Entrepôt de données : Uber Cab

Alexandre Cauty, Loic Schnuriger, Hugo Maitre et Lucas Aïssaoui

21 novembre 2021

Table des matières

1 Introduction **2**

2 Analyse **2**
2.1 Objectifs et position de Uber sur le marché. 2
2.2 Actions et opérations à tracer pour récupérer ces informations 2
2.3 Traitements possibles pour chaque action et opération 2
2.4 Ordre d'importance potentiel des actions à traiter 3

3 Conception **3**
3.1 Actions et opérations à analyser 3
3.2 Conception des Data-mart 3
3.3 Dimensions nécessaires aux modèles. 4
3.4 Pertinence du modèle mis en place. 5
3.5 Estimation de la taille de l'entrepôt. 7

4 Implémentation et requêtage **8**
4.1 Implémentation avec Oracle. 8
4.2 Requêtes analytiques. 8
4.3 Vues matérialisées. 10

5 Conclusion **10**

1 Introduction

Ce rapport rend compte du projet "Projet DW" durant l'unité d'enseignement "HAI708I - Entrepôt de données et big data". Les membres du groupe sont : Alexandre Cauty, Loic Schnuriger, Aïssaoui Lucas et Hugo Maitre .

2 Analyse

2.1 Objectifs et position de Uber sur le marché.

Uber est une plateforme de mise en relation entre chauffeurs et clients. C'est actuellement le leader du marché des VTC. Le service proposé est le plus simple et le plus fiable dans ce domaine. La plate-forme a actuellement plusieurs concurrents Heetch, Kapten et Bolt mais seulement dans les grandes villes d'Europe. Le service proposé par l'entreprise est le transport de particulier et le principal revenu de l'entreprise est la commission de 25% prélevée sur chaque course. Uber est une multinationale qui propose des services dans 65 pays et plus de 600 villes. Il s'adresse à tous les âges des passagers, mais il est principalement utilisé par les clients âgés de 16 à 34 ans. Ils représentent les deux tiers de l'activité de l'entreprise. Les objectifs de cet entrepôt de données pour Uber seraient de maximiser les gains et de optimiser le développement de l'offre sur le territoire. L'entreprise souhaiterait avoir des informations sur l'état du marché pour lui permettre d'étendre son offre là où la demande est la plus forte mais aussi de mieux répartir les chauffeurs sur le territoire pour diminuer le temps d'attente et potentiellement augmenter le nombre de course effectuées chaque jour. De plus Uber souhaiterait avoir des informations concernant les heures creuses et les heures de forte influence pour mettre en place des systèmes de promotion pour limiter la surcharge du réseaux et ainsi réguler l'affluence.

2.2 Actions et opérations à tracer pour récupérer ces informations

Pour notre entrepôt de données, nous aurons besoin de tracer les courses et les recherches faites par les utilisateur de la plate-forme. Pour les courses, nous avons besoin de connaître l'utilisateur qui a effectué cette course avec son lieu de départ et son lieu d'arrivée, la date et l'heure des différents évènements (heure de commande, heure de départ et heure d'arrivée) ainsi que le temps, d'attente le prix, et la distance de la course. Concernant les recherches, nous aurons besoin des informations sur l'état du marché pour chaque heure et pour chaque quartier, pour cela nous avons besoin de connaître le nombre de recherches effectuées, avec le nombre de recherches qui ont abouti à une commande et celles qui n'ont pas abouti, on pourra calculer et stocker ce ratio. Avec cela, nous aurons également besoin de connaître le nombre de chauffeurs présent dans le quartier, avec le nombre de chauffeurs occupés et le nombre de chauffeurs libres, on pourra également calculer et stocker ce ratio.

2.3 Traitements possibles pour chaque action et opération

Tout d'abord, nous allons tracer les courses et toutes les données qui leur sont liées. Cela permettra de faire des requêtes sur le nombre de courses par jour, par quartier et aussi par ville afin de mettre en évidence les jours où le nombre de courses est le plus important mais aussi dans quels quartiers et dans quelles villes il y en a le plus. Les résultats de ces requêtes pourront être croisés avec la période de l'année (saison, vacances, jours fériés) mais aussi avec l'heure (jour, nuit) ou même avec la population et le revenu moyen du quartier qui sont deux données stockées dans la table lieu. Nous pourrons aussi détecter les quartier avec les moins de courses ou qui rapporte le moins d'argent à l'entreprise.

Ensuite, en traçant les recherches faites sur la plate-forme nous allons, comme pour les courses, effectuer des requêtes sur le nombre de recherches par jour, par quartier et par ville. Nous nous intéresserons plus particulièrement aux recherches non abouties ainsi qu'au temps d'attente moyen. Connaître les recherches non abouties en fonction du lieu et du quartier permettra de savoir à quels endroits, quels jours et à quelle heure il manque des chauffeurs. Pour chaque lieu, nous connaîtrons le nombre de chauffeurs disponibles ainsi que le temps d'attente moyen et il sera donc possible de mettre en évidence d'éventuels dysfonctionnements, par exemple des chauffeurs libres mais des recherches non abouties ou un temps d'attente trop long.

Finalement, avec tous ces traitements, nous pourrons facilement construire des courbes et des histogrammes afin d'analyser les résultats et lancer des actions concrètes, comme la mise en place

de primes pour les chauffeurs ou de promotions pour les utilisateurs, pour faire augmenter le chiffre d'affaires de l'entreprise.

2.4 Ordre d'importance potentiel des actions à traiter

L'action la plus importante à traiter semble être les courses, car le nombre de courses réalisées impacte directement les gains de la plate-forme. Ensuite viennent l'étude des recherches qui sont très complémentaires avec celles des courses, car les recherches peuvent potentiellement permettre d'augmenter le nombre de courses. Les deux actions que l'on traite dans cet entrepôt de données sont complémentaires pour augmenter les gains de la plate-forme puisque si l'offre se développe comme nous l'attendons, le nombre de courses va également augmenter, ainsi que le nombre de recherches et les gains suivront la même tendance. Donc les deux actions ont le potentiel nécessaire pour remplir les objectifs de l'entreprise.

3 Conception

3.1 Actions et opérations à analyser

Les deux actions les plus importantes que nous avons décidé d'analyser sont les courses ainsi que les recherches faites par les utilisateurs. Nous avons vu précédemment que ces deux actions peuvent augmenter les gains pour la plate-forme.

3.2 Conception des Data-mart

Pour représenter les courses, nous avons choisi d'utiliser une table de fait avec pour mesure le prix, la note et la distance d'une course. Concernant les dimensions, nous sommes parti du principe qu'une course concerne un utilisateur et un chauffeur à une date précise, avec cela s'ajoute trois relevés heures qui correspondent à l'heure de commande, l'heure de départ et l'heure d'arrivée à destination. Nous avons donc 5 dimensions pour cette représentation chacune représentant respectivement : la date, l'heure, le lieu, l'utilisateur et le chauffeur.

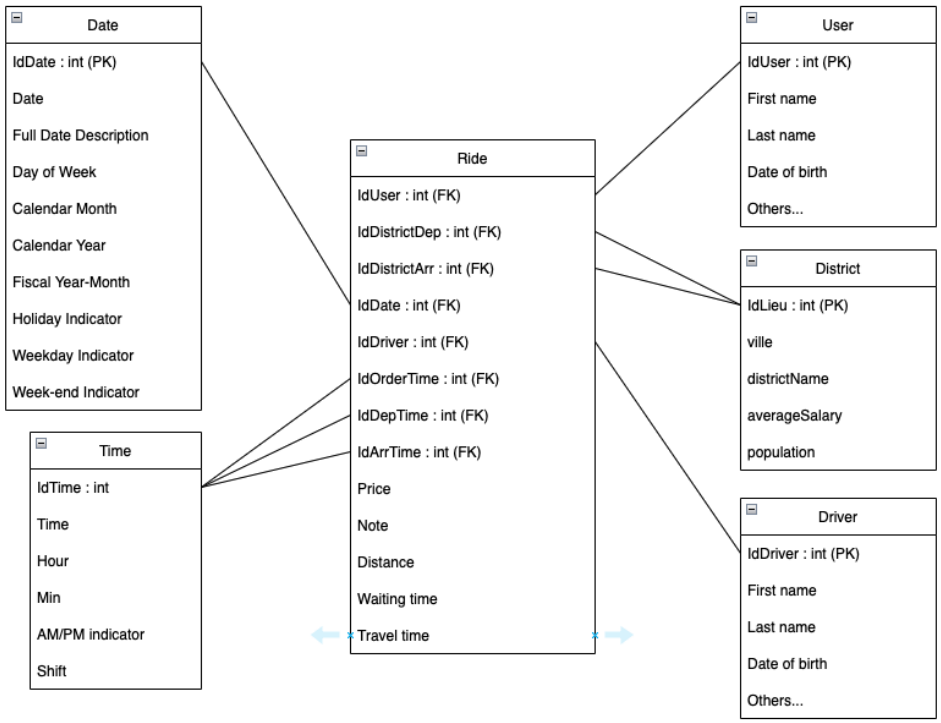

FIGURE 1 – Data-mart de l'action course.

Quant aux recherches, nous avons choisi d'implémenter une représentation snapshot. Pour chaque jour à chaque heure et pour chaque quartier, on relève le nombre de recherches effectuées et si elles ont abouti ou non. Et en parallèle de ça, nous allons relever le nombre de chauffeurs présents dans le quartier et si ils sont libres ou non. On obtient donc la représentation suivante :

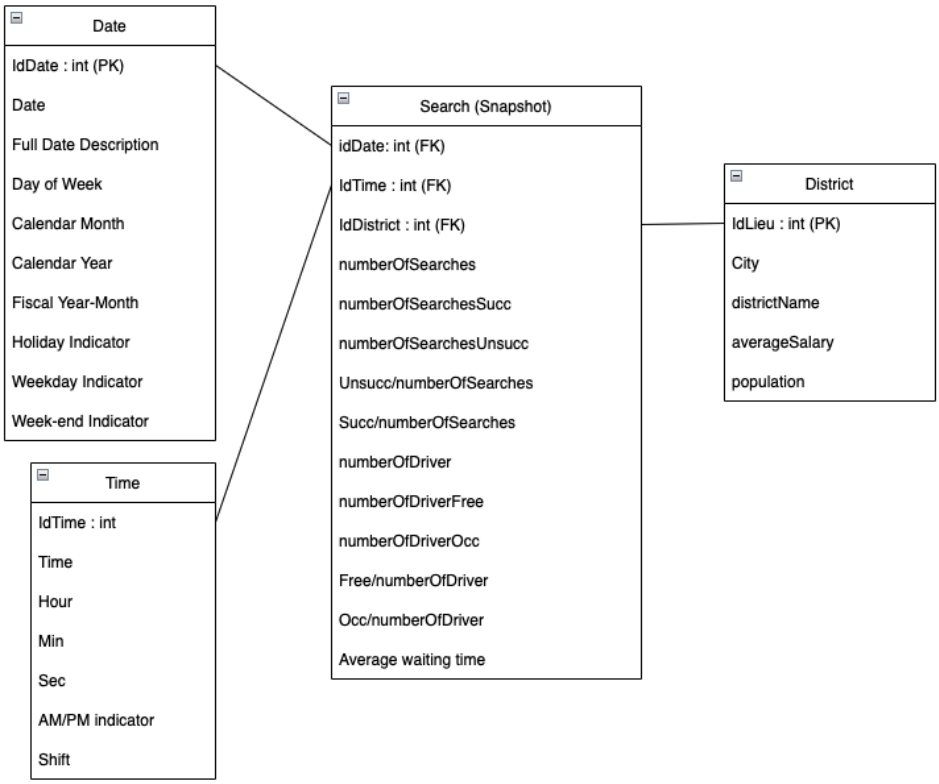

FIGURE 2 – Data-mart de l'action recherche.

3.3 Dimensions nécessaires aux modèles.

Les dimensions que nous allons donc modéliser pour notre implémentation sont : Date, Time, District, User et Driver. Nous avons fait le choix de séparer la date et l'heure pour éviter un produit cartésien qui ferait grossir inutilement notre entrepôt. Pour les tables User et Driver, nous avons choisi une représentation classique. Quant à notre dimension District, elle représente de manière générale, une zone géographique d'intérêt pour l'entreprise.

3.4 Pertinence du modèle mis en place.

Pour mettre en évidence la cohérence de notre entrepôt de données, nous avons décidé d'implémenter une première maquette de test, pour exécuter les requêtes et ainsi vérifier si elles étaient réalisables. Voici des exemples d'instance de nos tables de fait Ride et Search avec leurs dimensions.

District

ID	city	districtName	averageSalary	
1	Montpellier	Boutonnnet	1934.8	
2	Montpellier	Port-Marianne	2465.3	
3	Montpellier	Mosson	1503.2	

Driver

ID	lastName	firtName
1	Michel	Ayoub
2	Jean	Eric

Ride

IdUser	IdDistrictDep	IdDistrictArr	IdDate	IdDriver	IdOrderTime	IdDepTime	IdArrTime	Price	Note	Distance	Waiting time	Travel time
2	2	3	1	2	1	2	3	18.93	5	8.32	13	25

Time

ID	time	hour	min	AM/PM	Shift
1	12h10	12	10	TRUE	TRUE
2	12h23	12	23	TRUE	TRUE
3	12h48	12	48	TRUE	TRUE

Date

ID	date	heure	minutes	secondes
1	12/10/2021	10	32	24
2	12/10/2021	10	40	12
3	12/10/2021	11	02	57

User

ID	lastName	firstName	date	sex
1	Cauty	Alexandre	08/02/2000	Homme
2	Juliat	Elsa	10/12/1999	Homme

FIGURE 3 – Instance de la table Ride.

Date	Hour	District	Q
01/01/20	12h00	Boutonnet	12
01/01/20	12h00	Port-Marianne	150
01/01/20	12h00	Mosson	1
01/01/20	13h00	Boutonnet	40
01/01/20	13h00	Port-Marianne	15
01/01/20	13h00	Mosson	2
02/01/20	14h00	Boutonnet	48
02/01/20	14h00	Port-Marianne	35
02/01/20	14h00	Mosson	34
02/01/20	15h00	Boutonnet	62
02/01/20	15h00	Port-Marianne	117

FIGURE 4 – Instance de la table Search.

Sur cette première instance d'entrepôt de données, nous avons réalisé une requête pour connaître le nombre de courses par jour de la semaine et nous avons obtenu l'histogramme suivant :

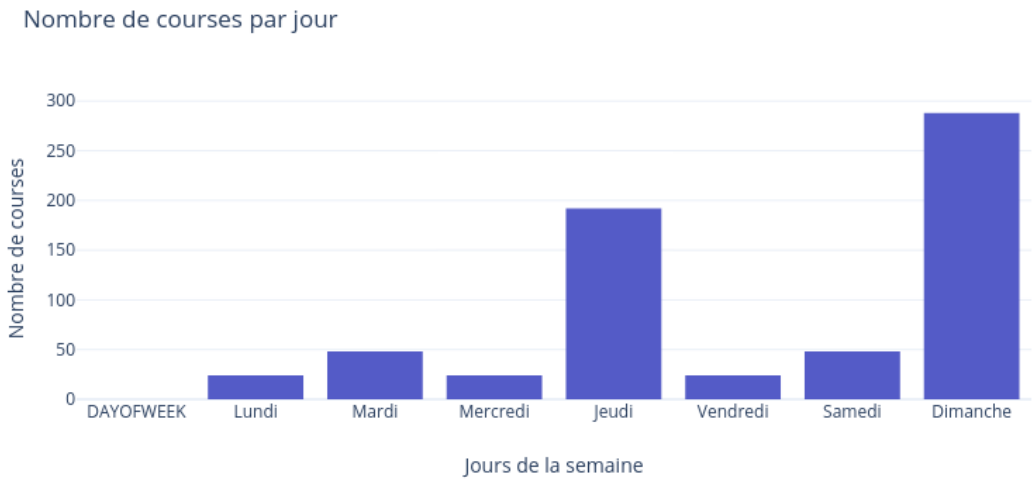

FIGURE 5 – Résultat de la requête pour connaître le nombre de courses par jour de la semaine.

Les résultats précédents nous montrent que cette modélisation peut répondre aux attentes d'Uber. Nous allons donc continuer sur cette implémentation et générer des volumes de données plus importants pour la suite du projet.

3.5 Estimation de la taille de l'entrepôt.

En 2015, Uber a enregistré 1 million de courses ce qui équivaut donc à 1 million de lignes dans la table course sur 12 mois. La table des recherches, dans sa représentation snapshot, avec une fréquence de un relevé par heure, aurait donc pour 12 mois 365 jours fois 24 heures fois le nombre de quartier, que l'on peut fixer arbitrairement à 2000. Ce qui implique de que la table District aurait 2000 lignes. Pour la table User on sait que Uber compte environ 1.5 millions d'utilisateurs en France, soit plus de 1.5 millions de lignes pour la table User. Le nombre de chauffeurs inscrits sur la plate-forme en 2016 était de 13 500, on peut donc compter tout autant de lignes pour cette dimension. Date et Time ont des tailles relativement petites, pour 12 mois Time compte 1440 lignes et Date 365 lignes.

Le nombre de lignes engendrées par la table Ride avec ses dimensions serait donc de : 11 517 296 lignes (1 000 000 + 1 500 000 + 365 + 1440 + 2000 + 13 500 = 11 517 296)

Concernant le nombre de lignes engendrées par la taille de la table Search et de ses dimensions, on peut facilement déduire que sa taille va varier en fonction du temps. Chaque année la table va grandir au pire de 365 * 24 * 2000 lignes sachant que les dimensions comptent environ 3805 lignes. On peut donc déduire que le nombre de lignes engendrées suit une fonction linéaire 365*24*2000* X + 3805 lignes avec X le nombre de mois. Ce qui donne le graphique suivant :

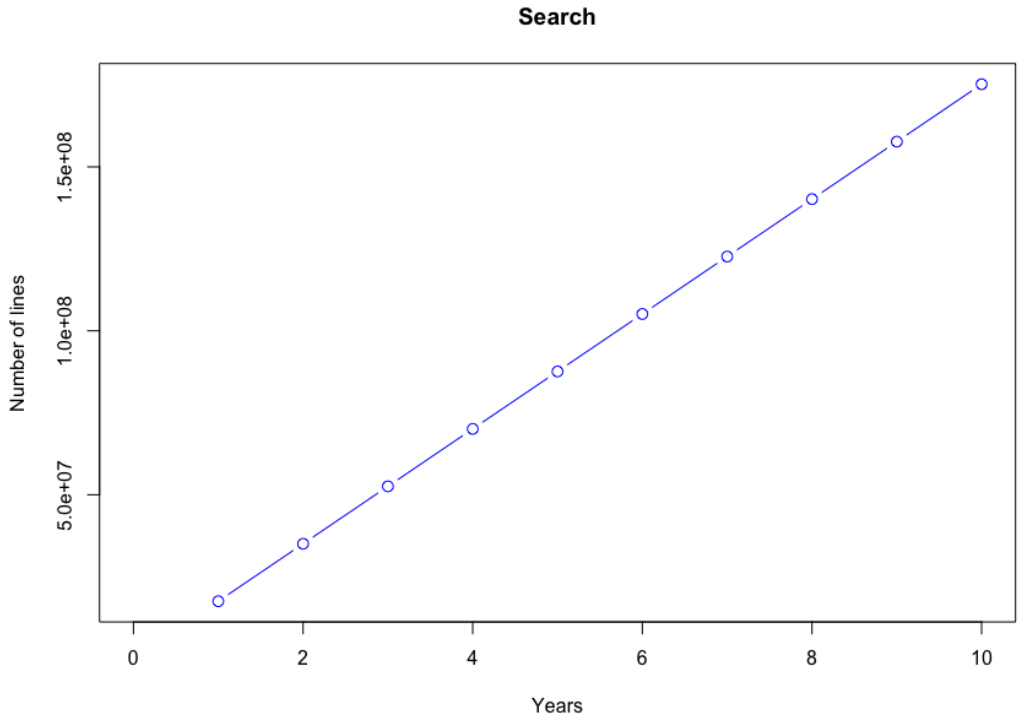

FIGURE 6 – Nombre de lignes engendrées par la table Search.

Cette taille d'entrepôt nous semble raisonnable car le nombre de lignes n'évolue pas de manière exponentielle, on peut voir sur la courbe que le nombre de lignes est d'environ 150 millions de lignes pour 10 ans.

4 Implémentation et requêtage

4.1 Implémentation avec Oracle.

L'implémentation avec Oracle s'est faite sans aucunes difficultés, nous avons créé les tables puis ajouté les clés étrangères. Nous avons utilisé des vues virtuelles pour différencier, dans la table Ride, les lieux de départ, d'arrivée et les temps de commande de départ et de trajet et aussi pour différencier les dates de courses et recherches et les lieux de recherche, de départ et d'arrivée.

```
01 |  CREATE VIEW ride_date AS SELECT * FROM dateC;
02 |  CREATE VIEW search_date AS SELECT * FROM dateC;
03 |
04 |  CREATE VIEW search_time AS SELECT * FROM timeOfDay;
05 |  CREATE VIEW order_time AS SELECT * FROM timeOfDay;
06 |  CREATE VIEW dep_time AS SELECT * FROM timeOfDay;
07 |  CREATE VIEW arr_time AS SELECT * FROM timeOfDay;
08 |
09 |  CREATE VIEW search_district AS SELECT * FROM district;
10 |  CREATE VIEW dep_district AS SELECT * FROM district;
11 |  CREATE VIEW arr_district AS SELECT * FROM district;
```

4.2 Requêtes analytiques.

Voici les 10 requêtes analytiques que nous avons mis en place pour notre entrepôt de données.

Le nombre de courses par villes et par quartier :

```
01 |  SELECT dep_district.city , dep_district.districtName, count(ride.idDistrictDep)
02 |  FROM ride , dep_district
03 |  WHERE ride.idDistrictDep = dep_district.idDistrict
04 |  GROUP BY CUBE(  dep_district.city, dep_district.districtName);
```

Le nombre de courses en fonction du salaire moyen dans le quartier :

```
01 |  SELECT dep_district.averageSalary , dep_district.districtName, count(ride.
       idDistrictDep)
02 |  FROM ride , dep_district
03 |  WHERE ride.idDistrictDep = dep_district.idDistrict
04 |  GROUP BY CUBE(  dep_district.averageSalary, dep_district.districtName);
```

Le nombre de courses par jours en fonction du jour de la semaine :

```
01 |  SELECT ride_date.dayOfWeek, COUNT(ride.idDate)
02 |  FROM  ride_date,ride
03 |  WHERE ride.idDate = ride_date.idDate
04 |  GROUP BY dayOfWeek;
```

Le chiffre d'affaire réalisé par Uber par jours et par quartiers :

```
01 |  SELECT ride_date.dayOfWeek, dep_district.districtName, SUM(ride.price)
02 |  FROM ride_date,ride, dep_district
03 |  WHERE ride.idDate = ride_date.idDate
04 |  and dep_district.idDistrict = ride.idDistrictDep
05 |  GROUP BY CUBE(ride_date.dayOfWeek, dep_district.districtName);
```

Le chiffre d'affaires par quartiers :

```
01 |  SELECT dep_district.districtName,SUM(ride.price)
02 |  FROM dep_district,ride
03 |  WHERE ride.idDistrictDep = dep_district.idDistrict
04 |  GROUP BY ROLLUP(dep_district.districtName);
```

La somme des recherches non abouties par jours :

```
01 |  SELECT search_date.dayOfWeek, SUM(search.nbSearchUnsucc)
02 |  FROM search,search_date
03 |  WHERE search.idDateSearch = search_date.idDate
04 |  GROUP BY(search_date.dayOfWeek);
```

Le nombre de courses non abouties par quartier et par villes

```
01 |  SELECT search_district.districtName, search_district.city, count(search.
       idDistrict)
02 |  FROM search,search_district
03 |  WHERE search.idDistrict = search_district.idDistrict
04 |  GROUP BY CUBE (search_district.districtName, search_district.city);
```

Le nombre de chauffeurs libres pour chaque jours de la semaine :

```
01 |  SELECT search_date.dayOfWeek, SUM(search.nbOccDriver)
02 |          FROM search, search_date
03 |                  WHERE search_date.idDate = search.idDateSearch
04 |                          GROUP BY search_date.dayOfWeek;
```

Le temps d'attente moyen par heures par quartiers :

```
01 |  SELECT search_district.districtName, search_time.hour, SUM(search.
       AverageWaiting)
02 |  FROM search,search_time,search_district
03 |  WHERE search_time.idTime = search.idDateSearch
04 |  AND search_district.idDistrict = search.idDistrict
05 |  GROUP BY (search_district.districtName, search_time.hour);
```

Le nombre courses en fonction de la population par quartier ou par ville :

```
01 |  SELECT dep_district.districtPopulation, dep_district.districtName, sum(ride.
       idDate)
02 |  FROM dep_district, ride
03 |  WHERE dep_district.idDistrict = ride.idDistrictDep
04 |  GROUP BY (dep_district.districtPopulation, dep_district.districtName);
```

Nombre de chauffeurs par district en fonction de la population Pour savoir le ration de chauffeurs en fonction de la population

```
01 |  SELECT order_time.hour ,dep_district.districtName, dep_district.
       districtPopulation, sum(search.nbDriver)
02 |  FROM order_time, dep_district, search
03 |  WHERE order_time.idTime = search.idTimeSearch
04 |  AND dep_district.idDistrict = search.idDistrict
05 |  GROUP BY (order_time.hour ,dep_district.districtName, dep_district.
       districtPopulation);
```

4.3 Vues matérialisées.

Le nombre de courses par jours et par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  courses_View (idDate, idDistrictDep, nb_Courses )
03 |  AS
04 |  SELECT idDate,idDistrictDep,count(idDate)
05 |  FROM ride
06 |  GROUP BY idDate, idDistrictDep;
```

Le CA par jours et par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  RideView (idDate, idDistrictDep, CA_district )
03 |  AS
04 |  SELECT idDate,idDistrictDep,SUM(price)
05 |  FROM ride
06 |  GROUP BY idDate, idDistrictDep;
```

Le prix moyen des cours par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  Prix_View ( idDistrictDep, PrixMoyen )
03 |  AS
04 |  SELECT idDistrictDep, idDate,AVG(price)
05 |  FROM ride
06 |  GROUP BY  idDistrictDep, idDate;
```

Le nombre de chauffeurs par jours et par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  driver_View ( idDateSearch, idDistrict, NbChauffeurs )
03 |  AS
04 |  SELECT idDateSearch, idDistrict,SUM(nbDriver)
05 |  FROM search
06 |  GROUP BY  idDateSearch, idDistrict;
```

Le nombre de chauffeurs libres par jours et par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  free_driver_View ( idDateSearch, idDistrict, NbChauffeursLibres )
03 |  AS
04 |  SELECT idDateSearch, idDistrict,SUM(nbFreeDriver)
05 |  FROM search
06 |  GROUP BY  idDateSearch, idDistrict;
```

Le temps d'attente moyen par jours et par districts.

```
01 |  CREATE MATERIALIZED VIEW
02 |  WaitingAVG_View ( idDateSearch, idDistrict, TempsAttenteMoyen )
03 |  AS
04 |  SELECT idDateSearch, idDistrict,AVG(AverageWaiting)
05 |  FROM search
06 |  GROUP BY  idDateSearch, idDistrict;
```

5 Conclusion

La mise en place d'un entrepôt de données à un coût élevé et nécessite une analyse importante en amont afin de bien cerner les besoins de l'entreprise. L'analyse que nous avons fait nous a permis de nous rendre compte que la action "recherche" est finalement au moins aussi importante que la partie "course" car sont traitement permet de trouver de nombreux points d'améliorations.