ONLINE STORE DATABASE PROJECT

Course: Database Introduction

Student: Naas Eyoub

Date: 15/09/2025

Instructor: Markus Schatten

TABLE DES MATIERES

EXTENDED ESSAY (TRANSACTIONS IN DBMS: ACID GUARANTEES AND CONTEMPORARY CHALLENGES)

Introduction

Database Management Systems (DBMS) rely on a fundamental mechanism to ensure the reliability of operations: transactions. A transaction represents a sequence of operations (reads, writes, updates) processed as an indivisible logical unit. Its role is to maintain data consistency and integrity, especially in multi-user environments or under failure conditions. Initially formalized in the 1970s, transactions are governed by fundamental properties known as ACID, and their implementation raises increasing technical challenges in light of the evolution of distributed architectures and Big Data.

The ACID Properties: Theoretical Foundation and Implementation

The ACID properties (Atomicity, Consistency, Isolation, Durability) form the theoretical foundation of transactions. Atomicity guarantees that a transaction is either fully executed or fully rolled back, preventing partial states. For example, in a bank transfer, debit and credit operations must both succeed or both fail. Consistency ensures that a transaction transitions the database from one valid state to another, respecting defined integrity constraints. Isolation requires that concurrent transactions do not interfere with each other, and each transaction should appear as if executed in isolation. Finally, durability guarantees that committed changes persist despite system failures, often through logging mechanisms.

Implementation differs across DBMS. Systems such as PostgreSQL or Oracle use locking or multiversion concurrency control (MVCC) to balance isolation and performance. Commit and rollback mechanisms allow validating or canceling transactions, while transaction logs record operations for recovery after a crash.

Challenges and Technical Issues
Concurrent execution of transactions may cause anomalies if not properly managed. Dirty reads, non-repeatable reads, and phantom reads are common problems. To address them, DBMS provide isolation levels (e.g., Read Committed, Serializable), each offering a trade-off between consistency and performance. For example, Read Uncommitted, though efficient, allows dirty reads and is rarely used in critical systems.

Beyond concurrency, failure management is another key challenge. Recovery algorithms, such as ARIES at IBM, or two-phase commit (2PC) protocols for distributed systems, ensure transaction durability. However, these mechanisms introduce complexity and can reduce throughput.

Applications and Contemporary Evolution
Transactions are omnipresent in demanding domains such as finance, e-commerce, or resource management. In a ticket reservation system, for example, a transaction must check seat availability, update the customer's account, and decrease seat inventory atomically to prevent double booking. Similarly, in distributed systems, extended architecture (XA) transactions coordinate operations across multiple resources, such as distinct databases.

With the rise of Big Data and cloud architectures, the ACID model is sometimes challenged. NoSQL systems, such as Cassandra, favor the BASE model (Basically Available, Soft state, Eventually consistent), which sacrifices immediate consistency for availability and scalability. Still, a recent trend has been to reintroduce transactional guarantees into these systems through hybrid approaches such as saga transactions or compensating mechanisms.

Extended Reflection
A major contemporary challenge is balancing **scalability** and **reliability**. While ACID remains essential in mission-critical applications, modern systems increasingly employ adaptive mechanisms where isolation levels are dynamically adjusted according to workload. Cloud providers such as AWS and Google Cloud have introduced "new SQL" systems (e.g., Spanner) that attempt to combine strong consistency with distributed scalability, illustrating that ACID is not obsolete but rather being reinterpreted. Additionally, the integration of machine learning into DBMS raises new issues: how to ensure ACID compliance when data is being updated in real time by predictive algorithms? These questions show that transactions are not just a legacy concept but an evolving research area.

Conclusion
Transactions remain a cornerstone of DBMS, ensuring data integrity in critical contexts. Their management relies on sophisticated mechanisms to reconcile ACID properties with performance and concurrency demands. In the era of distributed architectures and Big Data, transactions continue to evolve, adapting to new paradigms while preserving their theoretical rigor. For developers and architects, mastering these concepts remains indispensable to designing robust and reliable systems. Far from being outdated, transactions are at the heart of modern challenges, making them one of the most relevant and enduring concepts in the database field.

PART 2 – ENTITY-RELATIONSHIP DIAGRAM FOR THE ONLINE STORE DATABASE

Introduction
The conceptual design of the online store is represented using an Entity-Relationship Diagram (ERD). This diagram models the main business entities—such as clients, products, orders, payments, and deliveries—and the relationships between them. The ERD serves as a blueprint for the database implementation in PostgreSQL.

Entities and Attributes

1. **Client**
    - Attributes: id_client (PK), nom, email, telephone, date_inscription.
    - Represents registered customers.
2. **AdresseLivraison**
    - Attributes: id_adresse (PK), id_client (FK), rue, ville, code_postal, pays.
    - Each client can have multiple delivery addresses.
3. **CategorieProduit**
    - Attributes: id_categorie (PK), nom, description.
    - Organizes products into categories such as electronics or clothing.
4. **Produit**
    - Attributes: id_produit (PK), nom, description, prix, stock, id_categorie (FK).
    - Represents items available for purchase, with prices and stock levels.
5. **Commande**
    - Attributes: id_commande (PK), id_client (FK), date_commande, statut, id_adresse (FK).
    - Represents customer orders, linked to both the client and a delivery address.
6. **LigneCommande**
    - Attributes: id_commande (FK), id_produit (FK), quantite, prix_unitaire.
    - Junction table between *Commande* and *Produit*, recording items and quantities ordered.
7. **Paiement**
    - Attributes: id_paiement (PK), id_commande (FK), montant, mode_paiement, date_paiement.
    - Tracks payment transactions for each order.
8. **Livraison**
    - Attributes: id_livraison (PK), id_commande (FK), transporteur, date_expedition, date_livraison_prevue, statut.
    - Represents shipment details and delivery status.

Relationships

- One **Client** can have many **Commandes**.
- One **Client** can have many **Adresses de Livraison**.
- One **Commande** is associated with one **Adresse de Livraison**.
- One **CategorieProduit** can include many **Produits**.

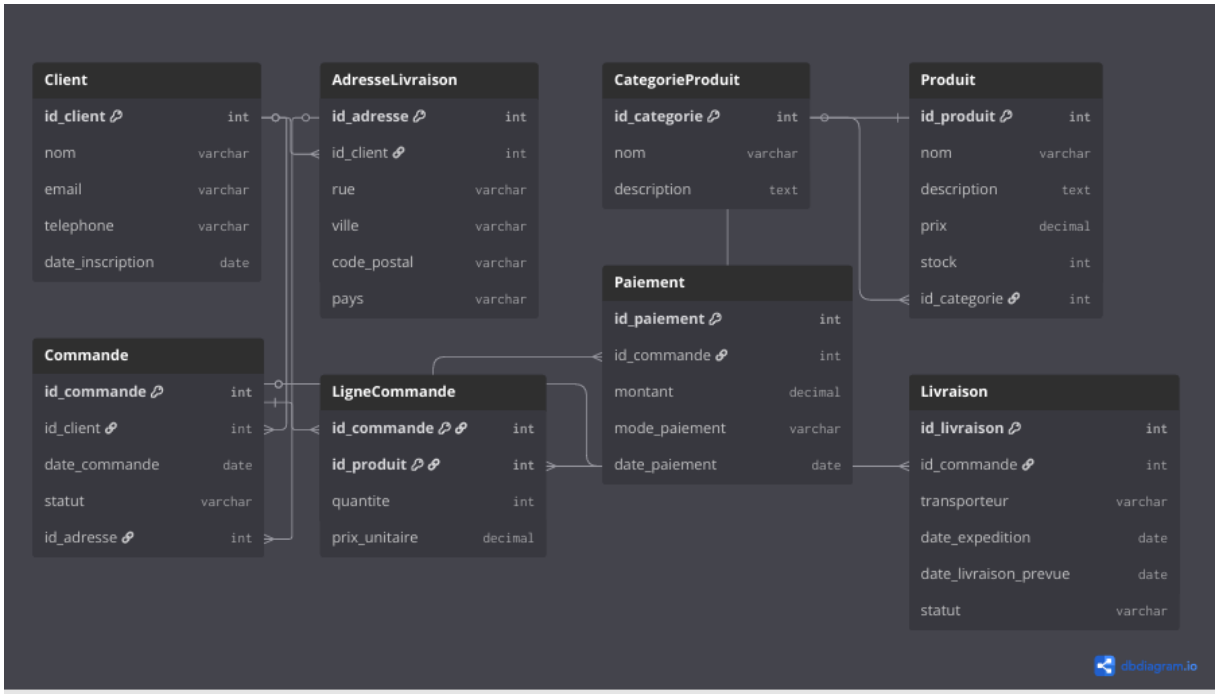
- One **Commande** includes one or more **Produits** via **LigneCommande** (many-to-many relationship).
- One **Commande** has one or more **Paiements**.
- One **Commande** has one **Livraison**.

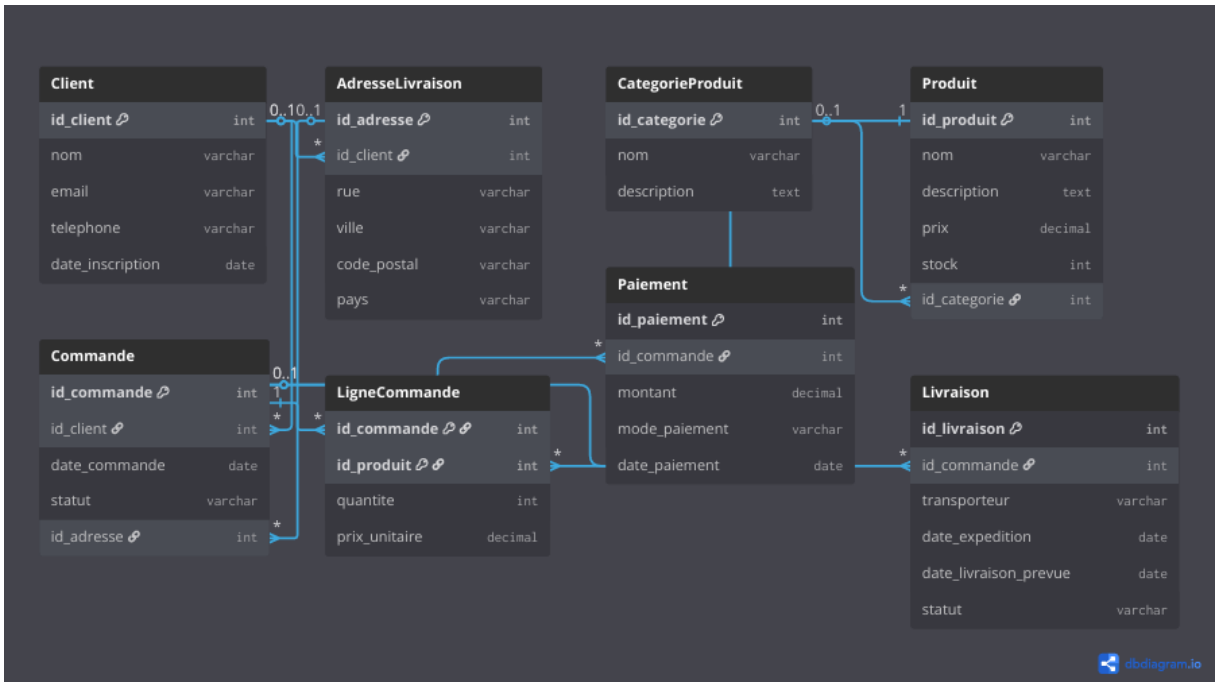
Keys and Constraints

- **Primary keys** uniquely identify each entity.
- **Foreign keys** enforce referential integrity between related entities.
- **Composite key** in *LigneCommande* (id_commande, id_produit) ensures that the same product cannot be added twice to the same order.

Conclusion

The ERD accurately captures the business logic of an online store. It clearly models the relationships between clients, products, orders, payments, and deliveries. This design provides a solid foundation for the database implementation and guarantees that data remains consistent while supporting core business processes.





PART 3 – DOCUMENTATION OF THE DATABASE IMPLEMENTATION

Introduction

The goal of this implementation is to transform the conceptual design of an online store into a functional relational database. Using PostgreSQL, the model includes entities for clients, products, orders, payments, and deliveries. This implementation ensures data integrity, supports core business operations such as order processing and payments, and allows querying for analytics and reporting.

Database Schema

The database schema was derived from the entity-relationship diagram (ERD). The following tables were created:

- **Client**: Stores customer details (name, email, phone number, registration date). Each client may place multiple orders.
- **AdresseLivraison**: Contains delivery addresses associated with clients. A client can have multiple delivery addresses.
- **CategorieProduit**: Organizes products into categories (e.g., electronics, clothing).
- **Produit**: Represents items available for purchase, with price, stock quantity, and category reference.
- **Commande**: Represents customer orders, linked to both a client and a delivery address.
- **LigneCommande**: Junction table linking orders and products, recording quantities and unit prices.
- **Paiement**: Stores payment details (amount, method, date) associated with an order.
- **Livraison**: Tracks deliveries, including carrier, shipping date, expected delivery date, and status.

Constraints and Integrity Rules

- **Primary Keys**: Each table has a unique identifier (e.g., id_client, id_commande).
- **Foreign Keys**: Relationships are enforced using foreign keys (e.g., id_client in *Commande* references *Client*).
- **Referential Integrity**: ON DELETE CASCADE ensures dependent records (e.g., delivery addresses) are automatically removed if a client is deleted.
- **Validation Constraints**: The schema enforces data rules such as CHECK (quantite > 0) in *LigneCommande* to prevent invalid values.

  Data Population
  The database was populated with representative sample data:

- Clients with corresponding delivery addresses.
- A set of products across different categories (electronics, books, clothing).
- Orders placed by clients with associated line items.
- Payment records to validate financial transactions.
- Delivery records to test logistics tracking.

  This dataset allows testing of relationships between entities and ensures that queries return meaningful results.

TABLES

```
-- Table Client
CREATE TABLE Client (
    id_client SERIAL PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    telephone VARCHAR(20),
    date_inscription DATE DEFAULT CURRENT_DATE
);

-- Table AdresseLivraison
CREATE TABLE AdresseLivraison (
    id_adresse SERIAL PRIMARY KEY,
    id_client INT REFERENCES Client(id_client) ON DELETE CASCADE,
```

```
    rue VARCHAR(150),
    ville VARCHAR(100),
    code_postal VARCHAR(20),
    pays VARCHAR(50)
);

-- Table CategorieProduit
CREATE TABLE CategorieProduit (
    id_categorie SERIAL PRIMARY KEY,
    nom VARCHAR(100),
    description TEXT
);

-- Table Produit
CREATE TABLE Produit (
    id_produit SERIAL PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    description TEXT,
    prix DECIMAL(10,2) NOT NULL,
    stock INT DEFAULT 0,
    id_categorie INT REFERENCES CategorieProduit(id_categorie)
);

-- Table Commande
CREATE TABLE Commande (
    id_commande SERIAL PRIMARY KEY,
    id_client INT REFERENCES Client(id_client),
    date_commande DATE DEFAULT CURRENT_DATE,
    statut VARCHAR(50),
    id_adresse INT REFERENCES AdresseLivraison(id_adresse)
```

```
);

-- Table LigneCommande
CREATE TABLE LigneCommande (
    id_commande INT REFERENCES Commande(id_commande) ON DELETE CASCADE,
    id_produit INT REFERENCES Produit(id_produit),
    quantite INT CHECK (quantite > 0),
    prix_unitaire DECIMAL(10,2),
    PRIMARY KEY (id_commande, id_produit)
);

-- Table Paiement
CREATE TABLE Paiement (
    id_paiement SERIAL PRIMARY KEY,
    id_commande INT REFERENCES Commande(id_commande),
    montant DECIMAL(10,2),
    mode_paiement VARCHAR(50),
    date_paiement DATE DEFAULT CURRENT_DATE
);

-- Table Livraison
CREATE TABLE Livraison (
    id_livraison SERIAL PRIMARY KEY,
    id_commande INT REFERENCES Commande(id_commande),
    transporteur VARCHAR(50),
    date_expedition DATE,
    date_livraison_prevue DATE,
    statut VARCHAR(50)
);
```

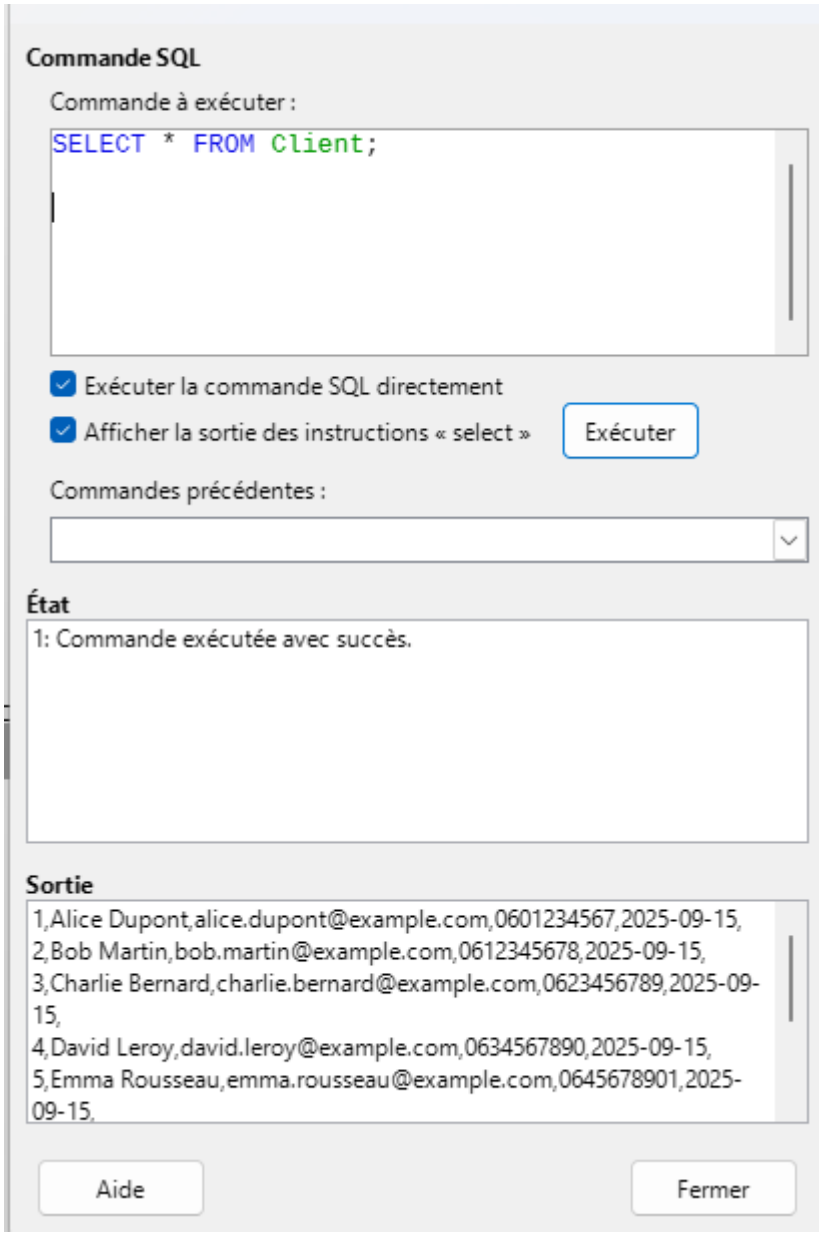
```
-- Ajouter d'autres clients
INSERT INTO Client (nom, email, telephone) VALUES
('David Leroy', 'david.leroy@example.com', '0634567890'),
('Emma Rousseau', 'emma.rousseau@example.com', '0645678901'),
('François Petit', 'francois.petit@example.com', '0656789012'),
('Julie Lambert', 'julie.lambert@example.com', '0667890123'),
('Karim Haddad', 'karim.haddad@example.com', '0678901234');

-- Adresses pour ces clients
INSERT INTO AdresseLivraison (id_client, rue, ville, code_postal, pays) VALUES
(4, '10 Rue Victor Hugo', 'Bordeaux', '33000', 'France'),
(5, '25 Rue Nationale', 'Lille', '59000', 'France'),
(6, '7 Avenue de la République', 'Toulouse', '31000', 'France'),
(7, '50 Boulevard Haussmann', 'Paris', '75009', 'France'),
(8, '88 Cours Mirabeau', 'Aix-en-Provence', '13100', 'France');

-- Nouveaux produits
INSERT INTO Produit (nom, description, prix, stock, id_categorie) VALUES
('Casque Audio', 'Casque sans fil Bluetooth', 120.00, 80, 1),
('E-book', 'Livre numérique en PDF', 9.99, 500, 2),
('Jeans', 'Pantalon denim homme', 49.90, 150, 3),
('Robe', 'Robe d’été femme', 39.90, 120, 3),
('Tablette', 'Tablette Android 10 pouces', 299.00, 60, 1);

-- Commandes supplémentaires
INSERT INTO Commande (id_client, statut, id_adresse) VALUES
(4, 'En cours', 4),
(5, 'En attente de paiement', 5),
(6, 'Expédiée', 6),
```

```
(7, 'Livrée', 7),
(8, 'Annulée', 8);

-- Lignes de commande supplémentaires
INSERT INTO LigneCommande (id_commande, id_produit, quantite, prix_unitaire) VALUES
(4, 5, 1, 299.00),   -- David a commandé une tablette
(5, 6, 3, 9.99),     -- Emma a commandé 3 e-books
(6, 7, 2, 49.90),    -- François a commandé 2 jeans
(7, 8, 1, 39.90),    -- Julie a commandé 1 robe
(8, 9, 2, 120.00);   -- Karim a commandé 2 casques audio

-- Paiements supplémentaires
INSERT INTO Paiement (id_commande, montant, mode_paiement) VALUES
(4, 299.00, 'Carte bancaire'),
(5, 29.97, 'Carte bancaire'),
(6, 99.80, 'Virement'),
(7, 39.90, 'PayPal'),
(8, 240.00, 'Carte bancaire');

-- Livraisons supplémentaires
INSERT INTO Livraison (id_commande, transporteur, date_expedition, date_livraison_prevue, statut)
VALUES
(4, 'DHL', '2025-09-11', '2025-09-16', 'En attente'),
(5, 'Colissimo', '2025-09-12', '2025-09-17', 'En attente'),
(6, 'UPS', '2025-09-09', '2025-09-14', 'En transit'),
(7, 'Chronopost', '2025-09-05', '2025-09-09', 'Livrée'),
(8, 'FedEx', '2025-09-08', '2025-09-13', 'Annulée');
```

Voici 5 requêtes que tu pourras tester et montrer :

1. **Lister tous les clients** :

SELECT * FROM Client;

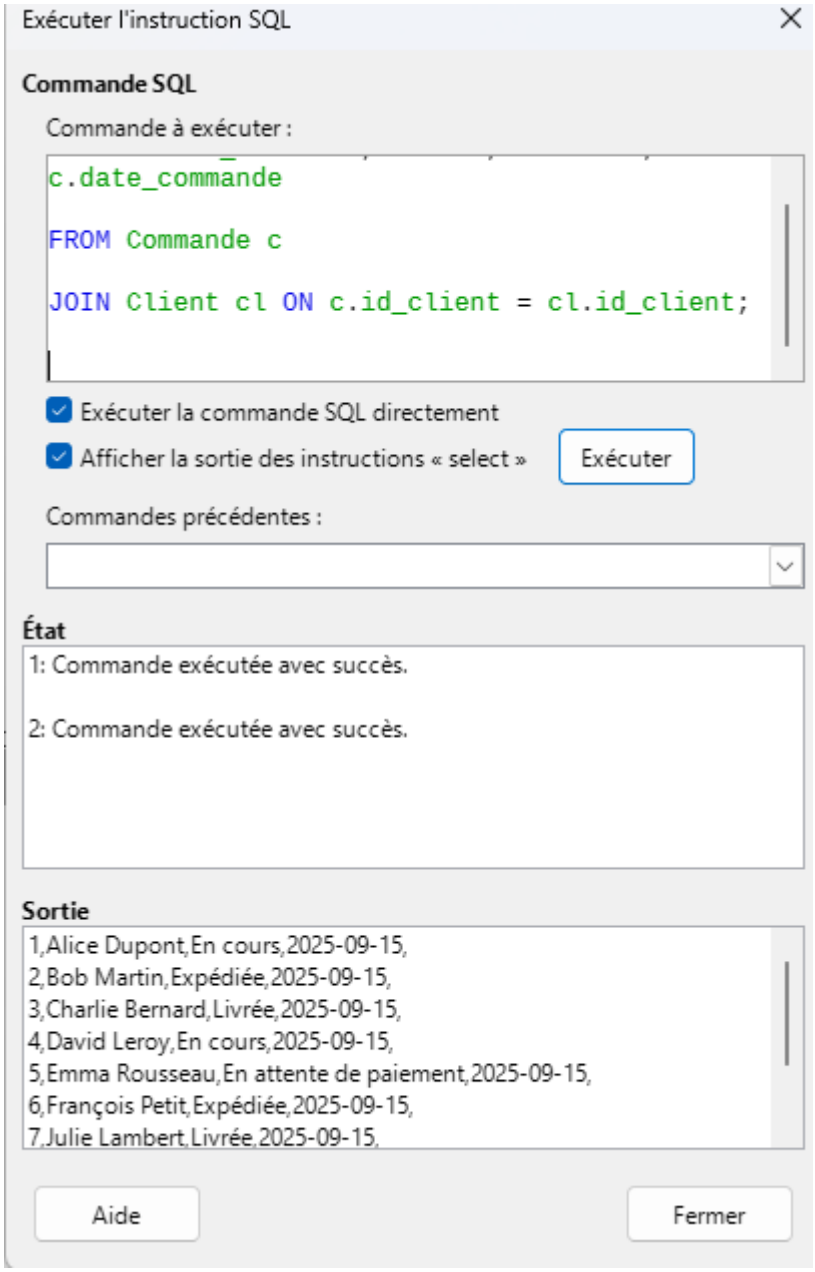


2. **Lister les commandes avec le nom du client et le statut** :

SELECT c.id_commande, cl.nom, c.statut, c.date_commande

FROM Commande c

JOIN Client cl ON c.id_client = cl.id_client;



3. **Lister les produits commandés par un client (ex. Alice Dupont)** :

SELECT cl.nom AS client, p.nom AS produit, lc.quantite, lc.prix_unitaire

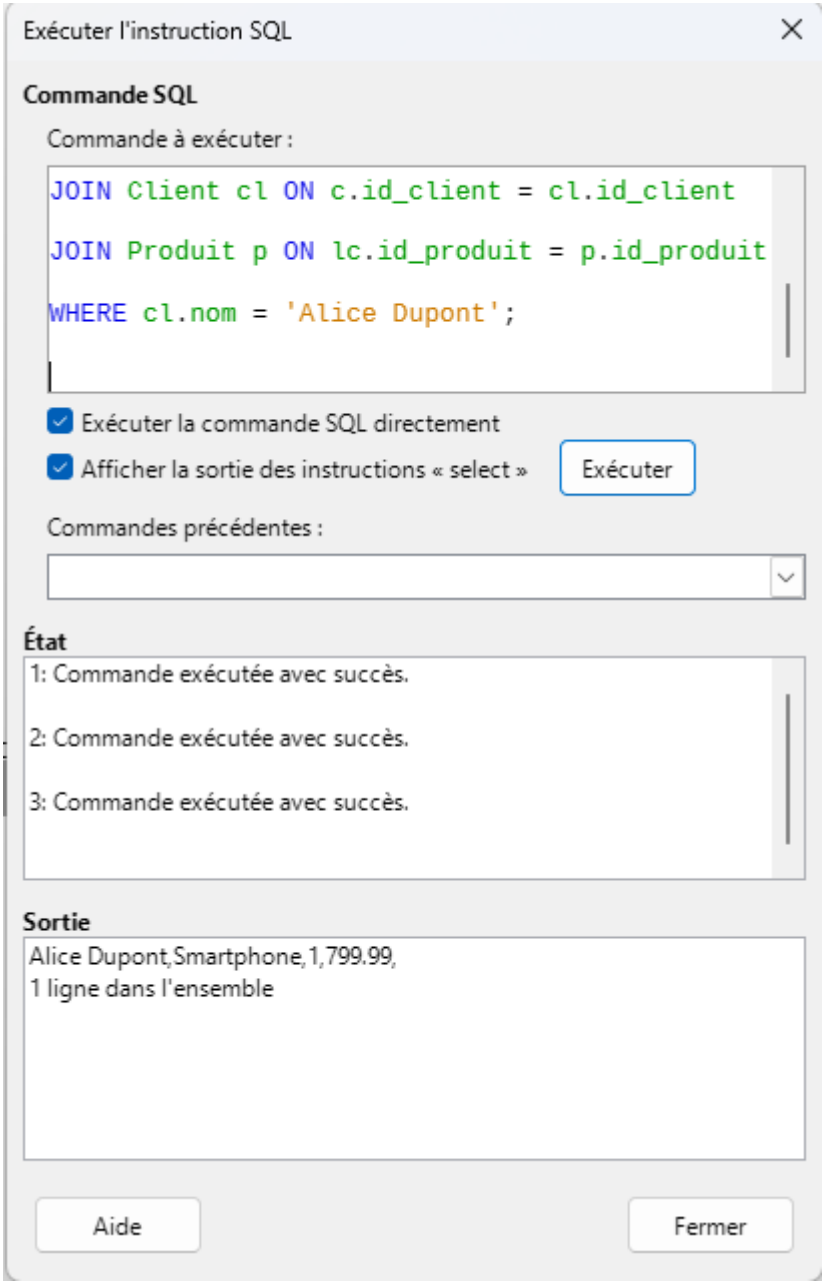
FROM LigneCommande lc

JOIN Commande c ON lc.id_commande = c.id_commande

JOIN Client cl ON c.id_client = cl.id_client

JOIN Produit p ON lc.id_produit = p.id_produit

WHERE cl.nom = 'Alice Dupont';



4. **Trouver le total dépensé par chaque client** :

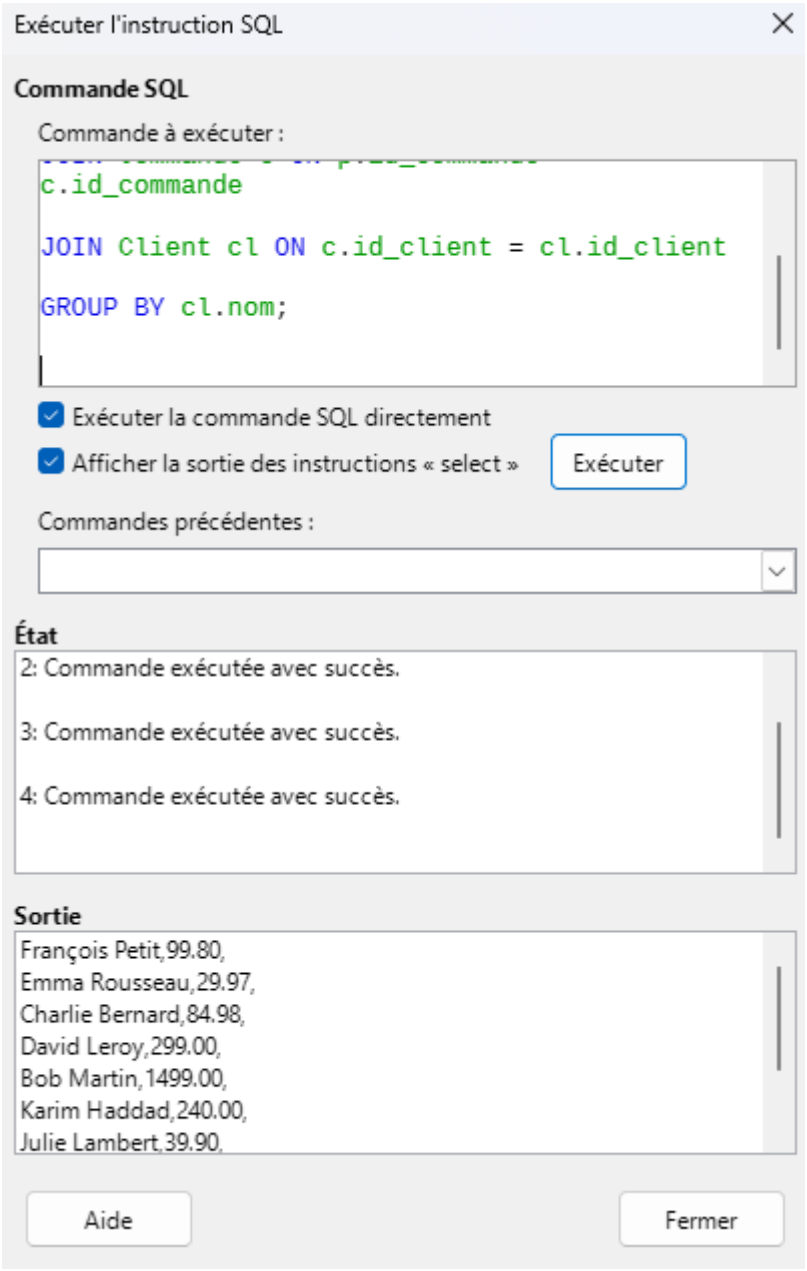
SELECT cl.nom, SUM(p.montant) AS total_depense

FROM Paiement p

JOIN Commande c ON p.id_commande = c.id_commande

JOIN Client cl ON c.id_client = cl.id_client

GROUP BY cl.nom;



5. **Lister les livraisons en retard** :

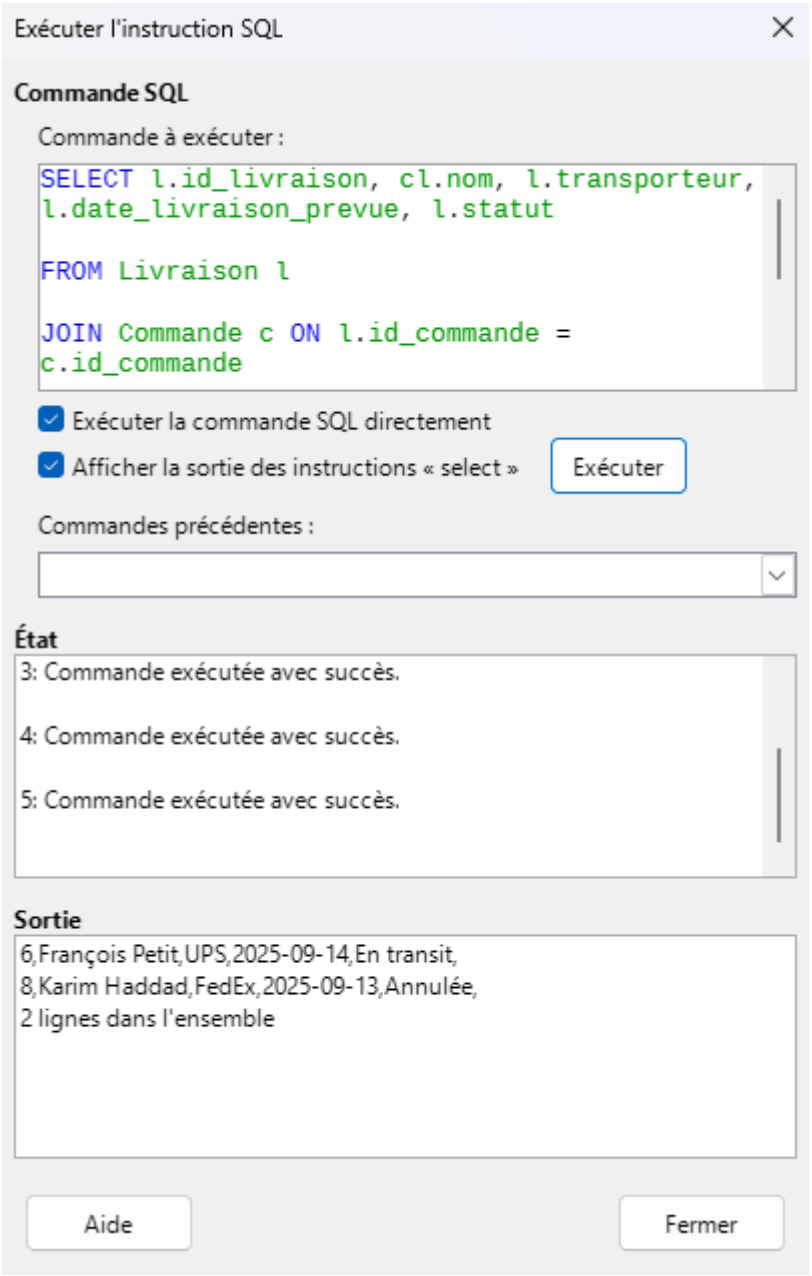
SELECT l.id_livraison, cl.nom, l.transporteur, l.date_livraison_prevue, l.statut

FROM Livraison l

JOIN Commande c ON l.id_commande = c.id_commande

JOIN Client cl ON c.id_client = cl.id_client

WHERE l.date_livraison_prevue < CURRENT_DATE AND l.statut <> 'Livrée';


Exécuter l'instruction SQL
Commande SQL
Commande à exécuter :
SELECT l.id_livraison, cl.nom, l.transporteur,
l.date_livraison_prevue, l.statut
FROM Livraison l
JOIN Commande c ON l.id_commande =
c.id_commande
Exécuter la commande SQL directement
Afficher la sortie des instructions « select »
Exécuter
Commandes précédentes :
État
3: Commande exécutée avec succès.
4: Commande exécutée avec succès.
5: Commande exécutée avec succès.
Sortie
6,François Petit,UPS,2025-09-14,En transit,
8,Karim Haddad,FedEx,2025-09-13,Annulée,
2 lignes dans l'ensemble
Aide
Fermer