UFR Sciences et Techniques de Dijon

Master BDIA — 2ème année

Informatique Décisionnelle

Auteurs :

RIAK Ilyas
KHOMSI ELHASSOUNI Abdelmajid
HAJAM Ayoub

Encadrant :

E. LECLERCQ
G. ANNABELLE

Année universitaire 2024 - 2025

Table des matières

1 Introduction et Analyse du Sujet

L'objectif de ce projet est de construire un **data warehouse (DW)** à partir des données du *Yelp Academic Dataset*, afin d'illustrer l'ensemble du processus de l'informatique décisionnelle. Ce processus comprend :

- La **modélisation** et la **conception** d'un schéma décisionnel (dimensionnel).
- L'**extraction** des données depuis plusieurs sources (PostgreSQL, JSON).
- La **transformation** et le **chargement** des données dans un SGBD cible (Oracle).
- L'**analyse** et la **présentation** des résultats via des requêtes et des tableaux de bord.
- L'**optimisation** des performances et l'amélioration de la structure si nécessaire.

1.1 Contexte et jeu de données

Le jeu de données *Yelp Academic Dataset* regroupe des avis d'utilisateurs sur un large éventail de commerces. Il est relativement volumineux et réparti sur plusieurs supports:

- **Base PostgreSQL**: contenant les tables `yelp.user`, `yelp.friend`, `yelp.elite` et `yelp.review`.
- **Fichiers JSON**: `business.json` et `checkin.json`, décrivant les commerces et les dates de visites (checkins).
- **Fichier CSV** `tip` (non utilisé directement dans ce projet, mais disponible).

Grâce à ces sources, il est possible d'obtenir à la fois des informations sur :

- Les commerces (**business**): localisation, catégories, horaires, attributs (parking, menus végétariens, etc.).
- Les utilisateurs (**user**): leur profil, leurs amis, les années durant lesquelles ils sont «élites», etc.
- Les avis (**review**): note (stars), texte libre, réactivité (cool/funny/useful) et date de publication.
- Les checkins (**checkin**): enregistrements de visites dans les commerces.

1.2 Analyse des besoins et choix des données

Étant donné l'ampleur du jeu de données initial, il est souvent nécessaire de **sélectionner** et de **charger uniquement les données pertinentes** pour l'analyse à réaliser. Dans notre cas, nous avons orienté les analyses autour des questions suivantes:

- Quels sont les utilisateurs les plus actifs?
- Quels sont les types de commerces les mieux ou les moins bien notés?
- Évolution temporelle des notes?
- Quelles villes bénéficient de commerces très bien notés et lesquelles souffrent d'un manque de certains types de services (épiceries, agences de location...)?

- Quels commerces reçoivent le plus de visites?
- Peut-on observer des tendances ou des regroupements d'utilisateurs en fonction de leurs préférences (catégories de commerces)?

Ces questions nous ont conduits à **ne charger qu'une partie des données** du jeu de données Yelp: - Nous avons volontairement omis les informations jugées peu cruciales pour notre analyse (ex. attributs très spécifiques comme «HairSpeciality», «AcceptsInsurance», etc.). - Nous avons conservé les *stars* (notes), les dates, les catégories (restaurants, car rentals, grocery, etc.), les identifiants des utilisateurs et commerces, ainsi que les données permettant les jointures (clés).

Cette sélection raisonnée s'inscrit dans un souci de **simplicité** et de **pertinence** en regard des analyses visées, tout en réduisant le temps d'exécution et le volume des traitements ETL.

1.3 Justification du choix du Data Warehouse

Pour ce projet, nous avons opté pour une **approche Kimball** et la construction d'un **modèle en étoile**.

- **Approche Kimball**: elle consiste à construire l'entrepôt de données comme un ensemble de *data marts* dimensionnels, centrés sur des processus métiers bien définis (avis, checkins, etc.). C'est une démarche bottom-up, favorisant la simplicité de conception et l'efficacité des analyses OLAP courantes (agrégations, drill-down, etc.).
- **Modèle en étoile**: il s'agit d'une organisation où l'on retrouve une table de faits centrale (contenant les mesures quantitatives) et des tables de dimensions (contenant les descripteurs, par exemple les commerces, les utilisateurs, les dates). Ce modèle est bien adapté aux requêtes analytiques car il minimise le nombre de jointures.

À titre de comparaison, **l'approche Inmon** prône la création d'un entrepôt de données d'entreprise (Enterprise Data Warehouse) très *normalisé*, puis la création de datamarts dérivés. Dans un cadre académique et pour un jeu de données comme Yelp, l'approche Kimball (avec un focus sur quelques processus clés) se prête mieux à des **analyses rapides** et orientées **décisionnel**.

2 Spécification du Data Warehouse

2.1 Modélisation adoptée : vue d'ensemble

Notre schéma global suit une forme **en étoile** (ou star schema). Nous avons identifié plusieurs tables de faits et leurs dimensions associées :

- **FACT_REVIEW**: regroupe les notes (stars), les réactions (cool/funny/useful), la date de l'avis et l'identifiant de l'utilisateur et du commerce.
- **FACT_CHECKIN**: regroupe les visites enregistrées par les utilisateurs, de manière agrégée ou quotidienne selon le besoin.

Les tables de dimensions majeures sont:

- **DIM_USER**: profil utilisateur (ID, nom, date de création du compte, etc.).
- **DIM_BUSINESS**: infos sur les commerces (nom, localisation, catégories, etc.).
- **DIM_DATE**: dimension temporelle (jour, mois, année, etc.).

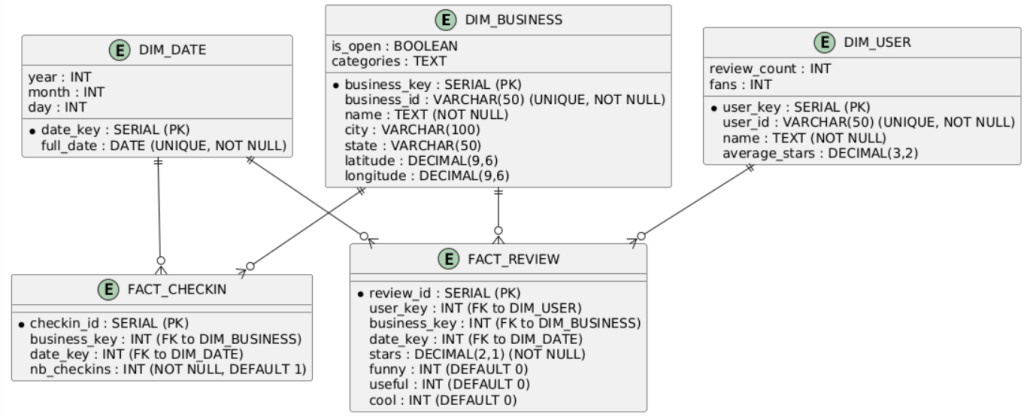

Figure 1: Schéma du Data Warehouse (modèle en étoile)

2.2 Tables de dimensions : détail

2.2.1 Dimension DIM_BUSINESS

Cette dimension répertorie toutes les informations descriptives d'un commerce :

- **BUSINESS_ID** (clé naturelle, venant de Yelp).
- **BUSINESS_KEY** (clé primaire substitut, générée pour le DW).
- **NAME** (nom du commerce).
- **CITY**, **STATE**, **LATITUDE**, **LONGITUDE**.
- **CATEGORIES** (un champ texte ou tableau transformé en chaîne).
- **REVIEW_COUNT** (nombre total d'avis reçus).
- **STARS** (note moyenne du commerce).

Les attributs plus spécialisés (parking, menus végétariens, etc.) ont été pour la plupart ignorés dans ce projet, afin de réduire la complexité et se concentrer sur nos questions principales (notes, localisation, catégories, etc.).

2.2.2 Dimension DIM_USER

Cette table contient les données décrivant l'utilisateur :

- **USER_ID** (identifiant naturel).
- **USER_KEY** (clé primaire substitut).
- **NAME** (nom ou prénom de l'utilisateur).
- **YELPING_SINCE** (date de création du compte).
- **FRIENDS_COUNT** (nombre d'amis).
- **REVIEW_COUNT** (nombre total d'avis rédigés).
- **ELITE_YEARS** (années où l'utilisateur est *elite*).

D'autres colonnes (par ex. fans, average_stars, etc.) peuvent être intégrées, selon les besoins d'analyse sur le profil de l'utilisateur.

2.2.3 Dimension DIM_DATE

Elle permet l'analyse temporelle (hiérarchies par année, trimestre, mois, etc.). De manière classique, on y retrouve :

- **DATE_KEY** (un entier au format YYYYMMDD, par exemple).
- **FULL_DATE** (type `DATE`).
- **YEAR**, **MONTH**, **DAY**, **QUARTER**.

Cette table est généralement populée via un script qui itère sur la plage de dates concernées. Dans notre cas, la période s'étale de 2004 à 2019 (selon la disponibilité des données Yelp).

2.3 Tables de faits : détail

2.3.1 FACT_REVIEW

C'est la table centrale pour toutes les analyses portant sur les évaluations (notes) :

- **REVIEW_ID** (identifiant naturel).
- **REVIEW_DATE** (ou **DATE_KEY** en FK vers DIM_DATE).
- **USER_KEY** (FK vers DIM_USER).
- **BUSINESS_KEY** (FK vers DIM_BUSINESS).
- **STARS** (la note attribuée, entre 1 et 5).
- **COOL**, **FUNNY**, **USEFUL** (réactions).

2.3.2 FACT_CHECKIN

Elle enregistre les visites déclarées dans Yelp :

- **CHECKIN_ID** (identifiant logique, éventuellement).
- **CHECKIN_DATE** (ou **DATE_KEY**).
- **BUSINESS_KEY** (FK vers DIM_BUSINESS).
- **NB_VISITS** (nombre de visites ce jour-là, si agrégé).

Les checkins ne contiennent pas toujours l'ID de l'utilisateur dans le fichier JSON, c'est pourquoi nous n'avons pas forcément établi de lien direct vers DIM_USER. Néanmoins, il serait possible de le faire si l'on disposait de la bonne source de jointure (par exemple, un champ `user_id` dans le checkin, ce qui n'est pas toujours le cas).

3 Déroulement du Projet

3.1 Extraction et Transformation des données (ETL)

Pour réaliser l'ETL, nous avons décidé d'utiliser **Spark en Scala**, car il permet d'orchestrer des lectures multi-sources (PostgreSQL, JSON, CSV) et de transformer les données en exploitant la parallélisation.

Exemple de lecture depuis PostgreSQL (table `yelp.review`):

```
val pgUrl = "jdbc:postgresql://stendhal.iem:5432/tpid2020"
val pgProps = new java.util.Properties()
pgProps.setProperty("user", "tpid")
pgProps.setProperty("password", "tpid")

val dfReviewRaw = spark.read.jdbc(pgUrl, "yelp.review", pgProps)
```

Exemple de lecture d'un fichier JSON (les commerces) :

```
val dfBusinessRaw = spark.read.json(
  "/home4/ir618188/Desktop/info_decis/yelp_academic_dataset_business.json"
)
```

Nous avons ensuite opéré **des filtres**, des **jointures** et des **sélections** de colonnes nécessaires à nos analyses. Par exemple, pour la dimension `DIM_BUSINESS` :

```
val dfBusiness = dfBusinessRaw.select(
  col("business_id"),
  col("name"),
  col("city"),
  col("state"),
  col("stars"),
  col("categories"),
  col("review_count"),
  col("latitude"),
  col("longitude")
)
```

3.2 Exécution de l'ETL avec Spark

Pour exécuter notre ETL, nous utilisons **Scala** et **Apache Spark**. Afin d'assurer un bon fonctionnement sur un dataset volumineux, nous avons alloué **16 Go de mémoire** à la JVM lors de l'exécution de SBT.

La commande utilisée est la suivante :

```
sbt -J-Xmx16G clean run
```

Cette commande :

- **Nettoie** le projet avant l'exécution (`clean`).
- **Alloue 16 Go de RAM** à l'ETL pour éviter les erreurs de mémoire (`-J-Xmx16G`).
- **Lance** le programme Spark (`run`).

L'optimisation mémoire est essentielle, car Spark effectue de nombreuses opérations en mémoire pour transformer et charger les données dans Oracle.

3.3 Chargement dans Oracle

Afin de charger les tables dimensionnelles et factuelles dans Oracle, nous avons utilisé le connecteur JDBC de Spark, en prenant soin de redéfinir le *mapping* des types pour éviter les incompatibilités (types `Boolean`, taille de `String`, etc.). Le code suivant illustre la configuration d'une classe `OracleDialect` :

```
class OracleDialect extends JdbcDialect {
  override def getJDBCType(dt: DataType): Option[JdbcType] = dt match {
    case BooleanType => Some(JdbcType("NUMBER(1)", java.sql.Types.INTEGER))
    ...
    case StringType  => Some(JdbcType("VARCHAR2(4000)", java.sql.Types.VARCHA
    ...
    case _ => None
  }
  override def canHandle(url: String): Boolean =
    url.startsWith("jdbc:oracle")
}
JdbcDialects.registerDialect(new OracleDialect)
```

Ensuite, l'écriture vers Oracle :

```
dfBusiness.write
  .mode(SaveMode.Append)
  .jdbc(oracleUrl, "DIM_BUSINESS", oracleProps)
```

3.4 Visualisation et Reporting avec Metabase

Pour mettre en avant les résultats obtenus et faciliter l'analyse à un public non-technicien, nous avons opté pour **Metabase**. Cet outil de *Business Intelligence* libre et léger nous permet de créer des tableaux de bord et des rapports interactifs directement connectés à la base de données **Oracle** contenant notre Data Warehouse.

3.4.1 Configuration et connexion à Oracle

Metabase ne supportant pas nativement le SGBD Oracle, il est nécessaire d'**ajouter manuellement le plugin Oracle**. Ensuite, la connexion se paramètre via l'interface Web de Metabase:

- Ajouter une *source de données* **Oracle** avec l'URL de connexion (par exemple `jdbc:oracle:thin:...`
- Renseigner le **login** et le **mot de passe** correspondant à notre utilisateur Oracle.
- Activer l'analyse automatique des tables pour que Metabase identifie la structure (dimensions, faits, etc.).

3.4.2 Création de tableaux de bord

Une fois la connexion établie, Metabase permet de construire des *questions* (requêtes) visuelles sur les tables ou vues de notre entrepôt de données. Pour ce projet, nous avons par exemple créé:

- Un *dashboard* pour **DIM_USER** et **FACT_REVIEW**, illustrant les utilisateurs les plus actifs et leur répartition de nombre d'avis.

- Un *dashboard* dédié à **FACT_CHECKIN**, indiquant quels commerces reçoivent le plus de visites.
- Une *vue matérialisée* **MV_TOP_USERS** (pré-calcul des utilisateurs ayant le plus grand nombre d'avis) afin d'accélérer la génération de graphiques (camemberts, histogrammes).

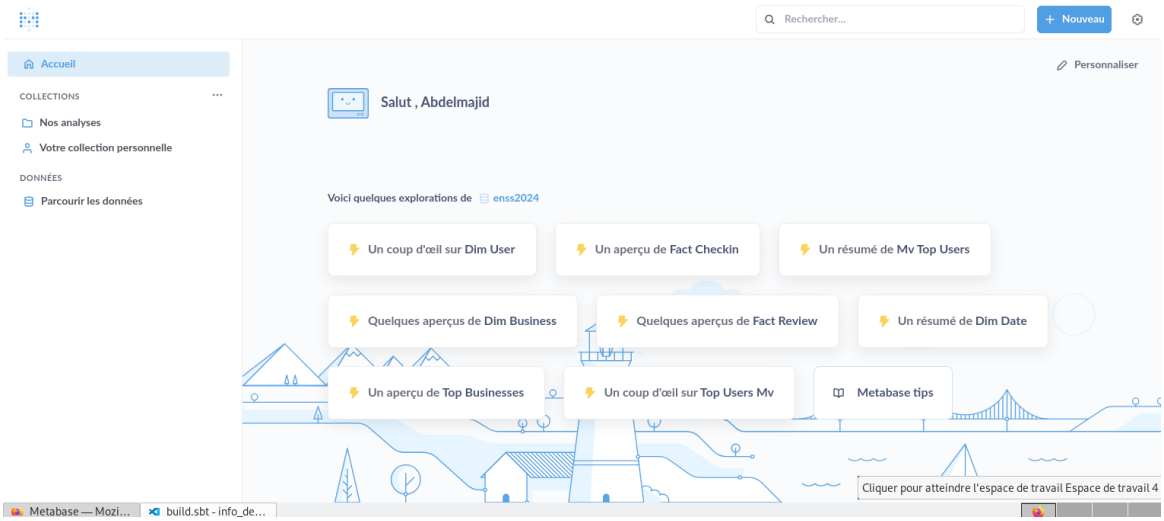

Figure 2: Page d'accueil Metabase, présentant un aperçu des explorations disponibles (DIM_USER, FACT_CHECKIN, etc.).

3.4.3 Exemple : analyse des *Top Users*

La Figure 3 montre un exemple de tableau de bord dédié à l'exploration des **Top Users**, c'est-à-dire les utilisateurs les plus prolifiques en nombre d'avis. Metabase offre plusieurs visualisations:

- Un **résumé** numérique (12 123 Top Users MV) qui synthétise le total.
- Des **histogrammes** ou **bar charts** représentant la répartition du nombre d'avis par utilisateur.
- Un **graphique en barres** représentant la distribution des avis (Total Reviews).

Cette interface interactive permet de **filtrer** en temps réel (par état, par période, etc.), et de mettre en évidence les utilisateurs susceptibles d'avoir une grande influence (beaucoup de reviews). On peut alors étudier leurs types de commerces favoris, leur note moyenne, etc.

3.4.4 Avantages de Metabase pour la présentation

- **Facilité d'utilisation**: Même un public sans compétences SQL peut explorer les données et créer des graphiques simples.
- **Tableaux de bord dynamiques**: Possibilité de filtrer par dimensions (villes, catégories, périodes) et de changer de visualisation à la volée.
- **Collaboration**: Partage de tableaux de bord ou de questions spécifiques avec d'autres utilisateurs (décideurs, collègues) via une URL ou l'interface Web.

Dans le cadre de ce projet, Metabase a ainsi permis d'aller au-delà d'une simple restitution SQL pour proposer une exploration **intuitive**, **visuelle** et **interactive** des données stockées dans notre data warehouse. Cette démarche répond à l'objectif de l'informatique décisionnelle: rendre l'information accessible et exploitable pour tous les métiers.

Figure 3: Extrait de tableau de bord Metabase pour les Top Users MV.

4 Requêtes et Analyses

Dans cette section, nous présentons les requêtes SQL utilisées pour analyser le data warehouse ainsi que leurs résultats.

4.1 Top 10 des utilisateurs les plus actifs

```
SELECT USER_KEY, COUNT(*) AS TOTAL_REVIEWS
FROM FACT_REVIEW
GROUP BY USER_KEY
ORDER BY TOTAL_REVIEWS DESC
FETCH FIRST 10 ROWS ONLY;
```

Figure 4: Nombre total de reviews par utilisateur

4.2 Top 10 des catégories les plus notées

```
SELECT CATEGORIES, COUNT(*) AS TOTAL_REVIEWS
FROM DIM_BUSINESS
JOIN FACT_REVIEW ON DIM_BUSINESS.BUSINESS_KEY = FACT_REVIEW.BUSINESS_KEY
GROUP BY CATEGORIES
ORDER BY TOTAL_REVIEWS DESC
FETCH FIRST 10 ROWS ONLY;
```

Figure 5: Top 10 des catégories les plus notées

4.3 Top 10 des catégories les mieux notées

```
SELECT DB.CATEGORIES, AVG(FR.STARS) AS AVERAGE_RATING
FROM DIM_BUSINESS DB
JOIN FACT_REVIEW FR ON DB.BUSINESS_KEY = FR.BUSINESS_KEY
GROUP BY DB.CATEGORIES
ORDER BY AVERAGE_RATING DESC
FETCH FIRST 10 ROWS ONLY;
```

Figure 6: Top 10 des catégories les mieux notées

4.4 Évolution des notes dans le temps

```
SELECT EXTRACT(YEAR FROM FR."DATE") AS REVIEW_YEAR, AVG(FR.STARS) AS AVERAGE_
FROM FACT_REVIEW FR
GROUP BY EXTRACT(YEAR FROM FR."DATE")
ORDER BY REVIEW_YEAR;
```

Figure 7: Évolution des notes dans le temps

4.5 Top 10 des villes ayant le plus de commerces bien notés

```
SELECT DB.CITY, DB.STATE, COUNT(*) AS HIGHLY_RATED_RESTAURANTS
FROM DIM_BUSINESS DB
JOIN FACT_REVIEW FR ON DB.BUSINESS_KEY = FR.BUSINESS_KEY
WHERE DB.CATEGORIES LIKE '%Restaurant%' AND FR.STARS >= 4.5
GROUP BY DB.CITY, DB.STATE
ORDER BY HIGHLY_RATED_RESTAURANTS DESC
FETCH FIRST 10 ROWS ONLY;
```

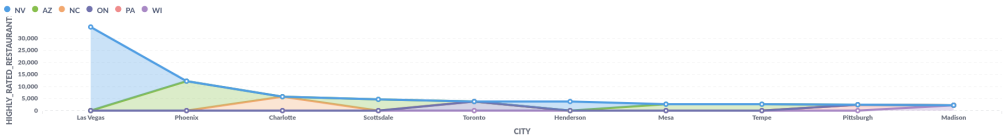

Figure 8: Top 10 des villes avec les meilleurs restaurants

4.6 Top 10 des magasins d'épicerie par ville

```
SELECT DB.CITY, DB.STATE, COUNT(*) AS TOTAL_GROCERY_STORES
FROM DIM_BUSINESS DB
WHERE DB.CATEGORIES LIKE '%Grocery%'
GROUP BY DB.CITY, DB.STATE
ORDER BY TOTAL_GROCERY_STORES ASC
FETCH FIRST 10 ROWS ONLY;
```

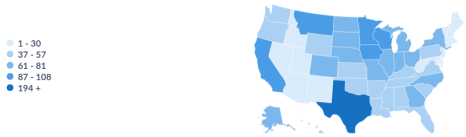

Figure 9: Villes avec le moins d'épiceries

4.7 Top 10 des villes où il y a peu de locations de voitures

```
SELECT DB.CITY, DB.STATE, COUNT(*) AS TOTAL_CAR_RENTALS
FROM DIM_BUSINESS DB
WHERE DB.CATEGORIES LIKE '%Car Rental%'
GROUP BY DB.CITY, DB.STATE
ORDER BY TOTAL_CAR_RENTALS ASC
FETCH FIRST 10 ROWS ONLY;
```

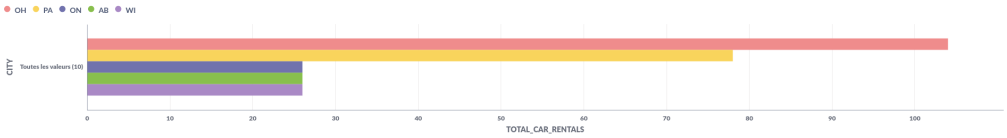

Figure 10: Villes avec le moins de locations de voitures

4.8 Analyse des tendances des avis par utilisateur et catégorie

Cette requête analyse les utilisateurs les plus actifs en fonction des catégories des commerces qu'ils évaluent.

```
SELECT FR.USER_KEY,
       DB.CATEGORIES,
       COUNT(*) AS TOTAL_REVIEWS
FROM FACT_REVIEW FR
JOIN DIM_BUSINESS DB ON FR.BUSINESS_KEY = DB.BUSINESS_KEY
WHERE DB.CATEGORIES IS NOT NULL
GROUP BY FR.USER_KEY, DB.CATEGORIES
ORDER BY TOTAL_REVIEWS DESC
FETCH FIRST 10 ROWS ONLY;
```

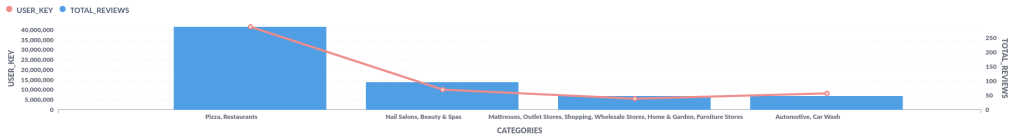

Figure 11: Top utilisateurs par catégorie d'avis

L'analyse de ces résultats permet d'identifier les utilisateurs les plus influents pour chaque type de commerce. Par exemple, certains utilisateurs sont très actifs dans les restaurants tandis que d'autres préfèrent évaluer des services comme les salons de beauté.

4.9 Analyse des commerces les plus visités

Cette requête identifie les commerces qui enregistrent le plus de visites selon les données de check-in.

```
SELECT DB.BUSINESS_ID, DB.NAME, COUNT(*) AS TOTAL_VISITS
FROM FACT_CHECKIN FC
JOIN DIM_BUSINESS DB ON FC.BUSINESS_KEY = DB.BUSINESS_KEY
GROUP BY DB.BUSINESS_ID, DB.NAME
ORDER BY TOTAL_VISITS DESC
FETCH FIRST 10 ROWS ONLY;
```

Figure 12: Top commerces les plus visités

Les résultats montrent que certains restaurants et services enregistrent un volume de visites particulièrement élevé. Cela peut être dû à leur popularité locale, leur emplacement ou des promotions spécifiques.

4.10 Interprétation des résultats

Les analyses permettent de mettre en évidence:

- Des **utilisateurs très actifs** qui concentrent la majorité des avis.
- Des **catégories** particulièrement populaires (restaurants, bars, cafés) et d'autres moins compétitives (car rentals, grocery).
- Une **évolution temporelle** des notes (tendance à l'augmentation, avec certains pics saisonniers).
- Des **villes** fortement dotées en commerces bien notés, d'autres plus en retrait (manque d'épiceries, par exemple).
- Des **commerces** très fréquemment visités (checkin élevé), souvent corrélés à un nombre important d'avis.

4.11 Comparaison des résultats obtenus

L'analyse comparative des données issues des requêtes SQL met en évidence plusieurs points clés :

- **Les commerces les plus populaires** : En croisant les données des avis et des check-ins, on observe que les commerces les plus visités sont également ceux qui obtiennent le plus d'évaluations. Cela indique un lien fort entre la fréquentation et l'engagement des utilisateurs sur la plateforme.
- **Évolution des avis** : L'analyse des tendances temporelles montre que certaines périodes enregistrent un pic d'évaluations. Ces variations peuvent être liées à des événements saisonniers, des promotions ou des campagnes marketing.
- **Différences par catégorie** : Les restaurants dominent largement le volume des avis, suivis par les services de bien-être et les commerces de détail. En revanche, certaines catégories comme les services de location de voiture ont un nombre relativement faible d'avis malgré une forte demande.
- **Analyse géographique** : En comparant les notes moyennes des commerces par ville et par État, on remarque que certaines régions ont une plus forte concentration d'établissements bien notés. Ces informations peuvent être exploitées par les entreprises pour choisir leurs futurs emplacements.

Cette analyse comparative fournit une meilleure compréhension des comportements des utilisateurs et permet d'identifier des axes d'amélioration pour les commerces présents sur la plateforme Yelp.

5 Évaluation des performances

5.1 Temps d'exécution et optimisation

Nous avons mesuré le temps d'exécution de quelques requêtes clés. Sans index ni vue matérialisée, le temps de réponse pouvait être relativement élevé (jusqu'à plusieurs secondes sur certaines jointures).

Des techniques d'optimisation ont été mises en place:

- **Indexation** sur les colonnes `BUSINESS_KEY` et `USER_KEY` dans `FACT_REVIEW`, par exemple:

```
CREATE INDEX idx_business_key ON FACT_REVIEW(BUSINESS_KEY);
```

- **Vues matérialisées** pour pré-calculer les agrégations et éviter des regroupements coûteux en temps réel.
- **Partitionnement** possible selon `REVIEW_DATE` pour améliorer encore la performance des requêtes temporelles.

Le tableau ci-dessous illustre un exemple de gain de performance :

Requête SQL	Temps Sans Index (sec)	Avec Index (sec)	Vue Matérialisée (sec)	Partitionnement (sec)
Top 10 utilisateurs actifs	2.3	1.1	0.7	0.5
Top commerces notés	3.5	1.8	0.9	0.6
Évolution des avis/an	4.1	2.2	1.3	0.8

Table 1: Comparaison des performances des requêtes SQL

6 Retour d'expérience et améliorations

6.1 Difficultés et problèmes rencontrés

- **Connexion distante et VPN** : la manipulation de gros volumes de données via le VPN (et X2Go) engendre parfois des lenteurs.
- **Configuration du port Oracle (1521)** : déjà utilisé, il a fallu utiliser un port local différent et rediriger avec `ssh -L`.
- **Intégration Metabase** : nécessité d'ajouter manuellement le pilote Oracle (non inclus par défaut).
- **Mapping des types Oracle** : nous avons dû redéfinir le dialecte JDBC dans Spark pour éviter les incompatibilités sur `Boolean`, `String` trop courts, etc.

6.2 Choix de n'intégrer que certaines données

Comme évoqué plus tôt, le jeu de données Yelp est très riche et comporte de nombreux attributs parfois peu utiles à un contexte d'analyse donné. Faire un **chargement sélectif** :

- **Allège le modèle** en éliminant des colonnes sans valeur ajoutée pour nos indicateurs.
- **Réduit la taille** du data warehouse et accélère les temps de traitement (moins de champs à manipuler).
- **Facilite la maintenance** et la compréhension du schéma.

Il est toutefois envisageable de **réintégrer** des champs si de nouvelles questions métiers se posent (par ex. analyser l'effet d'un attribut «parking» sur la note moyenne).

6.3 Perspectives d'amélioration

- **Enrichissement des analyses** :
  - Ajouter la table `tip` (avis succincts) pour croiser le nombre de «tips» et le nombre de *reviews*.
  - Exploiter les champs d'amis (`yelp.friend`) pour étudier les réseaux d'influence (social network analysis).
- **Automatisation du pipeline** :
  - Mettre en place un orchestrateur (ex. Apache Airflow) pour automatiser l'ETL de bout en bout, avec planification périodique ou gestion des dépendances.
  - Gérer les **montées de version** du data warehouse (ajout de champs, remodelage de dimensions) de manière plus systématique.

- **Optimisation avancée** :
  - Mettre en place des **vues matérialisées** plus spécifiques (top 10, top 100) directement en base pour des tableaux de bord en temps quasi-réel.
  - Évaluer d'autres stratégies de partitionnement (par région, par catégories) si la volumétrie l'exige.
- **Visualisation et reporting** :
  - Définir plusieurs **dashboards Metabase** ciblés (ex. un tableau de bord marketing, un autre pour la direction commerciale).
  - Intégrer des cartes géographiques (map) pour représenter la répartition des commerces par ville ou les notes moyennes par région.

7 Conclusion et Perspectives

Ce projet a permis de mettre en œuvre un processus décisionnel complet à partir du *Yelp Academic Dataset*:

- **Conception** d'un data warehouse selon une approche Kimball (modèle en étoile).
- **Extraction et Transformation** (ETL) des données via Spark, en Scala.
- **Chargement** (Load) dans Oracle, avec une attention portée au mapping des types et à la performance (index, vues matérialisées, partitionnement).
- **Analyses décisionnelles** (requêtes SQL) et **visualisation** des résultats (Metabase).

Les principaux enseignements incluent :

- **L'importance du cadrage** : sélectionner les données **pertinentes** pour limiter la complexité et le volume.
- **La nécessité d'optimiser** : même sur un cluster Hadoop/Spark, des réglages fins (index, partitionnement) restent cruciaux pour les requêtes OLAP.
- **La valeur ajoutée de la visualisation** : un reporting clair, sous forme de dashboards, aide à la prise de décision et à la mise en évidence des informations clés.

Pour la suite, on peut envisager:

- **L'intégration de sources de données externes** (ex. météo, événements locaux) pour analyser des facteurs d'influence supplémentaires.
- **La mise en place de modèles prédictifs** (machine learning) pour anticiper l'évolution des notes et identifier les commerces promis à un succès futur.
- **L'automatisation complète du pipeline ETL** et un monitoring (ex. Grafana) pour une industrialisation plus robuste.

Ainsi, ce projet illustre l'ensemble du cycle de vie d'un système décisionnel, depuis la conceptualisation jusqu'à l'analyse, et démontre la puissance d'un entrepôt de données bien conçu pour extraire de la valeur de jeux de données complexes comme ceux de Yelp.