Projet « RaPizz »

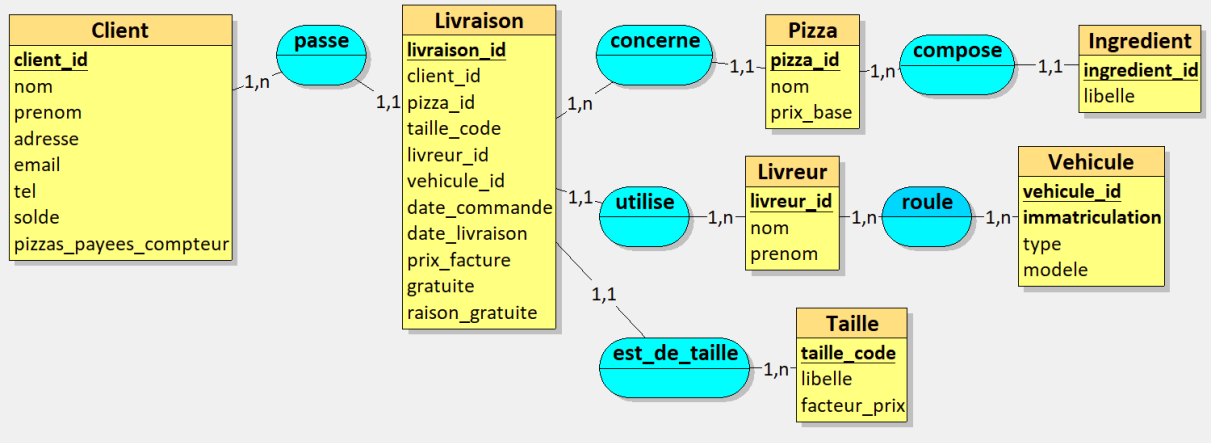
Richard HO, Ludovic REGNIER et Tom RAFFNER

Table des matières

1. Introduction & hypothèses

- **Unité d'opération** : le cahier des charges précise qu'« on considère que l'opération de base à modéliser est la *vente d'une unique pizza* ». Une *livraison* = 1 pizza.
- **Trois tailles** : naine (–1/3 du prix), humaine (prix de base), ogresse (+1/3 du prix).
- **Système prépayé** : le solde du client est débité à la création de la livraison, sauf si la pizza est gratuite.
- **Gratuités** :
    - **Fidélité** : une pizza gratuite est offerte toutes les 10 achetées.
    - **Retard > 30 min** : la pizza est gratuite et le client est remboursé.
- **Technologies cibles** : PostgreSQL 16 (SQL/PSM standard, PL/pgSQL), Java 21, Maven, JDBC, Swing.

2 . Modèle conceptuel E-A



Entités et attributs

Entité	PK	Attributs supplémentaires
CLIENT	client_id	nom, prenom, adresse, email, tel, solde, pizzas_payées_compteur
PIZZA	pizza_id	nom, prix_base
INGREDIENT	ingredient_id	libelle
LIVREUR	livreur_id	nom, prenom
VEHICULE	vehicule_id	immatriculation (UNIQUE), type ('AUTO', 'MOTO'), modele
TAILLE	taille_code	libelle, facteur_prix
LIVRAISON	livraison_id	client_id FK, pizza_id FK, taille_code FK, livreur_id FK, vehicule_id FK, date_commande, date_livraison, prix_facture, gratuite BOOL, raison_gratuite

Entité Client

Attribut	Type SQL	Utilité	Remarques
client_id	SERIAL (PK)	Identifiant unique du client	Généré automatiquement
nom	VARCHAR(50)	Nom de famille du client	Obligatoire

prenom	VARCHAR(50)	Prénom du client	Obligatoire
adresse	TEXT	Adresse de livraison	Utilisée pour les livraisons
email	VARCHAR(120) UNIQUE	Identifiant de contact	Peut servir à se connecter à un compte
tel	VARCHAR(20)	Téléphone du client	
solde	NUMERIC(8,2)	Montant en euros sur le compte	Décrémenté à chaque commande
pizzas_payees_compteur	SMALLINT	Nombre de pizzas payées	Sert pour la gratuité fidélité (toutes les 10 pizzas)

Entité Pizza

Attribut	**Type SQL**	**Utilité**	**Remarques**
pizza_id	SERIAL (PK)	Identifiant unique de la pizza	
nom	VARCHAR(60) UNIQUE	Nom de la pizza	Exemple : "Reine", "Margherita"
prix_base	NUMERIC(5,2)	Prix de base pour taille « humaine »	Les tailles ajustent ce prix avec un facteur multiplicatif

Entité Ingrédient

Attribut	**Type SQL**	**Utilité**	**Remarques**
ingredient_id	SERIAL (PK)	Identifiant unique de l'ingrédient	
libelle	VARCHAR(60) UNIQUE	Nom de l'ingrédient	

Entité Taille

Attribut	**Type SQL**	**Utilité**	**Remarques**
taille_code	CHAR(1) (PK)	Code court : N, H, O	
libelle	VARCHAR(20)	Nom de la taille	"Naine", "Humaine", "Ogresse"
facteur_prix	NUMERIC(4,3)	Multiplie le prix de base	Ex : 0.666, 1.000, 1.333

Entité Livreur

Attribut	Type SQL	Utilité	Remarques
livreur_id	SERIAL (PK)	Identifiant unique	
nom	VARCHAR(50)	Nom du livreur	
prenom	VARCHAR(50)	Prénom du livreur	

Entité Vehicule

Attribut	Type SQL	Utilité	Remarques
vehicule_id	SERIAL (PK)	Identifiant unique	
immatriculation	VARCHAR(15) UNIQUE	Permet d'identifier le véhicule	
type	VARCHAR(10)	Type de véhicule	Doit être 'AUTO' ou 'MOTO'
modele	VARCHAR(60)	Modèle du véhicule	Exemple : "Yamaha 125", "Fiat Panda"

Entité Livraison

Attribut	Type SQL	Utilité	Remarques
livraison_id	SERIAL (PK)	Identifiant de la livraison	
client_id	INT FK → CLIENT	Client concerné	
pizza_id	INT FK → PIZZA	Pizza livrée	
taille_code	CHAR(1) FK → TAILLE	Taille de la pizza	
livreur_id	INT FK → LIVREUR	Livreur assigné	
vehicule_id	INT FK → VEHICULE	Véhicule utilisé	
date_commande	TIMESTAMP	Quand la commande a été passée	Valeur par défaut = now()
date_livraison	TIMESTAMP	Quand elle a été livrée	Sert à détecter les retards
prix_facture	NUMERIC(6,2)	Prix réellement payé	
gratuite	BOOLEAN		
raison_gratuite	VARCHAR(20)	Soit 'FIDELITE', 'RETARD' ou NULL	

Associations détaillées

Association	Entités reliées	Cardinalité	Clé étrangère située dans	Attributs portés par la relation
passe	CLIENT — LIVRAISON	CLIENT (1,n) — LIVRAISON (1,1)	LIVRAISON.client_id	-
concerne	LIVRAISON — PIZZA	LIVRAISON (1,1) — PIZZA (1,1)	LIVRAISON.pizza_id	-
compose	PIZZA — INGREDIENT	PIZZA (1,n) — INGREDIENT (1,n)	Table associative PIZZA_INGREDIENT	-
utilise	LIVRAISON — LIVREUR	LIVRAISON (1,1) — LIVREUR (1,n)	LIVRAISON.livreur_id	-
roule	LIVREUR — VEHICULE	LIVREUR (1,n) — VEHICULE (1,n)	Aucun (association séparée)	-
est_de_taille	LIVRAISON — TAILLE	LIVRAISON (1,1) — TAILLE (1,n)	LIVRAISON.taille_code	-

3 . Modèle relationnel

Table CLIENT (

client_id PK,

nom,

prenom,

adresse,

email,

tel,

solde,

pizzas_payees_compteur

)

Table PIZZA (

pizza_id PK,

nom,

prix_base

)

Table INGREDIENT (

ingredient_id PK,

libelle

)

Table TAILLE (

taille_code PK,

libelle,

facteur_prix

)

Table LIVREUR (

livreur_id PK,

nom,

prenom

)

Table VEHICULE (

vehicule_id PK,

immatriculation,

type,

modele

)

Table LIVRAISON (

livraison_id PK,

client_id FK → CLIENT,

pizza_id FK → PIZZA,

taille_code FK → TAILLE,

livreur_id FK → LIVREUR,

vehicule_id FK → VEHICULE,

date_commande,

date_livraison,

prix_facture,

gratuite,

raison_gratuite

)

Table PIZZA_INGREDIENT (

pizza_id FK → PIZZA,

ingredient_id FK → INGREDIENT,

PRIMARY KEY (pizza_id, ingredient_id)

)

4 . Script SQL : création des tables (PostgreSQL-16)

```sql
CREATE TABLE client (
  client_id  SERIAL PRIMARY KEY,
  nom        VARCHAR(50) NOT NULL,
  prenom     VARCHAR(50) NOT NULL,
  adresse    TEXT NOT NULL,
  email      VARCHAR(120) UNIQUE NOT NULL,
  tel        VARCHAR(20),
  solde      NUMERIC(8,2) DEFAULT 0 CHECK (solde >= 0),
  pizzas_payees_compteur SMALLINT DEFAULT 0 CHECK (pizzas_payees_compteur >= 0)
);

CREATE TABLE pizza (
  pizza_id   SERIAL PRIMARY KEY,
  nom        VARCHAR(60) UNIQUE NOT NULL,
  prix_base  NUMERIC(5,2) NOT NULL CHECK (prix_base > 0)
);

CREATE TABLE ingredient (
  ingredient_id SERIAL PRIMARY KEY,
  libelle       VARCHAR(60) UNIQUE NOT NULL
);

CREATE TABLE pizza_ingredient (
  pizza_id      INT REFERENCES pizza(pizza_id) ON DELETE CASCADE,
  ingredient_id INT REFERENCES ingredient(ingredient_id) ON DELETE RESTRICT,
  PRIMARY KEY (pizza_id, ingredient_id)
);

CREATE TABLE taille (
  taille_code  CHAR(1) PRIMARY KEY,  -- N, H, O
  libelle      VARCHAR(20) NOT NULL,
  facteur_prix NUMERIC(4,3) NOT NULL CHECK (facteur_prix > 0)
);

CREATE TABLE livreur (
  livreur_id SERIAL PRIMARY KEY,
  nom        VARCHAR(50) NOT NULL,
  prenom     VARCHAR(50) NOT NULL
);
```

```sql
CREATE TABLE livreur (
  livreur_id SERIAL PRIMARY KEY,
  nom        VARCHAR(50) NOT NULL,
  prenom     VARCHAR(50) NOT NULL
);

CREATE TABLE vehicule (
  vehicule_id     SERIAL PRIMARY KEY,
  immatriculation VARCHAR(15) UNIQUE NOT NULL,
  type            VARCHAR(10) CHECK (type IN ('AUTO','MOTO')),
  modele          VARCHAR(60)
);

CREATE TABLE livraison (
  livraison_id SERIAL PRIMARY KEY,
  client_id    INT REFERENCES client(client_id),
  pizza_id     INT REFERENCES pizza(pizza_id),
  taille_code  CHAR(1) REFERENCES taille(taille_code),
  livreur_id   INT REFERENCES livreur(livreur_id),
  vehicule_id  INT REFERENCES vehicule(vehicule_id),
  date_commande   TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  date_livraison  TIMESTAMP,
  prix_facture NUMERIC(6,2) NOT NULL,
  gratuite     BOOLEAN NOT NULL DEFAULT FALSE,
  raison_gratuite VARCHAR(20)
);

CREATE INDEX idx_livraison_client ON livraison(client_id);
CREATE INDEX idx_livraison_vehicule ON livraison(vehicule_id);
```

5 . Script SQL : jeu d’essai minimal

```sql
-- Tailles
INSERT INTO taille VALUES
  ('N','Naine',0.666), ('H','Humaine',1.000), ('O','Ogresse',1.333);

-- Ingrédients
INSERT INTO ingredient (libelle) VALUES
 ('Tomate'),('Saumons'),('Jambon'),('Champignons'),('Olives'),
 ('Anchois'),('Poivrons'),('Merguez'),('Chorizo');

-- Pizzas
INSERT INTO pizza (nom,prix_base) VALUES
 ('Margherita',8.00),
 ('Reine',9.50),
 ('Chorizo Fiesta',10.50);

-- Composition
INSERT INTO pizza_ingredient VALUES
 -- Margherita
 (1,1),(1,2),
 -- Reine
 (2,1),(2,2),(2,3),(2,4),
 -- Chorizo Fiesta
 (3,1),(3,2),(3,8),(3,5);

-- Clients
INSERT INTO client (nom,prenom,adresse,email,solde) VALUES
 ('Regnier','Richard','1 rue des Lilas','richard@example.com',50),
 ('Ho','Ludovic','2 av. de Paris','ludovic@example.com',30);

-- Livreurs & véhicules
INSERT INTO livreur (nom,prenom) VALUES ('Lemoine','Céline'),('Perez','Diego');
INSERT INTO vehicule (immatriculation,type,modele) VALUES
 ('AB-123-CD','MOTO','Kawazaki 125'),
 ('EF-456-GH','AUTO','Toyota Yaris');
```

6 . Requêtes demandées

```sql
-- Menu
SELECT p.nom AS pizza,
       t.libelle AS taille,
       ROUND(p.prix_base * t.facteur_prix,2) AS prix,
       STRING_AGG(i.libelle, ', ' ORDER BY i.libelle) AS ingredients
FROM pizza p
JOIN pizza_ingredient pi ON pi.pizza_id = p.pizza_id
JOIN ingredient i        ON i.ingredient_id = pi.ingredient_id
CROSS JOIN taille t
GROUP BY p.pizza_id, p.nom, t.taille_code, t.libelle, t.facteur_prix
ORDER BY p.nom, t.taille_code;

-- Fiche de livraison
SELECT l.livraison_id,
       li.nom || ' ' || li.prenom      AS livreur,
       v.type || ' ' || v.modele       AS vehicule,
       c.nom || ' ' || c.prenom        AS client,
       l.date_commande,
       l.date_livraison,
       CASE WHEN l.date_livraison > l.date_commande + INTERVAL '30 minutes'
            THEN 'RETARD' ELSE 'OK' END AS statut,
       p.nom  AS pizza,
       p.prix_base                      AS prix_base
FROM livraison l
JOIN livreur li ON li.livreur_id = l.livreur_id
JOIN vehicule v ON v.vehicule_id = l.vehicule_id
JOIN client  c ON c.client_id = l.client_id
JOIN pizza   p ON p.pizza_id  = l.pizza_id
ORDER BY l.livraison_id;

-- Véhicules n'ayant jamais servi
SELECT v.*
FROM vehicule v
LEFT JOIN livraison l ON l.vehicule_id = v.vehicule_id
WHERE l.vehicule_id IS NULL;
```

```sql
-- Nombre de commandes par client
SELECT c.client_id, c.nom, c.prenom, COUNT(l.livraison_id) AS nb_commandes
FROM client c
LEFT JOIN livraison l ON l.client_id = c.client_id
GROUP BY c.client_id, c.nom, c.prenom;

-- Moyenne de commandes
WITH stats AS (
  SELECT COUNT(*) AS nb FROM livraison
)
SELECT AVG(nb) FROM (
  SELECT COUNT(*) AS nb
  FROM livraison
  GROUP BY client_id
) s;

-- Clients au-dessus de la moyenne
WITH nb_par_client AS (
  SELECT client_id, COUNT(*) AS nb
  FROM livraison
  GROUP BY client_id
),
  moyenne AS (
  SELECT AVG(nb) AS moy FROM nb_par_client
)
SELECT c.* , n.nb
FROM client c
JOIN nb_par_client n ON n.client_id = c.client_id
JOIN moyenne m ON n.nb > m.moy;
```

7 . Logique SQL : procédures & triggers

```sql
-- Procédure
CREATE OR REPLACE FUNCTION placer_livraison(
    _client INT,
    _pizza  INT,
    _taille CHAR(1),
    _livreur INT,
    _vehicule INT
) RETURNS INT AS $$
DECLARE
    _prix NUMERIC(6,2);
    _gratuite BOOLEAN := FALSE;
    _raison TEXT := NULL;
    _id INT;
BEGIN
  -- calcul du prix
  SELECT ROUND(p.prix_base * t.facteur_prix,2)
    INTO _prix
    FROM pizza p JOIN taille t ON t.taille_code = _taille
   WHERE p.pizza_id = _pizza;

  -- gestion fidélité
  SELECT pizzas_payees_compteur
    INTO STRICT _id
    FROM client WHERE client_id=_client;

  IF _id >= 9 THEN  -- 10e pizza
     _gratuite := TRUE;
     _raison   := 'FIDELITE';
     _prix     := 0;
     UPDATE client SET pizzas_payees_compteur = 0 WHERE client_id=_client;
  END IF;

  -- vérification solde
  IF NOT _gratuite THEN
    UPDATE client SET solde = solde - _prix,
                      pizzas_payees_compteur = pizzas_payees_compteur + 1
    WHERE client_id = _client AND solde >= _prix;
    IF NOT FOUND THEN
       RAISE EXCEPTION 'Solde insuffisant';
    END IF;
  END IF;
```

```sql
  -- insertion livraison
  INSERT INTO livraison (client_id,pizza_id,taille_code,livreur_id,vehicule_id,prix_facture,gratuite,raison_gratuite)
  VALUES (_client,_pizza,_taille,_livreur,_vehicule,_prix,_gratuite,_raison)
  RETURNING livraison_id INTO _id;

  RETURN _id;
END;$$ LANGUAGE plpgsql;

-- Trigger retard

CREATE OR REPLACE FUNCTION check_retard() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.date_livraison IS NOT NULL
     AND NEW.date_livraison > NEW.date_commande + INTERVAL '30 minutes'
     AND NEW.gratuite = FALSE THEN
       UPDATE client SET solde = solde + NEW.prix_facture
        WHERE client_id = NEW.client_id;
       NEW.gratuite := TRUE;
       NEW.raison_gratuite := 'RETARD';
       NEW.prix_facture := 0;
  END IF;
  RETURN NEW;
END;$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_retard
AFTER UPDATE OF date_livraison ON livraison
FOR EACH ROW EXECUTE FUNCTION check_retard();
```