

Projet n°1 : Gestion des Réservations et Interactions dans un Réseau de Co-Living

1. Une reformulation complète de la description du projet qui supprime toutes les ambiguïtés du présent document et définit exhaustivement les services rendus par la base de données, après négociations avec le client (≈ spécification des besoins complète). Et nommer les points flous et comment ils ont été réglés et clarifiés.

Contexte et Objectifs

Un réseau de logements partagés (co-living) souhaite déployer une base de données centralisée permettant de gérer efficacement l'occupation des logements, les réservations, les interactions entre résidents ainsi que la maintenance des logements. Cette base de données devra assurer une gestion optimale des logements en fonction de divers critères, tout en offrant une traçabilité précise des activités et des événements communautaires.

Etape 1 : Entretien client -

Lors de notre premier entretien avec notre 'client', nous avons pu lui poser les questions nécessaires afin de pouvoir concevoir la base de données le mieux possible et le plus simplement et optimisée pour le client.

Par exemple, quelques points étaient flou, comme le prix d'une réservation, est-ce que le prix dépend du type de logement ou du logement en lui-même ? C'est-à-dire, est-ce-que le prix de la nuit est fixe, et quels sont les facteurs à considérer ?

Nous avons également un point flou concernant la prolongation d'un séjour par un résident. Nous avons décidé de ne pas toucher à la date de fin initiale de la réservation, mais plutôt de créer une autre table qui gère cette action.

Concernant les équipements, on sait qu'il existe des équipements qui concernent le logement lui-même, ou qui sont propres au site globalement, ainsi nous avons choisi d'appliquer cette distinction dans différentes tables.

Nous avons tous contribué à lever les ambiguïtés du sujet afin de concevoir un modèle entité-association et son modèle relationnel correspondant. Ces modèles ont été modifiés et affinés à plusieurs reprises afin d'obtenir une base de données la plus structurée, claire et cohérente possible.

Services Rendus par la Base de Données

1. Gestion des logements

- Suivi des caractéristiques des logements : leur emplacement, type du logement (loft, bungalow, studio...), nombre de chambres et nombre de lits (doubles/simples), et surface (un des facteurs dans le calcul du prix...), équipements disponibles propres au logement.
- Association des logements à un site donné (localisation géographique - Adresse / Ville / Code postal / Pays / Nom du site).
- Classification des logements par type avec un indice forfaitaire d'occupation (un bungalow de 20m² est moins coûteux qu'un studio...).
- Association des équipements au logement.

2. Gestion des réservations

- Enregistrement des réservations avec date de début et de fin de la réservation.
- Association d'un logement à une réservation.
- Suivi des réservations effectuées par les résidents.
- Possibilité de prolongation des réservations. Gérée dans une table indépendante.

3. Gestion des résidents

- Suivi des informations des résidents (nom, prénom, contact - num tel).
- Gestion des interactions entre résidents via la participation aux événements communautaires.
- Enregistrement des conflits signalés et des résidents impliqués.

4. Suivi des conflits déclarés

- Date de signalement du conflit.
- Etat du conflit (réglé, en cours).
- Titre et description brève.
- Résidents impliqués dans le conflit.

5. Suivi des événements communautaires

- Enregistrement des événements (titre, lieu, date, description).
- Titre de l'évènement (spectacle, sport, cours...).
- Public (enfant / adulte / senior / tranche d'âge...).
- Association des événements à un site spécifique (location géographique - un certain immeuble / un certain camping...).
- Suivi de la participation des résidents aux événements.

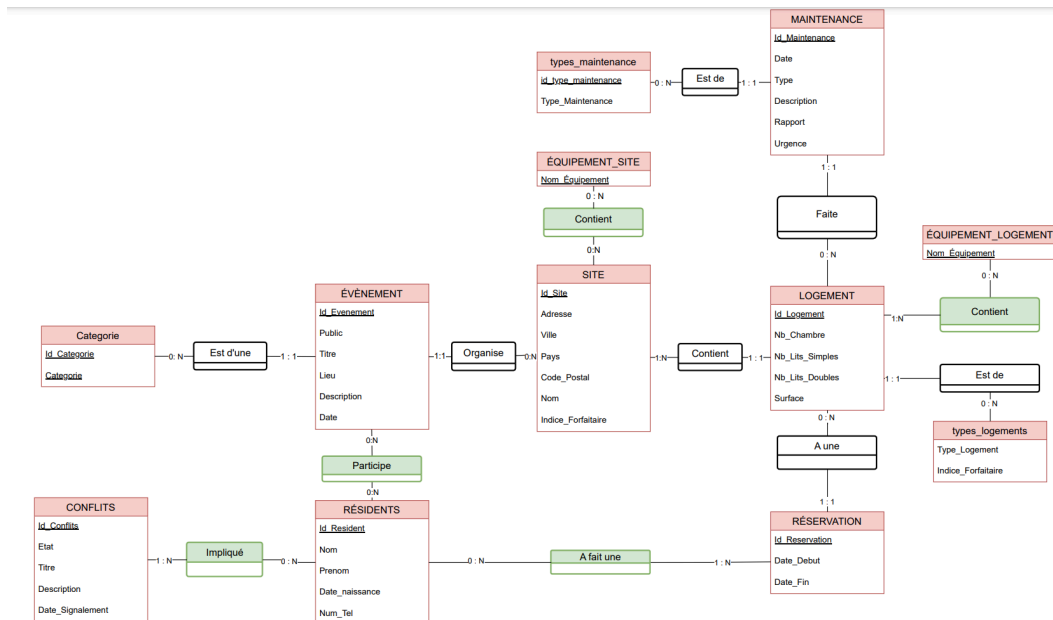
6. Gestion des interventions de maintenance

- Enregistrement des interventions de maintenance (date, description, urgence -oui/non nous avons choisi de le définir par un booléen, rapport).
- Association d'une intervention à un logement et à un type de maintenance prédéfini.
- Catégorisation des types de maintenance.

2. Le schéma de la base de données, exprimé d'une manière visuelle selon le modèle EA (vu en cours). Le commentaire expliquera et justifiera les choix effectués (≈ spécification détaillée)

Entité / Association -

Nous avons donc réalisé le schéma Entité/Association suivant :

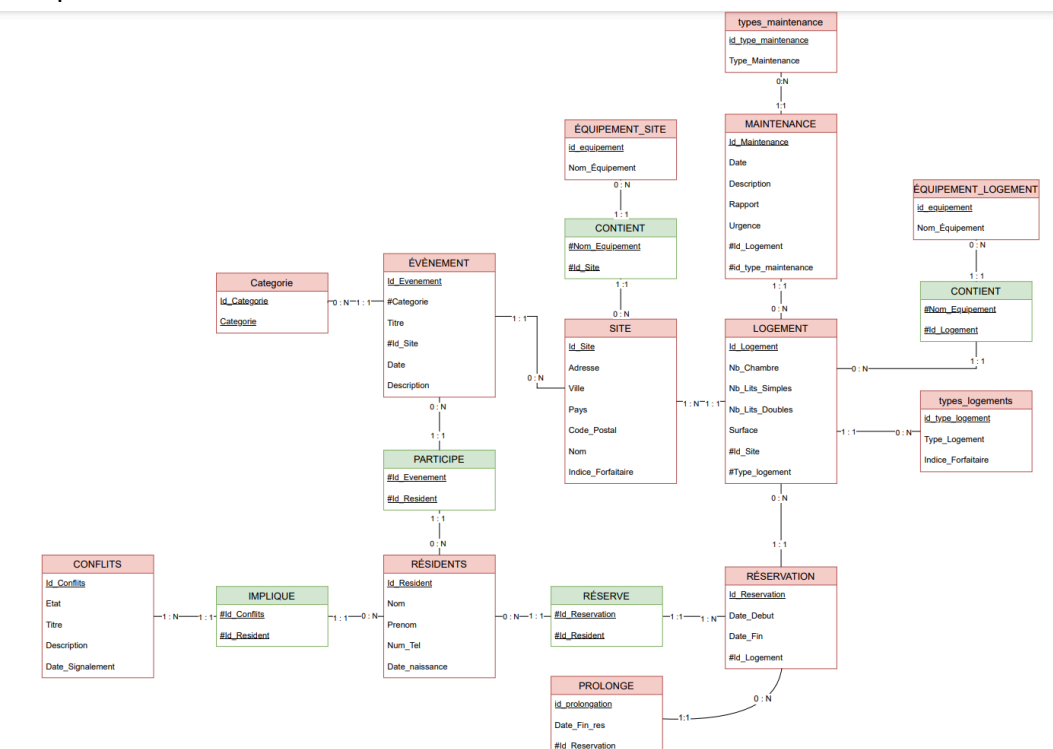


Normalisation -

En normalisant notre schéma, nous avons obtenu des relations en BCNF, la Forme Normale de Boyce-Codd (BCNF). Toutes nos dépendances respectent cette forme.

Schéma Relationnel -

A partir de cela nous avons fait le schéma relationnel suivant :



Nous en sommes venus à la conclusion que nous devons faire les tables suivantes :

- **Un LOGEMENT** est décrit par un identifiant unique (*Id_Logement*), un nombre de chambres (*Nb_Chambre*), un nombre de lits simples (*Nb_Lits_Simples*), un nombre de lits doubles (*Nb_Lits_Doubles*), une surface (*Surface*), et est associé à un site (*Id_Site*) ainsi qu'à un type de logement (*Id_Type_Logement*).
- **Un TYPE_LOGEMENT** est décrit par un identifiant unique (*Id_Type_Logement*), une désignation (*Type_Logement*) et un indice forfaitaire (*Indice_Forfaitaire*).
- **Un SITE** est décrit par un identifiant unique (*Id_Site*), une adresse (*Adresse*), une ville (*Ville*), un pays (*Pays*), un code postal (*Code_Postal*), un nom (*Nom*) et un indice forfaitaire (*Indice_Forfaitaire*).
- **Des RESERVATIONS** sont décrits par un identifiant unique (*Id_Reservation*), une date de début (*Date_Debut*), une date de fin (*Date_Fin*), et est liée à un logement (*Id_Logement*).
- **Un RÉSIDENT** est décrit par un identifiant unique (*Id_Resident*), un nom (*Nom*), un prénom (*Prenom*), un numéro de téléphone (*Num_Tel*), et une date de naissance (*Date_Naissance*).
- **Un lien RESIDENTS_RESERVATIONS** relie un résident (*Id_Resident*) à une réservation (*id_reservation*).
- **Des PROLONGATIONS** sont décrites par un identifiant unique (*Id_Prolongation*), une nouvelle date de fin (*date_fin_reservation*), et est associée à une réservation (*Id_Reservation*).
- **Un lien RESIDENTS_EVENEMENT** relie un résident (*Id_Resident*) à un événement (*Id_Evenement*).
- **Un ÉVÉNEMENT** est décrit par un identifiant unique (*Id_Evenement*), une catégorie (*#Categorie*), un titre (*Titre*), un site où il a lieu (*Id_Site*), une date (*Date*) et une description (*Description*).
- **Une CATÉGORIE** d'événement est décrite par un identifiant unique (*Id_Categorie*) et une désignation (*Categorie*).
- **Un CONFLIT** est décrit par un identifiant unique (*Id_Conflits*), un état (*Etat*), un titre (*Titre*), une description (*Description*) et une date de signalement (*Date_Signalement*).
- **Un lien RESIDENTS_CONFLITS** relie un résident (*Id_Resident*) à un conflit (*Id_Conflits*).
- **Un ÉQUIPEMENT_LOGEMENT** est décrit par un identifiant unique (*Id_Equipement*) et un nom (*Nom_Equipement*).
- **Un ÉQUIPEMENT_SITE** est décrit par un identifiant unique (*Id_Equipement*) et un nom (*Nom_Equipement*).

(Il fallait une distinction entre les équipements du site et ceux du logement, en effet, un site peut avoir une piscine commune, ou un terrain de sport, mais un logement a des équipements qui lui sont propres et ne sont pas accessibles en 'dehors' du logement).

- **Un lien LOGEMENTS_EQUIPEMENTS** relie un équipement (*Id_Equipement*) à un logement (*Id_Logement*).
- **Un lien SITE_EQUIPEMENTS** relie un équipement (*Id_Equipement*) à un site (*Id_Site*).
- **Une MAINTENANCE** est décrite par un identifiant unique (*Id_Maintenance*), une date (*Date*), une description (*Description*), un rapport (*Rapport*), un niveau d'urgence

(*Urgence*), un logement concerné (*Id_Logement*), et un type de maintenance (*Id_Type_Maintenance*).

- Un **TYPE_MAINTENANCE** est décrit par un identifiant unique (*Id_Type_Maintenance*) et une désignation (*Type_Maintenance*).

3. L'expression en SQL des requêtes sur la base de données, qui sont utiles pour rendre les services susmentionnés, avec des explications bien structurées de chaque requête (≈description du codage).

&

4. La description du jeu de données utilisé pour tester les différentes requêtes sous des conditions typiques. La taille de ce jeu de données peut être faible mais on justifiera soigneusement le choix de ces données pour la complétude des tests (= validation)

Une fois notre modèle relationnel finalisé, nous avons procédé à la création des tables correspondantes. Pour répartir efficacement le travail, chaque membre de l'équipe a été chargé de créer une partie des tables. Pour la gestion du projet nous avons créé un repository et utilisons git.

Afin de simplifier l'initialisation de la base de données et la mise en commun des scripts SQL, nous avons développé un fichier `createAllTables.sql` que nous exécutons : `python link.py createAllTables.sql`. Ce script génère un fichier `createAllTables_combined.sql`, qui regroupe l'ensemble des instructions de création de tables, facilitant ainsi leur exécution en une seule commande, ainsi que le fichier `data.sql` qui est automatiquement mis à la fin de ce même fichier qui est créé.

En effet, pour **peupler** notre base de données avec un **jeu de données** réaliste et modulable (permettant de générer un **grand nombre d'entrées en modifiant une simple constante**), nous avons développé un script Python `main.py`. Ce script utilise la bibliothèque `Faker` pour générer des données fictives et réalistes pour chaque table. Lors de son exécution, il produit un fichier SQL, le fameux `data.sql` contenant les instructions d'insertion, automatisant ainsi le processus de remplissage de la base. La répartition de cette tâche a également été effectuée entre les membres du groupe, modifiant le même fichier mais pour nos tables respectives.

Pour être le plus réaliste, nous avons fait une distribution normale pour l'âge des personnes. Nous avons pris des dates qui sont actuelles pour les réservations. Toutes ces modifications nous permettent d'avoir des données fiables et cohérentes pour faire toutes nos requêtes.

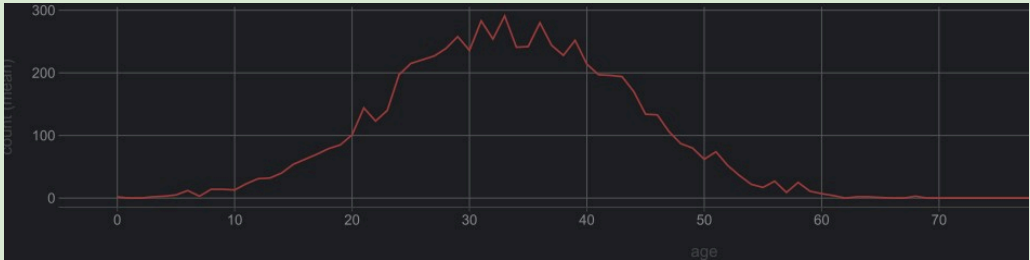
Quelle interaction avec les étudiants ?

Chaque membre de l'équipe-projet devra prendre en charge l'explication détaillée du fonctionnement d'une des requêtes SQL et le document rendu devra indiquer qui a rédigé chaque explication de requête.

Enfin, nous avons élaboré plusieurs requêtes SQL réalistes, adaptées aux besoins du projet, permettant d'exploiter efficacement les données de la base.

Questions Fonctionnelles Couvertes - Requêtes imposées dans le sujet :

1. **Disponibilité des logements** : Identification des logements disponibles selon période, type, emplacement et prix.
2. **Attribution optimisée des logements** : Affectation des logements aux réservations pour maximiser l'occupation.
3. **Suivi des interactions des résidents** : Quels résidents partagent un logement et quelles sont leurs interactions (événements, conflits) ?
4. **Analyse des interventions de maintenance** : Quels logements requièrent le plus d'interventions et pourquoi ?
5. **Impact des prolongations de séjour** : Identification des résidents ayant prolongé leur séjour et impact sur les futures réservations.
6. **Optimisation des événements communautaires** : Organisation des événements pour maximiser l'engagement des résidents.
7. **Amélioration de l'attractivité des logements** : Quels types de logements sont les plus demandés et quelles améliorations sont nécessaires ?

	Spécification des requêtes réalisées
Corentin	<p>Requête 6 : Optimisation des événements communautaires</p>  <p>(avec un nombre de résidents plus élevé que dans la base de données) La visualisation des résidents actuels pour un site donné permet d'organiser des événements communautaires. À l'aide de cette visualisation, il est possible de voir l'âge des résidents actuels pour un site donné. Et donc d'organiser des événements adaptés à la population.</p>
Anaïs	<p>Requête 2 : Attribution optimisée des logements</p> <p>Méthode utilisée : Procédure stockée avec la création d'une fonction <i>get_logement_optimise()</i>. L'objectif de cette fonction est d'optimiser l'attribution des logements en sélectionnant celui qui minimise la période d'inoccupation entre deux réservations.</p> <p>Entrées : <i>p_date_debut</i>, <i>p_date_fin</i> → Période de réservation souhaitée. <i>p_type</i>, <i>p_residence</i> → Type de logement et résidence ciblée (peuvent être <i>NULL</i>). <i>p_surface_min</i>, <i>p_capacite_lit_min</i> → Contraintes sur la surface et la capacité du logement.</p>

	<p>Sortie (<i>RETURNS TABLE(...)</i>) : Retourne une table contenant les caractéristiques du logement optimal.</p> <p>Étape 1 : Sélection des logements disponibles : Appelle la fonction <i>get_logements_disponibles()</i> (faite par Sabra) qui retourne les logements disponibles dans la période définie. Étape 2 : Trouver les réservations précédentes les plus proches. Étape 3 : Trouver les réservations suivantes les plus proches. Étape 4 : Sélectionner le logement optimisé.</p> <p>Un exemple d'utilisation de la fonction est donnée à la fin du fichier de cette même requête.</p> <p>Conclusion : Optimisation automatique des logements en minimisant les périodes d'inoccupation. Utilisation d'une combinaison de <i>WITH</i>, <i>JOIN</i> et <i>ORDER BY</i> pour trouver le logement optimal. Réutilisation de <i>get_logements_disponibles</i> pour identifier les logements compatibles avec les critères de recherche.</p> <p>Requête 4 : Analyse des interventions de maintenance</p> <p>Cette requête identifie les logements ayant le plus d'interventions de maintenance et affiche les types d'interventions effectuées. <i>l.id_logement</i> → Identifiant du logement. <i>COUNT(m.id_maintenance) AS nombre_interventions</i> → Compte le nombre total d'interventions de maintenance effectuées sur chaque logement. <i>STRING_AGG(DISTINCT tm.type_maintenance, ', ')</i> → Regroupe les différents types d'interventions sous forme d'une liste (séparée par des virgules). Jointure entre <i>maintenance</i> et <i>logements</i> → Pour récupérer le logement concerné par chaque intervention. Jointure avec <i>types_maintenance</i> → Pour associer chaque intervention à son type (ex: plomberie, électricité, etc.). Regroupe les résultats par logement → Chaque ligne correspond à un logement unique avec le nombre total d'interventions et leurs types. Trie les logements du plus grand au plus petit nombre d'interventions, mettant en avant ceux nécessitant le plus d'entretien. Si besoin, on peut ajouter un filtre (<i>HAVING</i>) pour afficher uniquement les logements ayant plus de n interventions.</p>
Alexis	<p>Requête 3 : Suivi des interactions des résidents Pour chaque logement actuellement occupé, on voit la liste des résidents qui le partagent, ce qui permet d'identifier les colocataires. La table affiche aussi les conflits signalés par ces résidents et les événements auxquels ils ont participé, offrant une vision de leurs interactions. De plus, une colonne indique si un conflit interne existe entre colocataires, en vérifiant si plusieurs résidents d'un même logement sont impliqués dans un même conflit non résolu. Cela permet d'identifier rapidement les logements où des tensions existent et de mieux comprendre la dynamique entre les résidents.</p> <p>Requête 7 : Amélioration de l'attractivité des logements Pour répondre à cette question on a fait deux requêtes.</p>

	<p>La première permet de comparer les différents types de logements. Pour chaque type de logement on voit le nombre de réservations associé pour observer quels types sont les plus et moins demandés. Si on remarque qu'un type de logement est beaucoup moins demandé par exemple, on peut essayer de trouver une raison à cela. Pour chercher s'il y a une raison, la table nous montre le nombre d'équipements moyen et le prix moyen d'une nuit par type de logement. On peut par exemple se rendre compte qu'un type est moins demandé car il y a peu d'équipement ou que son prix est trop élevé.</p> <p>La seconde requête permet de comparer chaque logement. On voit quels sont les logements les plus et moins demandés. Il y a écrit pour chaque logement leur nombre de réservations, leur site et leur type de logement. Pour essayer de comprendre pourquoi des logements marchent mieux que d'autres on voit leurs équipements, les équipements qu'ils ont sur leur site et leurs prix pour une nuit.</p>
Sabra	<p>Requête 1 : Disponibilité des logements Création d'une procédure <code>get_logements_disponibles</code> (dans un fichier du même nom) qui va recevoir en paramètre des caractéristiques souhaités pour le logement. Cette procédure va renvoyer une requête qui consistera à filtrer les logements disponibles selon les paramètres entrés.</p> <p>La procédure va sélectionner le numéro du logement (id logement), le type de logement, le nom du site, le nombre de chambres, de lits simples et de lits doubles. De plus, elle va également renvoyer le nombre de lits disponibles. La procédure calcule le nombre de lits réservés pour chaque logement à l'aide de <code>COUNT(*)</code>, puis soustrait ce total au nombre initial de lits disponibles (simples et doubles). Cette valeur est utilisée pour filtrer les logements disponibles.</p> <p>De plus, il y a aussi un filtrage selon les paramètres entrés (date de début de la réservation, date de fin, type de logement souhaité, le nom du site, la surface minimale et le nombre de chambres minimum) qui peuvent soit être `NULL` soit seront comparés aux caractéristiques des logements sélectionnés.</p> <p>Enfin, pour faire des requêtes sur cette fonction, des `select` d'exemples ont été créés dans <code>request_1.sql</code></p> <p>Requête 5 : Quels résidents ont prolongé leur séjour, et comment cela impacte les réservations futures ? Pour déterminer quels résidents ont prolongé leur séjour, on sélectionne le nom, prénom, numéro de téléphone, date de naissance, le type de logement, le nom du site, la date de début, la date de fin et enfin le numéro de réservation des résidents et de leur(s) réservation(s) qui ont été prolongés selon la table prolongation en faisant les jointures nécessaires avec les identifiants des tables.</p> <p>De plus, pour compléter cette question, la vue `reservations_dates_view` qui renvoie les réservations avec l'identifiant de la réservation, l'identifiant du logement réservé et enfin, les dates de début et de fin avec la date de fin de réservation mise à jour selon la date dans la table prolongations.</p>

De plus, deux triggers permettent de réguler les impacts des prolongations :

- **Premier trigger** : Lors d'une insertion ou d'une mise à jour dans la table `reservations`, ce trigger appelle la fonction `get_logements_disponibles` avec les nouvelles dates de début et de fin de la réservation. Il vérifie qu'un logement est bien disponible pour cette période. Si ce n'est pas le cas, une exception est levée.
- **Second trigger** : Lors d'une insertion ou mise à jour dans la table `prolongations`, ce trigger prend en paramètre uniquement la nouvelle date de fin de réservation. Il compare cette date avec les disponibilités du logement pour vérifier que la capacité maximale n'est pas dépassée. Si la prolongation entraîne une réservation sur des dates déjà occupées, une exception est levée.

Requêtes supplémentaires :

Nous avons également pensé des requêtes supplémentaires

request_supp1 (Anaïs) : Quelle est la résidence la plus rentable ?

request_supp2 (Alexis) : Tri des logements les plus cher à la location

request_supp3 (Alexis) : Taux d'occupation de chaque logement

request_supp4 (Anaïs) : Sites avec le plus de maintenances urgentes ce mois-ci (année...)

5. Une analyse critique du déroulement de votre projet : découpage et répartition du travail en équipe, difficultés rencontrées et réflexion sur des améliorations éventuelles de l'organisation des tâches au sein de l'équipe-projet... mais aussi les aspects bien réussis et les éléments clefs de cette réussite. Bref, comment vous y prendriez-vous « si c'était à refaire » ? Il ne s'agit pas de proposer des améliorations du produit logiciel en soi.

Concernant le déroulement du projet, nous estimons avoir bien structuré et réparti le travail de manière équilibrée. Chacun a contribué équitablement, tout en veillant à ce que les tâches de chacun n'impactent pas négativement celles des autres.

Les questions rencontrées ont été rapidement clarifiées, et la conception de notre base de données reflète fidèlement la réalité. Nous avons veillé à ce qu'elle soit facilement modulable et capable de gérer des milliers de tuples, permettant ainsi de formuler des requêtes à la fois fonctionnelles et réalistes.

L'élément clé de cette réussite réside sans aucun doute dans l'utilisation d'un programme Python exploitant une bibliothèque spécialisée, permettant de générer des tuples variés et proches de situations réelles. Le fait que les données ne soient pas statiques nous offre, d'une part, la possibilité de créer un nombre illimité de tuples, et d'autre part, d'élargir le champ des requêtes exploitables.

Nous avons aussi conçu le contrôle d'accès :

Concernant le contrôle d'accès, on a maintaineur, salarié, technicien et client. Le maintaineur a accès à toute la base de données co-living. Le salarié standard va pouvoir SELECT, INSERT, UPDATE sur toutes les tables. Ça lui servira à ajouter des logements ou modifier des logements si besoin. Le technicien va pouvoir modifier la maintenance uniquement. Il pourra modifier la catégorie rapport de la maintenance. Le client peut uniquement voir les activités.