Corrigé du PDF révision Big Data

2026-01-03

P.B
Université de Montpellier
2025

Contents

I. OPTIMISATION

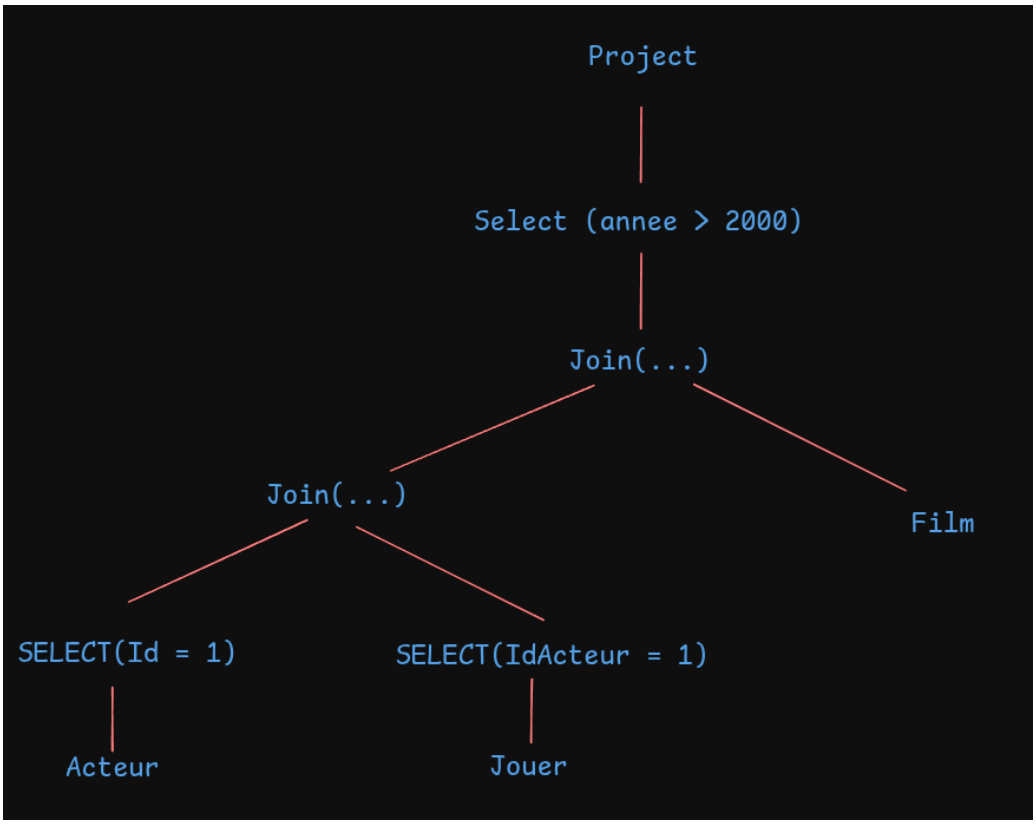
I.1. Plan d'execution

0 : est un select, donc un une requete
1 2 3 : Nested loop join entre
4 5 : Il cherche l'acteur dont l'identifiant est 1. Une fois trouvé dans l'index, il accède à la ligne complète dans la table ACTEUR
7 : Apres avoir trouve l acteur on va join avec jouer
8 : On join avec film
9 : on filtrer sur lannee > 2000

```sql
SELECT *
FROM   Acteur A
  JOIN Jouer J ON A.idA = J.idActeur
  JOIN Film F ON J.idFilm = F.idE
WHERE  A.idA = 1
  AND  F.annee > 2000;
```

I.2. Plan d'execution (couts)

I.2.1. Schema



I.2.2. Entre sortie

Select Acteur:

- Entre: |Acteur|

- Sortie: 1 car on select sur idA qui est une cle primaire

Select Jouer:
- Entre: |Jouer|
- Sortie: |Jouer|

Join entre Acteur et Jouer :
- Entre: $1 * |\text{Jouer}|$
- Sortie: |Jouer|

Join entre (Acteur JOIN Jouer) et Film :
- Entre: $|\text{Jouer}| * |\text{Film}|$
- Sortie: |Jouer| car on ya pas plus d'Acteur que de Film

Select Annee > 2000 :
- Entre: |Jouer|
- Sortie: |Jouer| car on a pas dhypothese, ptet tout la table est > 2000 par defaut avant de select

I.2.3. Expression algebrique

PROJECT(SELECT(ANNEE > 2000, JOIN(Film, JOIN(Jouer, SELECT(idA = 1, Acteur)))))

I.3. Exercice 3 : Plan dexecution (cout avec vues et index)

I.3.1. Index sur une cle primaire

C'est la méthode d'accès la plus classique pour rechercher une ligne unique.Le Scénario : On cherche une commande précise :

```
1  SELECT * FROM Commandes WHERE id_commande = 10542.
```

- Sans l'index (Full Table Scan) : Le système doit lire toutes les pages de la table (du début à la fin) pour trouver le numéro 10542. Si la table fait 100 Go, il lit 100 Go.Coût : $O(N)$ (Proportionnel au nombre de lignes).
- Avec l'index (Index Scan) : L'index (généralement un arbre B-Tree) agit comme le sommaire d'un livre. Le système parcourt l'arbre (racine -> branche -> feuille) pour trouver l'adresse exacte de la ligne.
- Coût : $O(\log N)$ (Généralement 3 ou 4 lectures disques seulement, même pour des millions de lignes).
- Gain : Passage de milliers de lectures à quelques lectures unitaires.

I.3.2. Vue materialisee

C'est la technique la plus radicale pour les requêtes d'agrégation (sommes, moyennes).

Le Scénario : On veut le total des ventes par an : SELECT annee, SUM(montant) FROM Ventes GROUP BY annee.

- Sans la vue : La table Ventes contient 1 milliard de lignes. Le système doit lire le milliard de lignes, les trier ou les hacher, et additionner les montants. C'est très lent.

- Avec la vue matérialisée : Le résultat du calcul a été stocké physiquement dans une petite table séparée (la vue) lors de la dernière mise à jour. Cette vue ne contient peut-être que 10 lignes (une par année).
- Gain : Au lieu de lire 1 milliard de lignes et de faire des calculs complexes, le système lit seulement 10 lignes. Le coût est divisé par un facteur énorme (ex: 1 000 000x plus rapide).

I.3.3. Index de type bitmap

Cet index est spécifique aux colonnes ayant peu de valeurs différentes (faible cardinalité), comme le "Sexe" (H/F) ou le "Statut" (Actif/Inactif).

Le Scénario : Compter les clients inactifs :

```
1  SELECT COUNT(*) FROM Clients WHERE statut = 'Inactif'.
```

- Sans l'index : Un index classique (B-Tree) est inefficace ici car il renverrait trop de résultats (si 50% sont inactifs, l'arbre est inutile, autant tout lire). Le système fait souvent un Full Scan.
- Avec l'index Bitmap : L'index stocke des tableaux de bits (0 et 1).
- Exemple : 1 0 0 1 1 (Le client 1 est inactif, le 2 actif, etc.). Le système lit juste cette petite chaîne de bits très compressée.
- Gain : Réduction drastique de l'espace disque lu. De plus, les opérations sur les bits (ET, OU) sont extrêmement rapides pour le processeur (CPU).

I.3.4. Index de type join-bitmap

C'est l'optimisation reine pour les entrepôts de données (Data Warehouses) en schéma en étoile.

Le Scénario : Une jointure entre une grosse table de faits (Ventes) et une petite dimension (Produits) : SELECT sum(ventes) FROM Ventes v, Produits p WHERE v.id_prod = p.id_prod AND p.categorie = 'Sport'.

- Sans l'index : Le système doit :
  - Trouver les ID des produits 'Sport' (ex: p1, p2).
  - Scanner la table Ventes ou utiliser un index standard pour trouver toutes les ventes liées à p1, puis p2... (Jointure coûteuse).
- Avec l'index Join-Bitmap : L'index est construit sur la table Ventes mais pointe vers les valeurs de la table Produits. Il "sait" déjà quelles lignes de Ventes correspondent à la catégorie 'Sport'.
- Gain : Élimination de la jointure. Le système identifie directement les lignes de la table de faits à lire sans avoir à croiser physiquement les deux tables au moment de l'exécution.

II. Entrepot de donnees

II.1. Questions ouvertes

1. Quelle est la différence entre une base de données relationnelle et un entrepôt de données ?

   La différence fondamentale réside dans leur objectif et leur utilisation :

   - Base de données relationnelle :
     - But : Gérer les opérations quotidiennes en temps réel (inscriptions, commandes, paiements).
     - Structure : Les données sont normalisées (divisées en plusieurs tables) pour éviter la redondance et garantir l'intégrité lors des mises à jour.
     - Performance : Optimisée pour traiter rapidement un grand nombre de petites transactions (lectures/écritures rapides et ponctuelles).
     - Données : Contient les données "courantes" ou actuelles.
   - Entrepôt de données :
     - But : Analyser les données pour la prise de décision (reporting, statistiques, business intelligence).
     - Structure : Les données sont souvent dénormalisées (modèle en étoile ou flocon) pour simplifier les requêtes complexes. On accepte la redondance pour gagner en vitesse de lecture.
     - Performance : Optimisée pour des requêtes lourdes (agrégations, sommes, moyennes) sur de très gros volumes.
     - Données : Contient des données historiques, nettoyées et consolidées.

2. Pourquoi les bases de données relationnelles ne sont-elles pas adaptées à la gestion des données massives ? Elles rencontrent des limites face aux "3 V" du Big Data (Volume, Variété, Vitesse) :

   - Mise à l'échelle rigide (Volume) : Les bases relationnelles "scalent verticalement" (pour être plus performant, il faut un ordinateur plus puissant : plus de RAM, plus de CPU). Cela devient très vite trop coûteux ou physiquement impossible.
   - Structure rigide (Variété) : Elles exigent un schéma strict (tables, colonnes, types définis) avant d'insérer les données. Elles gèrent mal les données non structurées (textes, vidéos, logs, réseaux sociaux) qui composent la majorité du Big Data.
   - Performance en écriture (Vitesse) : Les mécanismes ACID (qui garantissent la cohérence stricte des transactions) imposent des verrous et des vérifications qui ralentissent trop le système lorsqu'il faut ingérer des millions de données par seconde.

3. Pourquoi a-t-on introduit les plateformes de Big-Data ? Quels sont les avantages et les inconvénients par rapport aux entrepôts de données ?

   On a introduit les plateformes Big Data (comme Hadoop) pour traiter des volumes de données qui dépassent les capacités des serveurs traditionnels et pour gérer des données non structurées.

   - Avantages par rapport aux entrepôts de données :

     1. Scalabilité horizontale : On peut stocker des pétaoctets de données simplement en ajoutant des serveurs bon marché (“commodity hardware”) les uns à côté des autres.
     2. Flexibilité (Schema-on-read) : On peut stocker les données brutes sans définir de structure au préalable. On ne définit la structure qu’au moment de la lecture/analyse.
     3. Coût : Le stockage est généralement beaucoup moins cher que dans un entrepôt de données propriétaire.

     Variété : Capacité native à traiter du texte, des images, du son, etc.

   - Inconvénients par rapport aux entrepôts de données :

     1. Performance des requêtes simples : Un entrepôt de données est souvent beaucoup plus rapide pour répondre à une requête SQL standard qu’une plateforme Hadoop (latence plus élevée).
     2. Complexité : Les outils Big Data demandent des compétences techniques plus pointues (programmation, administration cluster) que le SQL standard.
     3. Qualité des données : Comme on stocke tout (“Data Lake”), cela peut devenir un “Data Swamp” (marécage) désorganisé si ce n’est pas bien géré, contrairement à l’entrepôt où les données sont propres et validées.

4. Pourquoi est-il nécessaire d’optimiser l’évaluation des requêtes dans les bases de données relationnelles ? Illustrez l’intérêt de l’optimisation avec un exemple.

   Le SQL est un langage déclaratif : vous dites au système ce que vous voulez (le résultat), mais pas comment l’obtenir (l’algorithme).

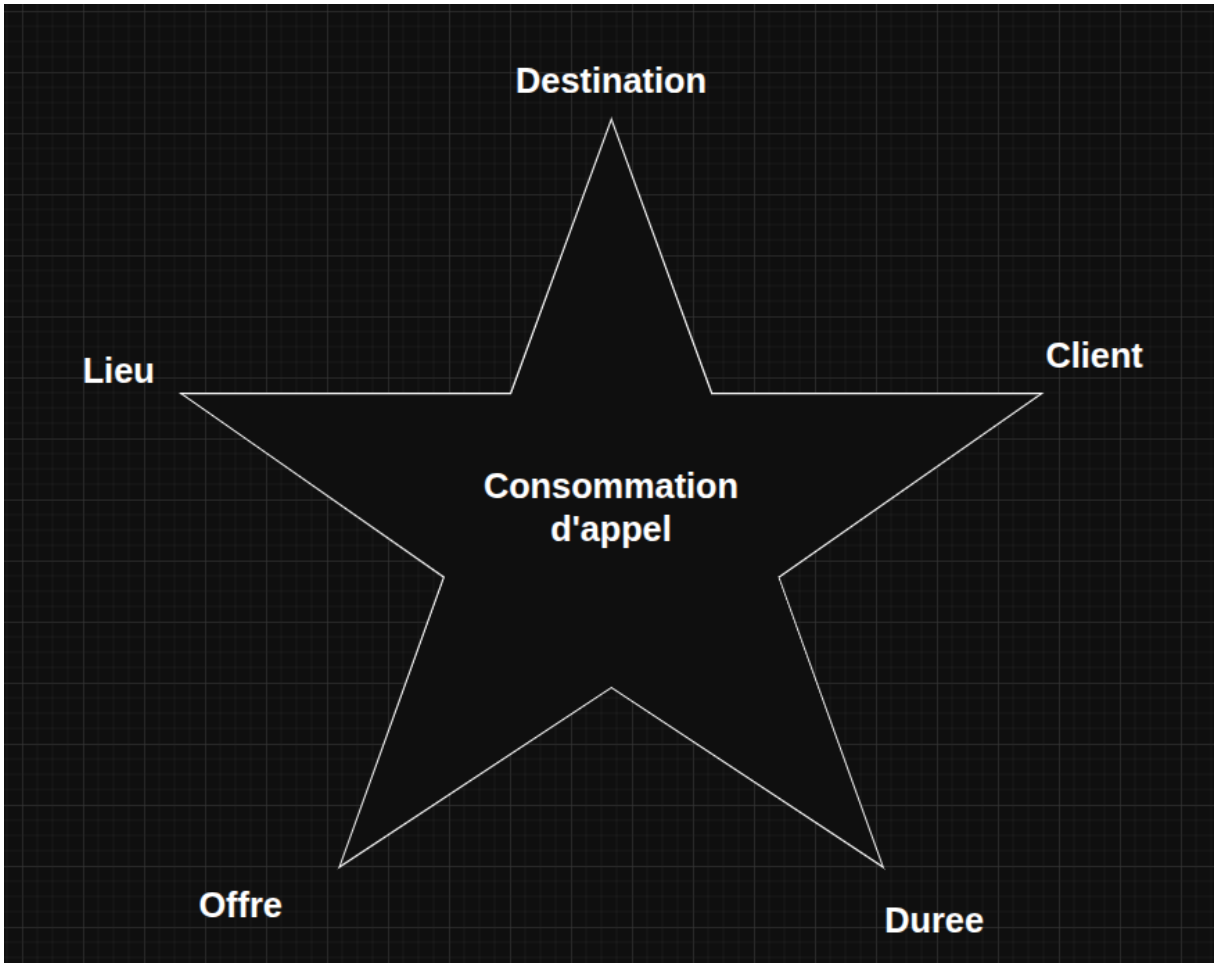
   Il est nécessaire d’optimiser car :

   1. Il existe des milliers de façons d’exécuter une même requête (plans d’exécution).
   2. La différence de temps entre un “bon” plan et un “mauvais” plan peut être gigantesque (quelques millisecondes vs plusieurs heures).
   3. Cela permet d’économiser les ressources matérielles (CPU, Mémoire, Disque).

II.2. Modelisation : Schema en Etoile

II.2.1. Exercice 1 : Telecommunication (Appels)

II.2.1.1. 1) Schema



Client	Destination	Localisation	Duree	Offre
id_client(PK)	id_destination(PK)	id_localisation(PK)	id_duree(PK)	id_offre(PK)
type_client(VARCHAR)	estParticulier (boolean)	ville (VARCHAR)	duree_heure (int)	id_forfait(VARCHAR)
nom_famille (varchar)	estEnFrance (boolean)	code_ville (varchar)	heure_dappel (time)	est_forfait_etudiant (boolean)
prenom (varchar)	estFixe (boolean)	departement_province (varchar)	heure_raccrochement (time)	est_forfait_pro (boolean)
sexe (char)	estMobile (boolean)	pays (varchar)	duree_minute (int)	methode_paiement (varchar)
date_de_naissance (date)	pays_destination (varchar)	continent (varchar)		est_illimite (boolean)
nom_legal (varchar)		fuseau_horraire (varchar)		debut_forfait (date)
numero_identification (varchar)		antenne (varchar)		
secteur_activite (varchar)				
email (varchar)				
telephone (varchar)				
date_inscription (date)				

Dimensions

Consommation d'Appel
id_client (FK)
id_destination (FK)
id_localisation (FK)
id_duree (FK)
id_offre (FK)
total_brute (decimal(10,2))
total_taxe (decimal(10,2))
remise (decimal(10,2))
total_net (decimal(10,2))

Table de fait

II.2.1.2. 2) Additivite

- total_brute : additive, on peut sommer les montants bruts des appels sur une période donnée.
- total_taxe : additive, les taxes peuvent être additionnées pour obtenir le total des taxes sur une période.
- remise : additive, les remises accordées peuvent être additionnées pour obtenir le total des remises sur une période.
- total_net : additive, le total net des appels peut être obtenu en sommant les totaux nets sur une période donnée.

II.2.1.3. 3) 5 requetes analytiques

1. Quel est le total net des appels passés par les clients professionnels au cours de l anne 2024 ?

```sql
SELECT SUM(C.total_net) AS total_net_pro
```

```sql
FROM Consommation_d_Appel C
JOIN Client Cl ON C._id_client_ = Cl.id_client
JOIN Offre O ON C._id_offre_ = O.id_offre
WHERE Cl.type_client = 'professionnel'
  AND C.date_appel BETWEEN '2024-01-01' AND '2024-12-31';
```

2. Quelle est la répartition des appels par continent de destination pour les clients particuliers ?

```sql
SELECT L.continent, COUNT(*) AS nombre_appels
FROM Consommation_d_Appel C
JOIN Client Cl ON C._id_client_ = Cl.id_client
JOIN Localisation L ON C._id_localisation_ = L.id_localisation
WHERE Cl.type_client = 'particulier'
GROUP BY L.continent;
```

3. Quel est le total des remises accordées aux clients ayant souscrit à un forfait illimité ?

```sql
SELECT SUM(C.remise) AS total_remises_illimite
FROM Consommation_d_Appel C
JOIN Offre O ON C._id_offre_ = O.id_offre
WHERE O.est_illimite = TRUE;
```

4. Quelle est la durée moyenne des appels passés entre 18h et 22h pour les clients en France ?

```sql
SELECT AVG(D.duree_minute) AS duree_moyenne
FROM Consommation_d_Appel C
JOIN Client Cl ON C._id_client_ = Cl.id_client
JOIN Localisation L ON C._id_localisation_ = L.id_localisation
JOIN Duree D ON C._id_duree_ = D.id_duree
WHERE L.estEnFrance = TRUE
  AND D.heure_dappel BETWEEN '18:00:00' AND '22:00:00';
```

5. Quel est le total brut des appels passés par les clients inscrits depuis plus d'un an ?

```sql
SELECT SUM(C.total_brute) AS total_brut_anciens_clients
FROM Consommation_d_Appel C
JOIN Client Cl ON C._id_client_ = Cl.id_client
WHERE Cl.date_inscription < CURRENT_DATE - INTERVAL '1 year';
```

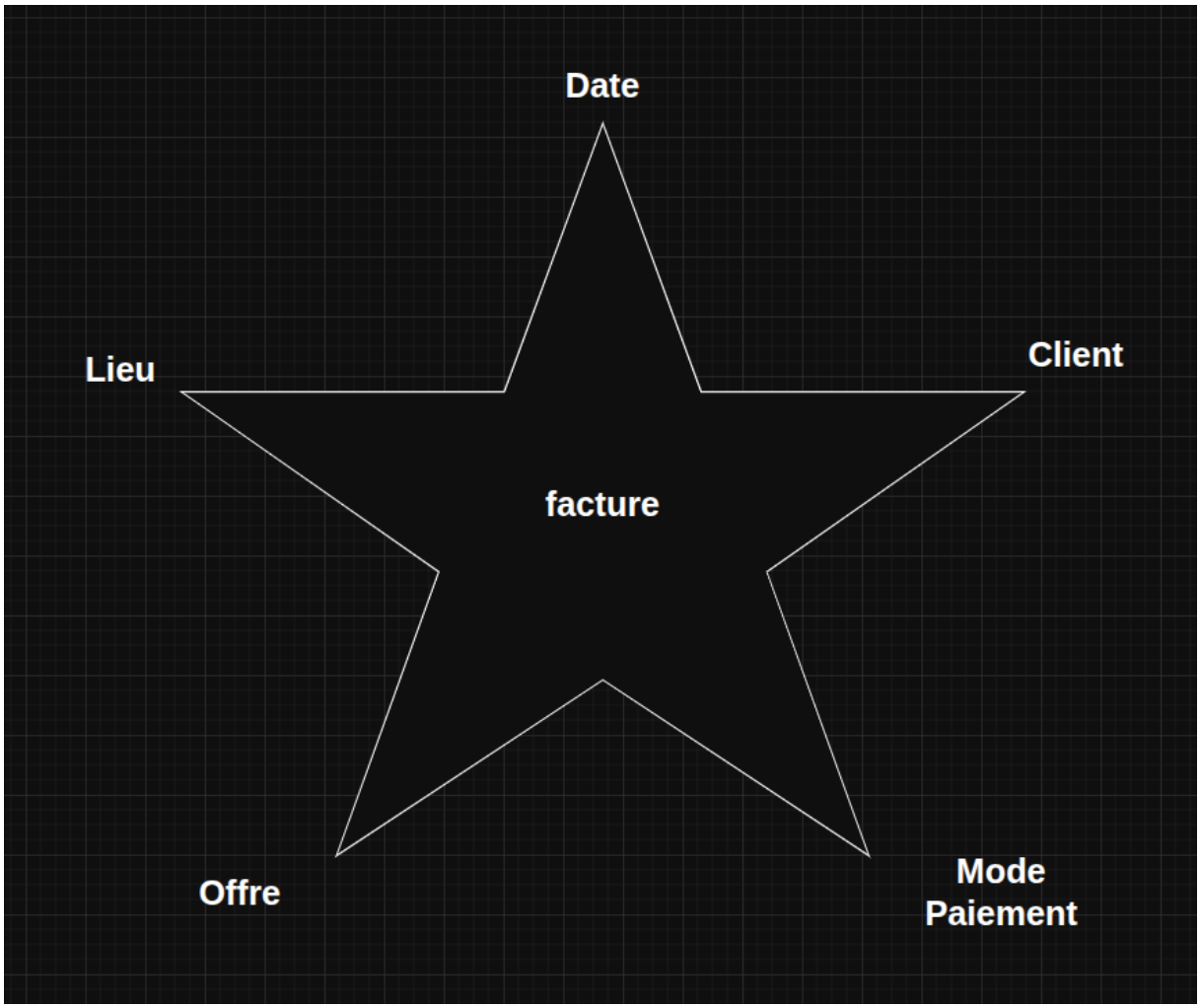
II.2.1.4. 4) Taille entrepot dans 10 ans

- Hypothèses :
  - Nombre de clients : 10 million
  - Nombre d'appels par client par jour : 3 (moyenne)
  - Nbr de jour par ans : 365

- Calcul du nombre total d'appels par an :
  - Total appels par an = Nombre de clients × Nombre d'appels par client par jour × Nbr de jour par ans
  - Total appels par an = $10,000,000 \times 3 \times 365 = 10,950,000,000$ appels par an
- Pour 10 ans on aura :
  - Total appels en 10 ans = $10,950,000,000 \times 10 = 109,500,000,000$ appels

II.2.2. Exercice 2 : Telecommunication (factures clients)

II.2.2.1. 1) Schema



Client	Mode Paiements	Localisation	Date	Offre
id_client (PK)	id_modePaiement (PK)	id_localisation (PK)	id_date (PK)	id_offre (PK)
type_client (VARCHAR)	date expritation(DATE)	ville (VARCHAR)	date_complete (DATE)	id_forfait(VARCHAR)
nom_famille (VARCHAR)	csv(VARCHAR)	code_ville(VARCHAR)	mois(INT)	est_forfait_etudiant(BOOLEAN
prenom(VARCHAR)	banque(VARCHAR)	departement_province (VARCHAR)	annee(INT)	est_forfait_pro(BOOLEAN)
sexe(CHAR(1))	nom_sur_carte(VARCHAR)	pays(VARCHAR)	jour(INT)	methode_paiement(VARCHAR
date_de_naissance(DATE)	numero_carte(VARCHAR)	continent(VARCHAR)		est_illimite(BOOLEAN)
nom_legal(VARCHAR)		fuseau_horraire(VARCHAR)		debut_forfait(DATE)
numero_identification(VARCHAR)		antenne(VARCHAR)		
secteur_activite(VARCHAR)				
email(VARCHAR)				
telephone(VARCHAR)				
date_inscription(DATE)				

Dimensions

Factures
id_facture (PK)
id_client (FK)
id_modePaiement (FK)
id_localisation (FK)
id_date (FK)
id_offre (FK)
montant_brute (DECIMAL)
montant_net (DECIMAL)
remise (DECIMAL)
retard_paiement (DECIMAL)

Table de fait

II.2.2.2. 2) Additivite

- montant_brute : additive
- montant_net : additive
- remise : additive
- retard_paiement : additive

II.2.3. 3) 5 requetes analytiques

- Quelle client a le plus de retard pour le mois janvier lannee 2025

```sql
SELECT Cl.id_client, Cl.nom_famille, Cl.prenom,
SUM(F.retard_paiement) AS total_retard
FROM Factures F
JOIN Client Cl ON F._id_client_ = Cl.id_client
JOIN Date D ON F._id_date_ = D.id_date
```

```
5  WHERE D.mois = 'janvier' AND D.annee = 2025
6  GROUP BY Cl.id_client, Cl.nom_famille, Cl.prenom
7  ORDER BY total_retard DESC
8  LIMIT 1;
```

....

II.2.3.1. 4) Estimation Taille

- 10 million user
- 1 user a 12 facture par ans
- $10,000,000 \times 12 = 120,000,000$ factures par ans
- $120,000,000 \times 10 = 1,200,000,000$ factures en 10 ans

II.2.4. Exercice 3 : Modelisation : Questions Ouvertes

- Que signifié dimension corrélée ? Comment gère-t-on cela dans la modélisation de l'entrepôt de données. Illustrer avec un exemple

  Dimension correle : Des dimensions corrélées sont des axes d'analyse qui possèdent une relation hiérarchique ou logique forte entre eux (ex: Produit et Marque, ou Ville et Pays), mais qui sont modélisés comme des dimensions séparées attachées directement à la table de faits. Cela crée une table de faits "mille-pattes" (avec trop de clés étrangères), ce qui nuit à la performance et complique les requêtes.

  Comment les gérer ? Il faut fusionner ces dimensions en une seule dimension hiérarchique dénormalisée.

  Exemple : Mauvaise pratique : Avoir une Dim_Modèle (Clio) et une Dim_Marque (Renault) liées séparément à la table de faits Ventes. Solution : Créer une seule dimension Dim_Véhicule qui contient les attributs : Id, Modèle, Marque, Gamme.

- Est-il conseillé d'avoir beaucoup de dimensions dans l'entrepôt de données ? Dans quels cas est-il conseillé d'utiliser une mini-dimension ? Illustrer avec deux exemples différents

  Est-il conseillé d'avoir beaucoup de dimensions ? Non. Avoir trop de dimensions (par exemple, plus de 15 ou 20) rend la table de faits très large (beaucoup de colonnes d'index) et les requêtes SQL complexes (trop de jointures). C'est ce qu'on appelle un schéma en "mille-pattes". Il vaut mieux regrouper les attributs corrélés dans des dimensions communes.

  Quand utiliser une Mini-Dimension ? On utilise une mini-dimension (ou Junk Dimension pour les indicateurs binaires, ou Demographic Mini-Dimension) pour sortir d'une dimension principale les attributs qui :

  1. Changent trop fréquemment (pour éviter de faire grossir la dimension principale avec l'historisation SCD Type 2).
  2. Sont utilisés pour des requêtes d'analyse de groupe (profiling) indépendamment de l'identité précise de l'individu.

  Deux exemples différents :

1. Profilage Client (Démographie) :
    - Dans Dim_Client, on garde les infos stables (Nom, Date de naissance).
    - On crée une Mini_Dim_Profil avec des tranches : Groupe d'âge (18-25), Tranche de revenu (Low/Med/High), Nombre d'enfants. Cela permet d'analyser les ventes par "profil" sans joindre la table client de 10 millions de lignes.
2. Indicateurs de Police d'Assurance :
    - Dans Dim_Police, on garde le numéro de contrat et les dates.
    - On crée une Mini_Dim_Risque contenant les combinaisons de risques : Zone à risque (Oui/Non), Conducteur novice (Oui/Non), Usage professionnel (Oui/Non).

- Qu'est une dimension dégénérée? Quelle est son utilité? Illustrer avec un exemple

  Une dimension dégénérée est une clé d'identification (généralement issue du système opérationnel) qui se trouve dans la table de faits mais qui ne pointe vers aucune table de dimension. Elle ne contient pas d'attributs descriptifs supplémentaires.

  Utilité : Elle sert à regrouper les lignes de la table de faits pour revenir au document original (audit, facture précise) ou pour effectuer des comptages distincts (ex: "Combien de paniers uniques avons-nous vendus ?").

  Exemple : Le Numéro de Facture (id_facture) ou le Numéro de Ticket de Caisse. Dans la table de faits Vente_Ligne, le numéro de ticket est stocké pour savoir quelles lignes appartiennent au même achat, mais on ne crée pas une table Dim_Ticket juste pour stocker ce numéro seul.

- Quels sont les avantages et les inconvénients des modèles en flocon de neige dans le cadre des entrepôts de données? Dans quel contexte particulier pourrait-on envisager leur utilisation ? Illustrer avec un exemple.

  Modèle en Flocon de Neige (Snowflake)

  - Avantages :
    1. Normalisation : Évite la redondance des données (on n'écrit "France" qu'une seule fois, pas pour chaque ville française).
    2. Gain d'espace : Moins de stockage (bien que moins critique avec les technologies modernes).
    3. Maintenance : Plus facile de mettre à jour une description (si une région change de nom, on le modifie à un seul endroit).
  - Inconvénients :
    1. Performance : Nécessite beaucoup plus de jointures (JOIN) pour récupérer les informations, ce qui ralentit les requêtes de lecture.
    2. Complexité : Rend le modèle plus difficile à comprendre pour les utilisateurs finaux et les outils de BI.

  - Contexte d'utilisation : On l'envisage quand une dimension est très volumineuse (des millions de lignes) et qu'elle contient des attributs à faible cardinalité (peu de valeurs distinctes) qui sont eux-mêmes très détaillés.
  - Exemple : La dimension Produit d'un géant du e-commerce (Amazon). Plutôt qu'une seule table géante, on peut avoir : Table Faits -> Dim_Produit -> Dim_Sous_Famille -> Dim_Famille -> Dim_Département.

- Quel est le rôle d'une table pont (bridge table) ? Illustrer avec un exemple.

  Table Pont (Bridge Table) :

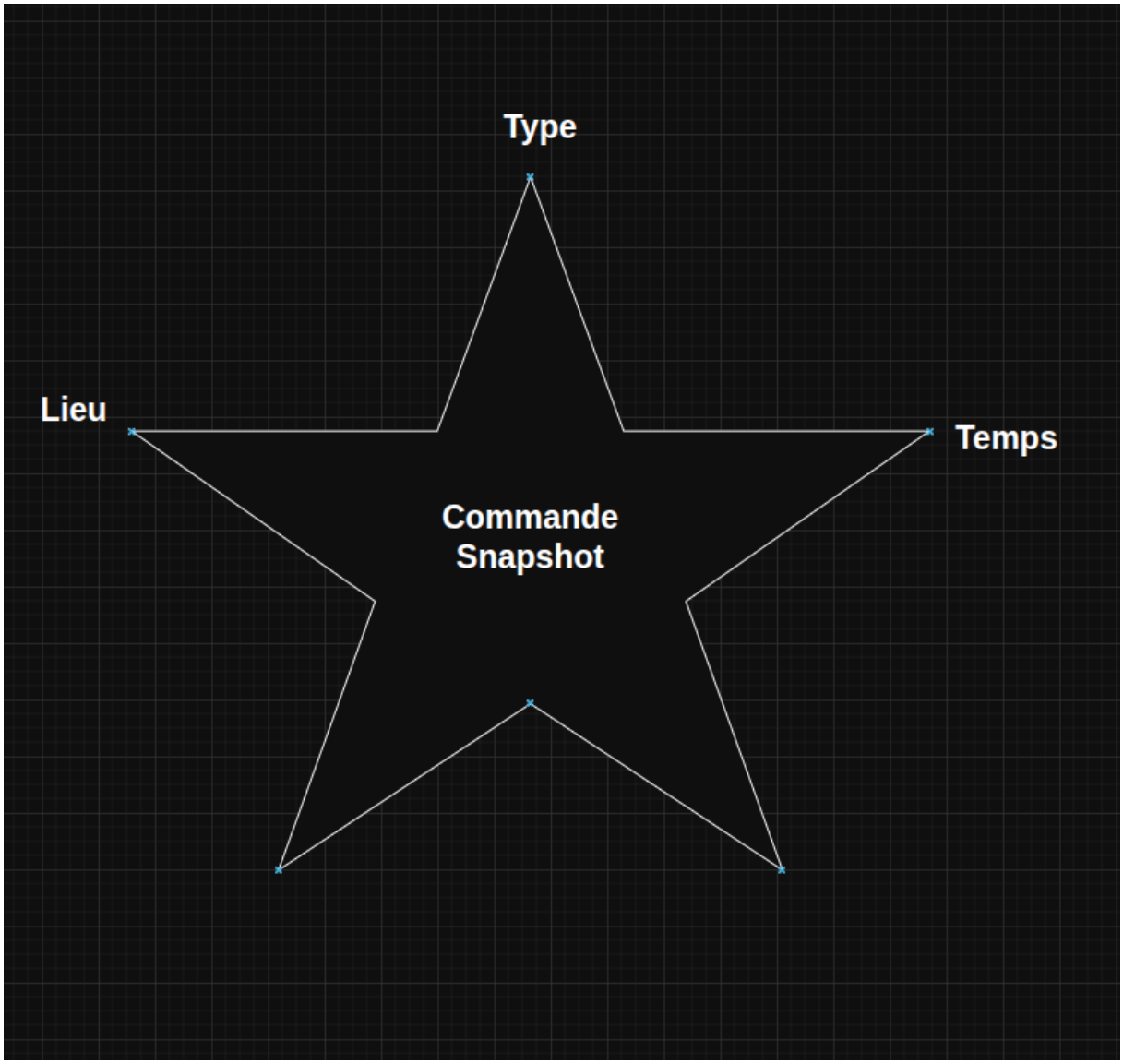
  Rôle : Une table pont sert à modéliser une relation "Plusieurs-à-Plusieurs" (Many-to-Many) entre une table de faits et une dimension. Cela arrive quand une ligne de fait ne correspond pas à une seule valeur de dimension, mais à plusieurs.

  Exemple : Le domaine Médical (Diagnostics).

  - Une consultation (Ligne dans la table de faits) concerne un patient, mais peut aboutir à plusieurs diagnostics (Grippe ET Déshydratation).
  - On ne peut pas mettre plusieurs colonnes "Diagnostic" dans la table de fait.
  - Solution : On crée une table pont Pont_Diagnostic_Consultation entre la table de faits Consultation et la dimension Dim_Maladie. Cette table liste toutes les maladies associées à chaque consultation.

II.2.5. Exercice 4 : Modele Snapshot : Deliveroo

II.2.5.1. 1) Schema Logique



Type	Temps	Lieu
id_type (PK)	id_temps (PK)	id_localisation (PK)
categorie_resto (VARCHAR)	date_complete (DATE)	ville (VARCHAR)
type_produit (VARCHAR)	heure_journee (INT)	code_ville (VARCHAR)
est_repas (BOOLEAN)	jour_semaine (VARCHAR)	departement_province (VARCHAR)
est_boisson (BOOLEAN)	est_weekend (BOOLEAN)	pays (VARCHAR)
est_fastfood (BOOLEAN)		continent (VARCHAR)
		fuseau_horraire (VARCHAR)
		zone_livraison (VARCHAR)

Dimensions

Commande Snapshot
id_commande_snapshot (PK)
id_type (FK)
id_temps (FK)
id_localisation (FK)
nb_total_commandes (INT)
nb_en_cours (INT)
nb_annulees (INT)
nb_terminees (INT)

Table de fait

II.2.5.2. 2) Additivite

- nb_total_commandes : additive, on peut sommer le nombre total de commandes sur différentes périodes ou zones géographiques.
- nb_en_cours : semi additive, si on a 30 commandes en cours à midi et 50 à 14h, on ne peut pas sommer pour dire qu'il y a eu 80 commandes en cours. On peut cependant prendre la valeur maximale sur une période donnée.
- nb_annulees : additive, les annulations peuvent être additionnées pour obtenir le total des annulations.
- nb_terminees : semi additive, si on a 20 commandes terminées à midi et 40 à 14h, on ne peut pas sommer pour dire qu'il y a eu 60 commandes terminées. On peut cependant prendre la valeur maximale sur une période donnée.

II.2.5.3. 3) 5 requetes analytiques

1. Quel est le nombre total de commandes passées dans la ville de Montpellier le 15 juin 2024 ?

```sql
SELECT SUM(CS.nb_total_commandes) AS total_commandes
FROM Commande_Snapshot CS
JOIN Localisation L ON CS._id_localisation_ = L.id_localisation
JOIN Temps T ON CS._id_temps_ = T.id_temps
```

```sql
5  WHERE L.ville = 'Montpellier'
6    AND T.date_complete = '2024-06-15';
```

2. Quel est le nombre de commandes annulées pour les restaurants de type “Fast Food” le week-end du 1er au 2 juillet 2024 ?

```sql
1  SELECT SUM(CS.nb_annulees) AS total_annulees
2  FROM Commande_Snapshot CS
3  JOIN Type Ty ON CS._id_type_ = Ty.id_type
4  JOIN Temps T ON CS._id_temps_ = T.id_temps
5  WHERE Ty.categorie_resto = 'Fast Food'
6    AND T.est_weekend = TRUE
7    AND T.date_complete BETWEEN '2024-07-01' AND '2024-07-02';
```

3. Quel est le nombre de commandes en cours à 19h dans la région Île-de-France le 10 août 2024 ?

```sql
1  SELECT SUM(CS.nb_en_cours) AS total_en_cours
2  FROM Commande_Snapshot CS
3  JOIN Localisation L ON CS._id_localisation_ = L.id_localisation
4  JOIN Temps T ON CS._id_temps_ = T.id_temps
5  WHERE L.departement_province = 'Île-de-France'
6    AND T.heure_journee = 19
7    AND T.date_complete = '2024-08-10';
```

4. Quel est le nombre total de commandes terminées pour les boissons dans le pays France en 2024 ?

```sql
1  SELECT SUM(CS.nb_terminees) AS total_terminees
2  FROM Commande_Snapshot CS
3  JOIN Type Ty ON CS._id_type_ = Ty.id_type
4  JOIN Localisation L ON CS._id_localisation_ = L.id_localisation
5  JOIN Temps T ON CS._id_temps_ = T.id_temps
6  WHERE Ty.est_boisson = TRUE
7    AND L.pays = 'France'
8    AND T.annee = 2024;
```

5. Quel est le nombre total de commandes passées pour les repas dans la zone de livraison “Centre-Ville” le 20 septembre 2024 ?

```sql
1  SELECT SUM(CS.nb_total_commandes) AS total_commandes
2  FROM Commande_Snapshot CS
3  JOIN Type Ty ON CS._id_type_ = Ty.id_type
4  JOIN Localisation L ON CS._id_localisation_ = L.id_localisation
5  JOIN Temps T ON CS._id_temps_ = T.id_temps
6  WHERE Ty.est_repas = TRUE
7    AND L.zone_livraison = 'Centre-Ville'
```

```
8     AND T.date_complete = '2024-09-20';
```

II.2.5.4. 4) Estimation Taille en 10 ans

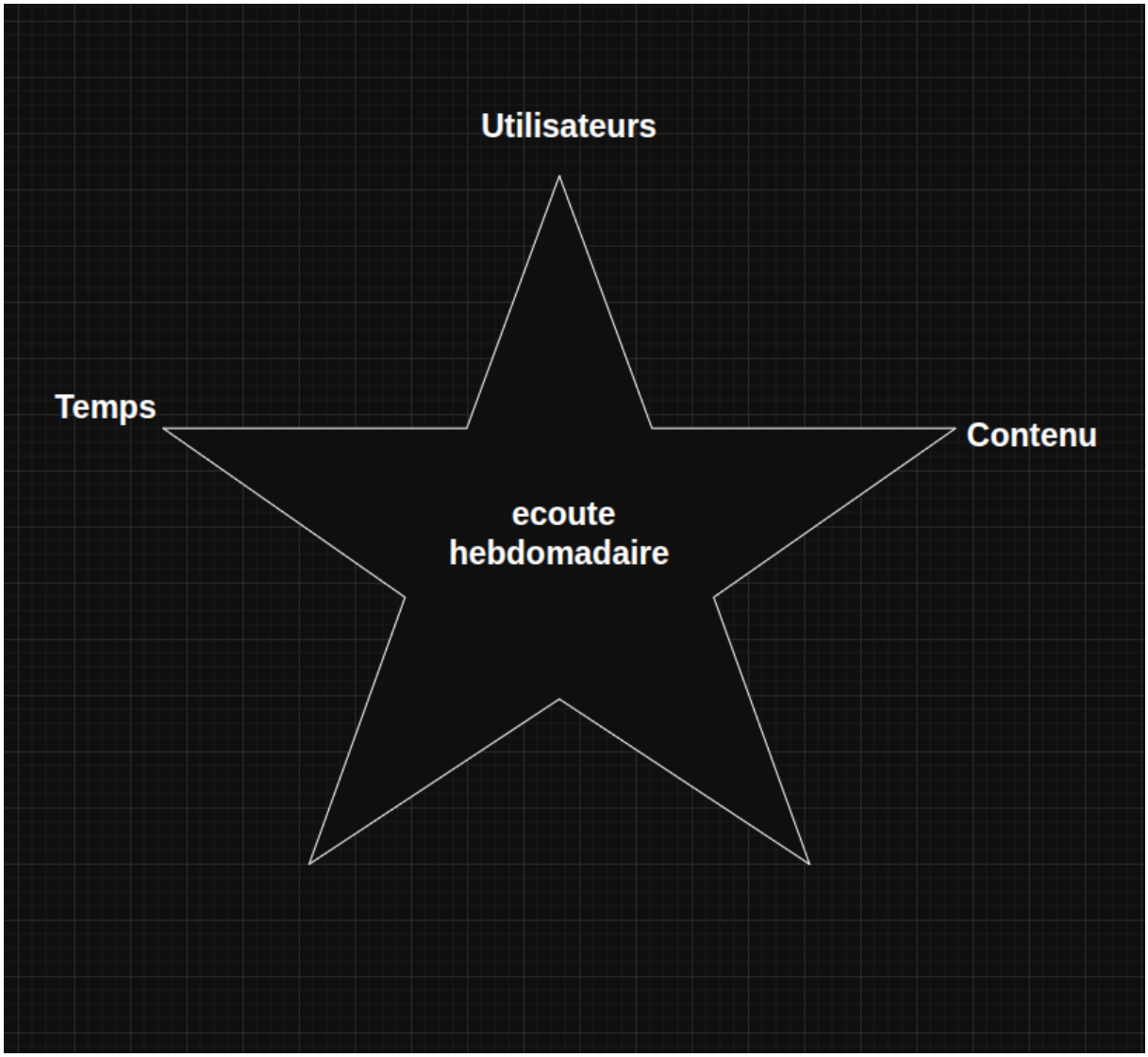
- Hypothèses :
  - Frequence : 1 snapshot par heure
  - Duree : 10 ans = $365 \times 10 = 3,650$
  - Dimension :
    - types : 20 categorie
    - secteur livraison : 5000 (10 secteur par ville, pour 500 ville)
  - Densite : 20% des secteur sont acrives sur 24h
- Calcul :
  - Ligne theorique par heure : 5,000 (Secteurs) $\times$ 20 (Types) = 100,000 lignes possibles/heure
  - Ligne reel : 100,000 $\times$ 0,20 = 20,000 lignes actives/heure
  - Ligne par jour : 20,000 $\times$ 24 = 480,000 lignes/jour
  - Ligne sur 10 ans : 480,000 $\times$ 3,650 = 1,752,000,000
- Poid stockage
  - id_snapshot (BIGINT) : 8 octets
  - id_temps (BIGINT) : 8 octets
  - id_lieu (INT) : 4 octets
  - id_type (INT) : 4 octets
  - nb_total (INT) : 4 octets
  - nb_en_cours (INT) : 4 octets
  - nb_annulees (INT) : 4 octets

  Total données brutes : 36 octets.

  On arrondi a 50 octet.
  - Taille totale = 1,752,000,000 $\times$ 50 octets = 87,600,000,000 octets = 87.6 Go

II.2.6. Exercice 5 : Modele Snapshot : Audible

II.2.6.1. 1) Schema Logique



utilisateur	Temps	Contenu
id_utilisateur (PK)	id_temps (PK)	id_contenu (PK)
Nom (VARCHAR)	date_complete (DATE)	titre (VARCHAR)
Prenom (VARCHAR)	date_debut_semaine (DATE)	auteur (VARCHAR)
est_forfait_etudiant (BOOLEAN)	numero_semaine (VARCHAR)	album (VARCHAR)
pays (VARCHAR)	mois (VARCHAR)	genre (VARCHAR)
tranche_age (INT)	annee (INT)	
type_abonnement (VARCHAR)		

Dimensions

Ecoute Hebdomadaire
id_ecoute_hebdomadaire (PK)
id_utilisateur (FK)
id_temps (FK)
id_conteu (FK)
nb_total_ecoute (INT)
dure_total_decoute (INT)
evaluation (INT)

Table de fait

II.2.6.2. 2) Additivite

- nb_total_ecoute : additive, on peut sommer le nombre total d'écoutes sur différentes périodes ou contenus.
- dure_total_decoute : additive, la durée totale d'écoute peut être additionnée pour obtenir le total sur une période donnée.
- evaluation : non additive, car il s'agit d'une note individuelle. On peut cependant calculer une moyenne des évaluations.

II.2.6.3. 3) 5 requetes analytiques

1. Quel est le nombre total d'écoutes pour le contenu "Le Petit Prince" en janvier 2024 ?

```sql
SELECT SUM(EH.nb_total_ecoute) AS total_ecoutes
FROM Ecoute_Hebdomadaire EH
JOIN Contenu C ON EH._id_conteu_ = C.id_contenu
JOIN Temps T ON EH._id_temps_ = T.id_temps
WHERE C.titre = 'Le Petit Prince'
  AND T.mois = 'janvier'
  AND T.annee = 2024;
```

2. Quelle est la durée totale d'écoute pour les utilisateurs âgés de 18 à 25 ans en France en 2024 ?

```sql
SELECT SUM(EH.dure_total_decoute) AS total_duree
```

```sql
FROM Ecoute_Hebdomadaire EH
JOIN Utilisateur U ON EH._id_utilisateur_ = U.id_utilisateur
JOIN Temps T ON EH._id_temps_ = T.id_temps
WHERE U.tranche_age BETWEEN 18 AND 25
  AND U.pays = 'France'
  AND T.annee = 2024;
```

3. Quelle est la note moyenne pour les albums du genre “Science-Fiction” en mars 2024 ?

```sql
SELECT AVG(EH.evaluation) AS note_moyenne
FROM Ecoute_Hebdomadaire EH
JOIN Contenu C ON EH._id_conteu_ = C.id_contenu
JOIN Temps T ON EH._id_temps_ = T.id_temps
WHERE C.genre = 'Science-Fiction'
  AND T.mois = 'mars'
  AND T.annee = 2024;
```

4. Quel est le nombre total d’écoutes pour les utilisateurs avec un abonnement étudiant en 2023

```sql
SELECT SUM(EH.nb_total_ecoute) AS total_ecoutes
FROM Ecoute_Hebdomadaire EH
JOIN Utilisateur U ON EH._id_utilisateur_ = U.id_utilisateur
JOIN Temps T ON EH._id_temps_ = T.id_temps
WHERE U.est_forfait_etudiant = TRUE
  AND T.annee = 2023;
```

5. Quel est le nombre total d’écoutes pour le contenu de l’auteur “J.K. Rowling” en 2024 ?

```sql
SELECT SUM(EH.nb_total_ecoute) AS total_ecoutes
FROM Ecoute_Hebdomadaire EH
JOIN Contenu C ON EH._id_conteu_ = C.id_contenu
JOIN Temps T ON EH._id_temps_ = T.id_temps
WHERE C.auteur = 'J.K. Rowling'
  AND T.annee = 2024;
```

II.2.6.4. 4) Estimation Taille en 10 ans

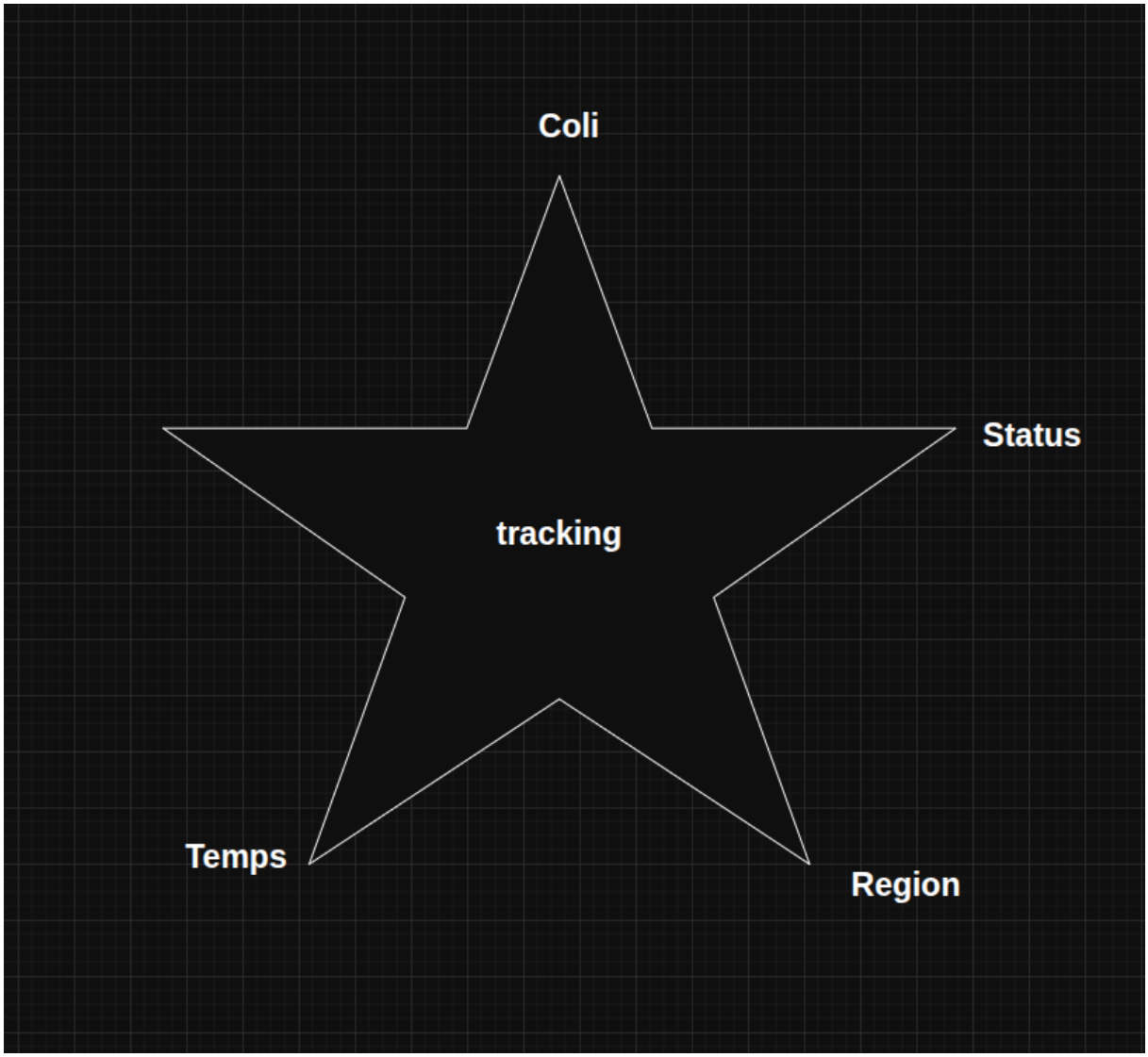
- Hypothèses :
  - Frequence : 1 snapshot par semaine
  - Duree : 10 ans = $52 \times 10 = 520$ (52 semaines par an)
  - Dimension :
    - utilisateurs : 5 million
    - contenu : 1 million
  - Densite : 100% des utilisateurs sont actifs chaque semaine
- Calcul :

  - Ligne par semaine : 5,000,000 (Utilisateurs) × 1,000,000 (Contenus) = 5,000,000,000,000 lignes possibles/semaine
  - Ligne sur 10 ans : 5,000,000,000,000,000 × 520 = 2,600,000,000,000,000,000 lignes en 10 ans
- Poid stockage
  - id_ecoute_hebdomadaire (BIGINT) : 8 octets
  - id_utilisateur (BIGINT) : 8 octets
  - id_temps (BIGINT) : 8 octets
  - id_contenu (BIGINT) : 8 octets
  - nb_total_ecoute (INT) : 4 octets
  - dure_total_decoute (INT) : 4 octets
  - evaluation (INT) : 4 octets

  Total données brutes : 44 octets. On arrondi a 50 octet.
  - Taille totale = 2,600,000,000,000,000,000 × 50 octets = 130,000,000,000,000,000,000 octets = 130 Exaoctets

II.2.7. Exercice 5.5 : Updated records DHL

II.2.7.1. 1) Schema Logique



Coli	Lieu
id_coli (PK)	id_lieu (PK)
poids (INT)	pays_destination (VARCHAR)
type (VARCHAR)	id_expediteur (INT)
id_client_receveur (INT)	

Dimensions

Tracking
id_coli (FK)
id_lieu (FK)
date_heure_reception (DATETIME)
date_heure_tri (DATETIME)
date_heure_expedition (DATETIME)
date_heure_livraison (DATETIME)
dure_total_livraison (INT)
nb_jour_retard (INT)
etat_actuelle (VARCHAR)

Table de fait

II.2.7.2. 2) Additivite

- date_heure_reception : non additive
- date_heure_tri : non additive
- date_heure_expedition : non additive
- date_heure_livraison : non additive
- dure_total_livraison : additive
- nb_jour_retard : additive
- etat_actuelle : non additive

II.2.7.3. 3) 5 requete

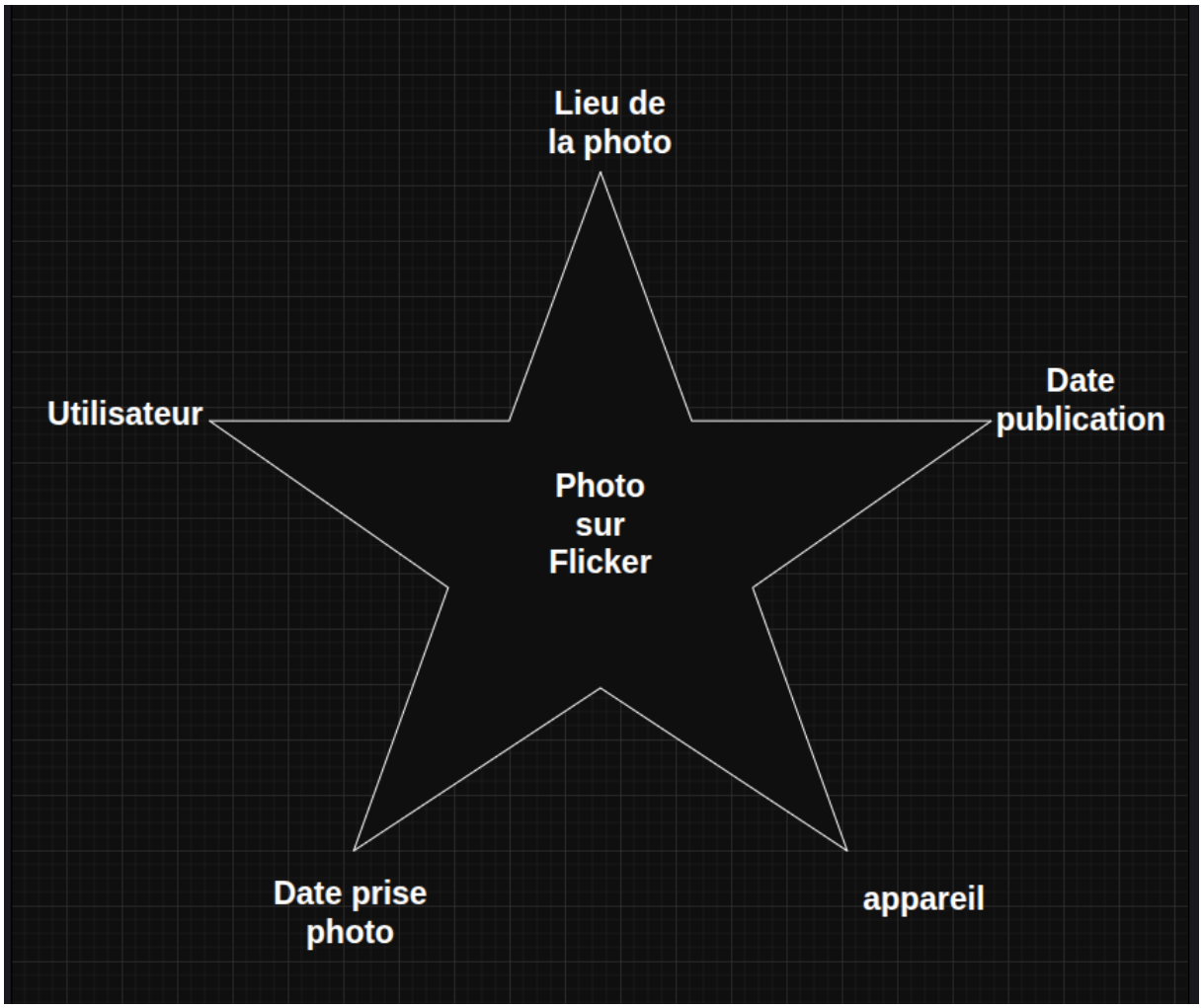
...

II.2.7.4. 4) Estimation Taille

...

III. Table Pont (Bridge table) pour hiérarchique

III.1. Hiearchie de Produit



Utilisateur	Lieu	Date Publication	Appareil	Date prise photo
id_user (PK)	id_lieu (PK)	id_date_publication (PK)	id_appareil (PK)	id_date_prise_photo (PK)
nom_flicker (VARCHAR)	pays (VARCHAR)	date_complete (DATE)	nom_appareil (VARCHAR)	date_complete (DATE)
date_achat_appareil (DATE)	ville (VARCHAR)	annee (INT)	derive_appareil (VARCHAR)	anne (INT)
nom (VARCHAR)	continent (VARCHAR)	mois (VARCHAR)	createur (VARCHAR)	mois (VARCHAR)

Dimensions

Photo
id_utilisateur (FK)
id_lieu (FK)
id_date_publication (FK)
id_appareil (FK)
id_date_prise_photo (FK)
nb_photo (INT)

Table de fait

III.1.1. 1)

```sql
SELECT nom_appareil, COUNT(nb_photo) FROM Appareil A
JOIN Photo P ON A.id_appareil = P._id_appareil
GROUP BY nom_appareil;
```

III.1.2. 2)

```sql
SELECT nom_appareil, COUNT(nb_photo) FROM Appareil A
JOIN Photo P ON A.id_appareil = P._id_appareil
WHERE createur = 'Eiji Fumio'
GROUP BY nom_appareil;
```

III.1.3. 3)

```sql
SELECT nom_appareil, COUNT(nb_photo),
date_prise_photo.date_complete FROM Appareil A
JOIN Photo P ON A.id_appareil = P._id_appareil
JOIN Date_Prise_Photo date_prise_photo ON P._id_date_prise_photo_ =
date_prise_photo.id_date_prise_photo
WHERE derive_appareil = 'Nikon 3000'
GROUP BY date_complete;
```

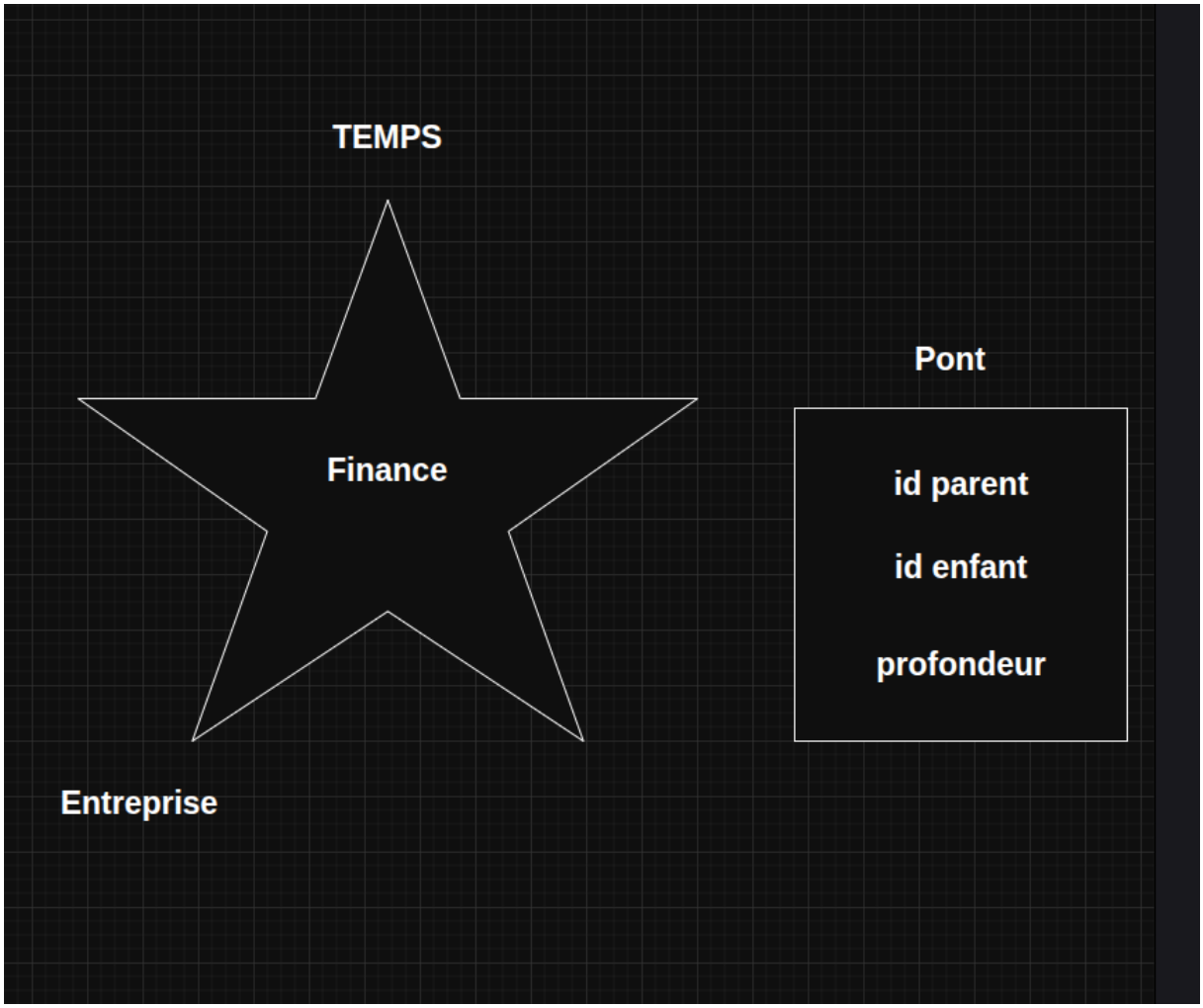
III.1.4. 4)

```sql
SELECT A.nom_appareil, COUNT(*) as degre_influence
FROM Appareil A
JOIN Photo P ON A.id_appareil = P._id_appareil
WHERE A.id_appareil IN (
    SELECT id_appareil
    FROM Appareil
    WHERE derive_appareil IS NULL
)
GROUP BY A.nom_appareil
ORDER BY degre_influence DESC;
```

III.1.5. 5)

On pourrai utilise une table pont sur les requetes 3,4 pour les hiearchie d'appareil. Vu que la hierachie est a plusieur niveau (derive de ... derive de ...) Une table pont permetrai de faire des jointure plus simple et eviterai les auto-jointure recursive.

III.2. Hiearchie des entre et filiales



Temps	Entreprise
id_temps (PK)	id_entreprise (PK)
date_complete (DATE)	nom_societe (VARCHAR)
annee (INT)	secteur_activite (VARCHAR)
mois (INT)	pays_siege (VARCHAR)

Dimensions

Hierarchie_Groupe	
• *id_parent* (FK) -> **Vers id_entreprise** • *id_enfant* (FK) -> **Vers id_entreprise**	• part_detention (DECIMAL) • niveau_profondeur (INT)

Table Pont

Finance	
• *id_temps* (FK) • *id_entreprise* (FK)	• montant_revenu (DECIMAL) • cout_infrastructure (DECIMAL) • benefice_net (DECIMAL)

Table de Fait

III.2.1. 1)

```sql
SELECT Enfant.nom_societe AS Filiale_Detenue,
Pont.niveau_profondeur, Pont.part_detention
FROM Entreprise Mere
JOIN Pont ON Mere.id_entreprise = Pont.id_parent
JOIN Entreprise Enfant ON Pont.id_enfant = Enfant.id_entreprise
WHERE Mere.nom_societe = 'Alphabet Inc';
```

III.2.2. 2)

```sql
SELECT DISTINCT E.nom_societe
FROM Entreprise E
JOIN Pont_Hierarchie P ON E.id_entreprise = P.id_parent
WHERE P.id_parent NOT IN (
    SELECT id_enfant
    FROM Pont_Hierarchie
    WHERE id_parent != id_enfant
);
```

III.2.3. 3)

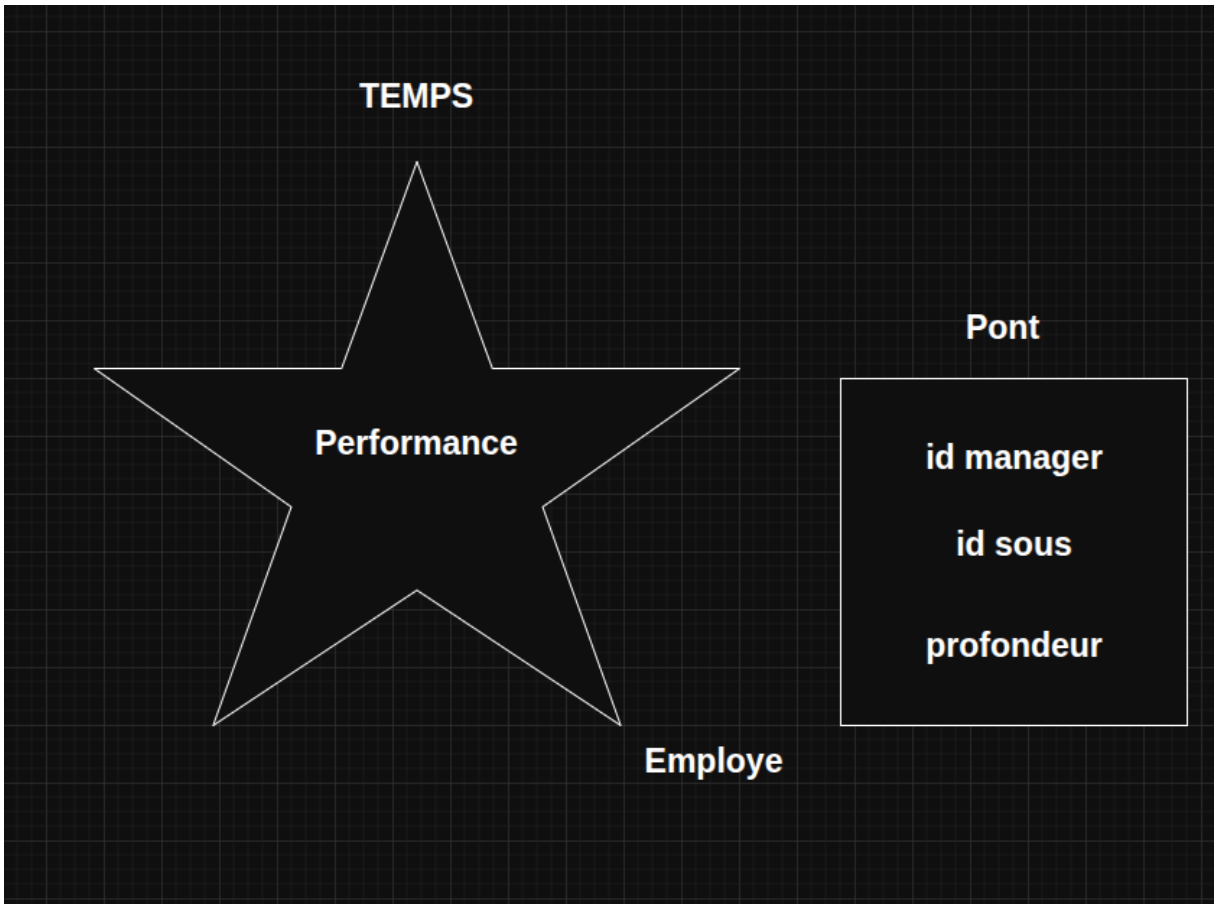
```sql
SELECT SUM(F.montant_revenu) AS CA_Total_Google_Group
FROM Entreprise Mere
JOIN Pont_Hierarchie Pont ON Mere.id_entreprise = Pont.id_parent
JOIN Finance F ON Pont.id_enfant = F.id_entreprise
```

```sql
5 WHERE Mere.nom_societe = 'Google LLC';
```

III.2.4. 4)

```sql
1 SELECT Enfant.nom_societe, COUNT(Pont.id_parent) AS
  Nombre_Parents_Directs
2 FROM Pont_Hierarchie Pont
3 JOIN Entreprise Enfant ON Pont.id_enfant = Enfant.id_entreprise
4 WHERE Pont.niveau_profondeur = 1
5 GROUP BY Enfant.nom_societe
6 HAVING COUNT(Pont.id_parent) > 1;
```

III.3. Hiearchie des employes



Temps	Employe
id_temps (PK)	id_employe (PK)
date_complete (DATE)	nom (VARCHAR)
annee (INT)	salaire (VARCHAR)
mois (INT)	role (VARCHAR)

Dimensions

Hierarchie_Employe	
• *id_manager* (FK) -> **Vers id_employe** • *id_sous* (FK) -> **Vers id_employe**	• niveau_profondeur (INT)

Table Pont

Performance	
• *id_temps* (FK) • *id_employe* (FK)	• montant_revenu (DECIMAL) • cout_infrastructure (DECIMAL) • benefice_net (DECIMAL)

Table de Fait

III.3.1. 1)

```sql
SELECT Manager.nom, AVG(Sous.salaire)
FROM Employe Manager
JOIN Pont_Hierarchie Pont ON Manager.id_employe = Pont.id_manager
JOIN Employe Sous ON Pont.id_sous = Sous.id_employe
GROUP BY Manager.nom;
```

III.3.2. 2)

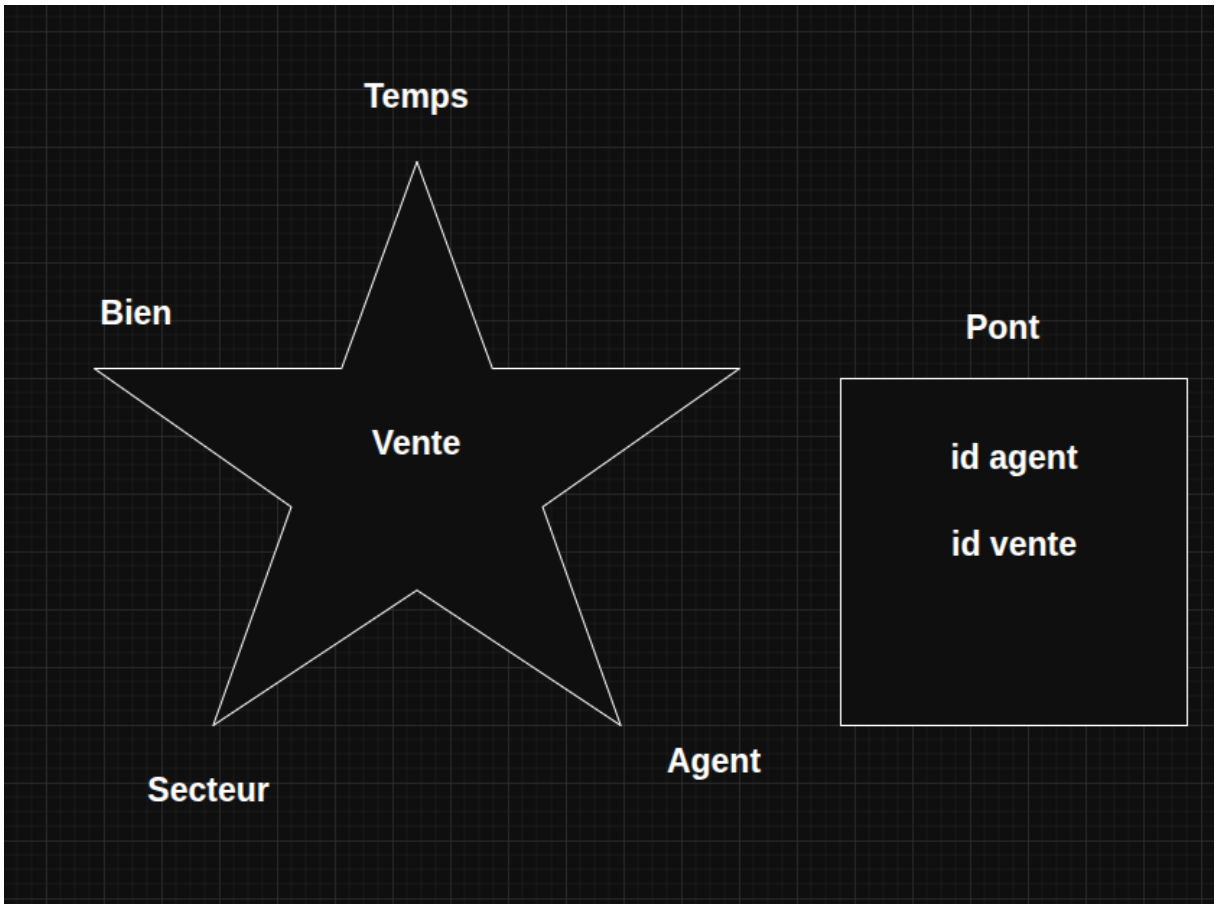
```sql
SELECT Manager.nom, SUM(montant_revenu) FROM Performance P
JOIN Hierarchie_Employe Pont ON P._id_employe_ = Pont.id_sous
JOIN Employe Manager ON Pont.id_manager = Manager.id_employe
WHERE Pont.niveau_profondeur > 0
GROUP BY Manager.nom;
```

III.3.3. 3)

Les 2

IV. Table-Pont (Bridge table) pour relation N:M

IV.1. Groupes d'agents



Temps	Secteur	Bien	Agent
id_temps (PK)	id_secteur (PK)	id_bien (PK)	id_agent (PK)
date_vente	nom_quartier	type_appart (T2, T3...)	nom_complet
mois/annee	ville	surface_m2	equipe

Dimensions contextuelles

Fait : Vente_Globale
• id_vente (PK) • *id_temps* (FK) • *id_secteur* (FK) • *id_bien* (FK) • montant_total_vente (DECIMAL)

Pont : Contribution_Agent
• *id_vente* (FK) -> **Vers Fait** • *id_agent* (FK) -> **Vers Agent** • part_contribution (DECIMAL) • montant_marge (DECIMAL)

La relation Vente < - > Agent passe par le Pont pour gérer le multi-agents

IV.1.1. 1)

```sql
SELECT id_vente, COUNT(DISTINCT CA._id_agent_) AS Nombre_Agents
FROM Contribution_Agent CA
JOIN Vente_Globale VG ON CA._id_vente_ = VG.id_vente
WHERE VG.montant_total_vente > 1000000;
GROUP BY id_vente;
```

IV.1.2. 2)

```sql
SELECT A.nom_complet, SUM(CA.montant_marge) AS Total_Marge
FROM Agent A
JOIN Contribution_Agent CA ON A.id_agent = CA._id_agent_
JOIN Vente_Globale VG ON CA._id_vente_ = VG.id_vente
JOIN Temps T ON VG.id_temps = T.id_temps
WHERE T.annee = 2019
GROUP BY A.id_agent, A.nom_complet;
```

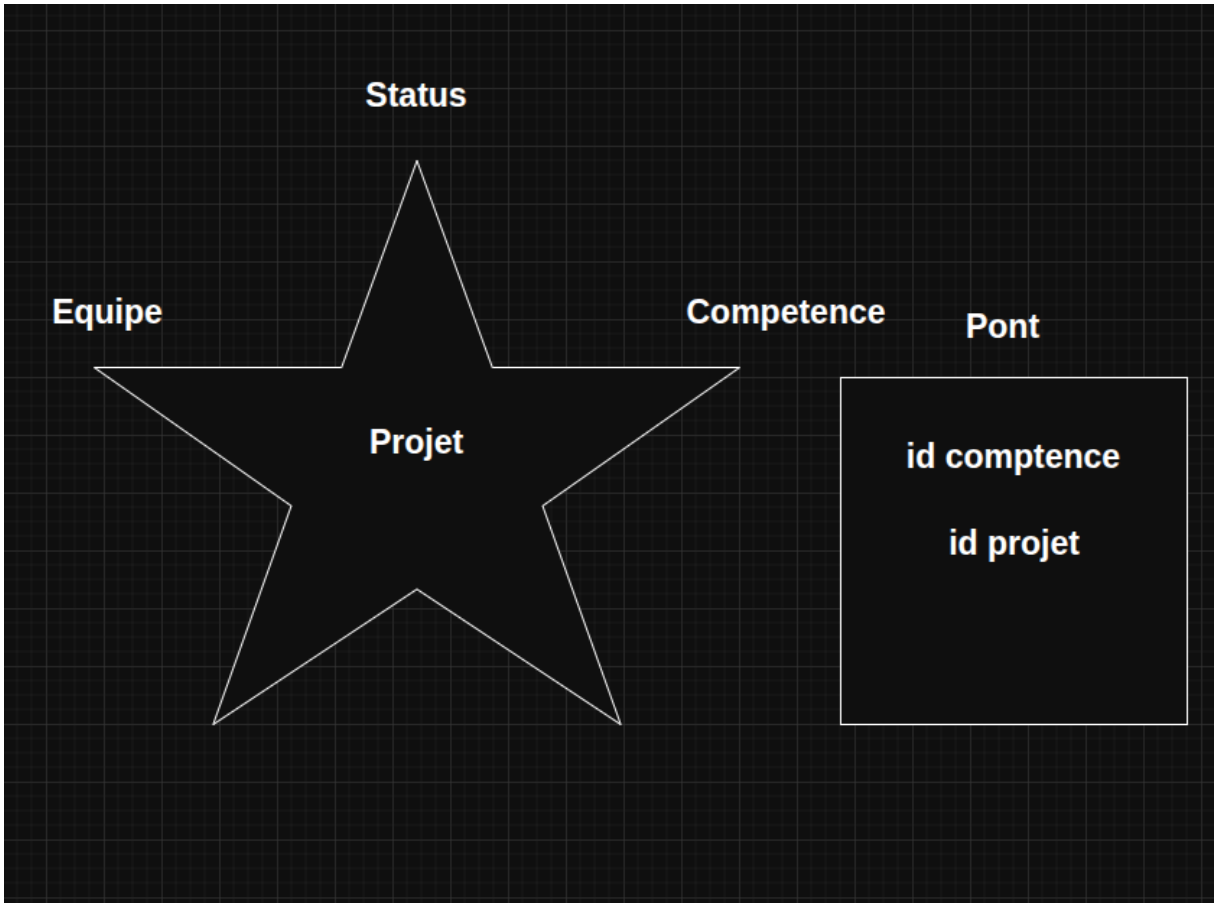
IV.1.3. 3)

```sql
SELECT id_vente, SUM(montant_marge) AS Marge_Totale
FROM Contribution_Agent
GROUP BY id_vente
```

IV.1.4. 4)

TOUS

IV.2. Projets et Compétences



Equipe	Status	Competence
id_equipe (PK)	id_status (PK)	id_competence (PK)
nom_equipe	etat (En cours/Fini)	technologie (Java, SQL)
chef_projet		domaine (Dev, Test, Réseau)

Les Dimensions contextuelles

Fait : Projet
• id_projet (PK)
• *id_equipe* (FK)
• *id_status* (FK)
• *id_temps_debut* (FK)
• budget_total (DECIMAL)
• duree_estimee (JOURS)

Pont : Utilisation_Comp
• *id_projet* (FK) -> **Vers Fait**
• *id_competence* (FK) -> **Vers Dim**
• pourcentage (DECIMAL)

Fait et Pont

IV.2.1. 1)

```sql
1  SELECT id_projet, COUNT(DISTINCT UC._id_competence_), pourcentage
```

```sql
2  FROM Utilisation_Comp UC
3  JOIN Projet P ON UC._id_projet_ = P.id_projet
```

IV.2.2. 2)

```sql
1  SELECT P.id_projet, C.technologie, UC.pourcentage
2  FROM Utilisation_Comp UC
3  JOIN Competence C ON UC._id_competence_ = C.id_competence
4  JOIN Projet P ON UC._id_projet_ = P.id_projet
5  WHERE P.budget_total < 100000;
```

IV.2.3. 3)

```sql
1  SELECT technologie, id_projet, SUM(budget_total * (pourcentage /
   100)) AS Budget_Attribue
2  FROM Utilisation_Comp UC
3  JOIN Competence C ON UC._id_competence_ = C.id_competence
4  JOIN Projet P ON UC._id_projet_ = P.id_projet
5  GROUP BY technologie, id_projet;
```

IV.2.4. 4)

```sql
1  SELECT technologie, SUM(pourcentage) AS Proportion_Cumulee
2  FROM Utilisation_Comp UC
3  JOIN Competence C ON UC._id_competence_ = C.id_competence
4  JOIN Projet P ON UC._id_projet_ = P.id_projet
5  WHERE P.budget_total > 100000
6  GROUP BY technologie;
7  ORDER BY Proportion_Cumulee DESC;
```

IV.2.5. 5)

tout

V. Mises a jour - Evolution de l'entrepot de donnees

V.1. Mise a jour des Dimensions (le cas de la dimension Client)

- Type 1 : Ecrasement (pas d'historique)
  - Avant : Jean | Paris
  - Après : Jean | Lyon
- Type 2 : L'Historisation par ligne (ADD ROW)
  - Avant : Jean | Paris | du 01/01/2020 au 31/12/2023 | Actif
  - Après :
    - Jean | Paris | du 01/01/2020 au 31/12/2023 | Inactif
    - Jean | Lyon | du 01/01/2024 au NULL | Actif
- Type 3 : L'Historisation par attribut (ADD ATTRIBUTE)
  - Avant : Jean | Ville_Actuelle : Paris
  - Après : Jean | Ville_Actuelle : Lyon | Ville_Precedente : Paris

V.1.1. 1)

- nom de famille : Type 1 (Ecrasement)
- date inscriptions : Type 1 (Ecrasement)
- adresse actuelles : Type 2 (Historisation par ligne)
- tranche_age : Type 1 (Ecrasement)
- profil : type 2 (Historisation par ligne)
- indice de confiance : Type 1 (Ecrasement)

V.1.2. 2)

- ```sql
  SELECT SUM(F.montant_vente) AS CA_Total_Clients_Fiables
  FROM Fait_Vente F
  JOIN Dim_Client C ON F.id_client = C.id_client_tech
  JOIN Dim_Temps T ON F.id_temps = T.id_temps
  WHERE T.annee = 2024 AND C.indice_confiance > 90;
  ```

- ```sql
  SELECT
    C.ref_client, C.nom_client, COUNT(*) AS Nombre_Statuts_Différents
    FROM Dim_Client C
    WHERE C.date_debut >= ADD_MONTHS(SYSDATE, -12)
    GROUP BY C.ref_client, C.nom_client
    HAVING COUNT(*) > 3; -- Il faut au moins 4 lignes pour avoir fait "plus de 2 changements" (3 changements)
  ```

- On suppose Jean a fait un paiement de 5000 euro en fevrier 2024, son score est de 95/100. En decembre 2024 il arrete de faire des achats et son score est de 30. Pour la premiere requete : Chiffre d'affaire des clients dont l'indice de confiance est supérieur à 90% en 2024, on aura pas Jean dans le calcul car en decembre son score est de 30. Donc on ne compte pas les 5000 dans le calcul ceux qui est faux dun point economique et reel.

- On suppose Jean a fait un gros paiement de 10000 euro en janvier 2024, son score est de 95/100. En mars 2024 il paye moins et son score descend a 50. Donc pour la ligne jean on a Score Actuelle = 50 | Score Precedant = 95. En decembre 2024 il arrete de faire des achats et son score est de 20. Sa ligne dans la db devient : Score Actuelle = 20 | Score Precedant = 50. Lors de la premiere requete : Chiffre d'affaire des clients dont l'indice de confiance est supérieur à 90% en 2024 on aura pas Jean dans le calcul car son score actuelle est de 20. Donc on ne compte pas les 10000 dans le calcul ceux qui est faux dun point economique et reel.

VI. Partitionnement

VI.1. Partitionnement par lignes et colonnes pour la dimension Clients

VI.1.1. 1)

VI.1.1.1. Partitionnement par colonnes

- ```sql
  CREATE TABLE Client_Statique (
      id_client INT PRIMARY KEY,
      nom_complet VARCHAR(100),
      date_inscription DATE,
      pays VARCHAR(50),
      region VARCHAR(50)
  );
  ```

- ```sql
   CREATE TABLE Client_Dynamique (
       id_client INT PRIMARY KEY,
       indice_confiance DECIMAL(5,2),
       derniere_connexion DATETIME,
       segment_marketing VARCHAR(20),

       CONSTRAINT fk_client_static
           FOREIGN KEY (id_client)
           REFERENCES Client_Statique(id_client)
   );
  ```

VI.1.1.2. Partitionnement par lignes

```sql
CREATE TABLE Client_Statique (
    id_client INT,
    nom_complet VARCHAR(100),
    date_inscription DATE,
    pays VARCHAR(50),
    region VARCHAR(50),

    PRIMARY KEY (id_client, region)
```

```
9 )
```

```sql
PARTITION BY LIST (region) (
    PARTITION p_nord VALUES ('Nord'),
    PARTITION p_sud  VALUES ('Sud'),
    PARTITION p_est  VALUES ('Est'),
    PARTITION p_ouest VALUES ('Ouest'),

    PARTITION p_autre VALUES (DEFAULT)
);
```

VI.1.2. 2)

```sql
SELECT CD.id_client, CS.nom_complet, CD.indice_confiance
FROM Client_Dynamique CD
JOIN Client_Statique CS ON CD.id_client = CS.id_client
WHERE CS.region = 'Sud' AND CD.indice_confiance > 80;
```

VII. Évolution de l'entrepôt de données

VII.1. Questions ouvertes

- Une nouvelle source de données a une granularité différente de celle des données existantes. Comment gérer cette différence ? Illustrer cela à l'aide d'un exemple

  Il faut harmoniser les données. Le plus simple est d'agréger les données les plus fines (somme ou moyenne) pour qu'elles correspondent au niveau des données les moins fines. Sinon, créez deux tables de faits distinctes.

  Exemple :

  - Existant : Ventes par Mois.
  - Nouveau : Ventes par Jour.
  - Action : On additionne les ventes journalières pour obtenir un total mensuel et l'intégrer à l'entrepôt.

- Si la nouvelle source contient des dimensions supplémentaires non présentes dans l'entrepôt actuel, comment intégrer ces dimensions tout en conservant la cohérence avec les dimensions existantes ? Illustrer cela à l'aide d'un exemple

  Ajoutez la nouvelle dimension au schéma. Pour les données existantes (qui ne possèdent pas cette info), liez-les à une ligne "fictive" dans la dimension appelée "Non Applicable" ou "Inconnu".

  Exemple :

  - Existant : Ventes en magasin (Pas de données de livraison).
  - Nouveau : Ventes Web (Avec dimension Livraison).

Action : Pour toutes les anciennes ventes magasin, la dimension Livraison pointe vers “Non Applicable”.

- Vous ajoutez un nouvel attribut dans une table dimensionnelle. Comment gérer les lignes existantes pour lesquelles cet attribut n’est pas disponible ? Illustrer avec un exemple.

  Ajoutez la colonne et remplissez les lignes existantes avec une valeur par défaut explicite comme “Inconnu”, “N/A” ou une valeur vide, pour éviter les NULLs qui peuvent gêner les requêtes.

  Exemple :

  - Dimension Client. On ajoute l’attribut Programme Fidélité (Gold, Silver…).
  - Action : Les anciens clients reçoivent la valeur “Inconnu” ou “Aucun” s’ils n’étaient pas inscrits.

- Vous ajoutez une nouvelle mesure dans une table de faits. Comment gérer les lignes existantes pour lesquelles cette mesure n’est pas disponible ? Illustrer avec un exemple.

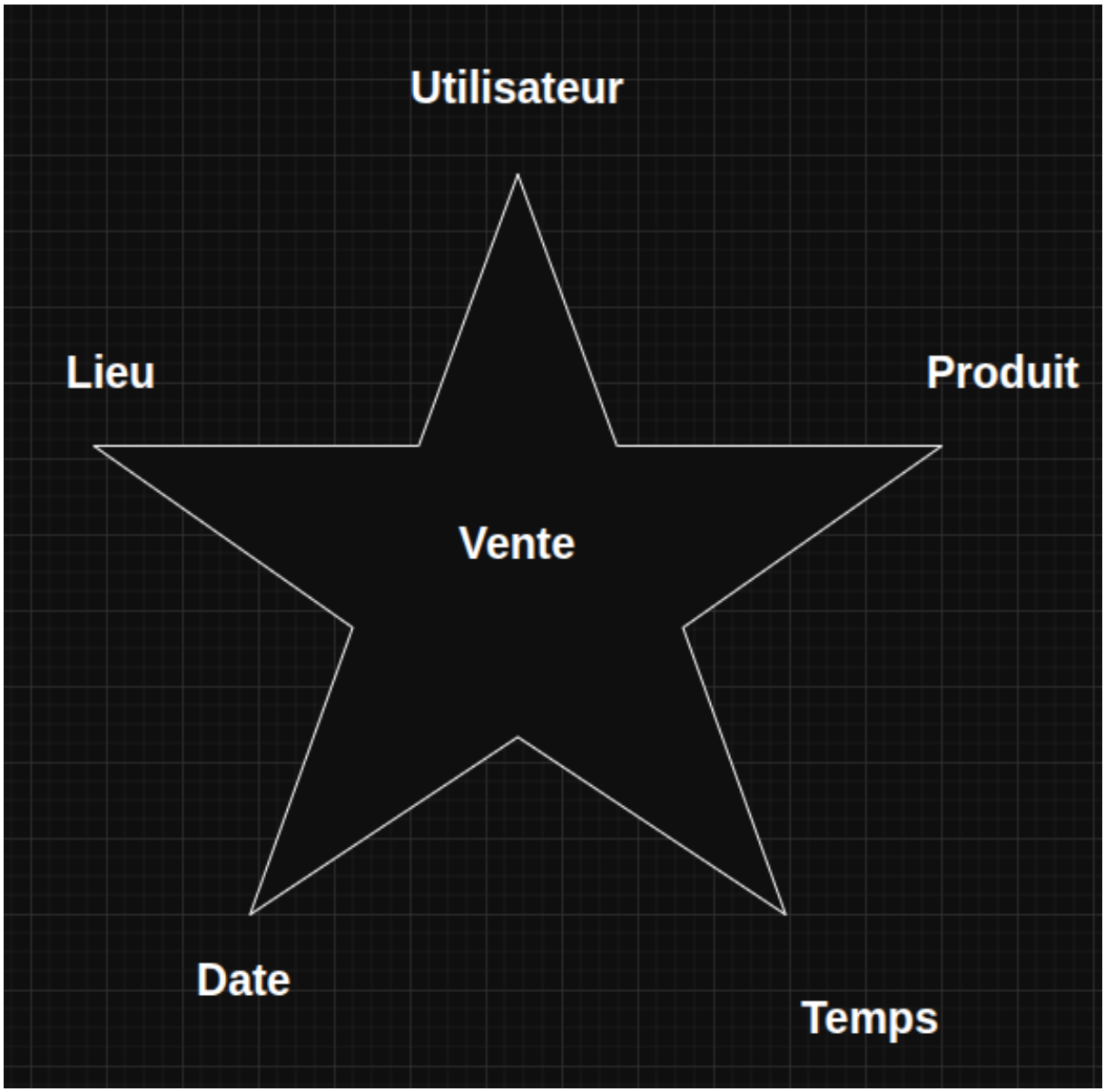
  Ajoutez la colonne dans la table de faits. Pour les anciennes lignes, mettez la valeur à NULL (si la donnée n’existait pas) ou 0 (si cela a du sens mathématiquement).

  Exemple :

  - Table Ventes. On ajoute la mesure Taxe Carbone.
  - Action : Les ventes des années précédentes (avant la taxe) auront NULL ou 0 dans cette colonne.

VIII. Vue Materialee

VIII.1. Exercice 1



VIII.1.1. 1)

- a.

```sql
SELECT COUNT(*), pays AS total_commandes
FROM Ventes
JOIN Lieu ON Lieu.id_lieu = U.id_lieu
WHERE Ventes.heure > 22
GROUP BY pays;
```

- b.

```sql
SELECT COUNT(id_vente), heure FROM Ventes
```

```
2  JOIN Temps ON Ventes.id_temps = Temps.id_temps
3  GROUP BY heure
```

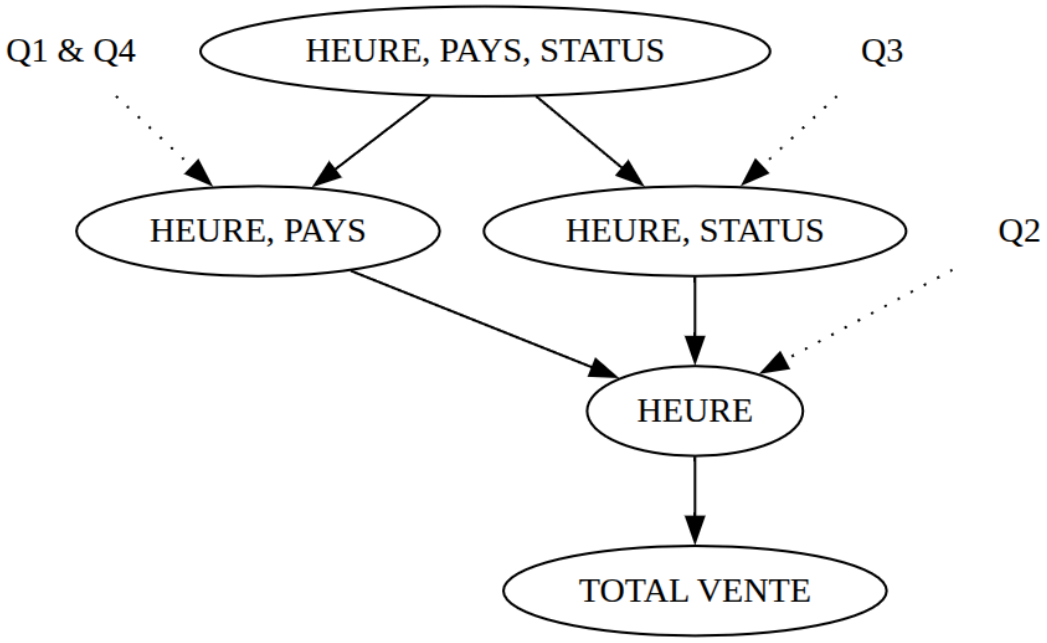
- c.

```SQL
1  SELECT COUNT(*), pays AS total_commandes
2  FROM Ventes
3  JOIN Lieu ON Lieu.id_lieu = U.id_lieu
4  JOIN Utilisateur U ON Ventes.id_utilisateur = U.id_utilisateur
5  WHERE Ventes.heure > 22 AND U.subscription = 'Premium'
6  GROUP BY pays;
```

- d.

```SQL
1  SELECT COUNT(id_vente), heure FROM Ventes
2  JOIN Temps ON Ventes.id_temps = Temps.id_temps
3  JOIN Lieu ON Ventes.id_lieu = U.id_lieu
4  WHERE pays = 'France'
```

VIII.1.2. 2)



VIII.1.3. 3)

```sql
-- Vue pour Q1, Q4 et Q2
CREATE MATERIALIZED VIEW V_HP AS
SELECT
    t.heure,
    c.pays,
    SUM(v.quantite_produit) as total_qte,
    SUM(v.montant) as total_mnt
FROM
    FAIT_VENTES v
    JOIN DIM_TEMPS t ON v.id_temps = t.id_temps
    JOIN DIM_CLIENT c ON v.id_client = c.id_client
GROUP BY
    t.heure, c.pays;

-- Vue pour Q3
CREATE MATERIALIZED VIEW V_HS AS
SELECT
    t.heure,
    c.status
    SUM(v.quantite_produit) as total_qte,
    SUM(v.montant) as total_mnt
FROM
    FAIT_VENTES v
    JOIN DIM_TEMPS t ON v.id_temps = t.id_temps
    JOIN DIM_CLIENT c ON v.id_client = c.id_client
GROUP BY
    t.heure, c.status;
```

VIII.1.4. 4)

- Table de fait Ventes : 1,000,000,000 lignes (1 milliard)
- Dimension :
  - Temps : 24 heures
  - Lieu : 100 pays
- Sans vue : Dans le pire des cas, la requête doit scanner toute la table de fait (1 milliard de lignes).
- Avec vue :
  - On aura pour Q1 : 24 (heures) x 100 (pays) = 2,400 lignes, vu que la vue est pré-agrégée.

IX. Index de type JOIN

IX.1. Exercice 1

IX.1.1. 1)

- Accès Index : Lire l'index bitmap idx pour la valeur 'p500'. On récupère immédiatement une suite de bits (0010010...) indiquant les positions des lignes dans la table de faits.
- Accès Table (Par RowID) : Aller chercher directement les lignes ciblées dans Fact_table_ventes pour récupérer la colonne amount.
- Le coût est très faible. Si la table de faits fait 1 milliard de lignes mais que le produit 'p500' n'a été vendu que 1000 fois.

  On lit l'index + 1000 accès disques. C'est quasi instantané.

IX.1.2. 2)

Le filtre est sur d.price.

L'index bitmap idx ne connait que les product_id. Il ne sait pas quels produits coûtent 50€.

- Scanner la table Dim_table_produit pour trouver tous les IDs des produits dont le prix est 50 (ex: p1, p12, p99).
- Il faut maintenant joindre ces résultats avec la table Fact_table_ventes. On peu pas utiliser le bitmap index car il est sur product_id uniquement.
- Le coût est élevé. Si 10,000 produits coûtent 50€, on doit faire 10,000 accès disques pour retrouver les lignes correspondantes dans la table de faits.

X. Hadoop

X.1. Question ouvertes

1. Pourquoi a-t-on besoin de la parallélisation pour exécuter des traitements sur des données massives ?

   Une seule machine (même puissante) est physiquement limitée en CPU, RAM et vitesse de disque pour traiter des pétaoctets de données. La parallélisation permet de diviser la tâche pour réduire le temps de traitement de plusieurs années à quelques heures ou minutes.

2. Quels sont les avantages des architectures shared-nothing pour l'exploitation des données massives ? Pourquoi les architectures à mémoire partagée ne sont-elles pas adaptées ?

   - Avantages Shared-nothing : Scalabilité linéaire (il suffit d'ajouter des nœuds courants pour augmenter la puissance), tolérance aux pannes (si un nœud tombe, les autres continuent), coût réduit.

- Problème Mémoire partagée : Goulot d'étranglement au niveau du bus mémoire (contention) quand trop de processeurs accèdent à la même RAM, limitant la scalabilité.

3. Indiquez trois différences entre les entrepôts de données et Hadoop en termes d'architecture ou de cas d'usage.

   - Données : DW gère des données structurées (Schema-on-write) ; Hadoop accepte tout (non-structuré, images, logs) (Schema-on-read).
   - Coût : DW repose souvent sur des serveurs propriétaires coûteux ; Hadoop tourne sur du matériel standard (commodity hardware).
   - Usage : DW est optimisé pour l'analyse SQL/BI ; Hadoop pour le traitement batch massif et complexe (ETL, Machine Learning).

4. Illustrer le principe de fonctionnement de HDFS : a. Comment se déroule le stockage d'un fichier dans HDFS ? Supposons de devoir stocker un fichier de 1GB avec facteur de réplication 3 et taille du split de 128MB sur un cluster de 10 machines. Comment ce fichier est géré par HDFS? b. Comment se déroule l'accès à un fichier dans HDFS ?

   a. Stockage (Fichier 1GB, réplication 3, split 128MB, 10 nœuds) :

   1. Le fichier est découpé en 8 blocs (1024 MB / 128 MB = 8).
   2. Chaque bloc est répliqué 3 fois, soit 24 blocs au total.
   3. Le NameNode distribue ces 24 blocs sur les 10 machines (en évitant de mettre deux copies identiques sur le même nœud si possible).

   b. Accès :

   1. Le client demande au NameNode "où est le fichier ?".
   2. Le NameNode renvoie la liste des DataNodes possédant les blocs.
   3. Le client lit directement les données depuis les DataNodes les plus proches.

5. Donnez un exemple de problème : a. Parallélisable avec communication entre nœuds. b. Non parallélisable.

   a. Parallélisable (avec comm) : Le tri de données (Sort) ou l'algorithme PageRank (les nœuds doivent échanger des résultats intermédiaires).

   b. Non parallélisable : Le calcul de la suite de Fibonacci (chaque étape dépend strictement de la précédente) ou le hachage en chaîne.

6. Expliquez comment les fonctions map et reduce permettent de paralléliser les problèmes. Quel est le rôle du map ? Quel est le rôle du reduce ?

   - Map (Diviser & Transformer) : Applique une fonction locale sur chaque bout de donnée indépendamment (ex: filtrer des lignes, extraire des mots). Produit des paires (clé, valeur).
   - Reduce (Agréger) : Reçoit toutes les valeurs associées à une même clé venant de tous les Mappers pour les synthétiser (ex: faire la somme, la moyenne).

8. Map/Reduce dans le cluster Une machine peut exécuter des tâches Map et Reduce (pas nécessairement au même moment, cela dépend des slots/conteneurs disponibles).

   a. Le Module : C'est le Resource Manager (YARN) ou le JobTracker (dans les vieilles versions) qui assigne les tâches.

   b. Critères : Le critère principal est la Localité des données. On envoie le code (la tâche Map) sur la machine qui possède déjà les données sur son disque dur pour éviter de saturer le réseau.

9. Pratique de Map/Reduce : a. Résolvez des problèmes classiques en Map/Reduce (comme ceux vus en travaux pratiques).

   Problème : WordCount (Compter les mots).

   - Map : Lit une ligne, découpe les mots, émet (mot, 1) pour chaque mot.
   - Shuffle (Auto) : Regroupe par clé : (chat, [1, 1, 1]).
   - Reduce : Somme la liste : ém - Utiliser Spark (qui garde les données en mémoire RAM entre les étapes).
   - Utiliser un Combiner (un "mini-reduce" local après le Map) pour réduire la quantité de données envoyées sur le réseau vers le Reducer.et (chat, 3).

10. Quelles sont les limites principales de l'approche Map/Reduce du point de vue de la consommation des ressources dans un cluster ? Comment peut-on atténuer ces problématiques ?

    - Limites : Très intensif en E/S Disque (Hadoop écrit sur le disque après le Map et après le Reduce), ce qui crée de la latence. Inadapté aux algorithmes itératifs (Machine Learning).
    - Atténuation :
      - Utiliser Spark (qui garde les données en mémoire RAM entre les étapes).
      - Utiliser un Combiner (un "mini-reduce" local après le Map) pour réduire la quantité de données envoyées sur le réseau vers le Reducer.

X.2. Exercice 2 programmation map reduce (taxi)

Pour obtenir la moyenne par jour pour chaque mois, la logique mathématique est la suivante : $\text{Moyenne} = \frac{\text{Total des passagers sur le mois}}{\text{Nombre de jours dans ce mois}}$

- Map : Grouper les données par mois.
- Reduce : Additionner tous les passagers du mois, puis diviser par le nombre de jours calendaires de ce mois (30, 31 ou 28).

X.2.1. Role du Map :

Le rôle du Map est d'extraire la clé de regroupement (le Mois) et la valeur utile pour le calcul (le Nombre de passagers).

Entrée (Input) :

- Clé : Offset de la ligne (généralement ignoré).
- Valeur : Une ligne du fichier CSV (ex: “2021-03-01_00:21:05, 2021..., 3, 5.8”).

Traitement :

- Découper la ligne (split) par la virgule pour accéder aux champs.
- Extraire la date de début (champ 0).
- Transformer la date pour ne garder que l’année et le mois (ex: de 2021-03-01... vers 2021-03). Ce sera notre Clé.
- Extraire le nombre de passagers (champ 2). Ce sera notre Valeur.

Sortie (Output) :

- Paire (Clé, Valeur) -> (“2021-03”, 3)

X.2.2. Role du Reduce :

Le framework Hadoop va regrouper (Shuffle) toutes les valeurs pour une même clé. Le Reducer reçoit donc un mois et une liste de tous les nombres de passagers enregistrés ce mois-là.

Entrée (Input) :

- Clé : Le mois (ex: “2021-03”).
- Valeurs : Liste des passagers pour toutes les courses du mois (ex: [3, 4, 1, 2, ...]).

Traitement :

- Initialiser une somme à 0.
- Parcourir la liste des valeurs et les additionner pour obtenir le total mensuel des passagers.
- Déterminer le nombre de jours dans le mois concerné (Jan=31, Fév=28 car 2021 n’est pas bissextile, Mars=31, etc.).
- Calculer la moyenne : Total Passagers / Jours dans le mois.

Sortie (Output) :

- Paire (Clé, Valeur) -> (“2021-03”, 1540.5)

X.3. Exercice 3 join sur plusieurs tables

SELECT S.B FROM S, R, T WHERE R.A = S.A AND R.B = T.A

X.3.1. Stratégie Globale

- Job 1 : Réaliser la jointure entre S et R sur la clé commune A.
- Job 2 : Réaliser la jointure entre le Résultat du Job 1 et T sur la clé R.B = T.A.

X.3.2. JOB 1 : Jointure S et R (Condition S.A = R.A)

L’objectif est de trouver les correspondances entre S et R, et de préparer les données pour la prochaine jointure (qui se fera sur la colonne B de R).

X.3.2.1. Fonction MAP (Job 1)

Le Map lit les fichiers S.csv et R.csv. Il doit utiliser la colonne A comme clé de jointure. Il doit aussi "taguer" la provenance (S ou R) pour que le Reducer s'y retrouve.

Entrée :

- Ligne de S ou ligne de R.

Traitement :

- Si la source est S : Lire colonnes (A, B). Émettre Clé = A, Valeur = ('S', B).
- Si la source est R : Lire colonnes (A, B). Émettre Clé = A, Valeur = ('R', B).

Sortie :

- Paires (A, ('S', S.B)) ou (A, ('R', R.B)).

X.3.2.2. Fonction REDUCE (Job 1)

Le Reducer reçoit toutes les données pour une clé A donnée. Il effectue le produit cartésien entre les lignes venant de S et celles venant de R.

Entrée :

- Clé A, Liste de valeurs [('S', 2), ('R', 2), ('S', 4)...].

Traitement :

- Séparer les valeurs dans deux listes : liste_S et liste_R.
- Faire une double boucle (boucle imbriquée) : Pour chaque élément val_S dans liste_S et chaque val_R dans liste_R.
- Point Crucial : On doit préparer la clé pour le Job 2. La prochaine jointure se fait sur R.B (qui est notre val_R).

Sortie :

- Clé = val_R (c'est-à-dire R.B), Valeur = val_S (c'est-à-dire S.B).

X.3.3. JOB 2 : Jointure Résultat 1 et T (Condition R.B = T.A)

Maintenant, on joint la sortie précédente avec la table T. La clé de jointure est désormais la valeur qui correspondait à R.B et T.A.

X.3.3.1. Fonction MAP (Job 2)

Lit la sortie du Job 1 et le fichier T.csv.

Entrée :

- Sortie Job 1 ou ligne de T.

Traitement :

- Si source Job 1 : L'entrée est déjà sous la forme R.B (clé) et S.B (valeur). On émet Clé = R.B, Valeur = ('Prev', S.B).
- Si source T : Lire (A, B). Ici la clé de jointure est T.A. On émet Clé = T.A, Valeur = ('T', 'exists') (on n'a pas besoin de T.B pour le SELECT final, juste de savoir que la ligne existe).

Sortie :

- Paires regroupées par la clé commune (valeur de R.B / T.A).

X.3.3.2. Fonction REDUCE (Job 2)

Le Reducer vérifie si une correspondance existe.

Entrée :

- Clé (valeur de jointure), Liste de valeurs [('Prev', S.B_val1), ('T', 'exists')...].

Traitement :

- Vérifier si la liste contient au moins une donnée venant de T.
- Si oui, pour chaque donnée venant de Prev (qui contient S.B), on l'émet.

Sortie Finale :

- S.B (le résultat demandé par la requête SELECT S.B).