COMPTE RENDU SQL : CREATION D'UNE BASE DE DONNEES

Entreprise
Boulangerie PAUL

MAUHIN Théo

GONZALEZ PAULO Rémy

Sommaire

TABLE DES MATIÈRES

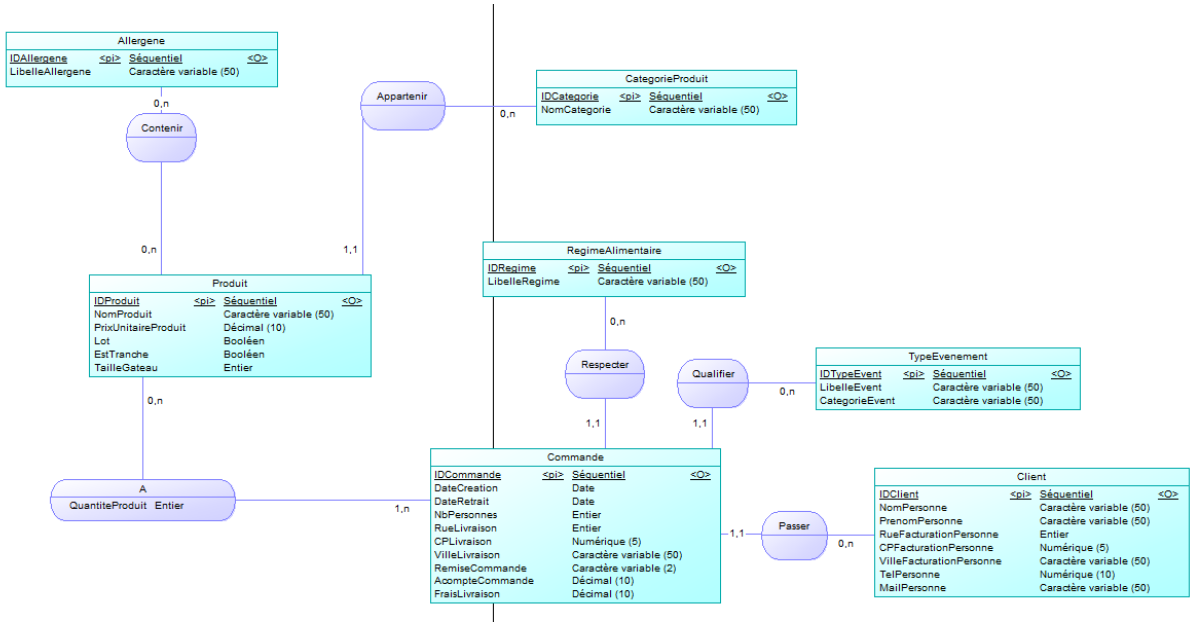
INTRODUCTION

Ce rapport a été écris en tant que contre rendu détaillant toutes les étapes de création d'une base de données fonctionnelle pour l'entreprise de boulangeries PAUL. Cela contient notamment la création du MCD, de la base de données, l'ajout des données nécessaires, les différentes requêtes, l'explication des statistiques demandées ainsi que l'explication de la création du rapport PowerBI.

1. CRÉATION DU MCD

La chaîne industrielle de boulangerie PAUL a mandaté l'entreprise pour la modernisation de son système de gestion.

Le MCD suivant est la version finale après plusieurs essais et modifications, nous permettant une modélisation optimale :



1.1. Référentiel Allergènes

Référentiel pour contenir les allergènes, contenus dans les produits. Elle est séparée des produits car un produit peut contenir plusieurs allergènes et également pour la simplicité de changer les noms des allergènes.

1.2. Table Produit

Table qui contient les produits et les spécificités comme si le produit est tranché (pain) ou bien pour les gâteaux la taille des tranches de ceux-ci. Ces deux `booléens` ont été implémentés à la suite d'un retour oral.

1.3. Référentiel Catégorie Produit

Ce référentiel contient les noms des différentes catégories de produits, elle est séparée de produits pour une modification des noms simplifiée.

1.4. Relation A

La relation A contient la propriété calculée de la quantité de chaque produit par commande.

1.5. Table Commande

La table `Commande` contient beaucoup d'informations concernant le client, l'évènement et la commande en elle-même. Il y a également les frais de livraison, l'acompte et le pourcentage de remise pris en compte.

1.6. Table Client

Cette table a pour objectif de contenir les informations classiques d'un client, tel que `son adresse de facturation, nom, prénom, téléphone, mail`. En raison des règles RGPD chaque client a le droit de demander la suppression de ces informations quand il le désire. Le rajout de la facturation a été réalisé à la suite d'une discussion orale.

1.7. Référentiel Régime Alimentaire

Ce référentiel a pour but de contenir les noms des différents régimes alimentaires présent dans un évènement, étant donné qu'il peut en avoir plusieurs et pour une facilité de changement des noms des libellés ceux-ci sont donc dans un référentiel au lieu dans la table `Commande`.

1.8. Référentiel Type évènement

Ce référentiel, comme les autres a un but de simplification pour le changement des noms. Il est lié à la table `Commande`, et non à l'intérieur pour la raison cité au-dessus.

2. CRÉATION DE LA BASE DE DONNÉES

Nous avons donc à partir de ce MCD générer la base de données, pour un souci de fluidité le script a été modifié de plusieurs façons.

2.1. Le rajout des contraintes

Pour des mesures de sécurité les tables ont subi des modifications avec des rajouts de contraintes qui sont les suivantes :

- Contraintes `UNIQUE` :
  - Contrainte sur le téléphone du client
  - Contrainte sur le mail du client
- Contrainte `CHECK` :
  - Contrainte sur le mail, pas d'espace

2.2. Modifications demandées

Des modifications nous ont été demandées qui sont les suivantes.

2.2.1 Incrémentation des ID

Nous n'avons pas eu de modification à faire mais si nous avions dû y faire nous aurions fait une séquence pour chaque `ID` et aurait utilisé `nextval`.

2.2.2 La baisse de la remise

On a rajouté une contrainte `CHECK` avec la remise commande qui devait se situer entre 0 et 30 pour limiter la remise à 30% maximum.

2.2.3 Score de comptabilité

Nous avons rajouté une table « Recommander » qui est une table d'association avec la référence du produit, du régime alimentaire.

Une contrainte `CHECK` pour limiter le score entre 0 et 100.

Le rajout des deux clés étrangères.

2.2.4 Suppression des lots et du pain tranché

Pour réaliser cela nous avons drop les colonnes de la table produit liées au pain tranché et au lots, et nous avons rajouté une colonne de description et une colonne de disponibilité dans la table produit pour cela.

2.3. Insertions dans la base de données

2.3.1 Les tables de références

Nous avons inséré dans les tables de références les différentes informations de la partie 1 du sujet ainsi que des informations supplémentaires.

2.3.2 Les adresses

Nous avons récupéré les adresses du département qui nous a été attribué, le département 24. Pour cela nous avons récupéré le .CSV, le reste a été généré par GenerateData.

Nous avons généré 500 clients pour avoir des clients avec des commandes et d'autres sans tout en ayant suffisamment de villes différentes pour le PowerBI. Une erreur s'était glissée dans l'importation du .CSV qui retirait tous les accents des villes, ce problème a été découvert lors de la création du PowerBI et a nécessité la modification du csv en question, la recréation des insert Clients ainsi que refaire la vue en question.

2.3.3 Les produits

Les produits ont été inséré, lié à leurs catégories avec une sous requête.

2.3.4 Les évènements

Ils ont été générés de manière aléatoire entre 2020 et aujourd'hui via un cross join et du random.

2.3.5 Les commandes

On a fait la création de 50 commandes en utilisant l'aléatoire sur les produits, le nombre de produits, le nombres de personne, le type d'évènement.

Nous avons eu un problème sur le type d'évènement qui prenait toujours le même ID donc nous avons réglé ça en modifiant la façon dont le random fonctionne pour qu'il génère un nombre aléatoire à chaque itération au lieu d'un seul pour tout.

2.3.6 Tables métiers, associations

Pour les tables d'associations nous avons réalisé plusieurs choses, la table allergènes subit un produit cartésien via un cross join pour prendre chaque ligne de la classe produit pour l'associer à chaque ligne de la classe

allergènes, et on a fait un random pour garder au maximum 30% des combinaisons possible pour de l'aléatoire.

Pour la recommandation nous avons inséré des scores réalistes et nuancés, pour cela nous avons appliqué une loi normale via la méthode de Box-Muller[1].

Cette méthode va transformer le hasard uniforme du random en une loi normale avec l'utilisation d'un écart type. Pour réaliser cela la requête réalise trois profils de produits, les produits peu chers à 5€ ou moins, les produits de milieu de gamme entre 5 et 20€ et enfin les produits hauts de gamme au-delà de 20€. Elle est implémentée par cette ligne :

```
sqrt(-2 * ln(random()+0.0001))
```

La loi normale peut en théorie inséré des nombres de -∞ à +∞ on utilise donc des filets de sécurité qui sont LEAST et GREATEST, GREATEST formate si le score est négatif ou nul il passe à 1 et LEAST si est supérieur à 100 donne 100.

Nous nous sommes aussi assurés que le résultat serait un entier via un CAST AS INT.

Enfin un filtre d'exclusion a été implémenté avec un produit cartésien entre les produits et les régimes et qui élimine environ 14% des combinaisons via un modulo 7 pour simuler que certains produits ne sont pas du tout adaptés à certains régimes et ils sont donc non notés au lieu de la note de 1.

Cette requête a pour but de générer les lignes de commande réalistes, et qui sont en accord avec les contraintes données.

Tout d'abord on sélectionne la quantité via un RANDOM, pour chaque ligne on génère une quantité aléatoire de produits entre 0 et 100. Le FLOOR arrondi à l'entier inférieur et ajoute 1, et le ::INT convertir en INT.

Le produit cartésien génère toutes les combinaisons entre les commandes et les produits.

Les contraintes elles sont gérés de deux manière via un OR dans le WHERE, tout d'abord la commande avec l'ID MIN permet d'ignorer le CROSS JOIN, cette commande possède donc 100% du catalogue. L'autre condition est un random qui conservera uniquement 25% des produits disponibles dans les commandes.

[1] Utilisation de l'IA Gemini pour l'utilisation et l'implémentation de cette formule

3. REQUÊTAGE DES DONNÉES

Une fois les insertions terminées, nous avons procédé à la réalisation des différentes requêtes demandées dans la consigne.

3.1. Les requêtes simples

3.1.1 Requête avec jointure

Pour cette première requête, la consigne était d'effectuer une jointure. Pour se faire, on a réalisé une requête permettant de lister les noms des clients ainsi que les dates de leurs commandes. Cette requête est la suivante :

```
SELECT c.NOMPERSONNE, co.DATECREATION
FROM CLIENT c
JOIN COMMANDE co ON c.IDCLIENT = co.IDCLIENT;
```

La première ligne sert à sélectionner les éléments NOMPERSONNE et DATECREATION dans leur table respective (C et CO, qui sont des raccourcis de client et commande). Ensuite, avec le from on identifie la table source (la table client, qui est ici raccourcis en c avec un alias), puis enfin on vient faire la jointure entre la table commande et la table client sur l'id du client (IDCLIENT).

Voici ci-dessous le retour de cette requête :

	nompersonne character varying (50)	datecreation date
1	Odysseus	2026-03-17
2	Ali	2026-03-25
3	Octavius	2026-03-12
4	Adam	2026-03-11
5	Camilla	2026-03-10

Total rows: 50 | Query complete 00:00:00.037

3.1.2 Requête filtrante

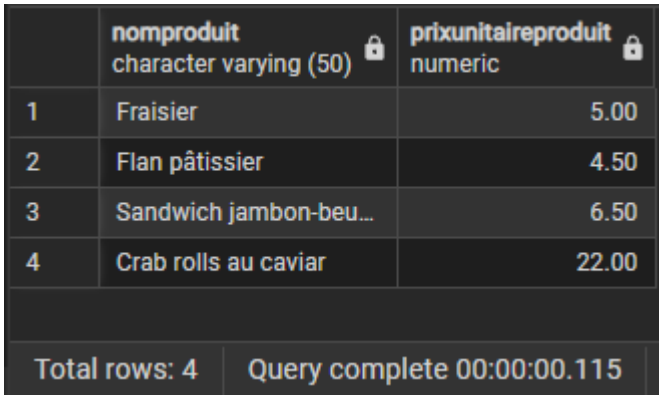
Pour cette seconde requête, il nous fallait faire une requête filtrante. Pour se faire, on a réalisé une requête permettant de sélectionner les produits dont le prix est strictement supérieur à 3€. Cette requête est la suivante :

```
SELECT NOMPRODUIT, PRIXUNITAIREPRODUIT
```

```
FROM PRODUIT

WHERE PRIXUNITAIREPRODUIT > 3;
```

La première ligne sert toujours à sélectionner les éléments, ici le nom du produit et son prix unitaire. La seconde ligne sert toujours à identifier la table à la source de cette sélection, donc ici la table produit. Et enfin, la troisième ligne sert à appliquer un filtre sur la valeur de PRIXUNITAIREPRODUIT, afin d'afficher uniquement ceux avec un prix supérieur à 3.

Voici ci-dessous le retour de la requête :



	nomproduit character varying (50)	prixunitaireproduit numeric
1	Fraisier	5.00
2	Flan pâtissier	4.50
3	Sandwich jambon-beu…	6.50
4	Crab rolls au caviar	22.00

Total rows: 4 | Query complete 00:00:00.115

3.1.3 Requête d'ensemble

Pour cette requête, il nous fallait faire une requête d'ensemble. Pour se faire, on a réalisé une requête permettant de sélectionner les rues de facturation et rue de livraison en une seule colonne nommée « Rue », et les villes de livraison ainsi que les villes de facturation en une seule colonne nommée « Ville ». Cette requête est la suivante :

```
SELECT RUEFACTURATIONPERSONNE AS RUE, VILLEFACTURATIONPERSONNE
AS VILLE

FROM CLIENT

UNION

SELECT RUELIVRAISON, VILLELIVRAISON

FROM COMMANDE;
```

La première ligne sert toujours à sélectionner les éléments, mais ici on va renommer la colonne (avec le AS), les deux colonnes affichées seront donc Rue et Ville, et non pas RUEFACTURATIONPERSONNE et VILLEFACTURATIONPERSONNE. La seconde ligne sert à identifier la table à la source de cette sélection, puis la troisième ligne sert quant à elle à dire d'ajouter à ce résultat la requête qui suit. Cette seconde requête se situe aux lignes 4 et 5, il s'agit de la sélection de RUELIVRAISON et VILLELIVRAISON de la table commande.

Voici ci-dessous le retour de cette requête :

	rue character varying (50)	ville character varying (50)
1	Route de l'Oc an	Siorac-de-Rib rac
2	Chemin du Bureau de la Mine	Veyrines-de-Domme
3	Route du Pignier des Rhodes	Sarlat-la-Can da
4	Rue des Ecoles	Tr lissac
5	Avenue de la Double	La Roche-Chalais

Total rows: 464 | Query complete 00:00:00.040

3.1.4 Requête d'exception

Cette fois-ci, il nous fallait faire une requête d'exception. Pour se faire, on a réalisé une requête permettant de sélectionner uniquement les clients qui n'ont jamais passé de commande (autrement dit, tous les clients sauf ceux qui ont déjà passé des commandes). Cette requête est la suivante :

```
SELECT NOMPERSONNE FROM CLIENT

EXCEPT

SELECT c.NOMPERSONNE FROM CLIENT c JOIN COMMANDE co ON
c.IDCLIENT = co.IDCLIENT;
```

La première ligne sert, comme d'habitude, à identifier ce que l'on sélectionne. La seconde sert à indiquer que dans les lignes sélectionner, on veut retirer celles qui correspondent à la requête qui va suivre, puis la troisième est la requête en question, dans laquelle on sélectionne le nom du client de la table client, puis enfin on fait une jointure avec la table commande sur l'id du client.

Voici le retour de cette requête :

	nompersonne character varying (50)
1	Hollee
2	Edward
3	Jackson
4	Avram
5	Olga

Total rows: 353 | Query complete 00:00:00.040

3.1.5 Requête d'agrégat

Cette fois-ci, il nous fallait faire une requête d'agrégat. Pour se faire, on a réalisé une requête permettant de compter le nombre de produit par catégorie. Cette requête est la suivante :

```
SELECT NOMCATEGORIE, COUNT(*) AS TOTAL_PRODUITS
FROM PRODUIT P
JOIN CATEGORIEPRODUIT C ON P.IDCATEGORIE=C.IDCATEGORIE
GROUP BY NOMCATEGORIE;
```

La première ligne sert à sélectionner le nom de la catégorie, mais également à sélectionner le nombre d'élément dans cette catégorie, grâce à la fonction count(). La seconde ligne sert à identifier la table source, puis la troisième est une jointure sur l'ID de la catégorie. Enfin, le résultat est trié par catégorie.

Voici le retour de la requête :

	nomcategorie character varying (50)	total_produits bigint
1	Snacking	1
2	Traiteur	2
3	Viennoiseries	2
4	Pains	2
5	Pâtisseries	2

Total rows: 5 | Query complete 00:00:00.042

3.1.6 Requête avec fonction

Cette fois-ci, il nous fallait des requêtes comprenant des fonctions (une scalaire, une conditionnelle, une Horodatée, et un transtypage). Toutes ces conditions ont été respecté en une seule requête, qui est la suivante :

```
SELECT
    UPPER(NOMPRODUIT) AS NOM_MAJ, -- Scalaire
    CASE WHEN PRIXUNITAIREPRODUIT < 2 THEN 'Eco' ELSE 'Premium'
END AS GAMME, -- Conditionnelle
    CURRENT_DATE AS DATE_CONSULTATION, -- Horodatée
    PRIXUNITAIREPRODUIT::VARCHAR || ' EUR' AS PRIX_TEXTE --
Transtypage (Cast)
FROM PRODUIT;
```

Cette requête a pour but de faire diverses sélections répondant chacune d'entre elle a une fonction. La deuxième ligne est un scalaire

permettant de mettre en majuscule une sélection. La troisième ligne est une fonction conditionnelle. La case accompagnée du when signifie le début de la condition, après il y a la condition (ici c'est PRIXUNITAIREPRODUIT < 2). Then exprime lui la conséquence si la condition est validée, et else exprime la conséquence dans le cas où elle n'est pas validée. La quatrième ligne est un appel de la fonction CURRENT_DATE qui renvoie la date actuelle. Enfin, la cinquième ligne est un transtypage qui récupère la valeur des prix unitaires des produits, le converti en VARCHAR (avec la fonction ::VARCHAR), puis le concatène avec l'opérateur || suivi de la seconde chaine de caractères.

Voici ce que cette requête retourne :

	nom_maj text	gamme text	date_consultation date	prix_texte text
1	BAGUETTE TRADITION	Eco	2026-06-07	1.35 EUR
2	PAIN AU MAÏS	Premium	2026-06-07	2.20 EUR
3	CROISSANT	Eco	2026-06-07	1.40 EUR
4	PAIN AU CHOCOLAT	Eco	2026-06-07	1.60 EUR
5	FRAISIER	Premium	2026-06-07	5.00 EUR

Total rows: 9 | Query complete 00:00:00.042

3.1.7 Requête imbriquée non synchronisée

Ici, il nous fallait faire une requête imbriquée non synchronisée. Pour se faire, on a réalisé une requête permettant de sélectionner tous les produits de la catégorie pain. La requête est la suivante :

```
SELECT NOMPRODUIT

FROM PRODUIT

WHERE IDCATEGORIE = (SELECT IDCATEGORIE FROM CATEGORIEPRODUIT
WHERE NOMCATEGORIE = 'Pains');
```

La première ligne sert donc à sélectionner le nom du produit, la seconde à identifier la table source, puis la troisième sert à renvoyer uniquement les produits pour lesquels l'id de catégorie est égal à l'id de la catégorie des pains (via la sous requête).

Voici ce que cela retourne :

	nomproduit character varying (50)
1	Baguette tradition
2	Pain au maïs

Total rows: 2 | Query complete 00:00:00.051

3.2. Les requêtes complexes

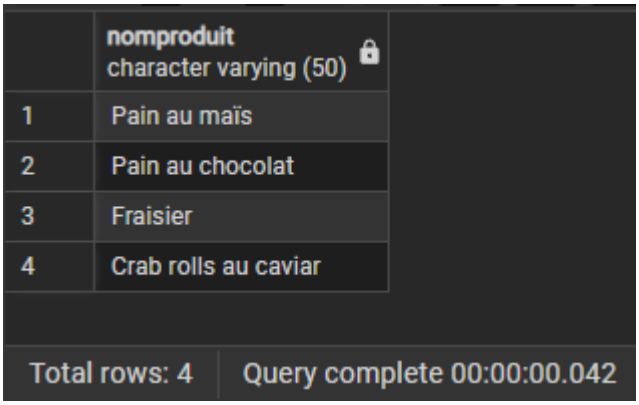
3.2.1 Requête imbriquée synchronisée

Pour cette requête, il nous fallait faire une requête imbriquée synchronisée. Pour se faire, on a réalisé une requête permettant de trouver les produits plus chers que la moyenne de leur propre catégorie. La requête est la suivante :

```
SELECT p1.NOMPRODUIT
FROM PRODUIT p1
WHERE p1.PRIXUNITAIREPRODUIT > (
    SELECT AVG(p2.PRIXUNITAIREPRODUIT)
    FROM PRODUIT p2
    WHERE p2.IDCATEGORIE = p1.IDCATEGORIE
);
```

Ici, la première ligne sert comme habituellement à sélectionner ce que l'on souhaite, ici le nom du produit, et la seconde ligne sert à identifier la table source (ici Produit). La troisième ligne sert à appliquer une condition à la sélection, pour qu'une ligne soit retourné il faut que le prix unitaire de ce produit soit supérieur au résultat de la sous requête qui retourne le prix moyen (fonction AVG()) des produits dans la catégorie de ce produit (Where p2.idcategorie = p1.idcategorie).

Cette requête retourne le résultat suivant :



	nomproduit character varying (50)
1	Pain au maïs
2	Pain au chocolat
3	Fraisier
4	Crab rolls au caviar

Total rows: 4 | Query complete 00:00:00.042

3.2.2 Division

Pour réaliser cette requête nous devions utiliser une division, nous avons donc choisi d'afficher les clients ayant une commande qui contient tous les produits du catalogue. La requête est la suivante :

```
SELECT c.IDCLIENT, cl.NOMPERSONNE
FROM COMMANDE c
JOIN CLIENT cl ON c.IDCLIENT = cl.IDCLIENT
WHERE NOT EXISTS (
    SELECT p.IDPRODUIT FROM PRODUIT p
    EXCEPT
    SELECT a.IDPRODUIT FROM A a WHERE a.IDCOMMANDE =
    c.IDCOMMANDE
);
```

On va sélectionner l'id du client et son nom pour l'affichage à partir de la table commande avec une jointure sur la table Client. On applique ensuite un `WHERE NOT EXISTS` pour effectuer le filtrage via une sous requête.

Dans la sous requête on `SELECT` tout les `ID` produits mais on utilise `EXCEPT` qui va faire office de soustraction entre le `SELECT` du haut et celui en dessous. Le `SELECT` en dessous lui récupère la liste des produits dans la commande en test.

Pour finir cela revient à faire Tout les produits existants – les produits de la commande X. SI le résultat est égal à 0 alors cela veut dire que la commande en question possède tous les produits du catalogue, et on regarde donc le client lié à cette commande.

Le résultat de la requête est la suivante :

	idclient integer	nompersonne character varying (50)
1	1	Odysseus
2	2	Ali
3	12	Destiny
4	14	Ray
5	23	Cullen
6	24	Rowan
7	28	Gary
8	34	Colt
9	39	George
10	41	Yoshio
11	44	Barclay
12	49	Uriel

Total rows: 12 | Query complete 00:00:00.047

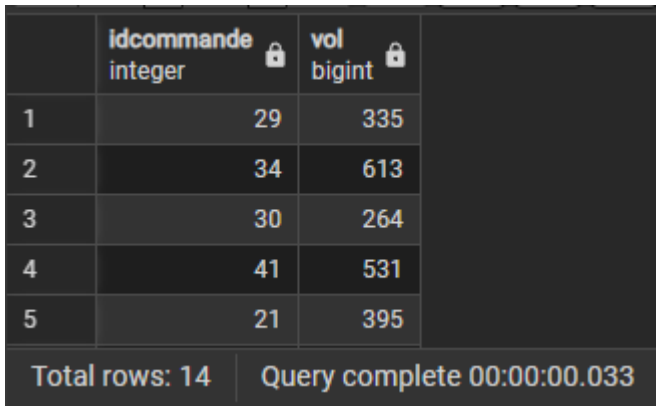
3.2.3 Table temporaire

Pour cette requête, il nous fallait faire une table temporaire. Pour se faire, on a réalisé une requête permettant de stocker dans une table temporaire le volume total de produit vendu par commande. La requête est la suivante :

```
DROP TABLE IF EXISTS TEMP_VOLUME cascade;

CREATE TEMP TABLE TEMP_VOLUME AS

SELECT IDCOMMANDE, SUM(QUANTITEPRODUIT) AS VOL

FROM A GROUP BY IDCOMMANDE;

SELECT * FROM TEMP_VOLUME;
```

Ici, la première ligne supprime la table temporaire (TEMP_VOLUME) dans le cas où elle existe afin d'éviter les erreurs. Ensuite, la seconde ligne sert à créer la table (Create temp table), puis à la définir avec le mot clef « AS ». Les deux lignes suivantes sont la définition de la table, qui est donc une requête sélectionnant l'id des commandes, et la somme de leur produit (avec la fonction SUM()) à partir de la table « A », puis le résultat est regroupé par id de commande. Enfin, la dernière ligne sert à afficher ce que contient cette nouvelle table.

Le résultat de la requête est le suivant :



	idcommande integer	vol bigint
1	29	335
2	34	613
3	30	264
4	41	531
5	21	395

Total rows: 14 | Query complete 00:00:00.033

3.2.4 Vue

Pour cette requête, il nous fallait faire une vue. Pour se faire, on a réalisé une requête permettant de créer une vue simplifiée du détail des commandes (en ne visualisant que l'id de la commande, le nom du client, le nom du produit acheté, et la quantité acheté de ce produit dans cette commande, et ce pour chaque produit de chaque commande). La requête est donc la suivante :

```
CREATE OR REPLACE VIEW V_SYNTHESE_COMMANDES AS

SELECT co.IDCOMMANDE, cl.NOMPERSONNE, p.NOMPRODUIT,
a.QUANTITEPRODUIT

FROM COMMANDE co

JOIN CLIENT cl ON co.IDCLIENT = cl.IDCLIENT

JOIN A a ON co.IDCOMMANDE = a.IDCOMMANDE

JOIN PRODUIT p ON a.IDPRODUIT = p.IDPRODUIT;
```

La première ligne de cette requête permet de créer la vue, ou si elle existe déjà, de la remplacer par la requête qui suit cette ligne. La seconde ligne est la sélection de ce que l'on souhaite mettre dans la vue comme valeurs, et la troisième ligne est la table source de cette sélection. À cela s'ajoute trois jointures : Une première entre la table client et commande sur l'id du client, une seconde entre la table « A » et la table commande sur l'id de la commande, puis la troisième est entre la table produit et la table « A » sur l'id du produit.

Voici ci-dessous un exemple de ce que la vue retourne pour une sélection de la commande dont l'id est 1 (`select * from V_SYNTHESE_COMMANDES where idcommande = 1`) :

	idcommande integer	nompersonne character varying (50)	nomproduit character varying (50)	quantiteproduit integer
1	1	Odysseus	Baguette tradition	22
2	1	Odysseus	Pain au maïs	6
3	1	Odysseus	Croissant	70
4	1	Odysseus	Pain au chocolat	99
5	1	Odysseus	Fraisier	76

Total rows: 9 Query complete 00:00:00.038

3.2.5 Table dérivée

Pour cette requête, il nous fallait faire une table dérivée. Pour se faire, on a réalisé une requête permettant de calculer la moyenne des quantités commandées à partir d'un résultat intermédiaire. La requête est la suivante :

```
SELECT AVG(TOTAL_QTE)
FROM (SELECT SUM(QUANTITEPRODUIT) AS TOTAL_QTE FROM A GROUP BY IDCOMMANDE) AS TABLE_DERIVEE;
```

La première ligne de cette requête sert à sélectionner la moyenne de TOTAL_QTE, qui est défini dans la seconde ligne. Dans la seconde ligne, le from indique la table source, et dans ce from figure une sous requête qui sélectionne la somme de QUANTITEPRODUIT figurant dans la table « A ». Ce résultat sera renommé TOTAL_QTE et sera groupé par id de commande. En l'occurrence le fait de renommer la table (as TABLE_DERIVEE) n'est pas réellement utile dans ce cas-ci, car après exécution la table est supprimée.

Voici ci-dessous le retour de cette requête :

	avg numeric
1	435.3750000000000000

3.3. Insertions et modifications

3.3.1 Transactions métiers

Ici il était demandé de proposer des transactions pour insérer des lignes dans la BD avec une nouvelle ligne dans une table de référence puis appeler cette ligne dans une table métier. Le script est le suivant :

```
BEGIN;

INSERT INTO CATEGORIEPRODUIT (NOMCATEGORIE) VALUES ('Snacking');

INSERT INTO PRODUIT (IDCATEGORIE, NOMPRODUIT, PRIXUNITAIREPRODUIT)

VALUES(currval('categorieproduit_idcategorie_seq'),'Sandwich Jambon', 4.50);

COMMIT;
```

On utilise ici la table de référence catégorie produits pour insérer les snackings, on utilise ensuite la table métier produit pour insérer un nouveau produit : le sandwich jambon. Pour cela on utilise le CURRVAL, cette fonction extrait la valeur courante de la séquence d'auto incrémentation de l'id de catégorie, donc le dernier utilisé, donc celui des snackings, liant le produit à la bonne catégorie.

3.3.2 Erreur d'insertion

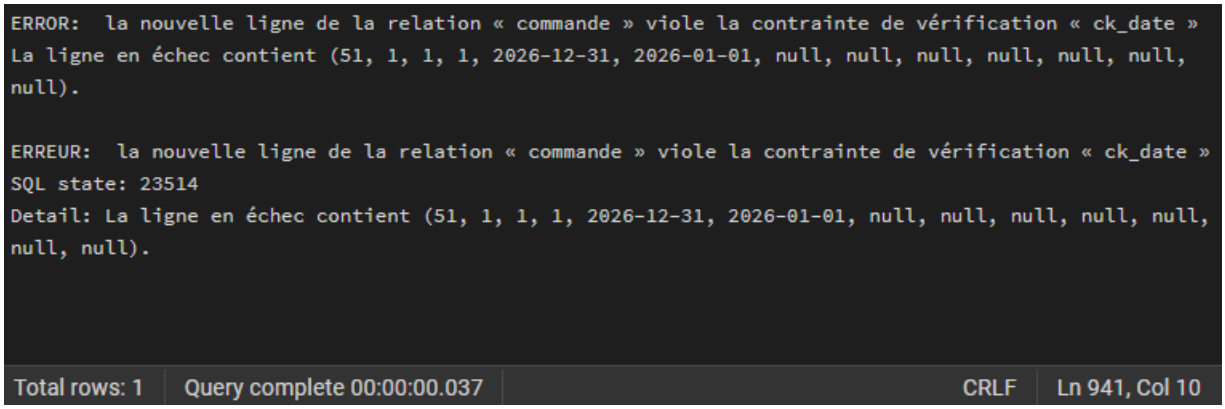
Ici, il était demandé de commettre volontairement une erreur d'insertion. Pour ce faire, on a fait une requête violant la contrainte check sur la temporalité entre la date de retrait et celle de création dans la table commande. La requête est la suivante :

```
BEGIN;

INSERT INTO COMMANDE (IDCLIENT, IDEVENEMENT, IDREGIME, DATECREATION, DATERETRAIT)

VALUES (1, 1, 1, '2026-12-31', '2026-01-01'); -- Doit échouer

ROLLBACK;
```

Ici, la première ligne sert à mettre un point de contrôle : en cas de rollback, l'état de la base de données reviendra à ce point, avant l'exécution des lignes qui suivent, puis cette fois-ci ignorera la séquence du begin au rollback. Ensuite, la seconde ligne est la déclaration d'une insertion de données dans la table commande. Dans celle-ci, on spécifie que les données insérées seront : un id client, un id événement, un id de régime, une date de création et enfin, une date de retrait. À la suite de cela, dans la ligne

trois, on va entrer les valeurs dans l'ordre cité dans la ligne précédente. Cette requête échoue car la date de création est antérieure à la date de retrait, ce qui est impossible, donc un check a été mis pour éviter cela.

Voici donc le retour de cette requête :



4. STATISTIQUES

La partie statistique est contenu dans le fichier Excel suivant : « Données+Stats.xlsx »

Conformément à la consigne tous les réponses et les résultats se trouvent à l'intérieur.

Pour réaliser ce fichier Excel nous avons créé une vue des statistiques qui contient les informations suivantes :

- La date de création des commandes
- La date de retrait des commandes
- On réalise ici un calcul directement dans la requête car cela nous servira et dans le Excel et le PowerBI qui est le délai de préparation des commandes.
- Le nom de la personne
- Sa ville de facturation
- Le libellé de l'évènement
- Le nom de la catégorie de l'événement
- Le nom des produits.
- Un autre calcul qui réalise le chiffre d'affaires immédiatement en prenant le prix unitaire produit multiplié par sa quantité.

5. POWERBI

Le rapport PowerBI est présent conformément aux consignes sous la forme d'un fichier .pbix nommé « Tableau de bord exécutif.pbix ».

Nous avons réutilisé la vue utilisée pour les statistiques. Dans le PowerBI celui contient trois pages.

5.1. Page d'accueil

La page d'accueil qui répertorie les données clés comme le nombre de commandes distinctes, le chiffre d'affaires moyen par panier ainsi que le chiffre d'affaires total. Attention, les données du chiffre d'affaires moyen par panier et le total sont étranges, une vérification n'a pas pu avoir lieu, c'est un point d'amélioration.

5.2. Statistiques et performances

Cette page répertorie les informations chiffrées. Les données peuvent être filtrées par catégorie ou type d'évènements, un ajout d'un tri par année est envisageable.

5.2.1 Nuage de points

Le nuage de points affiche la moyenne du délai de préparation et la moyenne du chiffre d'affaires par numéro de commande. Cela permet de voir le ratio de temps passé sur la commande par rapport à l'argent rapporté par celle-ci.

5.2.2 Graphique en anneau

Le graphique en anneau lui représente le chiffre d'affaires total par catégorie de produits.

Cela permet d'avoir une représentation visuelle de quelle catégorie est la plus prolifique.

5.2.3 Matrice des données

La matrice permet de voir la somme du chiffre d'affaires par catégorie de produits dans chaque évènement. Cela complète le graphique en anneau et permet donc d'avoir une vue globale sur quel type d'évènements et le plus prolifique pour chaque catégorie.

CONCLUSION

Ce projet de modélisation d'une base de données pour les boulangeries PAUL a permis de créer et structurer efficacement l'ensemble du système d'information lié aux activités de l'entreprise comme la prise de commande.

La mise en place des différents scripts plus ou moins complexe ont permis de simuler un environnement de test cohérent, réaliste et rigoureux garantissant la viabilité de la structure de la base face à des scénarios concrets.

Cette base de données pourrait donc être implémenté dans une application pour l'enseigne PAUL avec une intégration d'autres fonctionnalités tel qu'une gestion des stocks poussée, ou encore un algorithme de recommandation prédictives pour personnaliser l'expérience client via une application mobile.

RÉPARTITON DES POINTS :

- Théo MAUHIN : 50%
- Rémy GONZALEZ-PAULO : 50%