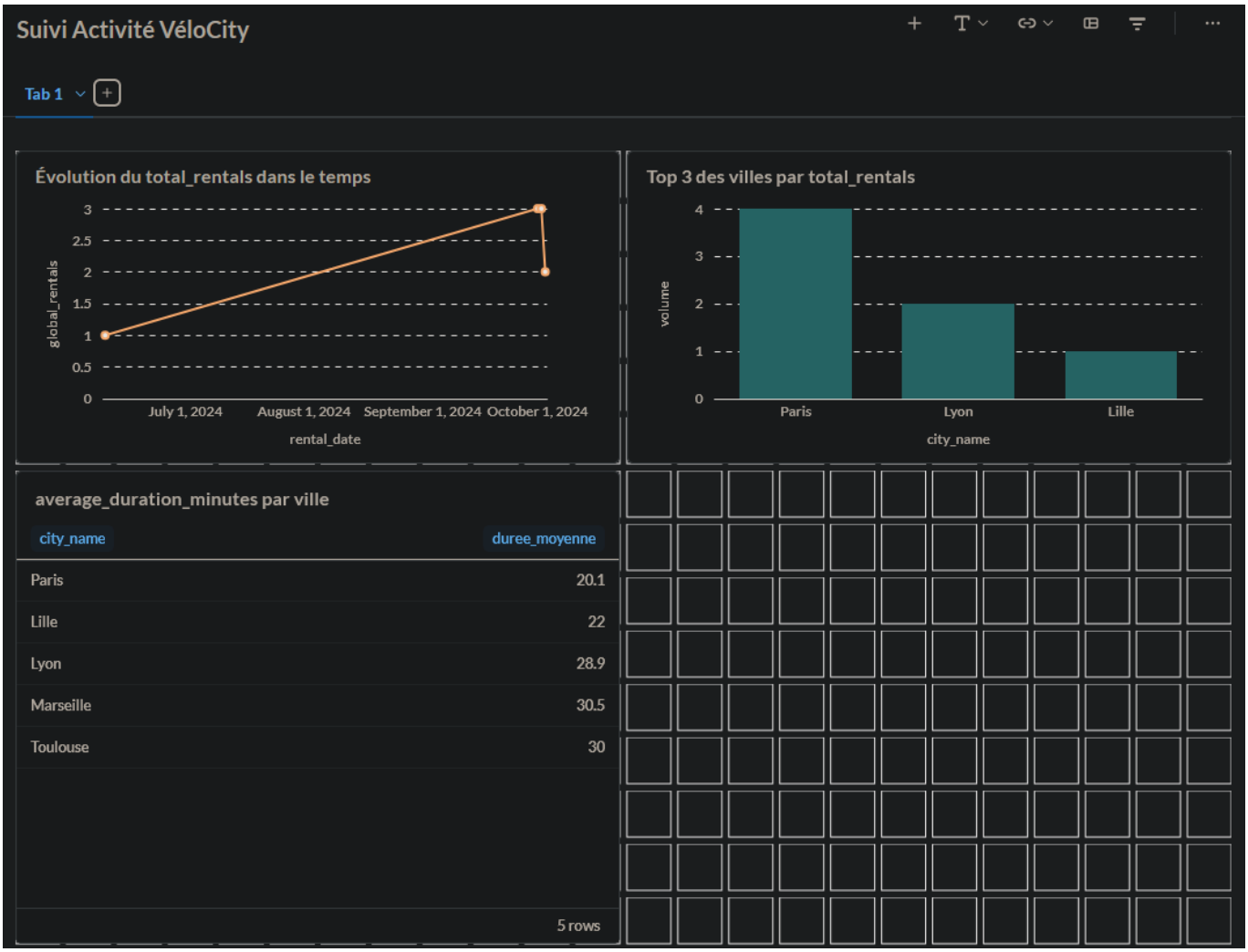


3. Création des charts & 4. Dashboard

Nous avons créé les trois charts suivants :

- Chart 1 (Courbe) : Évolution du total_rentals dans le temps (axe X = jour).
- Chart 2 (Bar) : Top 3 des villes par total_rentals.
- Chart 3 (Indicateur/KPI) : average_duration_minutes par ville.

Et avons construit un dashboard "Suivi Activité VéloCity" à partir de ces charts :



Partie 4 : Sécurité et Gouvernance (PostgreSQL +)

1. Scénario & 2. Audit (Simulation)

Si marketing_user essaie de faire `SELECT * FROM raw.user_accounts;`, il a le message *permission denied* et ne peut pas accéder aux données.

```sql
SET ROLE marketing_user;
-- Ce test DOIT afficher une erreur "Permission denied"
SELECT * FROM raw.user_accounts LIMIT 10;
```

Data Output Messages Notifications

```
ERROR:  permission denied for schema raw
LINE 3: SELECT * FROM raw.user_accounts LIMIT 10;
                      ^

SQL state: 42501
Character: 96
```

Si marketing_user fait `SELECT * FROM analytics_nom1_nom2.gold_daily_activity;`, il a accès aux données de la table *analytics_nom1_nom2.gold_daily_activity* uniquement.

```sql
SET ROLE marketing_user;
SELECT * FROM analytics_le_roux_boisgontier.gold_daily_activity
```

Data Output Messages Notifications

	rental_date date	city_name character varying (100)	station_name character varying (255)	bike_type text	subscription_type character varying	total_rentals bigint	average_duration_minutes numeric	unique_users bigint
1	2024-06-10	Lyon	Vieux Lyon	Electrique	Annuel	1	30.00	1
2	2024-10-01	Marseille	Vieux-Port	Electrique	Étudiant	1	30.50	1
3	2024-10-01	Paris	Hôtel de Ville Paris	Mecanique	Mensuel	2	15.12	1
4	2024-10-02	Lille	Gare Lille Flandres	Mecanique	Étudiant	1	22.00	1
5	2024-10-02	Paris	Hôtel de Ville Paris	Mecanique	Annuel	1	25.08	1
6	2024-10-02	Paris	Louvre-Rivoli	Mecanique	Mensuel	1	20.20	1
7	2024-10-03	Lyon	Vieux Lyon	Electrique	Annuel	1	27.75	1
8	2024-10-03	Toulouse	Capitole	Electrique	Mensuel	1	30.00	1

3. Tâche (Script SQL)

Nous avons écrit un script SQL pour :

- Implémenter cette règle de sécurité
- Créer un rôle manager_lyon

- Ne lui donner accès qu'à la table GOLD, pour la ville 'Lyon'

Cf. le fichier script `SQL_unique.sql` qui contient la création des schémas, tables, transformations et commandes GRANT et REVOKE pour les accès.

Puis nous avons testé le rôle pour nous assurer que celui-ci fonctionne bien comme convenu.