CEG 2532 - Bases de données I

Projet - Livrable 2

Date de remise: 6 avril 2024

Felix Allard - 300146250

Jean Markello Létang - 300245679

Emma Seaibi - 300273060

Samuel Rose - 300173591

Table des matières

Le SGBD et les langages de programmation que nous avons utilisés dans notre application

Dans le cadre de notre projet, notre équipe a adopté la pile technologique PERN pour le développement de notre application web. PERN est un acronyme qui résume les technologies principales utilisées: **PostgreSQL, Express, React et Node.js**. Cette combinaison offre un ensemble cohérent et puissant pour le développement full stack d'applications web modernes.

PostgreSQL est un SGBD très populaire et reconnu pour sa robustesse, sa conformité aux standards SQL, et sa capacité à gérer efficacement de grandes quantités de données. Il sert de fondement solide pour le stockage et la manipulation des données de notre application. Bien que notre base de données était gérée sur PostgreSQL, nous avons décidé de la faire "hoster" dans le cloud avec les services Google Cloud SQL.

Express est un framework pour Node.js qui simplifie la création de serveurs web et la gestion des requêtes HTTP. Il fournit un ensemble d'outils robuste pour le développement backend, facilitant la définition des routes, la gestion des erreurs et l'intégration de middleware. Le middleware est un élément essentiel pour notre application et son interaction avec la base de données.

React est un outil très populaire pour le développement frontend. C'est une bibliothèque JavaScript conçue pour construire des interfaces utilisateurs réactives et dynamiques. React permet de créer une expérience utilisateur fluide et interactive et c'est pourquoi nous l'avons choisie.

L'adoption de la pile PERN pour notre projet a permis une intégration fluide entre la base de données, le serveur et le frontend, offrant une solution complète pour le développement de notre application. La synergie entre PostgreSQL, Express, React et Node.js a été un facteur clé dans le succès de notre projet, nous permettant de développer une application robuste et hautement performante.

Étapes spécifiques pour l'installation de notre application

1. **Prérequis**
    a. Assurez-vous que Node.js est installé sur le système. Vous pouvez le vérifier en exécutant 'node -' dans un terminal.
2. **Configurer le Backend (Express.js / Node.js)**
    a. Naviguez dans le dossier backend dans votre terminal.
    b. Installer les dépendances qui suivent dans le dossier backend depuis le terminal: npm install
    c. Si votre terminal est dans le dossier backend, il ne reste qu'à faire la commande: node index.js
    d. Le serveur du backend est maintenant actif.
3. **Configurer le Frontend (React)**
    a. Naviguez dans le dossier frontend dans votre terminal.
    b. Installer les dépendances qui suivent dans le dossier frontend depuis le terminal:
        i. npm install
    c. Si votre terminal est le dossier backend, lancez l'application React avec la commande: npm start. React devrait ouvrir automatiquement l'application dans le navigateur.
4. **Tester l'application**
    a. Naviguez dans l'application Web pour vérifier que le frontend communique correctement avec le backend et que la base de données hébergée sur Google Cloud SQL est accessible et fonctionnelle.
    b. Testez les fonctionnalités clés de l'application pour vous assurer que tout fonctionne comme prévu. Quelques exemples des fonctions de l'application sont:
        i. La recherche d'hôtels et de chambres disponibles.
        ii. La modification des chambres, des réservations, des locations, des comptes clients. La manipulation des hôtels.
        iii. La création de réservations et de locations.
        iv. La création de comptes clients.

Liste des DDLs qui créent notre base de données

Tables de la base de données

```
-- ********** TABLES **********
CREATE TABLE Chaine (
    nomChaine VARCHAR(255) PRIMARY KEY,
    nombreHotel INT,
    numRue INT,
    nomRue VARCHAR(255),
    ville VARCHAR(255),
    cp VARCHAR(10)
);

CREATE TABLE ChaineContact (
    contactid SERIAL PRIMARY KEY,
    nomChaine VARCHAR(255),
    contactPhone VARCHAR(20), -- can be null
    contactEmail VARCHAR(100), -- can be null but phone and email cant both be null
    FOREIGN KEY (nomChaine) REFERENCES Chaine(nomChaine) ON DELETE CASCADE
);

CREATE TABLE Hotel (
    nomHotel VARCHAR(255) PRIMARY KEY,
    classification INT CHECK(classification BETWEEN 1 AND 5),
    nombreChambre INT CHECK(nombreChambre >= 0),
    numRue INT,
    nomRue VARCHAR(100),
    ville VARCHAR(100),
    cp VARCHAR(10),
    phoneContact VARCHAR(20),
    emailContact VARCHAR(100),
    nomChaine VARCHAR(255) REFERENCES Chaine(nomChaine) ON DELETE CASCADE
);

CREATE TABLE Chambre (
    numeroChambre INT,
    prixParNuit DECIMAL CHECK (prixParNuit > 0),
    vue VARCHAR(255) CHECK (vue IN ('mer', 'paysage', 'ville', 'montagne') OR vue IS NULL),
    dommages TEXT,
    capacite INT,
    extPhone INT CHECK (extPhone >= 1000 AND extPhone <= 9999),
    nomHotel VARCHAR(255) NOT NULL,
    occupied BOOLEAN DEFAULT FALSE,
    superficie INTEGER,
    PRIMARY KEY(numeroChambre, nomHotel),
    FOREIGN KEY (nomHotel) REFERENCES Hotel(nomHotel) ON DELETE CASCADE
);

CREATE TABLE HasCommodite (
    numeroChambre INT,
    nomHotel VARCHAR(255),
    commodite VARCHAR(50) CHECK (commodite IN ('fridge', 'tv', 'ac', 'safe', 'patio', 'hot tub')),
    PRIMARY KEY(numeroChambre, nomHotel, commodite),
    FOREIGN KEY (numeroChambre, nomHotel) REFERENCES Chambre(numeroChambre, nomHotel) ON DELETE
CASCADE
);

CREATE TABLE Employe (
    NAS VARCHAR(255) PRIMARY KEY CHECK (NAS ~ '^[0-9]{9}$'),
    prenom VARCHAR(255),
    nom VARCHAR(255),
    numRue INT,
    nomRue VARCHAR(255),
```

```
    ville VARCHAR(255),
    CP VARCHAR(10),
    role VARCHAR(255),
    nomHotel VARCHAR(255),
    FOREIGN KEY (nomHotel) REFERENCES Hotel(nomHotel) ON DELETE SET NULL
);

CREATE TABLE GestionnaireHotel(
    nomHotel VARCHAR(255),
    NASgestionnaire VARCHAR(255),
    FOREIGN KEY (nomHotel) REFERENCES Hotel(nomHotel) ON DELETE CASCADE,
    FOREIGN KEY (NASgestionnaire) REFERENCES Employe(NAS) ON DELETE SET NULL,
    PRIMARY KEY (nomHotel, NASgestionnaire)
);

CREATE TABLE Client(
    NAS VARCHAR(255) PRIMARY KEY CHECK (NAS ~ '^[0-9]{9}$'),
    prenom VARCHAR(255),
    nom VARCHAR(255),
    numRue INT,
    nomRue VARCHAR(255),
    ville VARCHAR(255),
    cp VARCHAR(10),
    registerDate DATE DEFAULT CURRENT_DATE CHECK(registerDate <= CURRENT_DATE)
);

CREATE TABLE Reservation(
    resID SERIAL PRIMARY KEY,
    status VARCHAR(50) CHECK (status in ('active', 'archived')),
    resStart DATE CHECK (resStart >= CURRENT_DATE),
    resEnd DATE CHECK (resEnd > resStart),
    NASclient VARCHAR(255),
    numeroChambre INT,
    nomHotel VARCHAR(255),
    FOREIGN KEY (NASclient) REFERENCES Client(NAS) ON DELETE NO ACTION,
    FOREIGN KEY (numeroChambre, nomHotel) REFERENCES Chambre(numeroChambre, nomHotel) ON DELETE NO
ACTION
);

CREATE TABLE Location(
    locationID SERIAL PRIMARY KEY,
    NASclient VARCHAR(255),
    NASemploye VARCHAR(255),
    numChambre INT,
    locationStart DATE CHECK (locationStart >= CURRENT_DATE),
    locationEnd DATE CHECK (locationEnd > locationStart),
    nomHotel VARCHAR(255),
    paymentID VARCHAR(100),
    FOREIGN KEY (NASclient) REFERENCES Client(NAS) ON DELETE NO ACTION,
    FOREIGN KEY (NASemploye) REFERENCES Employe(NAS) ON DELETE NO ACTION,
    FOREIGN KEY (numChambre, nomHotel) REFERENCES Chambre(numeroChambre, nomHotel) ON DELETE NO
ACTION
);
```

Indexes de la base de données

```
--index creation
CREATE INDEX idx_reservation_resstart ON Reservation(resStart);
CREATE INDEX idx_reservation_resend ON Reservation(resEnd);
CREATE INDEX idx_reservation_nasclient ON Reservation(NASclient);
CREATE INDEX idx_reservation_numchambre_nomhotel ON Reservation(numeroChambre,
nomHotel);
CREATE INDEX idx_location_start ON Location(locationStart);
CREATE INDEX idx_location_end ON Location(locationEnd);
CREATE INDEX idx_location_numchambre_nomhotel ON Location(numChambre, nomHotel);
```

Fonctions pour assurer le bon fonctionnement de la base de données

```
-- Function to update the 'occupied' status when a reservation is made
CREATE OR REPLACE FUNCTION update_chambre_reserved_status()
RETURNS TRIGGER AS $$
BEGIN
    -- Assuming 'occupied' is the correct field to update based on reservation dates
    IF (NEW.resStart <= CURRENT_DATE AND NEW.resEnd >= CURRENT_DATE) THEN
        UPDATE Chambre SET occupied = TRUE
        WHERE numeroChambre = NEW.numeroChambre AND nomHotel = NEW.nomHotel;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Function to check reservation dates
CREATE OR REPLACE FUNCTION check_reservation_dates()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.resEnd < NEW.resStart THEN
        RAISE EXCEPTION 'The end date of a reservation must be some time after the start date.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Function to check room availability
CREATE OR REPLACE FUNCTION check_room_availability()
RETURNS TRIGGER AS $$
DECLARE
    room_count INT;
BEGIN
    SELECT COUNT(*) INTO room_count FROM Reservation
    WHERE numeroChambre = NEW.numeroChambre AND nomHotel = NEW.nomHotel
    AND NOT (resEnd <= NEW.resStart OR resStart >= NEW.resEnd);
    IF room_count > 0 THEN
        RAISE EXCEPTION 'Room is already reserved for the specified dates.';
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Function to update the 'occupied' status on check-in
CREATE OR REPLACE FUNCTION update_room_status_on_checkin()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Chambre SET occupied = TRUE
    WHERE numeroChambre = NEW.numChambre AND nomHotel = NEW.nomHotel;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Function to validate employee role
CREATE OR REPLACE FUNCTION validate_employee_role()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.role NOT IN ('Manager', 'Receptionist', 'Housekeeping', 'Valet') THEN
        RAISE EXCEPTION 'Invalid role for employee.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Function to automatically update the 'occupied' status on check-out
CREATE OR REPLACE FUNCTION update_room_occupation()
RETURNS TRIGGER AS $$
BEGIN
    -- Check if the location status is being updated to 'archived'
    IF NEW.status = 'archived' THEN
        -- Update the corresponding room in the chambre table to set occupied = False
        UPDATE chambre
        SET occupied = FALSE
        WHERE numeroChambre = NEW.numchambre AND nomHotel = NEW.nomhotel;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Function to automatically update the number of rooms of a hotel when rooms are added to it
CREATE OR REPLACE FUNCTION update_hotel_chambre_count()
RETURN TRIGGER AS $$
BEGIN
    --Update nombrechambre conut for the hotel related to the inserted or deleted chambre
    UPDATE hotel h
    SET nombreChambre = (SELECT COUNT(*)
                         FROM chambre c
                         WHERE c.nomHotel = h.nomHotel
                         GROUP BY c.nomHotel)
    WHERE nomHotel = COALESCE(NEW.nomHotel, OLD.nomHotel);
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--check room availability
CREATE OR REPLACE FUNCTION check_room_availability()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1 FROM Reservation
    WHERE numeroChambre = NEW.numeroChambre
    AND (
      (NEW.DateArrivee BETWEEN DateArrivee AND DateDepart) OR
      (NEW.DateDepart BETWEEN DateArrivee AND DateDepart) OR
      (DateArrivee BETWEEN NEW.DateArrivee AND NEW.DateDepart) OR
      (DateDepart BETWEEN NEW.DateArrivee AND NEW.DateDepart)
    )
  ) THEN
    RAISE EXCEPTION 'La chambre est déjà réservée pour les dates spécifiées.';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- verifie s'il y a chevauchements de location pour la meme chambre
CREATE OR REPLACE FUNCTION check_location_availability()
RETURNS TRIGGER AS $$
BEGIN
  -- Vérifier les chevauchements de dates pour la chambre spécifiée
  IF EXISTS (
    SELECT 1 FROM Location
    WHERE numChambre = NEW.numChambre AND nomHotel = NEW.nomHotel
    AND (
      (NEW.locationStart BETWEEN locationStart AND locationEnd) OR
      (NEW.locationEnd BETWEEN locationStart AND locationEnd) OR
      (locationStart BETWEEN NEW.locationStart AND NEW.locationEnd) OR
      (locationEnd BETWEEN NEW.locationStart AND NEW.locationEnd)
    )
  ) THEN
    RAISE EXCEPTION 'La chambre % est déjà occupée pour les dates spécifiées.',
NEW.numChambre;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Triggers de la table pour déclencher les fonctions

```
--trigger for the update_chambre_reserved_status() function
CREATE TRIGGER trigger_update_chambre_reserved
AFTER INSERT OR UPDATE ON Reservation
FOR EACH ROW EXECUTE FUNCTION update_chambre_reserved_status();
--creation du declencheur trigger_check_resdates
CREATE TRIGGER trigger_check_resdates
BEFORE INSERT OR UPDATE ON Reservation
FOR EACH ROW EXECUTE FUNCTION check_reservation_dates();
--trigger for checking room availability before insert in reservation
CREATE TRIGGER trigger_check_room_availability
BEFORE INSERT ON Reservation
FOR EACH ROW EXECUTE FUNCTION check_room_availability();
--trigger for auto-update on room status on check-in/out
CREATE TRIGGER trigger_room_checkin
AFTER INSERT ON Location
FOR EACH ROW EXECUTE FUNCTION update_room_status_on_checkin();
--trigger for validating employee role
CREATE TRIGGER trigger_validate_employee_role
BEFORE INSERT OR UPDATE ON Employe
FOR EACH ROW EXECUTE FUNCTION validate_employee_role();
--trigger for detecting changed statuses in the location table
CREATE TRIGGER trigger_update_room_occupation
AFTER UPDATE ON location
FOR EACH ROW
WHEN (OLD.status IS DISTINCT FROM NEW.status)
EXECUTE FUNCTION update_room_occupation();
--update nombrechambre when item is inserted to chambre
CREATE TRIGGER trigger_update_hotel_chambre_count_after_insert
AFTER INSERT ON chambre
FOR EACH ROW
EXECUTE FUNCTION update_hotel_chambre_count();
--update nombrechambre also when item is deleted from chambre
CREATE TRIGGER trigger_update_hotel_chambre_count_after_delete
AFTER DELETE ON chambre
FOR EACH ROW
EXECUTE FUNCTION update_hotel_chambre_count();
-- check the availability of the room before reservation
CREATE TRIGGER check_availability_before_reservation
BEFORE INSERT ON Reservation
FOR EACH ROW EXECUTE FUNCTION check_room_availability();
-- check the availability of the room before location
CREATE TRIGGER check_availability_before_location
BEFORE INSERT ON Location
FOR EACH ROW EXECUTE FUNCTION check_room_availability();
```

Exemples de requêtes exécutées dans le backend de l'application

Tel que demandé dans l'énoncé du projet, voici le code SQL pour 4 exemples de requêtes qui sont exécutées dans notre application:

- Requête pour supprimer une chambre à un hôtel spécifié:
    - ("DELETE FROM chambre WHERE nomHotel = $1 AND numeroChambre = $2 RETURNING *", [nomHotel, numeroChambre]);
- Requête pour mettre à jour une chambre:
    - (UPDATE chambre SET prixParNuit = COALESCE($1, prixParNuit), vue = COALESCE($2, vue), dommages = COALESCE($3, dommages), capacite = COALESCE($4, capacite), extPhone = COALESCE($5, extPhone), occupied = COALESCE($6, occupied) WHERE numeroChambre = $7 AND nomHotel = $8 RETURNING *', [prixParNuit, vue, dommages, capacite, extPhone, occupied, id, name]);
- Requête pour trouver toutes les informations sur les chambres d'un hotel:
    - ("SELECT * FROM chambre WHERE nomhotel = $1 AND occupied=False", [nomhotel]);
- Requête pour mettre à jour le status d'une réservation:
    - ("UPDATE reservation SET status = $1 WHERE resid = $2 RETURNING * ", [status, resID]);

Document d'éthique de soumission

Accord d'éthique personnelle concernant les travaux universitaires

Projet de groupe

Nous soumettons ce travail et attestons que nous avons appliqué toutes les règles appropriées de citation et de référencement en usage à l'Université d'Ottawa https://www.uottawa.ca/etudiants-actuels/integrite-academique. Nous attestons que ce travail est conforme au règlement sur l'intégrité académique de l'Université d'Ottawa. Nous comprenons que cette tâche ne sera pas acceptée ou notée si elle est soumise sans la signature de tous les membres du groupe.

__FELIX ALLARD_____ ___300146250___
Nom, lettres majuscules Numéro d'étudiant

___Felix Allard_____ _06/04/2024__
Signature Date

__SAMUEL ROSE_____ 300173591
Nom, lettres majuscules Numéro d'étudiant

_SamuelRose_____ 06/04/2024
Signature Date

__EMMA SEAIBI_____ 300273060
Nom, lettres majuscules Numéro d'étudiant

__Emma Seaibi_____ 06/04/2024
Signature Date

__JEAN MARKELLO LÉTANG_____ 300245679
Nom, lettres majuscules Numéro d'étudiant

Jean Markello Létang_____ 06/04/2024
Signature Date