UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

RAPPORT S3 APP2

Base de données
GIF325

Présenté à
Bernard Beaulieu

Présenté par
Bendjeddou Mohamed El Hadi - benm2043
Bryan Gasse – gasb3002

Sherbrooke – 17 septembre 2024

TABLE DES MATIÈRES

1. INTRODUCTION

La problématique traite la conception et de l'implémentation d'une base de données relationnelle pour le système de gestion des réservations de salles au sein d'une faculté de génie. Ce rapport analysera les étapes suivies pour sa mise en place, allant du Modèle Conceptuel de Données (MCD), à sa transformation en Modèle Logique de Données (MLD), la normalisation de ce dernier, l'intégration des contraintes d'intégrité et d'unicité, jusqu'à la création et la manipulation de la base de données. Le respect des règles de conception sera examiné, et les choix techniques seront justifiés pour assurer la cohérence et la performance.

2. MODÈLE CONCEPTUEL DE DONNÉES

Afin d'éviter les redondances et garantir l'intégrité des données, il est essentiel d'adopter un formalisme de modélisation qui permet de définir et structurer les données impliquées dans les flux opérationnels. Le Modèle Conceptuel de Données (MCD), illustré par la Figure 1 ci-dessous, représente graphiquement les données extraites de l'énoncé de la problématique. Il met en évidence les entités principales, leurs attributs, et les relations qui les relient. Ce modèle organise les données de manière claire et cohérente, facilitant leur implémentation ultérieure dans la base de données. En particulier, il définit :

- **Entités principales :**
  - LOCAL : Représente les salles avec leurs caractéristiques telles que le pavillon, le numéro et la capacité
  - MEMBRE : Désigne les utilisateurs, identifiés par un CIP et définis par leur nom, prénom et courriel
  - CUBICULE : Un espace enfant lié à un LOCAL, avec une relation de cardinalité 1:1
  - STATUT et DEPARTEMENT : Définit respectivement le statut d'un membre et son département d'appartenance
  - CATEGORIE et CARACTÉRISTIQUE : Attributs supplémentaires associés aux locaux pour une meilleure décomposition et gestion
- **Associations :**
  - RESERVER : Relie les membres et les locaux à travers une réservation

- POSSEDER : Gère la relation entre un membre et ses statuts
- APPARTENIR : Associe un membre à un département
- ATTRIBUER : Lie des caractéristiques spécifiques à des locaux
- CONTENIR : Gestion de la relation entre un local et un cubicule

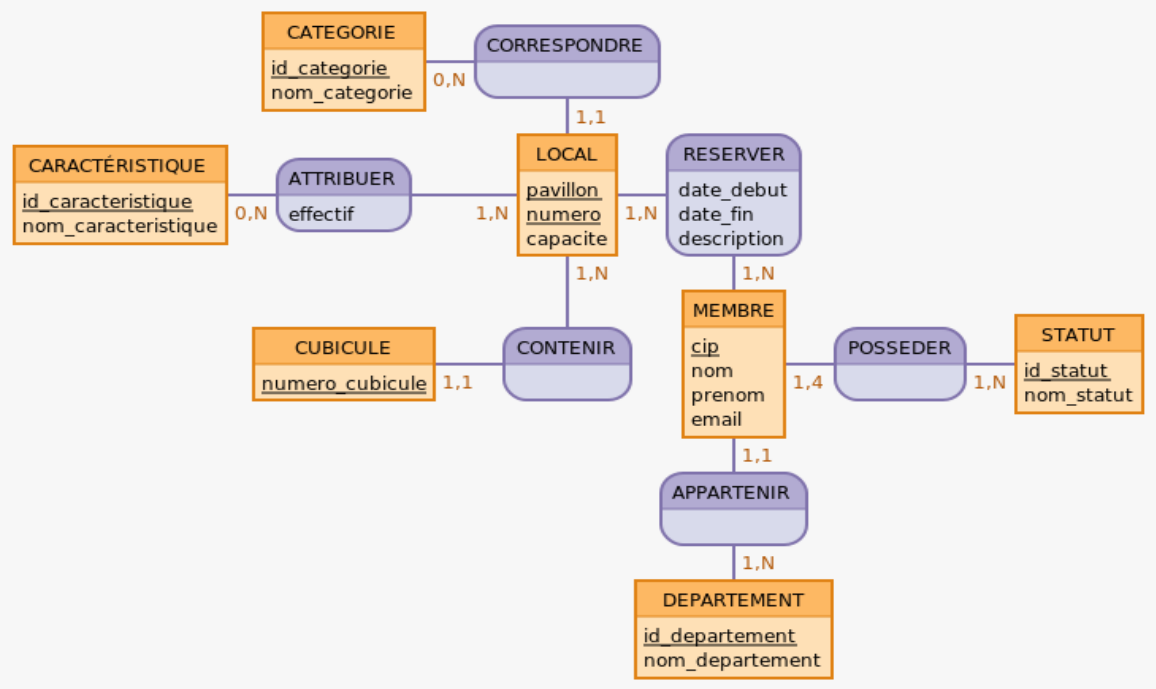

Figure 1: Modèle Conceptuel de Données pour la réservation des salles

3. MODÈLE RELATIONNEL DE DONNÉES

Le modèle relationnel de données est un cadre conceptuel permettant d'organiser et de structurer les données de manière logique via des relations. Pour optimiser les performances de la base de données, différentes méthodes d'indexation sont utilisées en fonction des caractéristiques des tables. Les index linéaires sont employés pour des tables avec peu de changements et un nombre limité de données, telles que CARACTÉRISTIQUE, CATÉGORIE ou STATUT. En revanche, des B-Trees sont principalement appliqués à la table RESERVER, qui peut contenir un grand volume de données, afin d'améliorer la rapidité des recherches en évitant les scans séquentiels. Il est représenté par :

CUBICULE (numero_cubicule, #pavillon, #numero)

- Le champ *numero_cubicule* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *CUBICULE*.
- Les champs *pavillon* et *numero* sont des clés étrangères. Ils ont migré par l'association de dépendance fonctionnelle *CONTENIR* à partir de l'entité *LOCAL* en perdant leur caractère identifiant.

ATTRIBUER (#pavillon, #numero, #id_caracteristique, effectif)

- Les champs *pavillon* et *numero* font partie de la clé primaire de la table. Ce sont des clés étrangères qui ont migré directement à partir de l'entité *LOCAL*
- Le champ *id_caracteristique* fait partie de la clé primaire de la table. C'est une clé étrangère qui a migré directement à partir de l'entité *CARACTÉRISTIQUE*
- Le champ *effectif* était déjà un simple attribut de l'association *ATTRIBUER*

CARACTÉRISTIQUE (id_caracteristique, nom_caracteristique)

- Le champ *id_caracteristique* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *CARACTÉRISTIQUE*
- Le champ *nom_caracteristique* était déjà un simple attribut de l'entité *CARACTÉRISTIQUE*

CATEGORIE (id_categorie, nom_categorie)

- Le champ *id_categorie* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *CATEGORIE*
- Le champ *nom_categorie* était déjà un simple attribut de l'entité *CATEGORIE*

DEPARTEMENT (id_departement, nom_departement)

- Le champ *id_departement* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *DEPARTEMENT*
- Le champ *nom_departement* était déjà un simple attribut de l'entité *DEPARTEMENT*

LOCAL (pavillon, numero, capacite, #id_categorie)

- Les champs *pavillon* et *numero* constituent la clé primaire de la table. C'étaient déjà des identifiants de l'entité *LOCAL*
- Le champ *capacite* était déjà un simple attribut de l'entité *LOCAL*
- Le champ *id_categorie* est une clé étrangère. Il a migré par l'association de dépendance fonctionnelle *CORRESPONDRE* à partir de l'entité *CATEGORIE* en perdant son caractère identifiant

MEMBRE (cip, nom, prenom, email, #id_departement)

- Le champ *cip* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *MEMBRE*
- Les champs *nom*, *prenom* et *email* étaient déjà de simples attributs de l'entité *MEMBRE*.
- Le champ *id_departement* est une clé étrangère. Il a migré par l'association de dépendance fonctionnelle *APPARTENIR* à partir de l'entité *DEPARTEMENT* en perdant son caractère identifiant

POSSEDER (#cip, #id_statut)

- Le champ *cip* fait partie de la clé primaire de la table. C'est une clé étrangère qui a migré directement à partir de l'entité *MEMBRE*
- Le champ *id_statut* fait partie de la clé primaire de la table. C'est une clé étrangère qui a migré directement à partir de l'entité *STATUT*

RESERVER (#id_reservation, #cip, #pavillon, #numero, date_debut, date_fin, description)

- Le champ *cip* fait partie de la clé primaire de la table. C'est une clé étrangère qui a migré directement à partir de l'entité *MEMBRE*
- Les champs *pavillon* et *numero* font partie de la clé primaire de la table. Ce sont des clés étrangères qui ont migré directement à partir de l'entité *LOCAL*
- Les champs *date_debut*, *date_fin* et *description* étaient déjà de simples attributs de l'association *RESERVER*

STATUT (id_statut, nom_statut)

- Le champ *id_statut* constitue la clé primaire de la table. C'était déjà un identifiant de l'entité *STATUT*
- Le champ *nom_statut* était déjà un simple attribut de l'entité *STATUT*

4. PROCÉDURE POUR LE TABLEAU DE RÉSERVATION

L'objectif ici est de visualiser rapidement la disponibilité des locaux pour faciliter la gestion des réservations. Le système doit fournir une vue tabulaire qui reflète les réservations effectuées pour un ensemble de locaux dans une plage horaire donnée. Le tableau est organisé en créneaux de 15 minutes, chaque cellule représentant un local à une heure donnée, contenant soit la description de la réservation, soit NULL s'il n'y a pas de réservation. Pour implémenter cette fonctionnalité il est mieux de la décomposé en sous-fonctions, chacune a une responsabilité claire, ce qui facilite la compréhension et la maintenance. L'approche utilisée le fait en 4 étapes :

1. **Générer des Plages Horaires** : Le moyen le plus simple d'atteindre cet objectif est d'utiliser la fonction **generate_series()** de PostgreSQL, qui est efficace pour générer des séquences de valeurs.

```
Untitled
CREATE OR REPLACE FUNCTION generate_time_slots(
    date_debut TIMESTAMP,
    date_fin TIMESTAMP
)
RETURNS TABLE (time_slot TIMESTAMP) AS $$
BEGIN
    RETURN QUERY
    SELECT generate_series(date_debut, date_fin, '15 minutes'::interval) AS time_slot;
END;
$$ LANGUAGE plpgsql;
```

Figure 2: Fonction generate_time_slots()

2. **Obtenir les Locaux:** Une fonction **get_local_data(**id_categorie**)** pour récupérer tous les locaux dans une catégorie spécifique via une simple condition.

```
Untitled
CREATE OR REPLACE FUNCTION get_local_data(id_categorie INT)
RETURNS TABLE(pavillon VARCHAR, numero VARCHAR) AS $$
#variable_conflict use_variable
BEGIN
    RETURN QUERY
    SELECT l.pavillon, l.numero
    FROM local l
    WHERE l.id_categorie = id_categorie;
END;
$$ LANGUAGE plpgsql;
```

Figure 3: Fonction get_local_data()

3. **Obtenir les Réservations:** Une fonction **get_reservation_data(**start_timestamp TIMESTAMP, end_timestamp TIMESTAMP**)** pour obtenir les réservations qui se chevauchent avec les plages horaires.

```
Untitled
CREATE OR REPLACE FUNCTION get_reservation_data(start_timestamp TIMESTAMP, end_timestamp TIMESTAMP)
RETURNS TABLE(pavillon VARCHAR, numero VARCHAR, date_debut TIMESTAMP, date_fin TIMESTAMP, description VARCHAR) AS $$
#variable_conflict use_variable
BEGIN
    RETURN QUERY
    SELECT r.pavillon, r.numero, r.date_debut, r.date_fin, r.description
    FROM reserver r
    WHERE r.date_debut < end_timestamp
    AND r.date_fin > start_timestamp;
END;
$$ LANGUAGE plpgsql;
```

Figure 4: Fonction get_reservation_data()

4. **Combiner les Données:** Une fonction principale **tableau (TIMESTAMP, TIMESTAMP, INT)** qui joint les plages horaires et les locaux (CROSS JOIN), puis les associe aux réservations en utilisant une jointure gauche (LEFT JOIN). La fonction COALESCE permet de montrer NULL si aucune réservation n'est trouvée pour la combinaison donnée.

```
Untitled
drop function tableau(TIMESTAMP, TIMESTAMP, INT);
CREATE OR REPLACE FUNCTION tableau(
    q_date_debut TIMESTAMP,
    q_date_fin TIMESTAMP,
    q_id_categorie INT
)
RETURNS TABLE(
    pavillon VARCHAR,
    numero VARCHAR,
    time_slot TIMESTAMP,
    description VARCHAR
) AS $$
#variable_conflict use_column
BEGIN
    RETURN QUERY
    WITH
    time_slots AS (
        SELECT time_slot
        FROM generate_time_slots(q_date_debut, q_date_fin)
    ),
    local_data AS (
        SELECT pavillon, numero
        FROM get_local_data(q_id_categorie)
    ),
    reservation_data AS (
        SELECT pavillon, numero, date_debut, date_fin, description
        FROM get_reservation_data(q_date_debut, q_date_fin)
    )
    SELECT
        l.pavillon,
        l.numero,
        ts.time_slot,
        COALESCE(r.description, NULL) AS description
    FROM time_slots ts
    CROSS JOIN local_data l
    LEFT JOIN reservation_data r
        ON l.pavillon = r.pavillon
        AND l.numero = r.numero
        AND ts.time_slot ⩾ r.date_debut
        AND ts.time_slot < r.date_fin
    ORDER BY l.pavillon, l.numero, ts.time_slot;
END;
$$ LANGUAGE plpgsql;
```

Figure 5: fonction tableau ()

Pour vérifier les plages horaires des locales de catégorie “1” entre 08h et 13h, il suffit de faire un

```
SELECT * FROM tableau (
'2024-10-16 08:00',
'2024-10-16 13:00',
1
);
```

pavillon	numero	time_slot	description
C1	1936	2024-10-16 08:00:00.000	
C1	1936	2024-10-16 08:15:00.000	
C1	1936	2024-10-16 08:30:00.000	Travail en equipe
C1	1936	2024-10-16 08:45:00.000	Travail en equipe
C1	1936	2024-10-16 09:00:00.000	Travail en equipe
C1	1936	2024-10-16 09:15:00.000	Travail en equipe
C1	1936	2024-10-16 09:30:00.000	Travail en equipe
C1	1936	2024-10-16 09:45:00.000	Travail en equipe
C1	1936	2024-10-16 10:00:00.000	
C1	1936	2024-10-16 10:15:00.000	
C1	1936	2024-10-16 10:30:00.000	
C1	1936	2024-10-16 10:45:00.000	
C1	1936	2024-10-16 11:00:00.000	Just chilling
C1	1936	2024-10-16 11:15:00.000	Just chilling
C1	1936	2024-10-16 11:30:00.000	Just chilling
C1	1936	2024-10-16 11:45:00.000	Just chilling
C1	1936	2024-10-16 12:00:00.000	
C1	1936	2024-10-16 12:15:00.000	

Figure 6: résultat de la requête

5. GESTION DES CUBICULES

Le modèle proposé, avec l'entité CUBICULE, présente des limitations significatives pour la réservation des sous-locales. La principale difficulté réside dans la correspondance et l'intégration des réservations entre les locaux et leurs cubicules associés. Cette approche entraîne des complexités lors de l'agrégation des données, et des incohérences peuvent apparaître lorsqu'on tente de synchroniser les réservations entre les locaux et leurs sous-locaux, la gestion des relations hiérarchiques devient compliquée.

Toutefois, il est possible de surmonter cette limitation en adoptant une approche de modélisation plus cohérente basée sur une **association réflexive** dans la table LOCAL. Cela consiste à remplacer l'entité CUBICULE par une auto-association dans la table LOCAL, permettant de gérer les relations hiérarchiques entre les locaux et leurs sous-locaux de manière intégrée avec l'ajoute des **triggers** pour assurer la propagation des réservations (Lors de l'insertion d'une réservation dans un local parent, le trigger met à jour les réservations de tous les sous-locaux associés et inversement)

6. CHEVAUCHEMENT DES RÉSERVATIONS

Pour assurer la gestion efficace des réservations et éviter les conflits de réservation, un algorithme simple a été mis en place. L'objectif est de garantir qu'aucunes deux réservations ne se chevauchent pour le même local, tout en respectant les permissions d'accès des utilisateurs. Voici, l'algorithme de gestion des chevauchements :

1. **Vérification de Permission:** vérifiez que l'utilisateur a les autorisations nécessaires pour effectuer une réservation sur le local cible. Si l'utilisateur ne possède pas les permissions requises, générez une erreur et annulez la tentative.

2. **Vérification de Disponibilité:** Interrogez la table des réservations pour obtenir toutes les réservations existantes sur le local en question pendant la plage horaire souhaitée.
    a. Identifiez la réservation la plus récente qui pourrait chevaucher la nouvelle demande.
    b. Comparez les dates de cette réservation existante avec les dates demandées. Si un chevauchement est détecté, générez une erreur. Sinon, la réservation est considérée comme valide et peut être ajoutée à la base de données.
3. **Insertion de la Réservation :** Si les vérifications précédentes sont toutes satisfaites (aucun chevauchement détecté et permission accordée), procédez à l'insertion de la nouvelle réservation dans la table appropriée.

Untitled

```
CREATE OR REPLACE FUNCTION public.handle_reservation()
 RETURNS trigger
 LANGUAGE plpgsql
AS $function$
DECLARE
    is_permissible BOOLEAN;
    is_available BOOLEAN;
    dates_check BOOLEAN;
BEGIN
    -- Check if the user has permission (manfully implemented)
    is_permissible := public.check_permission(NEW.cip, NEW.pavillon, NEW.numero);
    IF NOT is_permissible THEN
        RAISE EXCEPTION 'Permission denied: The user does not have permission to book this local';
    END IF;

    -- Check if the local is available (manfully implemented)
    is_available := public.check_availability(NEW.pavillon, NEW.numero, NEW.date_debut, NEW.date_fin);
    IF NOT is_available THEN
        RAISE EXCEPTION 'Local is not available: The local is already booked during the proposed dates.';
    END IF;

    -- Check if the dates are valid (manfully implemented)
    dates_check := public.check_dates(NEW.date_debut, NEW.date_fin);
    IF NOT dates_check THEN
        RAISE EXCEPTION 'Invalid reservation dates: Reservation must be less than 24 hours.';
    END IF;

    RETURN NEW;
END;
$function$
;
-- Ajouter le trigger avant chaque insertion ou modification
create trigger trg_handle_reservation
    before insert or update
    on public.reserver for each row execute function handle_reservation()
```

Figure 7: le trigger trg_handel_reservation()

7. MODÈLE EN ÉTOILE DE L'ENTREPÔT DE DONNÉES

Le modèle en étoile est une architecture de schéma conçue pour organiser les données de manière à optimiser l'analyse et les requêtes OLAP (Online Analytical Processing). Ce modèle est largement utilisé dans les entrepôts de données pour faciliter l'analyse multidimensionnelle, en

offrant un équilibre entre simplicité, efficacité des requêtes et redondance des données. Le schéma proposé est représenté par la figure 8 ci-dessous.

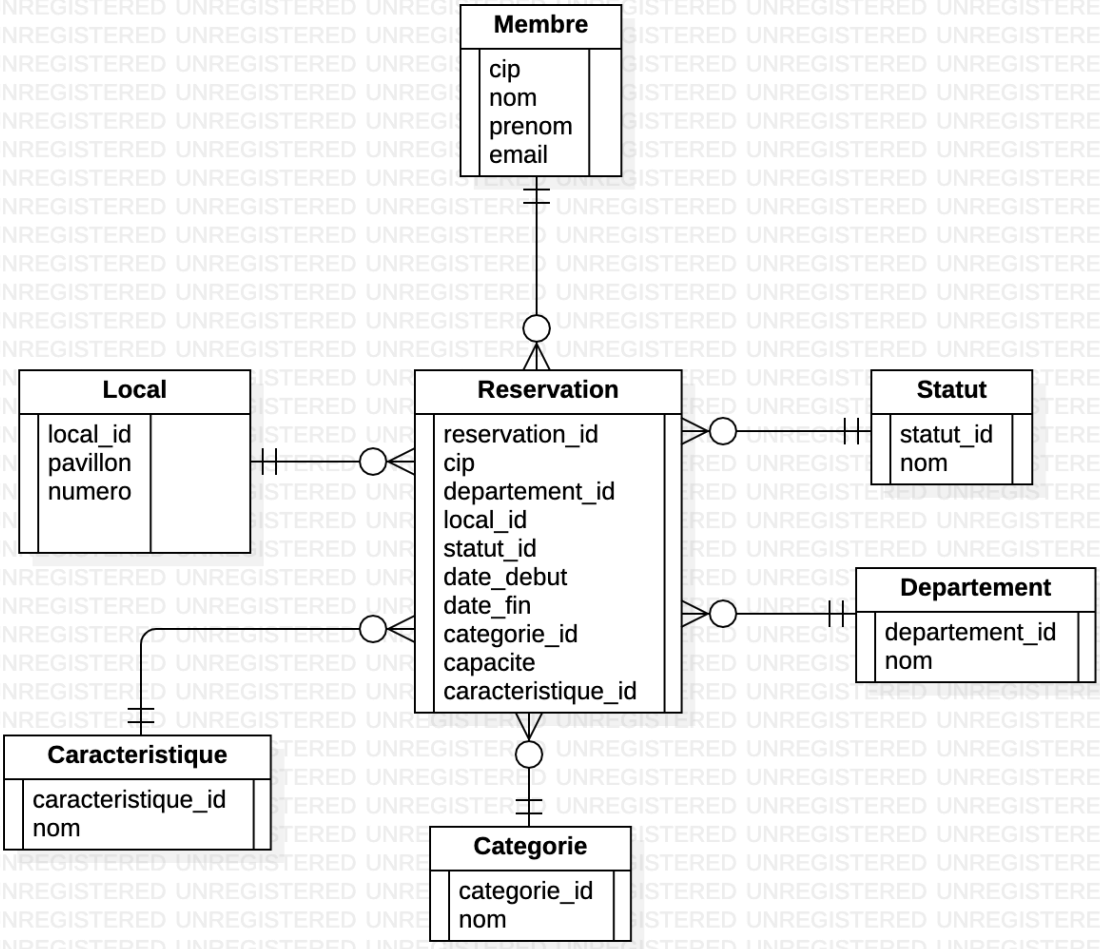

Figure 8: Schéma en étoile

8. DISCUSSION SUR L'ENTREPÔT DE DONNÉES ET L'INDEXATION

Quant à lui, l'entrepôt de données est un système centralisé dédié à la gestion et au stockage des données, visant à soutenir la prise de décision et l'analyse au sein d'une organisation. Il intègre des données provenant de diverses sources externes, telles que les bases de données transactionnelles. Les données sont organisées, nettoyées et préparées pour des requêtes analytiques, des rapports et des analyses avancées. Contrairement aux bases de données

transactionnelles, un entrepôt de données est optimisé pour des requêtes complexes plutôt que pour des opérations fréquentes de lecture et d'écriture.

Nous pourrions intégrer un entrepôt de données à notre système de réservation pour analyser des informations telles que la fréquence de réservation des locaux ou les membres les plus actifs, les périodes les plus fréquentes des réservations, etc. Cet entrepôt se connecterait à notre base de données durant les périodes de faible activité, comme la nuit, pour collecter et analyser les nouvelles données.

Étant donné que les entrepôts de données peuvent contenir des millions de lignes, des requêtes non optimisées peuvent devenir extrêmement lentes. L'indexation joue un rôle crucial en garantissant que les analyses se déroulent efficacement, sans être entravées par le volume massif de données. Les types d'index, tels que les B-trees, les index bitmap, et les index multicolonne, sont sélectionnés en fonction de la nature des requêtes et des dimensions les plus souvent examinées. Ces index optimisent l'accès aux données et améliorent les performances des systèmes en permettant une recherche rapide des informations pertinentes, sans nécessiter un parcours complet des tables.

9. CONCLUSION

Ce projet a impliqué la conception et l'implémentation d'une base de données relationnelle pour la gestion des réservations de salles, en suivant méthodiquement les étapes de la conception à l'exécution. Le Modèle Conceptuel de Données (MCD) a été transformé en Modèle Logique de Données (MLD), avec une attention particulière portée à la normalisation et à l'intégration des contraintes d'intégrité et d'unicité. Cette démarche s'inscrit dans un cadre où l'optimisation de la gestion des locaux, la prévention des conflits de réservation, et la traçabilité des modifications sont des priorités.