

**Projet de cours:**  
**Livrable 2 – Rapport final**

**CSI2532 – Base de données**  
**Hiver 2024**

**École de génie électrique et d'informatique**

**Professeur: Kalonji H. Kalala**

**Membres du groupe 22:**

Choueb Salah (8749973)

Darine Fourar Laidi (300203113)

Fatima Al-Zahra Ghadbawi (300301842)

Rahima Salah (300287692)

**Assistants d'enseignement:**

Ali Abdeddaïm

Amina Mahamane

Tawfiq Abubaker

**Date de soumission : 6/4/24**

**a. Le SGBD et les langages de programmation que vous avez utilisés dans votre implémentation de l'application.**

- Le SGBD qui a été utilisé pour l'implémentation de l'application est **PostgreSQL**.
- Les langages de programmation qui ont été utilisés pour l'implémentation de l'application sont: **JavaScript , EJS , CSS**.
- Pour le développement du backend de notre application, nous avons choisi **Node.js**, une plateforme logicielle open-source qui permet l'exécution de code JavaScript côté serveur. Node.js nous a permis de développer une API RESTful en utilisant le framework Express.js ainsi que le module postgres pour communiquer avec notre base de données PostgreSQL, offrant ainsi une interface flexible et puissante pour les opérations de base de données. Nous avons aussi utilisé le body-parser middleware pour analyser les requêtes entrantes du côté serveur.

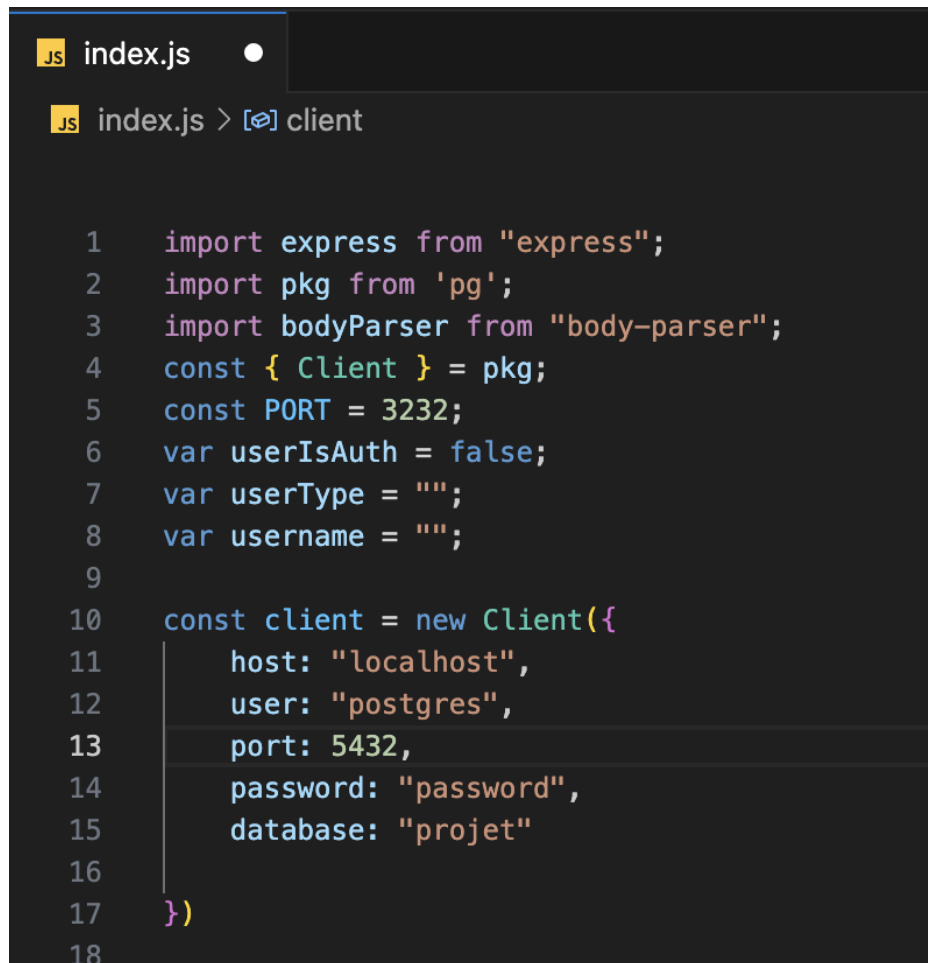
**b. Étapes spécifiques pour guider quelqu'un dans l'installation de vos applications.**

**Instructions pour la base de données:**

1. Créer une nouvelle base de données PostgreSQL nommé « projet ».
2. Créer un schema nommé « projet » dans cette même base de données.
3. Afin d'utiliser ce nouveau schema créé comme schema par défaut pour les requêtes au lieu du schema « public », nous devons d'abord exécuter la phrase suivante dans l'outil SQL: SET search\_path = "projet".
4. S'assurer d'intégrer dans la base de données «projet» les fichiers SQL dans l'ordre suivant:
  - Schemas.sql
  - Constraints.sql
  - Data.sql
  - Triggers.sql
  - Indexes.sql
  - Vues.sql

### Instructions pour l'application:

1. Télécharger et installer le fichier compressé «projetSQL2»
2. Ouvrir le fichier dans Visual Studio Code ou dans votre IDE favori
3. Ouvrez le fichier index.js du serveur et attribuez un numéro de port à la constante PORT si le port 3232 est déjà utilisé.



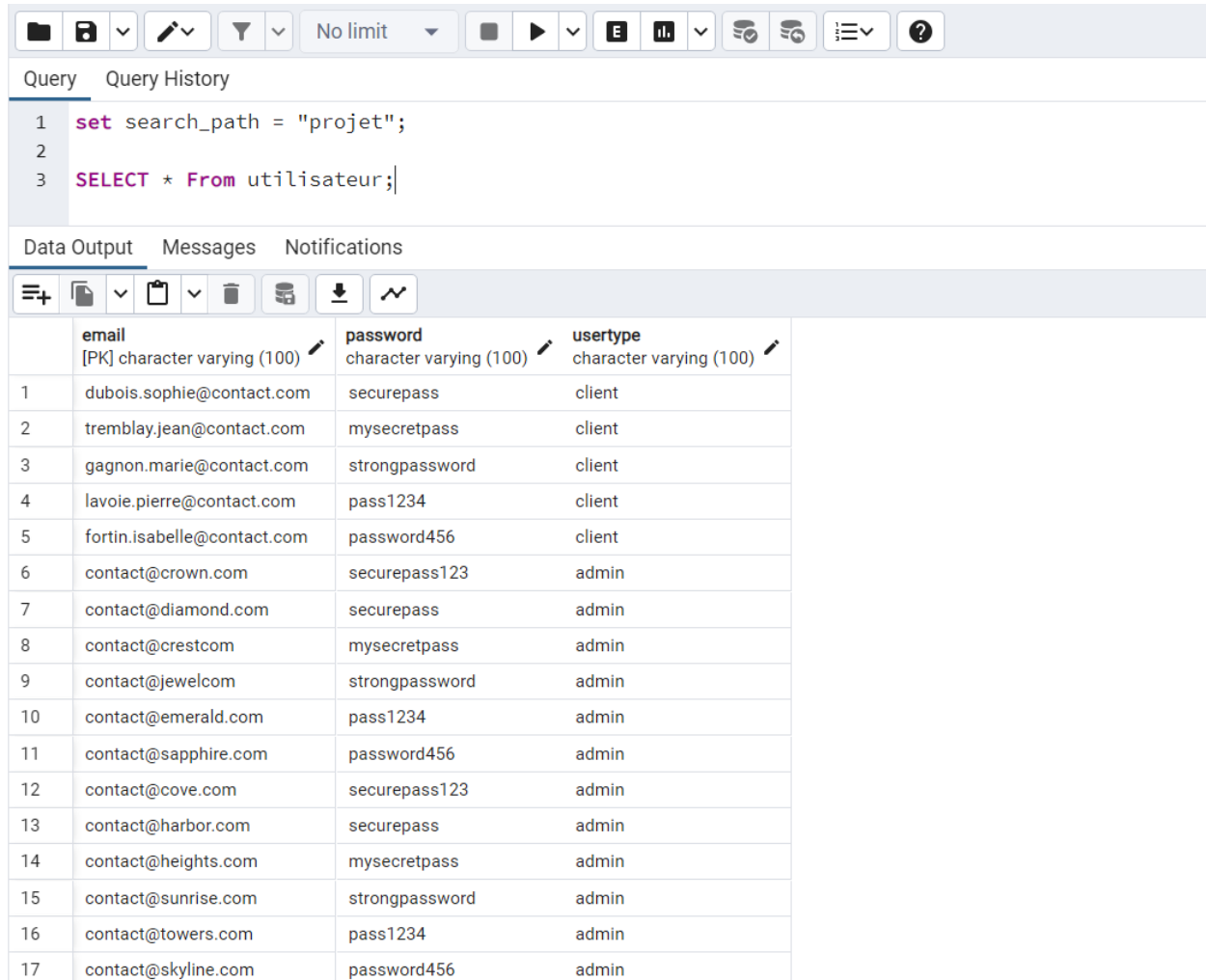
```
JS index.js
JS index.js > [🔗] client

1  import express from "express";
2  import pkg from 'pg';
3  import bodyParser from "body-parser";
4  const { Client } = pkg;
5  const PORT = 3232;
6  var userIsAuth = false;
7  var userType = "";
8  var username = "";
9
10 const client = new Client({
11   host: "localhost",
12   user: "postgres",
13   port: 5432,
14   password: "password",
15   database: "projet"
16 })
17
18
```

4. Si vous avez modifié le port par défaut pour PostgreSQL, veuillez le spécifier sous l'objet client dans l'attribut port.
5. Sous l'attribut password de l'objet client inscrivez votre mot de passe associé à PostgreSQL.
6. Assurez-vous d'avoir installé Node.js si ce n'est pas déjà fait.
7. Ouvrez un terminal dans votre IDE, en vous assurant d'être dans le répertoire du projet SQL, puis saisissez la commande npm i.
8. Exécutez la commande node index.js dans le terminal.

9. Ouvrez un navigateur et accédez à l'URL <http://localhost:3232>. Modifiez la valeur 3232 pour correspondre au port associé à la constante PORT du serveur.
10. Vous êtes maintenant en ligne. Connectez-vous à un compte client ou administrateur (voir ci-dessous)

Afin de vous connecter à un compte client ou administrateur, veuillez utiliser les comptes qui existent dans la table *utilisateur*. Il existe un *usertype* (client ou admin) pour chaque compte. Ils peuvent être obtenus en utilisant la requête: *SELECT \* FROM utilisateur*. Voici un exemple de l'implémentation de cette requête ainsi qu'une partie des résultats:



The screenshot shows a database client interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 set search_path = "projet";
2
3 SELECT * From utilisateur;
```

The results pane displays the output of the query, showing 17 rows of data from the *utilisateur* table. The columns are *email*, *password*, and *usertype*.

	email [PK] character varying (100)	password character varying (100)	usertype character varying (100)
1	dubois.sophie@contact.com	securepass	client
2	tremblay.jean@contact.com	mysecretpass	client
3	gagnon.marie@contact.com	strongpassword	client
4	lavoie.pierre@contact.com	pass1234	client
5	fortin.isabelle@contact.com	password456	client
6	contact@crown.com	securepass123	admin
7	contact@diamond.com	securepass	admin
8	contact@crestcom	mysecretpass	admin
9	contact@jewelcom	strongpassword	admin
10	contact@emerald.com	pass1234	admin
11	contact@sapphire.com	password456	admin
12	contact@cove.com	securepass123	admin
13	contact@harbor.com	securepass	admin
14	contact@heights.com	mysecretpass	admin
15	contact@sunrise.com	strongpassword	admin
16	contact@towers.com	pass1234	admin
17	contact@skyline.com	password456	admin

c. Une liste avec les DDL qui créent votre base de données.

## 1. CREATE TABLE

```
CREATE TABLE Role(
ID_role INTEGER,
Nom_role VARCHAR(100),
```

```

Description VARCHAR(255),
Salaire INTEGER,
PRIMARY KEY (ID_role));

CREATE TABLE Gestionnaire(
ID_gestionnaire INTEGER,
Nom VARCHAR(100),
Prenom VARCHAR(100),
Nom_hotel VARCHAR(100),
Adresse VARCHAR(100),
Email VARCHAR(100),
Num_tel VARCHAR(100),
PRIMARY KEY (ID_gestionnaire));

CREATE TABLE ChaîneHoteliere(
ID_chaine INTEGER,
Nom_chaine VARCHAR(100),
Adresse VARCHAR(100),
Nombre_hotel INTEGER,
Email VARCHAR(100),
Num_tel VARCHAR(100),
PRIMARY KEY (ID_chaine));

CREATE TABLE Hotel(
ID_Hotel INTEGER,
ID_chaine INTEGER,
Nom_hotel VARCHAR(100),
Nombre_chambre INTEGER,
Adresse VARCHAR(100),
Num_tel VARCHAR(100),
Nombre_etoile INTEGER,
Email VARCHAR(100),
Categorie VARCHAR(100),
Nom_chaine VARCHAR(100),
Description VARCHAR(255),
PRIMARY KEY (ID_Hotel),
FOREIGN KEY (ID_chaine) REFERENCES ChaîneHoteliere);

CREATE TABLE Employee(
NAS_employee INTEGER,
ID_role INTEGER,
Nom VARCHAR(100),
Prenom VARCHAR(100),
Adresse VARCHAR(100),
Num_tel VARCHAR(100),

```

```

Nom_hotel VARCHAR(100),
PRIMARY KEY (NAS_employee),
FOREIGN KEY (ID_role) REFERENCES Role);

CREATE TABLE Client(
NAS_client INTEGER,
Nom VARCHAR(100),
Prenom VARCHAR(100),
Age INTEGER,
Adresse VARCHAR(100),
Num_tel VARCHAR(100),
Date_enregistrement DATE,
email VARCHAR(100),
PRIMARY KEY (NAS_client));

CREATE TABLE Chambre(
num_chambre INTEGER,
ID_hotel INTEGER,
Prix Numeric(10,2),
Commodites VARCHAR(100),
Capacite INTEGER,
Vue VARCHAR(100),
Extensible BOOLEAN,
Problemes BOOLEAN,
Disponibilite BOOLEAN,
Superficie_m2 VARCHAR(100),
Description VARCHAR(255),
Nom_hotel VARCHAR(100),
PRIMARY KEY (num_chambre),
FOREIGN KEY (ID_hotel) REFERENCES Hotel);

CREATE TABLE Reservation(
ID_reservation INTEGER,
num_chambre INTEGER,
NAS_client INTEGER,
Date_reservation DATE,
Date_arrivee DATE,
Date_depart DATE,
Statut VARCHAR(100),
PRIMARY KEY (ID_reservation),
FOREIGN KEY (num_chambre) REFERENCES Chambre,
FOREIGN KEY (NAS_client) REFERENCES Client);

CREATE TABLE Location(
ID_location INTEGER,

```

```

num_chambre INTEGER,
NAS_client INTEGER,
NAS_employee INTEGER,
Date_location DATE,
Date_arrivee DATE,
Date_depart DATE,
Statut VARCHAR(100),
PRIMARY KEY (ID_location),
FOREIGN KEY (num_chambre) REFERENCES Chambre,
FOREIGN KEY (NAS_client) REFERENCES Client,
FOREIGN KEY (NAS_employee) REFERENCES Employee);

CREATE TABLE Paiement(
ID_paiement INTEGER,
ID_location INTEGER,
Montant NUMERIC(10,2),
Date_paiement DATE,
Methode_paiement VARCHAR(100),
Credit_card_number VARCHAR(20),
PRIMARY KEY (ID_paiement),
FOREIGN KEY (ID_location) REFERENCES Location);

CREATE TABLE Utilisateur(
    email VARCHAR(100),
    password VARCHAR(100),
    userType VARCHAR(100),
    PRIMARY KEY (email));

```

Code SQL pour la création des tables pour la base de données.

## 2. ALTER TABLE

```

ALTER TABLE Gestionnaire
ADD CONSTRAINT CHK_FormatEmail_Gestionnaire CHECK (email LIKE '%_@_%.%');

ALTER TABLE ChaineHoteliere
ADD CONSTRAINT CHK_FormatEmail_Chaine CHECK (email LIKE '%_@_%.%');

ALTER TABLE Hotel
ADD CONSTRAINT CHK_FormatEmail_Hotel CHECK (email LIKE '%_@_%.%');

ALTER TABLE client
ADD CONSTRAINT CHK_FormatEmail_client CHECK (email LIKE '%_@_%.%');

```

```

ALTER TABLE Client ADD CONSTRAINT CHK_FormatTelephone_Client CHECK (num_tel
SIMILAR TO '(\+?\d{1,3}\s?)?(?(\d{3}\s?)?[\s.-]? \d{3}[\s.-]? \d{4}');

ALTER TABLE Employee ADD CONSTRAINT CHK_FormatTelephone_Employee CHECK (num_tel
SIMILAR TO '(\+?\d{1,3}\s?)?(?(\d{3}\s?)?[\s.-]? \d{3}[\s.-]? \d{4}');

ALTER TABLE Gestionnaire ADD CONSTRAINT CHK_FormatTelephone_Gestionnaire CHECK
(num_tel SIMILAR TO '(\+?\d{1,3}\s?)?(?(\d{3}\s?)?[\s.-]? \d{3}[\s.-]? \d{4}');

ALTER TABLE ChaineHoteliere ADD CONSTRAINT CHK_FormatTelephone_Chaine CHECK
(num_tel SIMILAR TO '(\+?\d{1,3}\s?)?(?(\d{3}\s?)?[\s.-]? \d{3}[\s.-]? \d{4}');

ALTER TABLE Hotel ADD CONSTRAINT CHK_FormatTelephone_Hotel CHECK (num_tel SIMILAR
TO '(\+?\d{1,3}\s?)?(?(\d{3}\s?)?[\s.-]? \d{3}[\s.-]? \d{4}');

ALTER TABLE Reservation ADD CONSTRAINT CHK_FormatDateArrivee CHECK
(to_char(date_arrivee, 'YYYY-MM-DD') ~ '^ \d{4}-\d{2}-\d{2}$');

ALTER TABLE Reservation ADD CONSTRAINT CHK_FormatDateDepart CHECK
(to_char(date_depart, 'YYYY-MM-DD') ~ '^ \d{4}-\d{2}-\d{2}$');

ALTER TABLE Location ADD CONSTRAINT CHK_FormatDateArrivee CHECK
(to_char(date_arrivee, 'YYYY-MM-DD') ~ '^ \d{4}-\d{2}-\d{2}$');

ALTER TABLE Location ADD CONSTRAINT CHK_FormatDateDepart CHECK
(to_char(date_depart, 'YYYY-MM-DD') ~ '^ \d{4}-\d{2}-\d{2}$');

ALTER TABLE paiement
ADD CONSTRAINT CHK_FormatCreditCardNumber
CHECK (credit_card_number ~ '^ \d{4}-\d{4}-\d{4}-\d{4}$');

ALTER TABLE Hotel ADD CONSTRAINT CHK_RatingRange CHECK (nombre_etoile >= 1 AND
nombre_etoile <= 5);

ALTER TABLE Client
ADD CONSTRAINT CHK_NAS_Client_Length CHECK (NAS_Client >= 100000000 AND
NAS_Client <= 999999999);
ALTER TABLE Client
ADD CONSTRAINT UNQ_NAS_Client_Unique UNIQUE (NAS_Client);

```



```

ALTER TABLE Employee
ADD CONSTRAINT CHK_NAS_Employee_Length CHECK (NAS_employee >= 100000000 AND
NAS_employee <= 999999999);
ALTER TABLE Employee
ADD CONSTRAINT UNQ_NAS_Employee_Unique UNIQUE (NAS_employee);

ALTER TABLE Client
ADD CONSTRAINT check_age CHECK (age >= 18);

ALTER TABLE Reservation
ADD CONSTRAINT CHK_Reservation_Date CHECK (Date_Depart > Date_Arrivee);

ALTER TABLE Location
ADD CONSTRAINT CHK_Location_Date CHECK (Date_Depart > Date_Arrivee);

ALTER TABLE Paiement
ADD CONSTRAINT CHK_MethodePaiementValide CHECK (methode_paiement IN
('Mastercard', 'Visa'));

ALTER TABLE Reservation ADD CONSTRAINT CHK_ReservationUnique UNIQUE (num_chambre,
date_arrivee, date_depart);

```

Code SQL pour l'ajout des contraintes dans la base de données.

### 3. CREATE OR REPLACE FUNCTION et CREATE TRIGGER

```

CREATE OR REPLACE FUNCTION update_nombre_chambres_disponibles()
RETURNS TRIGGER AS $$
DECLARE
    hotel_id INT;
BEGIN
    SELECT id_hotel INTO hotel_id FROM Chambre WHERE num_chambre =
NEW.num_chambre LIMIT 1;

    IF hotel_id IS NOT NULL THEN
        UPDATE Hotel
        SET nombre_chambre = nombre_chambre - 1
        WHERE id_hotel = hotel_id;
    END IF;

    RETURN NEW;
END;

```

```

$$ LANGUAGE plpgsql;

CREATE TRIGGER update_nombre_chambres_disponibles
AFTER INSERT ON Reservation
FOR EACH ROW
EXECUTE FUNCTION update_nombre_chambres_disponibles();

CREATE OR REPLACE FUNCTION restore_nombre_chambres_disponibles()
RETURNS TRIGGER AS $$
DECLARE
    hotel_id INT;
BEGIN
    SELECT id_hotel INTO hotel_id FROM Chambre WHERE num_chambre =
OLD.num_chambre LIMIT 1;

    IF hotel_id IS NOT NULL THEN
        UPDATE Hotel
        SET nombre_chambre = nombre_chambre + 1
        WHERE id_hotel = hotel_id;
    END IF;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restore_nombre_chambres_disponibles
AFTER DELETE ON Reservation
FOR EACH ROW
EXECUTE FUNCTION restore_nombre_chambres_disponibles();

```

Code SQL pour la création des Triggers dans la base de données.

- **update\_nombre\_chambres\_disponibles**: soustraire par 1 le nombre de chambres disponibles (nombre\_chambre) dans la table hotel lorsqu'il y a une réservation ajouté.
- **restore\_nombre\_chambres\_disponibles**: ajouter 1 au nombre de chambres disponibles (nombre\_chambre) dans la table hotel à chaque fois qu'il y a une réservation qui se fait supprimé.

#### 4. CREATE OR REPLACE VIEW

```

--Vue 1: Nombre de chambres disponibles par zone
--Cette vue donnera le nombre de chambres disponibles dans chaque hôtel, groupées
par zone.

```

```

CREATE OR REPLACE VIEW chambres_disponibles_par_zone AS
SELECT H.Adresse AS Zone,
       H.Nom_hotel,
       H.Nombre_chambre AS Nombre_Chambres_Disponibles
FROM Hotel H
WHERE EXISTS (
    SELECT 1
    FROM Chambre C
    WHERE H.ID_Hotel = C.ID_hotel
    AND C.Disponibilite = TRUE
);

--Vue 2: Capacité de toutes les chambres d'un hôtel spécifique
CREATE OR REPLACE VIEW capacite_par_chambre_par_hotel AS
SELECT C.num_chambre,
       C.Capacite,
       H.Nom_hotel
FROM Chambre C
INNER JOIN Hotel H ON C.ID_hotel = H.ID_Hotel;

```

Code SQL pour la création des Vues dans la base de données.

- **chambres\_disponible\_par\_zone:** cette vue donne le nombre de chambres disponibles dans chaque hotel, groupées par zone.
- **capacite\_par\_chambre\_par\_hotel:** cette vue donne la capacité de toutes les chambres d'un hôtel spécifique.

## 5. CREATE INDEX

```

CREATE INDEX idx_reservation_date_arrivee_depart ON Reservation(date_arrivee,
date_depart) WHERE statut = 'confirmed';

CREATE INDEX idx_location_date_arrivee_depart ON Location(date_arrivee,
date_depart) WHERE statut = 'confirmed';

CREATE INDEX idx_client_id ON Client(NAS_client);

CREATE INDEX idx_chambre_hotel_id ON Chambre(ID_hotel);

CREATE INDEX idx_paiement_methode ON Paiement(methode_paiement);

```

Code SQL pour la création des index dans la base de données.

**Justification de l'utilisation de ces index:**

- **idx\_reservation\_date\_arrivee\_depart:** cet index est créé sur la table réservation avec les colonnes date\_arrivee et date\_depart. La clause WHERE restreint l'index pour seulement inclure les rangées où la colonne statut est 'confirmed'. Cet index peut être utile pour les requêtes qui incluent un filtrage ou un classement par date\_arrivee et date\_depart pour les réservations qui sont confirmées. Ceci peut accélérer les requêtes impliquant l'analyse de ces colonnes.
- **idx\_location\_date\_arrivee\_depart:** cet index est créé sur la table location avec les colonnes date\_arrivee et date\_depart. La clause WHERE restreint l'index pour seulement inclure les rangées où la colonne statut est 'confirmed'. Cet index peut être utile pour les requêtes qui incluent un filtrage ou un classement par date\_arrivee et date\_depart pour les locations qui sont confirmées. Ceci peut accélérer les requêtes impliquant l'analyse de ces colonnes.
- **idx\_client\_id:** Cet index a été créé sur la table client avec la colonne NAS\_client. L'indexation de cette colonne va permettre d'accélérer les requêtes qui nécessitent un filtrage ou une jointure sur NAS\_client. Ainsi, ça peut améliorer les performances des requêtes qui récupèrent l'information des clients selon le NAS\_client.
- **idx\_chambre\_hotel\_id:** Cet index a été créé sur la table chambre avec la colonne id\_hotel. L'indexation de cette colonne peut être utile pour les requêtes qui nécessitent un filtrage ou une jointure sur id\_hotel, par exemple, la récupération de toutes chambres qui appartiennent à un hôtel spécifique. Elle peut également être utile pour optimiser les requêtes qui nécessitent des jointures entre les tables chambres et hôtel selon id\_hotel.
- **idx\_paiement\_methode:** Cet index a été créé sur la table paiement avec la colonne methode\_paiement. L'indexation de cette colonne peut améliorer la performance des requêtes qui nécessitent un filtrage ou une jointure sur methode\_paiement, par exemple, la récupération de paiements effectués avec une méthode de paiement spécifique. Ça peut accélérer les recherches qui nécessitent cette colonne.

### Exemples de requêtes:

```
INSERT INTO Employee(nas_employee,id_role,nom,prenom,adresse,num_tel,nom_hotel)
VALUES (800000000, 1001,'Marie','John','888 Forest Drive,Vancouver, British
Columbia, Canada V5C 0E6','604-433-7890','Tranquil Oasis Hotel');

INSERT INTO
client(nas_client,nom,prenom,age,adresse,num_tel,date_enregistrement,email)
VALUES (993394444, 'James', 'Pierre', 34, '555 York Street, Toronto, ON, Canada
M6J 4E3', '416-777-7990',CURRENT_DATE, 'james.pierre@contact.com');

INSERT INTO
reservation(id_reservation,num_chambre,nas_client,date_reservation,date_arrivee,d
ate_depart,statut)
```

```
VALUES
    (02,27,993394444,'2024-04-18','2024-04-19','2024-04-23','pending');

DELETE FROM Reservation WHERE id_reservation = 456;

UPDATE Hotel SET nombre_etoile = 5 WHERE id_hotel = 123;

UPDATE Client SET adresse = '222 Willow Way, Vancouver, British Columbia, Canada
V5B 8E9' WHERE NAS_client = 999911111;
```

Code SQL pour des exemples de requêtes dans la base de données.

- L'ajout d'un nouveau employé.
- L'ajout d'un nouveau client.
- L'ajout d'une nouvelle réservation.
- Supprimer une réservation.
- Modifier les informations reliées à un hôtel spécifique, par exemple, le nombre d'étoiles.
- Modifier les information reliés à un client spécifique, par exemple, l'adresse.

```
UPDATE Client
SET nom = 'new name', prenom = 'new surname', adresse = 'new address'
WHERE NAS_client = 'Client_NAS';

UPDATE Hotel
SET nom_hotel = 'new hotel name', adresse = 'new hotel address', nombre_etoile =
'newStarRating'
WHERE ID_Hotel = 'Hotel_ID';
```

Code SQL pour modifier les informations d'un client et d'un hôtel.

```
SELECT h.ID_Hotel, h.Nom_hotel, ch.num_chambre
FROM Hotel h
JOIN Chambre ch ON h.ID_Hotel = ch.ID_hotel
WHERE ch.Disponibilite = TRUE
AND NOT EXISTS (
    SELECT 1
    FROM Reservation r
    WHERE r.num_chambre = ch.num_chambre
    AND (
        (r.Date_Arrivee < '2024-04-06' AND r.Date_Depart > '2024-04-04')
        OR (('2024-04-04' BETWEEN r.Date_Arrivee AND r.Date_Depart)
        OR ('2024-04-06' BETWEEN r.Date_Arrivee AND r.Date_Depart))
    )
)
```

```
ORDER BY h.ID_Hotel, ch.num_chambre;
```

Code SQL pour trouver les chambres disponibles dans les hotels selon une date d'arrivée et une date de départ spécifique.

```
SELECT l.*, h.nom_hotel, c.ID_hotel
FROM Location l
INNER JOIN Chambre c ON l.num_chambre = c.num_chambre
INNER JOIN Hotel h ON c.ID_hotel = h.ID_Hotel;
```

Code SQL pour une requête qui trouve toutes les informations sur les locations en utilisant le nom de l'hôtel (nom\_hotel) et l'identifiant du hotel (ID\_hotel).

```
SELECT r.*, h.nom_hotel, c.ID_hotel
FROM Reservation r
INNER JOIN Chambre c ON r.num_chambre = c.num_chambre
INNER JOIN Hotel h ON c.ID_hotel = h.ID_Hotel;
```

Code SQL pour une requête qui trouve toutes les informations sur les réservations en utilisant le nom de l'hôtel (nom\_hotel) et l'identifiant du hotel (ID\_hotel).