Ce schéma relationnel permet de structurer efficacement les données de l'application tout en assurant la traçabilité des commandes, des paiements et des opérations en cuisine.

Suite

Le diagramme de classes correspondant au schéma relationnel proposé pour l'application de gestion de restaurant en Java pourrait avoir les classes suivantes avec leurs attributs et méthodes. Voici une représentation textuelle, et ensuite je décrirai comment les classes interagissent entre elles.

### 1. \*\*Classe `Utilisateur`\*\*

```java

class Utilisateur {

int idUtilisateur;

String nom;

String email;

String motDePasse;

String role; // Administrateur, Serveur, Cuisinier, Caissier

// Méthodes

void ajouterUtilisateur();

void modifierUtilisateur();

void supprimerUtilisateur();

void connecter();

}

```

### 2. \*\*Classe `Menu`\*\*

```java

class Menu {

int idMenu;

String nom;

String description;

String categorie; // Entrée, Plat principal, Dessert, Boisson

double prix;

// Méthodes

void ajouterPlat();

void modifierPlat();

void supprimerPlat();

}

```

### 3. \*\*Classe `TableRestaurant`\*\*

```java

class TableRestaurant {

int idTable;

String numero;

String statut; // Libre, Occupée, Réservée

Utilisateur serveur; // FK vers la classe Utilisateur (Serveur)

// Méthodes

void reserverTable();

void libererTable();

}

```

### 4. \*\*Classe `Commande`\*\*

```java

class Commande {

int idCommande;

TableRestaurant table; // FK vers la classe TableRestaurant

Utilisateur serveur; // FK vers la classe Utilisateur (Serveur)

Date dateCommande;

String statut; // En préparation, Servi

double total;

// Méthodes

void ajouterCommande();

void modifierCommande();

void supprimerCommande();

void changerStatut();

}

```

### 5. \*\*Classe `CommandeDetail`\*\*

```java

class CommandeDetail {

int idCommandeDetail;

Commande commande; // FK vers la classe Commande

Menu plat; // FK vers la classe Menu

int quantite;

double prixUnitaire;

// Méthodes

void ajouterDetail();

void modifierDetail();

void supprimerDetail();

}

```

### 6. \*\*Classe `Facture`\*\*

```java

class Facture {

int idFacture;

Commande commande; // FK vers la classe Commande

Date dateFacture;

double montantTotal;

String modePaiement; // Espèces, Carte, Mobile

// Méthodes

void genererFacture();

void imprimerFacture();

}

```

### 7. \*\*Classe `Stock`\*\*

```java

class Stock {

int idStock;

Menu plat; // FK vers la classe Menu

int quantiteDisponible;

// Méthodes

void ajouterStock();

void modifierStock();

void supprimerStock();

}

```

---

### Relations entre les classes :

- \*\*Utilisateur\*\* : classe centrale qui interagit avec les tables et les commandes. Les serveurs sont liés aux tables qu'ils gèrent.

- \*\*Menu\*\* : chaque élément du menu est utilisé dans les commandes et affecte aussi la gestion du stock.

- \*\*TableRestaurant\*\* : représente les tables du restaurant, gérées par les serveurs. Chaque table peut avoir plusieurs commandes au fil du temps.

- \*\*Commande\*\* : représente les commandes passées par les serveurs pour une table donnée. Chaque commande a plusieurs détails (éléments du menu).

- \*\*CommandeDetail\*\* : décrit les plats et boissons spécifiques dans chaque commande.

- \*\*Facture\*\* : générée à partir des commandes, elle représente la somme à payer et le mode de paiement choisi.

- \*\*Stock\*\* : permet de gérer la quantité disponible des éléments du menu.

---

### Diagramme de classes visuel (description textuelle)

Voici une manière d’imaginer le diagramme de classes :

- \*\*Classes avec relations d'agrégation :\*\*

- `Commande` a une relation d'agrégation avec `TableRestaurant` (une table peut avoir plusieurs commandes) et avec `Utilisateur` (le serveur qui passe la commande).

- `CommandeDetail` a une relation d'agrégation avec `Commande` (une commande peut avoir plusieurs détails).

- `Facture` est liée à `Commande`, car chaque facture est associée à une commande unique.

- \*\*Classes avec associations simples :\*\*

- `Menu` est associé à `CommandeDetail` (chaque détail d'une commande correspond à un élément du menu).

- `Stock` est associé à `Menu`, car il gère les quantités des éléments disponibles dans le menu.

Ce modèle de classes reflète la structure de base de l'application de gestion d'un restaurant avec un système de facturation et une gestion complète des commandes et du personnel.

AVEC LE CLIENT

Pour permettre aux clients de \*\*réserver des tables\*\* et \*\*passer des commandes à distance\*\*, il faut étendre l'application avec de nouvelles fonctionnalités, classes et relations. Cela nécessite de prendre en compte la gestion des utilisateurs externes (clients), l'intégration de commandes en ligne et la gestion des réservations via une interface (desktop ou web). Voici les fonctionnalités, les modifications dans le diagramme de classes et le schéma relationnel adaptés.

### Fonctionnalités supplémentaires pour les clients

#### 1. \*\*Réservation de table à distance\*\*

- Les clients peuvent voir les tables disponibles et réserver à distance pour un créneau horaire spécifique.

- Suivi et gestion des réservations.

- Confirmation par email ou notification.

#### 2. \*\*Commande en ligne\*\*

- Consultation du menu à distance (via une application ou un site web).

- Passer des commandes en ligne pour livraison ou à emporter.

- Suivi de la commande (préparation, livraison, etc.).

- Paiement en ligne (intégration d'un système de paiement).

#### 3. \*\*Historique des commandes et des réservations\*\*

- Les clients peuvent consulter l'historique de leurs commandes et réservations passées.

---

### Nouvelles classes et modifications pour les clients à distance

#### 1. \*\*Classe `Client`\*\*

Cette nouvelle classe représente les clients enregistrés dans le système, avec la possibilité de s'authentifier et d'interagir avec les services à distance.

```java

class Client {

int idClient;

String nom;

String email;

String motDePasse;

String adresse; // Pour la livraison

String telephone;

// Méthodes

void creerCompte();

void seConnecter();

void reserverTable();

void passerCommande();

void consulterHistorique();

}

```

#### 2. \*\*Classe `Reservation`\*\*

Gère les réservations effectuées par les clients, soit directement dans le restaurant, soit à distance.

```java

class Reservation {

int idReservation;

Client client; // FK vers la classe Client

TableRestaurant table; // FK vers la classe TableRestaurant

Date dateReservation;

String statut; // Confirmée, Annulée, Terminée

// Méthodes

void creerReservation();

void modifierReservation();

void annulerReservation();

}

```

#### 3. \*\*Modification de la classe `Commande`\*\*

Ajouter une relation avec la classe `Client` pour que les commandes puissent être passées par les clients à distance. Les clients peuvent ainsi passer des commandes en ligne et les lier à leur compte.

```java

class Commande {

int idCommande;

TableRestaurant table; // FK vers la classe TableRestaurant

Utilisateur serveur; // FK vers la classe Utilisateur (Serveur)

Client client; // FK vers la classe Client (nouvelle relation)

Date dateCommande;

String statut; // En préparation, Servi, Livré

double total;

// Méthodes

void ajouterCommande();

void modifierCommande();

void supprimerCommande();

void changerStatut();

void passerCommandeEnLigne(); // Nouvelle méthode pour la commande à distance

}

```

#### 4. \*\*Classe `Paiement`\*\*

Gère les paiements en ligne ou en personne pour les commandes et les réservations.

```java

class Paiement {

int idPaiement;

Commande commande; // FK vers la classe Commande

Client client; // FK vers la classe Client

double montant;

String modePaiement; // Carte, Mobile, Espèces (pour les commandes en personne)

String statut; // Réussi, Échoué

// Méthodes

void traiterPaiement();

void verifierPaiement();

}

```

---

### Schéma relationnel modifié

Pour intégrer les clients à distance et les nouvelles fonctionnalités, voici les nouvelles tables et relations :

1. \*\*Table `Client`\*\*

- `id\_client` (PK)

- `nom`

- `email`

- `mot\_de\_passe`

- `adresse`

- `telephone`

2. \*\*Table `Reservation`\*\*

- `id\_reservation` (PK)

- `id\_client` (FK vers `Client`)

- `id\_table` (FK vers `Table\_Restaurant`)

- `date\_reservation`

- `statut` (Confirmée, Annulée, Terminée)

3. \*\*Modification de la table `Commande`\*\*

- Ajout d'une colonne `id\_client` (FK vers `Client`).

4. \*\*Table `Paiement`\*\*

- `id\_paiement` (PK)

- `id\_commande` (FK vers `Commande`)

- `id\_client` (FK vers `Client`)

- `montant`

- `mode\_paiement` (Carte, Mobile, Espèces)

- `statut`

---

### Diagramme de classes mis à jour

- \*\*Classe `Client`\*\* : Le client peut réserver une table ou passer une commande en ligne. Il interagit avec les classes `Reservation`, `Commande`, et `Paiement`.

- \*\*Classe `Reservation`\*\* : Nouvelle classe associée à `Client` et à `TableRestaurant`. Elle gère la réservation des tables à distance ou en personne.

- \*\*Classe `Commande`\*\* : La relation avec `Client` permet au client de passer des commandes à distance. Cette classe inclut également la gestion des paiements.

- \*\*Classe `Paiement`\*\* : Gère les paiements en ligne et en personne pour les commandes et les réservations.

### Nouvelles interactions

1. \*\*Clients (via application web ou mobile)\*\*

- \*\*Réservation\*\* : Les clients peuvent voir les tables disponibles, réserver et recevoir des confirmations.

- \*\*Commande\*\* : Les clients peuvent passer une commande en ligne pour livraison ou à emporter.

- \*\*Paiement\*\* : Les clients peuvent payer en ligne via carte ou autres moyens électroniques.

2. \*\*Serveurs (via l'application desktop)\*\*

- Gèrent les commandes en personne et surveillent les commandes passées à distance.

3. \*\*Administrateurs (via l'application desktop)\*\*

- Surveillent toutes les réservations, commandes et paiements.

---

Avec ces modifications, l'application devient plus complète en permettant une interaction à distance pour les clients tout en conservant la gestion locale au sein du restaurant