Expression des requêtes SQL et Explications

Rajaa

Dans cette partie du document, nous explorons les différentes requêtes SQL et procédures stockées utilisées pour gérer les réservations et l'occupation des logements. L'objectif est d'optimiser l'attribution des logements, d'analyser les tendances de réservation et de mieux comprendre le comportement des résidents.

Voici donc quelques unes des requêtes traitées:

-Quels logements sont disponibles pour une période donnée, selon des critères spécifiques

(type, emplacement, prix)

La **procédure stockée logements_disponibles**()

permet d'obtenir la liste des logements disponibles pour une période donnée, en filtrant selon certains critères : Type de logement (appartement, studio, etc.), emplacement (ville, quartier) ,loyer maximum (budget).

La liste des logements disponibles est stockée dans une table temporaire **temp_logements_disponibles**, ce qui permet d'exécuter des requêtes dessus après l'appel de la procédure

Filtrage des logements disponibles: On sélectionne tous les logements qui sont marqués comme "**Disponible**" avec WHERE l.etat = 'Disponible'

Filtrage dynamique

- Si un **type de logement** est spécifié (p_type_logement), on ne garde que ceux qui correspondent.
- Si un **emplacement** est précisé (p_emplacement), on ne garde que les logements situés dans cette zone.
- Si un **loyer maximum** est fixé (p_loyer_max), on exclut les logements plus chers.

Exclusion des logements déjà réservés

- On élimine les logements qui sont déjà réservés sur la période demandée.

- La clause OVERLAPS permet de détecter si une **période de réservation** chevauche la période demandée.

Stockage des résultats dans une table temporaire

- Les logements trouvés sont enregistrés dans temp_logements_disponibles, une table temporaire accessible après l'exécution.

- Comment gérer les réservations et attribuer les logements aux nouveaux résidents en

optimisant l'occupation ?

La **procédure stockée** permet attribuer_logement()sert à attribuer un logement disponible à un résident en minimisant le coût de location. Cette procédure sélectionne le logement disponible le moins cher et vérifie sa disponibilité via NOT IN. Une fois attribué, l'état du logement est mis à jour automatiquement (Occupé) pour indiquer qu'il est occupé. La gestion d'erreurs est également intégrée pour signaler l'absence de logement disponible.

Sélection du logement disponible le moins cher

On filtre les logements disponibles qui **ne sont pas réservés** sur la période demandée.

On trie ces logements par loyer (ORDER BY l.loyer ASC) pour prendre **le moins cher**.

On utilise LIMIT 1 pour sélectionner **un seul logement**.

Vérification de la disponibilité

Si aucun logement disponible n'a été trouvé, on lève une erreur avec RAISE EXCEPTION.

Création de la réservation

On ajoute une entrée dans la table Reservation avec la période demandée et l'ID du logement trouvé.

Mise à jour de l'état du logement

Une fois attribué, le logement est marqué comme **"Occupé"** (UPDATE Logement SET etat = 'Occupé').

Affichage d'un message de confirmation

RAISE NOTICE permet d'afficher un message confirmant l'attribution du logement.

-Quel est le taux d'occupation actuel des logements ?

Le taux d'occupation actuel des logements est calculé grâce à une jointure LEFT JOIN, qui permet d'inclure tous les logements, qu'ils soient réservés ou non. Le ratio entre les logements occupés et le total des logements est obtenu à l'aide de COUNT(DISTINCT r.id_logement), ce qui évite de compter plusieurs réservations pour le même logement.

- Quel est le temps moyen de séjour d'un résident dans un logement ?

Le temps moyen de séjour d'un résident dans un logement est déterminé en utilisant DATE_PART('day', date_fin - date_debut), qui calcule la durée en jours. La moyenne de ces durées sur l'ensemble des réservations est obtenue grâce à AVG().

-Quel est le profil des résidents qui prolongent leur séjour par rapport à ceux qui ne prolongent pas ?

Le profil des résidents qui prolongent leur séjour par rapport à ceux qui ne le prolongent pas est identifié en détectant les cas où une réservation commence le jour où une autre finit (r1.date_fin = r2.date_debut). Une agrégation des données par profession et secteur d'activité permet d'analyser les tendances et de mieux comprendre les caractéristiques des résidents ayant tendance à prolonger leur séjour.Nous notons aussi l'utilisation de GROUP BY qui permet de regrouper les resultats

-Quelles sont les tendances de réservation sur différentes périodes (mois, trimestre, année) ?

Tendances Mensuelles :

SELECT DATE_TRUNC('month', date_debut) AS mois, COUNT(*) AS nombre_reservations
FROM Reservation
GROUP BY mois
ORDER BY mois;

Tendances Trimestrielles et Annuelles :

SELECT DATE_TRUNC('quarter', date_debut) AS trimestre, COUNT(*) AS nombre_reservations

```
FROM Reservation
GROUP BY trimestre
ORDER BY trimestre;

SELECT DATE_TRUNC('year', date_debut) AS annee, COUNT(*) AS nombre_reservations
FROM Reservation
GROUP BY annee
ORDER BY annee;
```

Les tendances de réservation sur différentes périodes (mois, trimestre, année) sont étudiées à l'aide de plusieurs requêtes. Pour analyser les tendances mensuelles, la requête `SELECT DATE_TRUNC('month', date_debut) AS mois, COUNT(*) AS nombre_reservations FROM Reservation GROUP BY mois ORDER BY mois;` est utilisée. Pour les tendances trimestrielles et annuelles, `DATE_TRUNC('quarter', date_debut)` et `DATE_TRUNC('year', date_debut)` permettent d'agréger les données respectivement par trimestre et par année. La fonction `COUNT(*)` mesure le volume de réservations pour chaque période.

-Comment les saisons affectent-elles la demande des logements ?

L'effet des saisons sur la demande des logements est étudié grâce à une requête qui regroupe les mois en fonction des saisons :

```
SELECT
  CASE
    WHEN EXTRACT(MONTH FROM r.date_debut) IN (12, 1, 2) THEN 'Hiver'
    WHEN EXTRACT(MONTH FROM r.date_debut) IN (3, 4, 5) THEN 'Printemps'
    WHEN EXTRACT(MONTH FROM r.date_debut) IN (6, 7, 8) THEN 'Été'
    ELSE 'Automne'
  END AS saison,
  COUNT(*) AS nombre_reservations
FROM Reservation r
GROUP BY saison;
```

L'opérateur CASE permet de regrouper les mois par saison. La fonction EXTRACT(MONTH FROM r.date_debut) extrait le mois de début de réservation et le classe dans la saison correspondante. Ensuite, COUNT(*) mesure le nombre de réservations par saison.

Ces requêtes SQL et procédures stockées permettent une meilleure gestion des logements et des réservations, en facilitant l'analyse des tendances et en optimisant l'attribution des logements. Elles offrent des outils clés pour améliorer la rentabilité et l'efficacité du système de gestion immobilière.

Assia

- Logement ayant meilleur rapport qualité/prix en fonction des avis des résidents:

Cette requête récupère les logements en calculant leur rapport qualité/prix, défini comme la moyenne des notes des résidents divisée par le loyer. Si aucun avis n'est disponible, COALESCE remplace NULL par 0, et si le loyer est nul, le rapport est NULL.

Elle utilise un LEFT JOIN entre Logement et Note pour inclure tous les logements, même ceux sans avis. Les résultats sont regroupés avec GROUP BY et triés par rapport qualité/prix décroissant.

Cette requête permet d'identifier les logements avec le meilleur rapport qualité/prix. Cela aide à optimiser l'attractivité des logements, en ajustant les prix ou les services pour rendre certains logements plus compétitifs ou attrayants pour les résidents.

-Le taux de satisfaction des résidents en ce qui concerne les logements, les services et les événements :

Cette requête calcule deux taux de satisfaction :

1. **Taux de satisfaction des logements** : La moyenne des notes attribuées par les résidents aux logements, calculée via AVG(n.score).
2. **Taux de participation aux événements** : Le pourcentage de résidents ayant participé à des événements communautaires, calculé comme la proportion des résidents participants par rapport au nombre total de résidents

Cette requête répond aux besoins concernant la gestion des réservations, le taux de satisfaction des résidents, et la participation aux événements.

-Disponibilité des logements :

La requête sélectionne l'emplacement des logements et compte ceux qui sont disponibles dans chaque zone géographique en utilisant COUNT(*). Elle exclut les logements réservés en comparant les dates de réservation actuelles avec CURRENT_DATE à travers une sous-requête. Seuls les logements non réservés pour la période actuelle sont considérés comme disponibles. Enfin, les résultats sont regroupés par emplacement avec GROUP BY, permettant de calculer le nombre de logements disponibles dans chaque zone géographique.

Cette requête répond à la question "Quels logements sont disponibles pour une période donnée, selon des critères spécifiques (type, emplacement, prix) ?", en listant les logements disponibles à la location dans un quartier donné, en excluant ceux qui sont déjà réservés pour la période actuelle.

- Résidents avec des comportements problématiques ou signalés des conflits récurrents

Cette requête permet d'identifier les résidents ayant des conflits récurrents et non résolus. Elle effectue un **JOIN** entre les tables **Resident**, **Resident_conflicts** et **Conflit** pour relier chaque résident à ses conflits. La fonction **COUNT(c.id_conflit)** compte le nombre de conflits non résolus pour chaque résident, et **HAVING COUNT(c.id_conflit) > 1** filtre les résultats pour ne conserver que les résidents ayant plus d'un conflit non résolu.

Les résultats sont ensuite triés par **nombre conflits** de manière décroissante, mettant en avant les résidents ayant le plus de conflits.

Cette requête répond à la question **C** ("Quels résidents partagent actuellement un logement et quelles sont leurs interactions ?") en identifiant les résidents avec des conflits récurrents.

-Organisation des événements communautaires pour maximiser la participation des résidents dans un logement donné:

L'objectif de cette requête est de maximiser la participation des résidents aux événements en analysant leur taux de participation. Elle utilise un **JOIN** avec la table **participation** pour relier chaque résident aux événements auxquels ils ont pris part. La fonction

COUNT(DISTINCT re.id_resident) permet de compter le nombre de résidents uniques ayant participé à chaque événement, afin d'évaluer l'engagement.

Le taux de participation est calculé en divisant le nombre de participants par le nombre total de résidents, à l'aide de la fonction **ROUND**. Les résultats sont ensuite triés par **nombre_participants** de manière décroissante, mettant en avant les événements les plus populaires.

Cette requête répond à la question **F** ("Comment organiser les événements communautaires pour maximiser la participation des résidents dans un logement donné ?") en identifiant les événements qui attirent le plus de résidents.

-Identification des logements les plus rentables:

Cette requête permet d'identifier les types de logements les plus rentables en combinant les revenus générés par les réservations et les coûts des interventions de maintenance. Elle repose sur l'utilisation de **vues** et de **requêtes imbriquées** pour structurer les calculs.

La première sous-requête **revenus** (vue) calcule le revenu total pour chaque type de logement en multipliant la durée de chaque réservation par le loyer, et en regroupant les résultats par **type_logement**. La deuxième sous-requête **interventions** (vue) calcule le nombre d'interventions de maintenance par type de logement en comptant les interventions associées.

La requête principale joint ces deux résultats et calcule la rentabilité estimée en soustrayant les coûts d'entretien (500 par intervention) des revenus totaux. Les résultats sont triés par rentabilité estimée en ordre décroissant, mettant ainsi en avant les types de logements les plus rentables.

Arij

-Liste des logements avec le nombre d'interventions:

Question traitée : *Quels logements nécessitent le plus d'interventions de maintenance ?*

Cette requête permet de récupérer le nombre d'interventions effectuées pour chaque logement

.

- **LEFT JOIN** est utilisé pour s'assurer que tous les logements sont inclus, même ceux qui n'ont pas d'interventions associées. Si un logement n'a pas d'intervention, le résultat du COUNT pour ce logement sera 0.
- Le **GROUP BY** regroupe les résultats par identifiant et emplacement de logement, permettant ainsi de calculer le nombre d'interventions pour chaque logement.
- L'**ORDER BY nb_interventions DESC** trie les résultats par nombre d'interventions, du plus élevé au plus faible, ce qui permet d'identifier les logements les plus utilisés.

- Liste des logements disponibles pour une période spécifique

Question traitée : *Quels logements sont disponibles pour une période donnée?*

Cette requête permet de trouver les logements disponibles pendant une période spécifique.

- **JOIN** entre la table Logement et Type_logement pour obtenir des informations sur le type de logement.
- La condition **NOT EXISTS** vérifie qu'aucune réservation ne chevauche les dates spécifiées. Cela permet d'assurer que les logements retournés ne sont pas réservés pendant la période recherchée.
- La condition **AND** impose de filtrer les résultats selon le type de logement (ici, "Studio"), l'emplacement et un loyer maximal.

-Liste des résidents ayant prolongé leur séjour:

Question traitée : *Quels résidents ont prolongé leur séjour*,

Cette requête identifie les résidents qui ont prolongé leur séjour en comparant les dates de fin des réservations successives.

- **JOIN** entre deux instances de la table Reservation pour associer chaque prolongation à sa réservation initiale.
- Le **HAVING** filtre les résultats pour ne garder que ceux où la date de sortie prolongée

-Liste des logements avec le nombre de demandes en attente

Question traitée : *Quels logements ont le plus de demandes en attente ?*
Cette requête retourne le nombre de demandes de réservation en attente pour chaque logement.

- **LEFT JOIN** garantit que tous les logements sont inclus, même ceux sans demande en attente.
- Le **WHERE** filtre les réservations dont la date de fin est dans le futur, signifiant qu'elles sont encore en cours ou à venir.
- Le **GROUP BY** regroupe les résultats par logement, et le **COUNT** calcule le nombre de réservations pour chaque logement.

- Liste des logements avec le nombre d'interventions et la moyenne des notes

Question traitée :Quels logements ont le meilleur rapport qualité/prix en fonction des avis des résidents ?

Cette requête calcule le nombre d'interventions et la moyenne des notes attribuées à chaque logement.

- **LEFT JOIN** est utilisé pour s'assurer que tous les logements sont inclus, même ceux sans intervention ou note.
- **COALESCE** est utilisé pour remplacer les valeurs NULL par 0 si aucune intervention ou note n'est présente.
- **AVG(N.score)** calcule la moyenne des notes, et **ROUND** permet d'arrondir cette moyenne à deux décimales.

-Liste des logements avec le nombre de résidents actifs

Question traitée : *Combien de résidents vivent actuellement dans chaque logement ?*

Cette requête permet de trouver le nombre de résidents actuellement actifs dans chaque logement.

- **JOIN** entre Logement, Reservation, et Resident pour relier les logements aux résidents.
- Le **WHERE** filtre les réservations en cours, basées sur la date actuelle.
- Le **COUNT(DISTINCT)** garantit que chaque résident est compté une seule fois, même s'il a fait plusieurs réservations.

- Trigger pour mettre à jour le statut du logement après une intervention

Ce trigger met à jour automatiquement l'état d'un logement en "**En maintenance**" lorsqu'une intervention y est ajoutée.

Déclencheur :

Il se déclenche **après chaque insertion** (AFTER INSERT) dans la table Logement_Intervention.

Mécanisme :

- Lorsqu'une nouvelle intervention est insérée dans Logement_Intervention, la fonction update_logement_status est exécutée.
- Elle met à jour le statut du logement concerné (UPDATE Logement SET statut = 'En maintenance').
- Elle utilise NEW.id_logement, qui correspond au logement concerné par la nouvelle intervention.
- **CREATE FUNCTION** définit une fonction qui change le statut du logement en "En maintenance".
- **AFTER INSERT** déclenche la fonction après qu'une nouvelle intervention ait été ajoutée à Logement_Intervention.

-Trigger pour libérer le logement après une annulation de réservation

Question traitée : *Comment gérer automatiquement la disponibilité des logements après une annulation ?*

Ce trigger libère un logement (le rend disponible) après l'annulation d'une réservation.

- **CREATE FUNCTION** définit la logique de mise à jour du statut du logement.
- **AFTER DELETE** déclenche la fonction après la suppression d'une réservation.

- Profil démographique des résidents

Question traitée : *Quel est le profil démographique des résidents (âge, profession, etc.) ?*

Cette requête génère un profil démographique des résidents en joignant les tables Resident et Profil.

- **EXTRACT(YEAR FROM AGE())** calcule l'âge des résidents à partir de leur date de naissance.
- Le **JOIN** entre Resident et Profil permet d'obtenir des informations supplémentaires sur le résident, comme la profession et le secteur d'activité.

Rayen

Sélection des personnes partageant un même logement avec une réservation en cours

Cette requête récupère les résidents ayant une réservation active dans un logement partagé avec au moins une autre personne. Elle vérifie que la date actuelle (CURRENT_DATE) est comprise entre la date de début et de fin de la réservation. Ensuite, elle s'assure que le logement contient plus d'un résident via une sous-requête. Elle enrichit les résultats avec d'autres informations, comme les événements auxquels les résidents participent et les éventuels conflits liés à ces résidents. (version similaire traité par Assia mais avec une approche plus liée aux conflits alors que ici on considère les événements)

Sélection de tous les résidents ayant déjà partagé ou partageant encore un logement

Cette requête liste les logements où plusieurs personnes ont déjà cohabité à un moment donné. Elle regroupe les résidents par logement et concatène leurs noms. Grâce à la condition EXISTS, elle identifie les cas où un autre résident a occupé le même logement durant une période de chevauchement des réservations. Seuls les logements ayant accueilli plus d'un résident sont affichés, grâce à la clause HAVING COUNT(*) > 1. Cette requête peut être utile dans le cas ou on veut grouper les résidents par logements

Agrégation des réservations et calcul de la note générale des logements

Cette requête crée une table **logement_reservations** qui regroupe les logements avec le nombre total de réservations et la moyenne des notes attribuées par les

clients. L'agrégation se fait via COUNT(*) pour le nombre de réservations et AVG(n.score) pour la note générale des logements. logement_reservations permet d'être utilisé dans d'autres requêtes pour analyser la popularité des logements.

Sélection des logements les plus demandés

En utilisant logement_reservations, cette requête sélectionne les logements les plus réservés, en affichant également leurs détails (emplacement, surface, loyer, nombre de chambres). Pour éviter les valeurs nulles dans la note moyenne, elle utilise COALESCE(n.score, 0). Le tri est effectué d'abord par nombre de réservations (DESC), puis par la note moyenne en cas d'égalité. Seuls les trois logements les plus demandés sont retournés grâce à LIMIT 3.

Sélection des 3 mois avec le plus de réservations (toutes années confondues)

Cette requête identifie les mois les plus populaires pour les réservations, indépendamment de l'année. Elle extrait le mois sous format numérique (TO_CHAR(date_debut, 'MM')) et compte le nombre total de réservations pour chaque mois. Après avoir regroupé les données par mois, elle trie les résultats en ordre décroissant et affiche les trois mois ayant enregistré le plus grand nombre de réservations. Donc on pourra déterminer les périodes avec le plus de demandes et ajuster les prix.

Mise à jour du loyer et des factures

Cette transaction garantit la cohérence des données lors de la modification du loyer d'un logement (id_logement = 10). D'abord, le loyer est mis à jour dans la table logement. Ensuite, les factures liées à ce logement sont recalculées en fonction du nouveau loyer et de la durée des séjours. La transaction assure que les deux mises à jour sont appliquées ensemble ou annulées en cas d'erreur.

Génération des factures à partir des réservations

Cette requête insère de nouvelles factures en récupérant les informations des tables Reservation, Resident et logement. Le prix total est calculé en multipliant le loyer par la durée du séjour. Chaque facture est associée au résident via son CIN et à une réservation spécifique. Cette opération permet d'automatiser la facturation des séjours enregistrés.