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

RAPPORT APP1 – S3

Système répartis et bases de données
GIF390, GIF620

Présenté à
Domingo Palao Munoz

Présenté par
Équipe numéro 01
Marie-Eve Castonguay – casm1907
Julien Lavoie – lavj2421

TABLE DES MATIÈRES

1. Diagramme de déploiement 1

2. Modèles d'architecture logicielle proposés 2

3. Considérations par rapport à la transparence, homogénéité, couplage 3

4. Modèle conceptuel de données 4

5. Modèle relationnel de données 4

6. Procédures pour le tableau de réservation 5

6.1 Gestion de cubicules 6
6.2 Chevauchement des réservations 7
6.3 Méthodes d'indexation 7

7. Modèle en étoile de l'entrepôt de données 8

LISTE DES FIGURES

Figure 1: Diagramme de déploiement du système de réservation de locaux proposé 1

Figure 2: Diagramme Entité - Relation de la base de données implémentée sur PostgreSQL 4

Figure 3: Modèle relationnel de la base de données implémentée sur PostgreSQL 4

Figure 4: Extrait de code de la vue TABLEAU affichant la plage de réservations selon la catégorie de local 5

Figure 5: Schéma des contraintes logiques empêchant une réservation par chevauchement 7

Figure 6: Schéma étoile de l'entrepôt de données autour de la table Réservation 8

1. DIAGRAMME DE DÉPLOIEMENT

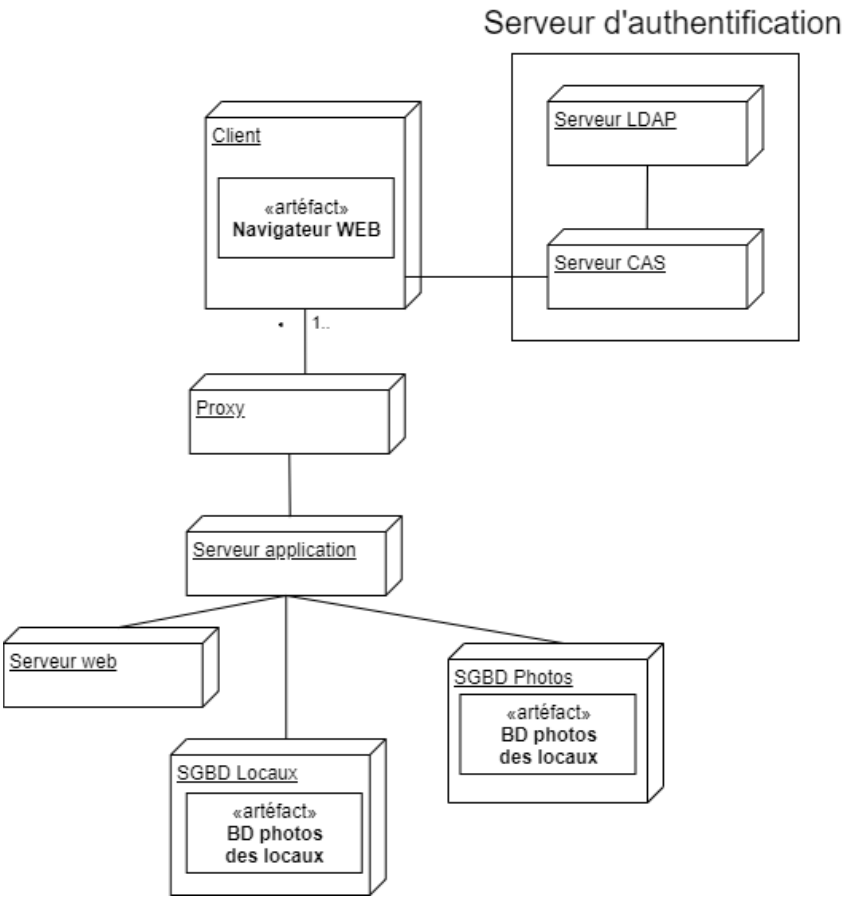

Figure 1: Diagramme de déploiement du système de réservation de locaux proposé

Notre diagramme de déploiement commence tout d'abord avec le client. Celui-ci communique avec le système en utilisant un navigateur web. Lorsque celui-ci veut accéder à un service de l'université, celui-ci sera redirigé vers une page d'authentification afin de vérifier qu'il dispose des droits pour accéder à la ressource demander. C'est donc ici que le serveur d'authentification joue son rôle dans le diagramme. Suite à l'approbation du serveur d'authentification, l'usager peut accéder, via un proxy, aux différents services du système. Dans notre cas, le serveur d'application redirige l'utilisateur vers les différentes ressources accessibles, le serveur web pour les différentes pages web que l'utilisateur veut accéder, ainsi que les bases de données.

2. MODÈLES D'ARCHITECTURE LOGICIELLE PROPOSÉS

Le modèle d'architecture que l'on propose ressemble beaucoup à celui d'une application «Three-Tier». Ce type d'architecture est un exemple d'architecture client-serveur, où les services offerts par le serveur sont constamment offerts et où le client se connecte quand il le souhaite. Le modèle que l'on propose est séparé en trois tiers isolés: l'interface usagé, la logique de l'application et l'accès aux données. Le fait que ces trois parties soient séparées fait en sorte que l'application est très modulaire, ce qui facilite grandement le développement d'une des trois parties indépendamment des deux autres. Les trois tiers sont distincts selon le type de logique qu'ils gèrent. L'interface usager permet au client de communiqué avec l'application et les données via un navigateur web. Dans notre cas, ce navigateur web communique aussi avec un serveur qui fournit un service d'authentification (CAS et LDAP). Le proxy est le middleware qui permet la communication entre le client et le serveur application, tandis que le serveur application comporte la logique des différentes fonctionnalités auxquelles le client aura accès. C'est aussi l'intermédiaire qui permet la communication entre le client/l'interface usagé et les bases de données. Finalement, les serveurs de gestions des bases de données gèrent tout ce qui concerne l'accès et la modification des données stockées dans les bases de données (comme nous l'avons fait dans la problématique 2).

3. CONSIDÉRATIONS PAR RAPPORT À LA TRANSPARENCE, HOMOGÉNÉITÉ, COUPLAGE

Les choix faits dans la conception de notre système permettent une certaine transparence pour l'utilisateur. On parle tout d'abord de transparence de localisation. En effet, le client ne communique jamais directement avec les différentes composantes du système. Il ne fait qu'accéder aux différents services offerts par les serveurs via un proxy. Il ne sait pas où se fait la communication et il voit simplement la réponse que le système lui renvoie. Puisque plusieurs clients peuvent tenter d'accéder à aux mêmes ressources (comme la plage de réservation des locaux), le système doit garder sa transparence de concurrence. Le design du système doit s'assurer que les différents clients ne ressentent pas que d'autres clients soient en train d'accéder aux mêmes ressources. Pour ce qui est du couplage des composants, notre système à une liaison forte, parce que toutes les composantes font partie du même système, à l'exception de la base de données externe de photo. Ainsi, le proxy connait chacune des adresses IP des différents serveurs, via un DNS, afin d'avoir un peu plus de flexibilité dans le futur, par exemple l'ajout d'un autre serveur. Dans le cas de l'hétérogénéité de notre système, on peut parler d'hétérogénéité de plateforme ainsi que de système d'exploitation. En effet, on peut accéder aux différents services avec plusieurs appareils différents un téléphone mobile, un ordinateur portable, un ordinateur fixe, etc. Puisque l'accès aux services est possible à partir d'une variété de système d'exploitation, notre système doit permettre à toutes les sortes de clients d'utiliser les services offerts. Les différentes composantes du système peuvent aussi faire affaire avec différents systèmes d'exploitation, selon le choix du concepteur logiciel. Par exemple, le client peut se connecter avec un ordinateur ayant Windows, sur un serveur ayant Linux, ce qui n'empêche pas non plus le processus inverse, un client Linux ou Mac et un serveur Windows.

4. MODÈLE CONCEPTUEL DE DONNÉES

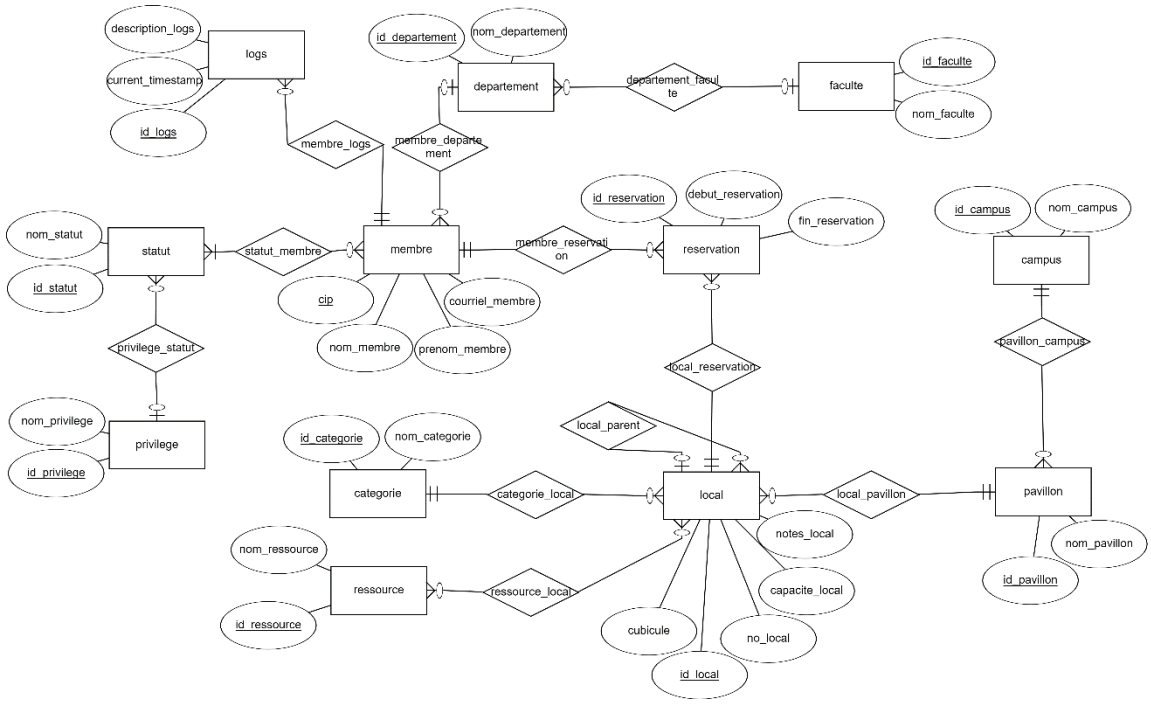

Figure 2: Diagramme Entité - Relation de la base de données implémentée sur PostgreSQL

5. MODÈLE RELATIONNEL DE DONNÉES

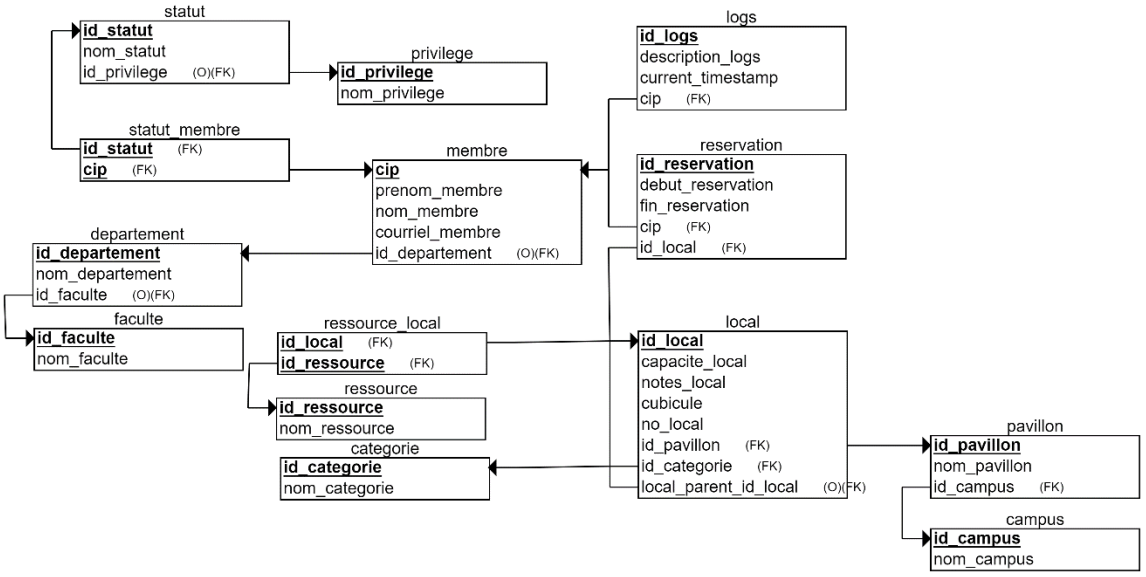

Figure 3: Modèle relationnel de la base de données implémentée sur PostgreSQL

6. PROCÉDURES POUR LE TABLEAU DE RÉSERVATION

```sql
CREATE OR REPLACE VIEW TABLEAU AS
    WITH local_reserve AS (
        SELECT local.id_local,
               local.id_pavillon,
               local.no_local,
               categorie.nom_categorie
        FROM local INNER JOIN categorie ON local.id_categorie = categorie.id_categorie
    ), temps AS (
        SELECT GS AS temp
        FROM generate_series(TIMESTAMP '2020-05-09 08:00:00-05', TIMESTAMP '2020-05-09 23:00:00-05', '15 minutes') GS
    )
    SELECT local_reserve.id_pavillon,
           local_reserve.no_local,
           local_reserve.nom_categorie,
           temp AS temps,
           reservation.cip
    FROM local_reserve
    CROSS JOIN temps
    LEFT JOIN reservation ON local_reserve.id_local = reservation.id_local AND temps.temp >= reservation.debut_reservation AND temps.temp <= reservation.fin_reservation
    ORDER BY id_pavillon, no_local, temps;
```

Figure 4: Extrait de code de la vue TABLEAU affichant la plage de réservations selon la catégorie de local

La procédure «TABLEAU» qui permet d'afficher toutes les réservations pour une catégorie de local, nous utilisons une «vue». Celle-ci permet de sélectionner les éléments voulus à l'affichage. Aussi, une vue crée une table, qui peut être considérée comme temporaire, ce qui permet ici de faire une requête sur cette table. Étant donné qu'une requête est possible sur cette table, on peut aussi filtrer les résultats qu'elle va retourner. On peut ensuite simplement passer la catégorie de local voulu dans la requête et obtenir tous les locaux de cette catégorie, avec les plages de réservation libres ou non. La colonne cip est NULL si le local est libre à cette heure, sinon elle contient le cip du membre ayant fait la réservation. La plage d'heures de réservation (les cases de 15 minutes) d'une journée n'est pas contenue dans une table, mais elle est plutôt générée grâce à la fonction « generate_series », qui permet de créer une colonne comportant les heures spécifiées par un intervalle de temps donnée. Par exemple, ici on mentionne que l'on veut les heures de 8h le matin à 23h le soir, avec des intervalles de 15 minutes.

6.1 GESTION DE CUBICULES

Tableau 1: Exemple d'une table «local»

Local_id (INT)	no_local (INT)	cubicule (VARCHAR)	
1	5012	NULL	Local en entier (Parent)
2	5012	'5012-1'	Cubicule #1
3	5012	'5012-2'	Cubicule #2
4	5012	'5012-3'	Cubicule #3

L'entité local possède un attribut local_id (sa clé primaire), no_local (un entier) et un identifiant cubicule (VARCHAR) qui est NULL si le local n'est pas un cubicule.

CAS DE GESTION DE CUBICULES #1 – PARENT RÉSERVÉ

Si un local parent est réservé, les cubicules individuels à l'intérieur de ce local parent ne peuvent être réservés en même temps. Ainsi, dans notre schéma ER, on établi une relation de l'entité local avec elle-même puisqu'un cubicule possède une relation avec un local parent (dont le no_local est le même, mais que son attribut cubicule est NULL). Si on tente de louer un cubicule alors que la parent est déjà réservé, le système retourne un notice « Local parent réservé, impossible de compléter la réservation » et la réservation demandée ne s'ajoute pas à la table réservation.

CAS DE GESTION DE CUBICULES #2 – CUBICULE RÉSERVÉ

Si un cubicule est réservé, il est possible de réserver d'autre cubicules dans le même local. Par contre, il n'est pas possible de réserver le local parent. Ainsi, lors d'une nouvelle réservation, on vérifie dans la table des réservations si un local portant le même numéro de local et ayant un attribut cubicule NULL (le local parent) est réservé. Si oui, le système retourne un notice « Cubicule déjà réservé dans ce local, impossible de compléter la réservation ». Si non, la réservation est possible et s'ajoute à la table réservation.

6.2 CHEVAUCHEMENT DES RÉSERVATIONS

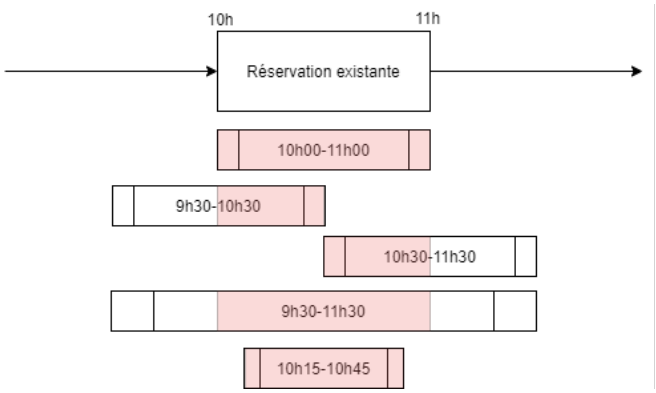

Figure 5: Schéma des contraintes logiques empêchant une réservation par chevauchement

Dans le schéma ci-dessus, les différentes contraintes logiques qui pourraient empêcher une réservation d'en chevaucher une autre sont illustrées. Premièrement, une réservation à exactement la même heure ne devraient pas fonctionner pour un même local. Ensuite, ni le début ni la fin d'une nouvelle réservation devrait être entre le début et la fin d'une réservation existante. Une nouvelle réservation qui chevauche l'entièreté d'une réservation existante n'est pas autorisée non plus. Celles-ci ont été implémentées à l'aide d'une clause BEFORE INSERT dans un trigger. Avant d'insérer le nouveau record dans la table réservation, la fonction check_reservation() vérifie que toutes les contraintes sont respectées. Si elles ne le sont pas, le système renvoie un *notice* « Reservation overlap » et la réservation ne s'effectue pas.

6.3 MÉTHODES D'INDEXATION

Lorsque c'était possible, nous avons utilisé les clés primaires comme façon d'indexer les records. Ces clés primaires sont toujours uniques, et le plus souvent possible, naturels par rapport à l'entité qui les concerne. Par exemple, les membres de la faculté sont indexés par leur cip, un identifiant qui est toujours composé de 8 caractères (4 lettres et 4 chiffres). Chaque combinaison de ces 8 caractères sont uniques et permettent d'identifier un seul membre. Pour d'autres entité qui ne possède pas nécessairement d'identifiant naturel, tels que les départements, les facultés, les campus et les statuts des membres, nous avons utilisé des SERIAL (entier qui s'auto-incrémente en insérant les données dans la table). De cette façon, nous n'avions pas à nous soucier de commettre une erreur en indexant manuellement les données.

7. MODÈLE EN ÉTOILE DE L'ENTREPÔT DE DONNÉES

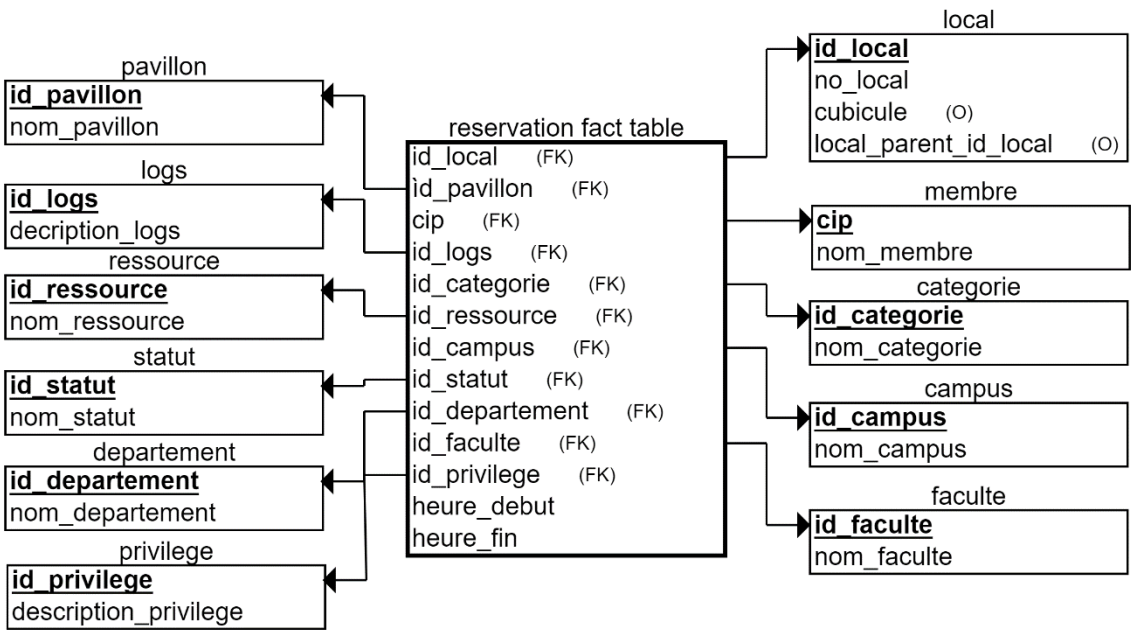

Figure 6: Schéma étoile de l'entrepôt de données autour de la table Réservation

Le modèle ci-dessus représente les informations accumulées dans la base de données sous forme d'un schéma étoile. L'application que nous avons développée pourrait servir de point de départ à la création d'un entrepôt de données puisque toutes les dimensions par rapport à la table de faits centrale sont fournies. Un entrepôt de données sert principalement à faire l'analyse des données stockées, tandis qu'une base de données tourne autour d'un concept de transactions (insert, delete, modify). Un entrepôt de données pourrait s'intégrer au système de gestion des salles de cours en faisant l'analyse des transactions dans les bases de données. Par exemple, elle pourrait créer des statistiques par rapport aux locaux les plus en demande, les heures les plus achalandées, les membres faisant le plus de réservations, leurs facultés, etc… Pour l'efficacité d'un point de vue de performance, il faut prendre en compte que plus l'entrepôt de données grossit, plus il y aura de traitement à faire. En conséquence, le système sera plus lent.