Rapport sur la Modélisation de la Base de Données pour la Gestion des Commandes et Produits

1. Objectifs visés

Les objectifs de ce projet sont de :

- Modéliser une base de données relationnelle pour stocker les informations relatives aux clients, aux commandes, aux produits (boîtes), ainsi que les calculs associés.
- Automatiser la gestion des contraintes et des calculs (surface des boîtes, prix des commandes, etc.).
- Produire des analyses pour valider la cohérence et l'utilité de la base de données.

2. Modélisation des données

Pour répondre aux besoins du projet, le modèle logique de données se compose de plusieurs entités principales : `CLIENTS`, `COMMANDES`, `BOITES`, `MATIERES`, `COULEURS`, `LIGNES_COMMANDE`, et une entité intermédiaire `MATIERE_COULEURS`.

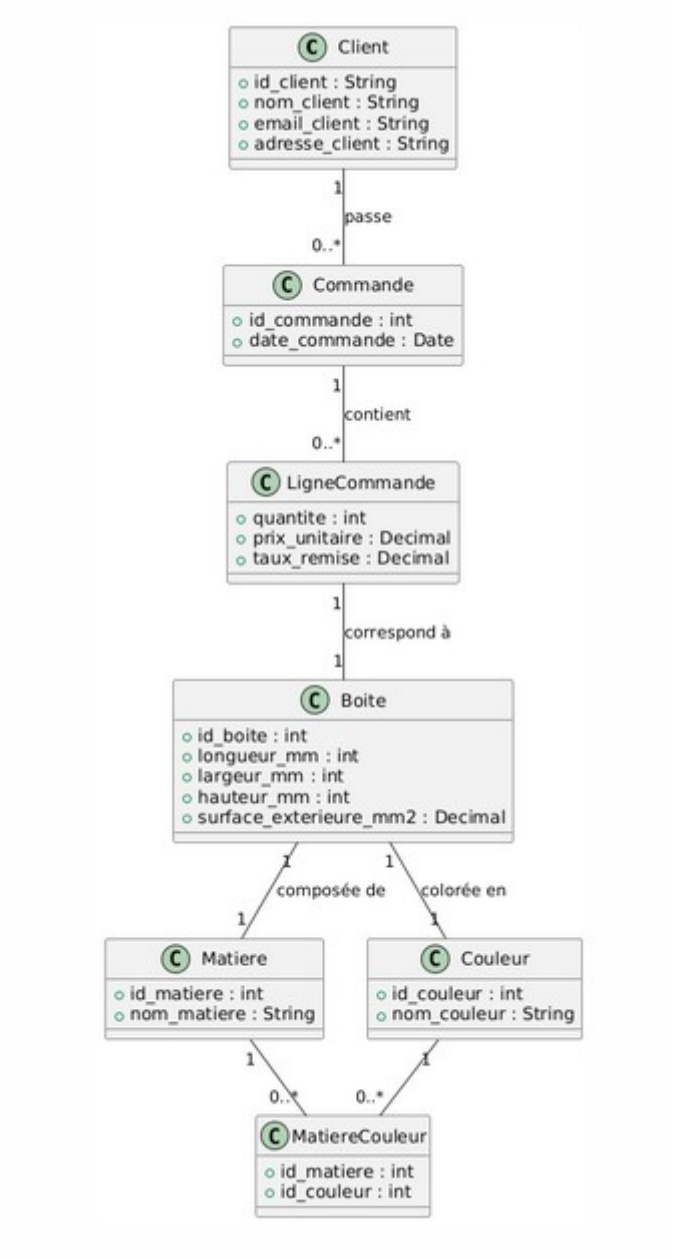
2.1 Schéma de base de données (Modèle logique)

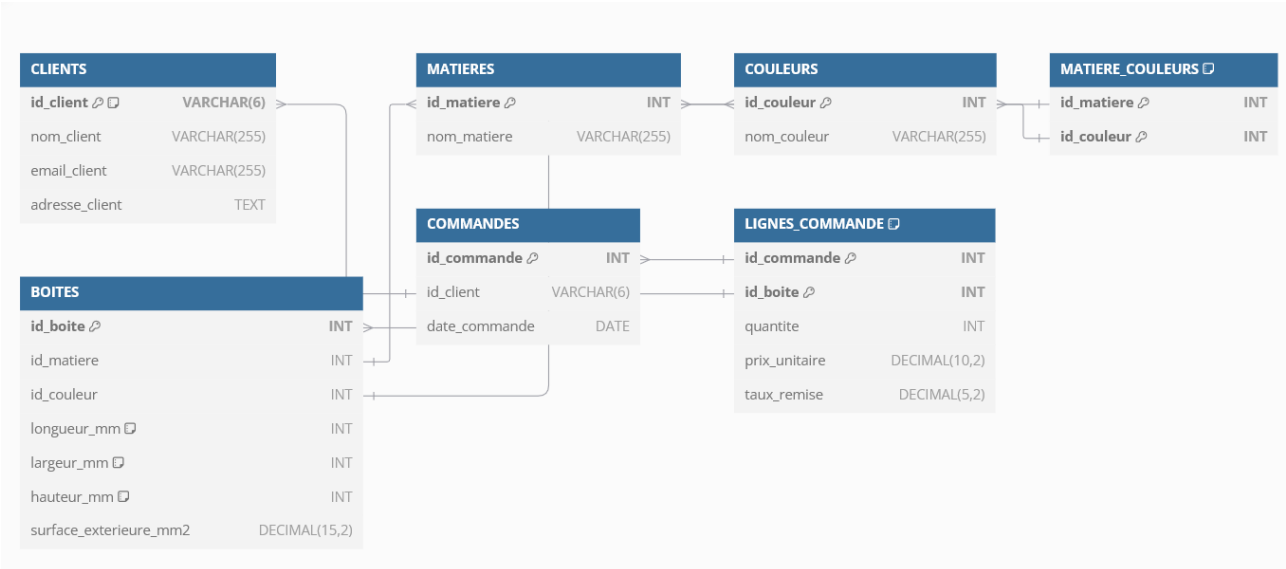
Le modèle logique des données a été conçu pour refléter les relations entre les clients, les produits (boîtes), et les commandes :

- **CLIENTS** : Cette table contient les informations sur les clients (nom, email, adresse) et un identifiant unique au format `xx-123`.
- **MATIERES et COULEURS** : Ces deux tables contiennent les informations sur les matières et les couleurs disponibles, respectivement.
- **MATIERE_COULEURS** : Cette table fait le lien entre les matières et les couleurs disponibles pour les boîtes.
- **BOITES** : Contient les informations détaillées sur les boîtes (matière, couleur, dimensions), avec une contrainte de dimension maximale (1 mètre).
- **COMMANDES** : Contient les commandes passées par les clients avec une référence à la table `CLIENTS`.
- **LIGNES_COMMANDE** : Relie les commandes aux boîtes et gère la quantité, le prix unitaire, et les remises associées.

2.2 Schéma visuel (Diagramme Entité-Relation ou UML)

Le schéma visuel de la base de données est présenté ci-dessous


Client
id_client : String
nom_client : String
email_client : String
adresse_client : String
1
passe
0..*
Commande
id_commande : int
date_commande : Date
1
contient
0..*
LigneCommande
quantite : int
prix_unitaire : Decimal
taux_remise : Decimal
1
correspond à
1
Boite
id_boite : int
longueur_mm : int
largeur_mm : int
hauteur_mm : int
surface_exterieure_mm2 : Decimal
1
composée de
1
1
colorée en
1
Matiere
id_matiere : int
nom_matiere : String
Couleur
id_couleur : int
nom_couleur : String
1
0..*
1
0..*
MatiereCouleur
id_matiere : int
id_couleur : int


CLIENTS
id_client VARCHAR(6)
nom_client VARCHAR(255)
email_client VARCHAR(255)
adresse_client TEXT
MATIERES
id_matiere INT
nom_matiere VARCHAR(255)
COULEURS
id_couleur INT
nom_couleur VARCHAR(255)
MATIERE_COULEURS
id_matiere INT
id_couleur INT
COMMANDES
id_commande INT
id_client VARCHAR(6)
date_commande DATE
LIGNES_COMMANDE
id_commande INT
id_boite INT
quantite INT
prix_unitaire DECIMAL(10,2)
taux_remise DECIMAL(5,2)
BOITES
id_boite INT
id_matiere INT
id_couleur INT
longueur_mm INT
largeur_mm INT
hauteur_mm INT
surface_exterieure_mm2 DECIMAL(15,2)

3. Structure et contraintes

- **Validation des dimensions** : La contrainte `CHECK` sur les dimensions (longueur, largeur, hauteur) garantit que la taille maximale de la boîte ne dépasse pas 1000 mm.
- **Validation des ID des clients** : Un contrôle est appliqué pour s'assurer que l'ID des clients suit le format `xx-123` (deux lettres minuscules suivies d'un numéro).

4. Calculs associés

Des fonctions et triggers sont utilisés pour calculer automatiquement certains paramètres :

- **Calcul de la surface extérieure** : Une fonction PL/pgSQL `calculer_surface_exterieure` est utilisée pour calculer la surface extérieure d'une boîte à partir de ses dimensions.
- **Trigger** : Un trigger est configuré pour mettre à jour la surface extérieure à chaque insertion ou mise à jour d'une boîte dans la table `BOITES`.

5. Analyse et vues

Des vues ont été créées pour faciliter l'analyse des données :

- **v1** : Nombre total de commandes par client.
- **v2** : Chiffre d'affaires total par client.
- **v3** : Chiffre d'affaires total par mois en 2024.
- **v4** : Top 5 des boîtes les plus vendues.
- **v5** : Top 5 des boîtes les plus rentables.
- **v6** : Nombre total de commandes et chiffre d'affaires par matière.
- **v7** : Nombre total de commandes et chiffre d'affaires par couleur.
- **v8** : Répartition des commandes par jour de la semaine.

6. Script SQL pour la création de la base de données

Le script SQL suivant permet de créer toutes les tables et les relations associées, avec les contraintes nécessaires :

```
-- Autoriser PostgreSQL à écouter sur toutes les interfaces réseau
ALTER SYSTEM SET listen_addresses = '*';

-- Ajouter une règle pour le réseau Docker dans pg_hba.conf
ALTER SYSTEM SET hba_file = '/var/lib/postgresql/data/pg_hba.conf';

-- Suppression des tables si elles existent (pour le développement/test)
DROP TABLE IF EXISTS LIGNES_COMMANDE;
DROP TABLE IF EXISTS COMMANDES;
DROP TABLE IF EXISTS BOITES;
DROP TABLE IF EXISTS MATIERE_COULEURS;
DROP TABLE IF EXISTS COULEURS;
DROP TABLE IF EXISTS MATIERES;
DROP TABLE IF EXISTS CLIENTS;

-- Création de la table CLIENTS
CREATE TABLE CLIENTS (
  id_client VARCHAR(6) PRIMARY KEY,
```

```
  nom_client VARCHAR(255),
  email_client VARCHAR(255),
  adresse_client TEXT,
  -- Contrainte CHECK avec expression régulière pour l'id_client
  CONSTRAINT chk_id_client CHECK (id_client ~ '^[a-z]{2}-\d+$')
);

-- Création de la table MATIERES
CREATE TABLE MATIERES (
  id_matiere SERIAL PRIMARY KEY,
  nom_matiere VARCHAR(255) UNIQUE
);

-- Création de la table COULEURS
CREATE TABLE COULEURS (
  id_couleur SERIAL PRIMARY KEY,
  nom_couleur VARCHAR(255) UNIQUE
);

-- Création de la table MATIERE_COULEURS
CREATE TABLE MATIERE_COULEURS (
  id_matiere INT,
  id_couleur INT,
  PRIMARY KEY (id_matiere, id_couleur),
  FOREIGN KEY (id_matiere) REFERENCES MATIERES(id_matiere),
  FOREIGN KEY (id_couleur) REFERENCES COULEURS(id_couleur)
);

-- Création de la table BOITES
CREATE TABLE BOITES (
  id_boite SERIAL PRIMARY KEY,
  id_matiere INT,
  id_couleur INT,
  longueur_mm INT NOT NULL,
  largeur_mm INT NOT NULL,
  hauteur_mm INT NOT NULL,
  surface_exterieure_mm2 DECIMAL(15,2),
  FOREIGN KEY (id_matiere) REFERENCES MATIERES(id_matiere),
  FOREIGN KEY (id_couleur) REFERENCES COULEURS(id_couleur),
  CONSTRAINT chk_longueur CHECK (longueur_mm <= 1000),
  CONSTRAINT chk_largeur CHECK (largeur_mm <= 1000),
  CONSTRAINT chk_hauteur CHECK (hauteur_mm <= 1000)
);

-- Création de la table COMMANDES
CREATE TABLE COMMANDES (
  id_commande SERIAL PRIMARY KEY,
  id_client VARCHAR(6),
  date_commande DATE,
  FOREIGN KEY (id_client) REFERENCES CLIENTS(id_client)
);
```

```sql
-- Création de la table LIGNES_COMMANDE
CREATE TABLE LIGNES_COMMANDE (
  id_commande INT,
  id_boite INT,
  quantite INT NOT NULL,
  prix_unitaire DECIMAL(10,2),
  taux_remise DECIMAL(5,2),
  PRIMARY KEY (id_commande, id_boite),
  FOREIGN KEY (id_commande) REFERENCES COMMANDES(id_commande),
  FOREIGN KEY (id_boite) REFERENCES BOITES(id_boite)
);

-- Fonction pour calculer la surface extérieure
CREATE OR REPLACE FUNCTION calculer_surface_exterieure(longueur INT, largeur INT, hauteur INT)
RETURNS DECIMAL(15,2) AS $$
BEGIN
  RETURN 2 * (longueur * largeur + longueur * hauteur + largeur * hauteur);
END;
$$ LANGUAGE plpgsql;

-- Trigger pour mettre à jour la surface extérieure à l'insertion ou la modification d'une boîte
CREATE OR REPLACE FUNCTION maj_surface_boite()
RETURNS TRIGGER AS $$
BEGIN
  NEW.surface_exterieure_mm2 := calculer_surface_exterieure(NEW.longueur_mm, NEW.largeur_mm, NEW.hauteur_mm);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER maj_surface_boite_trigger
BEFORE INSERT OR UPDATE ON BOITES
FOR EACH ROW
EXECUTE FUNCTION maj_surface_boite();

-- Vue pour simplifier l'accès aux informations des boîtes avec matière et couleur
CREATE OR REPLACE VIEW boites_details AS
SELECT
  b.id_boite,
  m.nom_matiere,
  c.nom_couleur,
  b.longueur_mm,
  b.largeur_mm,
  b.hauteur_mm,
  b.surface_exterieure_mm2
FROM
  BOITES b
JOIN
  MATIERES m ON b.id_matiere = m.id_matiere
JOIN
  COULEURS c ON b.id_couleur = c.id_couleur;
```

```sql
-- Insertion de 7 matières
INSERT INTO MATIERES (nom_matiere) VALUES
('Plastique'),
('Bois'),
('Métal'),
('Carton'),
('Verre'),
('Céramique'),
('Tissu');

-- Insertion de 7 couleurs
INSERT INTO COULEURS (nom_couleur) VALUES
('Rouge'),
('Bleu'),
('Vert'),
('Jaune'),
('Noir'),
('Blanc'),
('Violet');

-- Insertion de 7 clients
INSERT INTO CLIENTS (id_client, nom_client, email_client, adresse_client) VALUES
('ab-123', 'Alice Smith', 'alice@example.com', '123 Rue Principale'),
('cd-456', 'Bob Johnson', 'bob@example.com', '456 Avenue du Chêne'),
('ef-789', 'Charlie Brown', 'charlie@example.com', '789 Boulevard des Roses'),
('gh-101', 'Diana Prince', 'diana@example.com', '101 Rue de la Paix'),
('ij-112', 'Evan Davis', 'evan@example.com', '112 Avenue des Lilas'),
('kl-131', 'Fiona Green', 'fiona@example.com', '131 Rue des Érables'),
('mn-415', 'George White', 'george@example.com', '415 Boulevard du Soleil');

-- Insertion de 7 boîtes
INSERT INTO BOITES (id_matiere, id_couleur, longueur_mm, largeur_mm, hauteur_mm) VALUES
(1, 1, 100, 150, 50),   -- Plastique Rouge
(2, 3, 200, 200, 100),  -- Bois Vert
(3, 5, 300, 150, 200),  -- Métal Noir
(4, 2, 250, 250, 250),  -- Carton Bleu
(5, 6, 150, 150, 150),  -- Verre Blanc
(6, 4, 100, 100, 100),  -- Céramique Jaune
(7, 7, 200, 100, 50);   -- Tissu Violet

-- Insertion d'un nombre aléatoire de commandes par client avec des dates aléatoires en 2024
DO $$
DECLARE
  client_id VARCHAR(6);
  num_commandes INT;
  i INT;
  random_days INT;
  random_date DATE;
BEGIN
  FOR client_id IN SELECT id_client FROM CLIENTS LOOP
```

```
    num_commandes := floor(random() * 10) + 1; -- Génère entre 1 et 10 commandes par client
    FOR i IN 1..num_commandes LOOP
      random_days := floor(random() * 365); -- Génère un nombre aléatoire de jours entre 0 et 364
      random_date := '2024-01-01'::DATE + (random_days || ' days')::INTERVAL; -- Date aléatoire en 2024
      INSERT INTO COMMANDES (id_client, date_commande)
      VALUES (client_id, random_date);
    END LOOP;
  END LOOP;
END $$ LANGUAGE plpgsql;

-- Insertion d'un nombre aléatoire de lignes de commande par commande
DO $$
DECLARE
  commande_id INT;
  num_lignes INT;
  j INT;
  boite_id INT;
BEGIN
  FOR commande_id IN SELECT id_commande FROM COMMANDES LOOP
    num_lignes := floor(random() * 5) + 1; -- Génère entre 1 et 5 lignes de commande par commande
    FOR j IN 1..num_lignes LOOP
      boite_id := (commande_id + j - 1) % 7 + 1; -- Associe chaque ligne à une boîte différente
      INSERT INTO LIGNES_COMMANDE (id_commande, id_boite, quantite, prix_unitaire, taux_remise)
      VALUES (
        commande_id,
        boite_id,
        (commande_id + j) % 5 + 1, -- Quantité entre 1 et 5
        10.00 + ((commande_id + j) % 10), -- Prix unitaire entre 10.00 et 19.00
        0.10 * ((commande_id + j) % 3) -- Taux de remise entre 0.00 et 0.20
      );
    END LOOP;
  END LOOP;
END $$ LANGUAGE plpgsql;


-- VUES
-- 1. Nombre total de commandes par client
CREATE OR REPLACE VIEW v1 AS
SELECT
  c.id_client,
  c.nom_client,
  COUNT(co.id_commande) AS nombre_commandes
FROM
  CLIENTS c
LEFT JOIN
  COMMANDES co ON c.id_client = co.id_client
GROUP BY
  c.id_client, c.nom_client
ORDER BY
  nombre_commandes DESC;
```

```sql
-- 2. Chiffre d'affaires total par client
CREATE OR REPLACE VIEW v2 AS
SELECT
  c.id_client,
  c.nom_client,
  SUM(lc.quantite * lc.prix_unitaire * (1 - lc.taux_remise)) AS chiffre_affaires
FROM
  CLIENTS c
JOIN
  COMMANDES co ON c.id_client = co.id_client
JOIN
  LIGNES_COMMANDE lc ON co.id_commande = lc.id_commande
GROUP BY
  c.id_client, c.nom_client
ORDER BY
  chiffre_affaires DESC;

-- 3. Chiffre d'affaires total par mois en 2024
CREATE OR REPLACE VIEW v3 AS
SELECT
  TO_CHAR(co.date_commande, 'YYYY-MM') AS mois,
  SUM(lc.quantite * lc.prix_unitaire * (1 - lc.taux_remise)) AS chiffre_affaires
FROM
  COMMANDES co
JOIN
  LIGNES_COMMANDE lc ON co.id_commande = lc.id_commande
WHERE
  EXTRACT(YEAR FROM co.date_commande) = 2024
GROUP BY
  TO_CHAR(co.date_commande, 'YYYY-MM')
ORDER BY
  mois;

-- 4 Top 5 des boîtes les plus vendues
CREATE OR REPLACE VIEW v4 AS
SELECT
  b.id_boite,
  m.nom_matiere,
  c.nom_couleur,
  SUM(lc.quantite) AS quantite_vendue
FROM
  LIGNES_COMMANDE lc
JOIN
  BOITES b ON lc.id_boite = b.id_boite
JOIN
  MATIERES m ON b.id_matiere = m.id_matiere
JOIN
  COULEURS c ON b.id_couleur = c.id_couleur
GROUP BY
  b.id_boite, m.nom_matiere, c.nom_couleur
ORDER BY
```

```
  quantite_vendue DESC
LIMIT 5;

-- 5. Top 5 des boîtes les plus rentables

CREATE OR REPLACE VIEW v5 AS
SELECT
  b.id_boite,
  m.nom_matiere,
  c.nom_couleur,
  SUM(lc.quantite * lc.prix_unitaire * (1 - lc.taux_remise)) AS chiffre_affaires
FROM
  LIGNES_COMMANDE lc
JOIN
  BOITES b ON lc.id_boite = b.id_boite
JOIN
  MATIERES m ON b.id_matiere = m.id_matiere
JOIN
  COULEURS c ON b.id_couleur = c.id_couleur
GROUP BY
  b.id_boite, m.nom_matiere, c.nom_couleur
ORDER BY
  chiffre_affaires DESC
LIMIT 5;


-- 6. Nombre total de commandes et chiffre d'affaires par matière

CREATE OR REPLACE VIEW v6 AS
SELECT
  m.nom_matiere,
  COUNT(DISTINCT co.id_commande) AS nombre_commandes,
  SUM(lc.quantite * lc.prix_unitaire * (1 - lc.taux_remise)) AS chiffre_affaires
FROM
  MATIERES m
JOIN
  BOITES b ON m.id_matiere = b.id_matiere
JOIN
  LIGNES_COMMANDE lc ON b.id_boite = lc.id_boite
JOIN
  COMMANDES co ON lc.id_commande = co.id_commande
GROUP BY
  m.nom_matiere
ORDER BY
  chiffre_affaires DESC;

-- 7. Nombre total de commandes et chiffre d'affaires par couleur

CREATE OR REPLACE VIEW v7 AS
SELECT
  c.nom_couleur,
```

```
  COUNT(DISTINCT co.id_commande) AS nombre_commandes,
  SUM(lc.quantite * lc.prix_unitaire * (1 - lc.taux_remise)) AS chiffre_affaires
FROM
  COULEURS c
JOIN
  BOITES b ON c.id_couleur = b.id_couleur
JOIN
  LIGNES_COMMANDE lc ON b.id_boite = lc.id_boite
JOIN
  COMMANDES co ON lc.id_commande = co.id_commande
GROUP BY
  c.nom_couleur
ORDER BY
  chiffre_affaires DESC;

-- 8. Répartition des commandes par jour de la semaine

CREATE OR REPLACE VIEW v8 AS
SELECT
  TO_CHAR(co.date_commande, 'Day') AS jour_semaine,
  COUNT(co.id_commande) AS nombre_commandes
FROM
  COMMANDES co
GROUP BY
  TO_CHAR(co.date_commande, 'Day')
ORDER BY
  nombre_commandes DESC;

select * from v8
```

7. Conclusion et recommandations

Le modèle relationnel proposé répond aux besoins fonctionnels et métiers du projet, en assurant la gestion cohérente des données et des contraintes. La base de données est prête à être utilisée pour le traitement des commandes et la gestion des produits. Les vues et les calculs automatisés permettront de simplifier les analyses et la prise de décision pour l'entreprise.