

Licence CC - Guillaume Paumier

Projet de bases de  
données  
ASI3 2013-2014

Manon ANSART  
Antoine AUGUSTI

# BD UNIVERSITÉ

## RAPPORT DE PROJET - GROUPE GRTT6

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Processus de réalisation . . . . .	2
1.2	Livrables attendus . . . . .	2
<b>2</b>	<b>Réalisation du projet</b>	<b>3</b>
2.1	Gestionnaire de versions . . . . .	3
2.2	Modèles Entité-Association et Relationnel . . . . .	3
2.2.1	Passage du modèle Entité-Association au modèle relationnel . . . . .	3
2.2.2	Critique du modèle Entité-Relationnel . . . . .	5
2.3	Noms des tables et des attributs . . . . .	5
2.3.1	Table piece . . . . .	5
2.3.2	Table personne . . . . .	5
2.3.3	Table tache . . . . .	5
2.3.4	Table appartient . . . . .	5
2.3.5	Table passePar . . . . .	6
2.3.6	Table reservation . . . . .	6
2.4	Choix des clés et des contraintes . . . . .	6
2.4.1	Choix des clés . . . . .	6
2.4.2	Choix des contraintes . . . . .	6
2.5	Droits des utilisateurs . . . . .	7
<b>3</b>	<b>Difficultés rencontrées</b>	<b>9</b>
3.1	Serveur PostgreSQL local . . . . .	9
3.2	Rapport d'intrusion . . . . .	9
3.3	Date de réservation dans l'application . . . . .	12
3.4	Report des exceptions dans l'application . . . . .	13
<b>4</b>	<b>Ce qui aurait pu être fait</b>	<b>14</b>
4.1	Contraintes supplémentaires . . . . .	14
4.1.1	Contraintes sur les dates . . . . .	14
4.1.2	Contraintes sur les réservations . . . . .	14
4.1.3	Utilisation d'incrémentation automatique pour les clés primaires . . . . .	14
<b>5</b>	<b>Répartition du travail et conclusion</b>	<b>15</b>
5.1	Répartition du travail . . . . .	15
5.2	Conclusion . . . . .	15
<b>6</b>	<b>Annexes</b>	<b>17</b>
6.1	Code source SQL . . . . .	17
6.1.1	creation.sql . . . . .	17
6.1.2	suppression.sql . . . . .	25

6.2	Code source C . . . . .	25
6.2.1	appliUniv.pgc . . . . .	25

# Chapitre 1

## Introduction

Ce mini-projet conclut les cours Bases de Données 1 et 2 de la première année du département ASI<sup>1</sup> de l'INSA de Rouen. Il vise à mettre en œuvre les concepts abordés lors de ces cours en réalisant une petite base de données et quelques fonctionnalités sur celle-ci. L'objectif final étant de proposer une application permettant d'interagir avec une base de données correctement modélisée.

### 1.1 Processus de réalisation

Le processus général pour mener la réalisation d'une base de données est ici simplifié et peut être décrit ainsi :

1. Passage d'un schéma Entité-Association à un modèle relationnel ;
2. Définition des domaines, des conditions initiales et des droits ;
3. Définition des index sur la base ;
4. Création de fonctions et des vues sur la base ;
5. Développement d'une application en C afin d'interagir avec la base de données conçue.

### 1.2 Livrables attendus

Les livrables attendus pour ce mini-projet sont les suivants :

- Un rapport d'une dizaine de pages expliquant la démarche suivie et les choix de conception effectués ;
- 2 fichiers SQL :
  - `creation.sql` qui crée la base, les fonctions, les droits, les index et la remplit ;
  - `suppression.sql` qui supprime tout ce qui a été créé.
- Un fichier `appliUniv.pgc` qui contient l'application permettant d'interagir avec la base de données créée. Ce fichier est à compiler.

---

1. ASI : Architecture des Systèmes d'Information.

# Chapitre 2

## Réalisation du projet

### 2.1 Gestionnaire de versions

Pour travailler de manière collaborative efficacement, nous avons choisi d'utiliser le gestionnaire de versions Git et d'héberger notre projet sur GitHub<sup>1</sup>. Nous avons choisi d'utiliser Git et GitHub car ils sont très en vogue dans le monde de l'open source.

Ainsi tout notre travail peut être vu et téléchargé à l'adresse <https://github.com/manonansart/BDUniv>. Il est par exemple possible de regarder nos commits, ceux-ci expliquant pas à pas comment nous avons mené ce projet. Nous avons également rédigé une documentation sommaire indiquant comment créer notre base de données, la supprimer, compiler notre application en C et l'utiliser.

Notre répertoire de travail Git peut par ailleurs être directement cloné sur votre ordinateur à l'aide de la commande : `git clone https://github.com/manonansart/BDUniv.git`.

### 2.2 Modèles Entité-Association et Relationnel

#### 2.2.1 Passage du modèle Entité-Association au modèle relationnel

Nous avions pour consigne d'utiliser un schéma Entité-Association précis et de réaliser le modèle relationnel correspondant.

---

1. GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git.

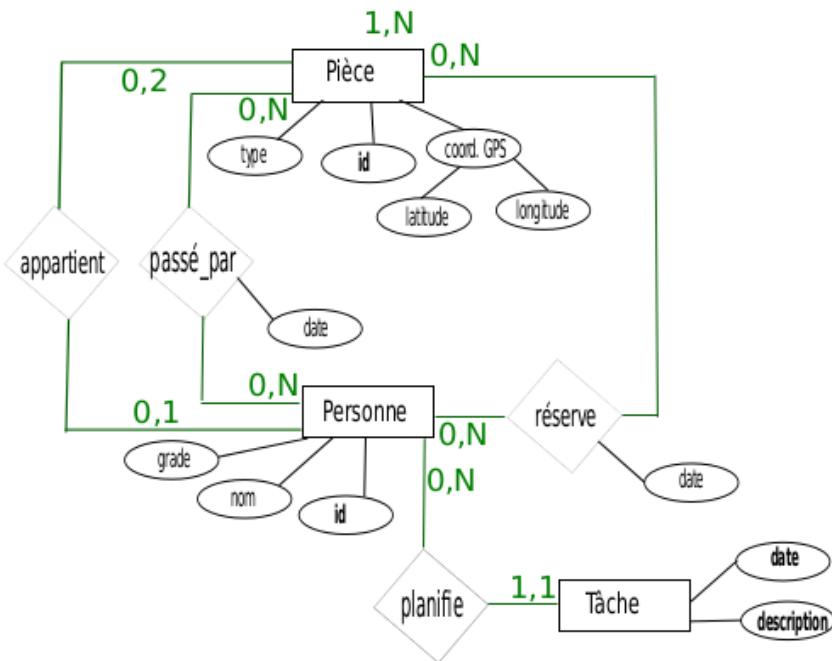


FIGURE 2.1 – Schéma EA

En suivant chaque étape du cours, nous avons obtenu les résultats suivants :

1. Passage des entités en relations :

- **piece**(idPiece, type, latitude, longitude) : l'attribut composite coord\_GPS est remplacé par ces 2 attributs, latitude et longitude.
- **personne**(idPers, nom, grade)
- **tache**(date, tache)

2. Passage des entités faibles en relations : rien à faire puisqu'il n'y a pas d'entités faibles

3. Association binaire 1,1 par clés étrangères :

- **piece**(idPiece, type, latitude, longitude) : schéma inchangé
- **personne**(idPers, nom, grade) : schéma inchangé
- **tache**(date, tache, id) : ajout de la clé étrangère idPers, clé de **personne** car il y a une cardinalité 1,1

4. Associations binaires M,N :

- **piece**(idP, type, latitude, longitude) : schéma inchangé
- **personne**(idPers, nom, grade) : schéma inchangé
- **tache**(date, tache, idPers) : schéma inchangé
- **appartient**(idP, idPers) : création de la relation **appartient** à cette étape. Étant une table de liaison entre **piece** et **personne**, on y met les clés étrangères de ces tables. Ces seules clés permettent de modéliser la possession d'une pièce par une personne donc on n'ajoute aucun attribut.
- **passeparr**(idP, idPers, date) : création de la relation **passeparr** à cette étape. Étant une table de liaison entre **piece** et **personne**, on y met les clés étrangères de ces tables. Nous devons ajouter un attribut **date** car le passage d'une personne dans une pièce doit être daté.

- **reservation(idP, idPers, date)** : création de la relation **reservation** à cette étape. Étant une table de liaison entre **piece** et **personne**, on y met les clés étrangères de ces tables. Nous devons ajouter un attribut **date** car la réservation d'une salle par une personne d'une pièce est pour une date précise.

5. Il n'y a pas d'attributs multi-valués ni d'association n-aire.

## 2.2.2 Critique du modèle Entité-Relationnel

On remarque que le modèle semble bien conçu et n'engendre aucun problème. Par exemple, il n'y a pas de redondance des données, les informations concernant les personnes (nom, grades) ou les pièces (coordonnées, type) ne sont pas répétées pour chaque réservation de salle avec ce modèle et le modèle relationnel qui en découle.

De ce fait, les mises à jour n'induisent pas d'incohérence : si le nom de la personne est modifié, il ne faut pas le changer à différents endroits pour rester cohérent.

On remarque également l'absence d'anomalie de suppression. Une personne peut n'être concernée par aucune association, c'est-à-dire qu'elle ne possède pas de bureau, n'est passée par aucune pièce et n'a rien réservé, on peut tout de même conserver les informations la concernant puisqu'elles sont séparées.

## 2.3 Noms des tables et des attributs

Retrouvez ci-dessous la description des tables utilisées dans ce projet ainsi que leurs attributs.

### 2.3.1 Table piece

- **idP** : l'identifiant de la pièce.
- **type** : le type de la pièce. Les valeurs possibles sont : Bureau, Salle de Cours, Autre.
- **gps\_lat** : latitude de la pièce.
- **gps\_long** : longitude de la pièce.

### 2.3.2 Table personne

- **idPers** : l'identifiant de la personne.
- **nom** : le nom de la personne.
- **grade** : le grade de la personne. Les valeurs possibles sont : Etudiant, MCF, PU, BIATOSS, PE

### 2.3.3 Table tache

- **date** : la date de la tâche.
- **tache** : la description de la tâche planifiée. Les valeurs possibles sont : Enseignement, Recherche, Réunion.
- **idPers** : l'identifiant de la personne.

### 2.3.4 Table appartient

- **idP** : l'identifiant de la pièce.
- **idPers** : l'identifiant de la personne.

### 2.3.5 Table passePar

- **idP** : l'identifiant de la pièce.
- **idPers** : l'identifiant de la personne.
- **date** : la date du passage.

### 2.3.6 Table reservation

- **idP** : l'identifiant de la pièce.
- **idPers** : l'identifiant de la personne.
- **date** : la date de la réservation.

Les types des attributs peuvent être retrouvés dans le fichier de création des tables : `src/SQL/creation.sql`.

## 2.4 Choix des clés et des contraintes

### 2.4.1 Choix des clés

Pour les tables **piece** et **personne** on choisit comme clés primaires **idP** et **idPers** car ils identifient clairement une pièce ou une personne. On choisit ces attributs comme clés primaires et ces attributs seront utilisés comme clés étrangères dans d'autres tables.

Pour les tables **passePar** et **reservation** nous devons définir comme clé primaire les attributs **idP**, **idPers** et **date** car on référence un événement qui se déroule à une date et qui lie une pièce à une personne. Il est nécessaire d'utiliser les 3 attributs comme clés primaires dans ces deux tables pour identifier un tuple. Ces tables sont des tables de liaison donc on utilise les clés primaires des tables que l'on veut lier.

Pour la table **tache** nous devons définir comme clé primaire les attributs **tache**, **idPers** et **date** car on référence une tache planifiée par une personne à une date. L'explication est très similaire au paragraphe précédent. Cette table est une table de liaison donc on utilise les clés primaires des tables que l'on veut lier.

Enfin, pour la table **appartient** on choisit comme clé primaire les attributs **idP** et **idPers** pour permettre de dire qu'une personne est propriétaire d'une pièce. Cette table est une table de liaison donc on utilise les clés primaires des tables que l'on veut lier.

### 2.4.2 Choix des contraintes

#### Clés primaires et étrangères

Parmi les contraintes qu'il nous a semblé primordial de respecter, la première était l'unicité des clés primaires qui devaient également être non nulles. C'est pourquoi pour chaque table nous avons renseigné les clés primaires décrites précédemment par PRIMARY KEY.

Nous avons également veillé au respect des contraintes pour les clés étrangères en utilisant REFERENCES. Ainsi, pour la table **tache**, nous avons indiqué que l'attribut **idPers** était une clé étrangère faisant référence à **idPers**, la clé de **personne**. De même, pour les tables **appartient**, **passePar** et **reservation**, **idP** et **idPers** sont deux clés étrangères correspondant respectivement aux clés primaires de **piece** et **personne**.

## Table des pièces

Nous vérifions ici plusieurs contraintes grâce au script de création de la table.

1. Le type de la pièce ne peut être que : *Bureau*, *Salle De Cours* ou *Autre*.
2. La latitude de la position GPS de la pièce peut être entre 0 et 90 degrés.
3. La longitude de la position GPS de la pièce peut être entre -180 et 180 degrés.

## Table des propriétaires

Concernant la table appartient, seules les personnes dont le grade est MCF, PU ou BIATOSS peuvent posséder une pièce et la pièce possédée doit être de type *Bureau*.

Afin de vérifier que ces conditions sont bien respectées nous avons créé le trigger `triggerAppartient` qui appelle la fonction `testAppartient()`. Cette fonction vérifie que la personne n'est pas de type *Etudiant* ou *PE* et que la pièce est bien de type *Bureau*. Nous vérifions également dans cette fonction que la personne n'est pas déjà propriétaire de la pièce et qu'on ne tente pas de créer un troisième propriétaire. Ces dernières contraintes nous sont imposées par le modèle Entité-Association.

## Table des réservations

La table réserve nécessite plusieurs contraintes : seules les personnes dont le grade est MCF, PU ou BIATOSS peuvent réserver une salle, il faut que la salle réservée ait un type cohérent avec le grade de la personne et la tâche, si celle-ci existe, pour laquelle elle est réservée. De plus la salle ne doit pas faire l'objet d'une autre réservation pour la même date. Nous vérifions chacune de ces contraintes à l'aide du trigger `triggerReservation`, correspondant à la fonction `testReservation()`.

Dans un premier temps, cette fonction vérifie que la personne pour laquelle on veut faire la réservation est bien de grade MCF, PU ou BIATOSS à l'aide de plusieurs si. Puis la fonction vérifie qu'on ne tente pas d'effectuer une réservation dans le passé. Elle compte ensuite le nombre de réservation pour cette salle à cette date afin de vérifier que ce nombre est nul et qu'on peut donc bien réserver la salle.

Dans une dernière partie, la fonction regarde si la personne a déjà planifié une tâche à cette date et si c'est le cas, que le type de la salle est bien cohérent avec celui de la tâche.

## Table des tâches

Enfin, un dernier trigger, `triggerTache`, associé à la fonction `testTache()`, vérifie les contraintes concernant la table tache, c'est-à-dire que la personne n'est pas de type PE et que la tâche planifiée est bien cohérente avec le grade de la personne. Nous vérifions que la tâche est : *Enseignement*, *Recherche* ou *Réunion*.

## 2.5 Droits des utilisateurs

Comme demandé dans les consignes, nous avons donné les droits de sélection et de mise à jour (uniquement UPDATE) pour toutes les tables et vues du projet à l'utilisateur `grtt42`. Nous avons également donné les droits de sélection sur nos tables et vues à l'utilisateur `grtt11`. Ces utilisateurs ne sont pas créés car on suppose qu'ils existent déjà comme notre code a vocation à être exécuté sur les bases de données de l'INSA de Rouen.

Le code SQL pour donner ces droits est basique et peut être retrouvé ci-dessous :

```
1 -- Les droits
2 GRANT SELECT, UPDATE ON tache TO grtt42;
3 GRANT SELECT, UPDATE ON passepar TO grtt42;
4 GRANT SELECT, UPDATE ON piece TO grtt42;
5 GRANT SELECT, UPDATE ON personne TO grtt42;
6 GRANT SELECT, UPDATE ON rapport_activite TO grtt42;
7 GRANT SELECT, UPDATE ON intrusion TO grtt42;
8 GRANT SELECT, UPDATE ON appartient TO grtt42;
9 GRANT SELECT, UPDATE ON reservation TO grtt42;
10 GRANT SELECT ON tache TO grtt11;
11 GRANT SELECT ON passepar TO grtt11;
12 GRANT SELECT ON piece TO grtt11;
13 GRANT SELECT ON personne TO grtt11;
14 GRANT SELECT ON rapport_activite TO grtt11;
15 GRANT SELECT ON intrusion TO grtt11;
16 GRANT SELECT ON appartient TO grtt11;
17 GRANT SELECT ON reservation TO grtt11;
```

# Chapitre 3

## Difficultés rencontrées

### 3.1 Serveur PostgreSQL local

Nous avons dû installer sur nos ordinateurs personnels PostgreSQL en client et en serveur pour pouvoir développer rapidement. En effet, nous ne voulions pas à avoir à monter des tunnels SSH vers l'INSA de Rouen pour travailler sur le serveur PostgreSQL de l'INSA.

L'installation de PostgreSQL sous Ubuntu est facile mais la configuration est un petit peu plus délicate. Nous avons utilisé la documentation en ligne de PostgreSQL et quelques forums trouvés grâce à des moteurs de recherche. Nous avons mis un petit peu de temps à changer le mot de passe de l'utilisateur par défaut, créer notre utilisateur `grtt6`, créer une base du même nom et définir les droits adéquats dessus. Mais maintenant nous maîtrisons parfaitement la procédure ! Une fois effectué nous n'avons plus eu aucun problème et nous avons pu écrire nos scripts SQL et les tester sans problème.

### 3.2 Rapport d'intrusion

La fonctionnalité nous ayant posé le plus de difficultés a été le rapport d'intrusion. En effet, après avoir écrit une première version de la fonction `estIntru` qui nous paraissait cohérente, nous avons constaté que les résultats obtenus ne correspondaient pas à ceux de l'exemple d'utilisation.

```

1 CREATE OR REPLACE FUNCTION estIntru(dateIntru date, pieceIntru VARCHAR(10), idIntru VARCHAR(10))
2 RETURNS Integer AS $$
```

3

```

4 DECLARE
5     gradeIntru VARCHAR;
6     proprio VARCHAR;
7     passageProprio Integer;
```

8

```

9
10 BEGIN
11
12     SELECT p.grade INTO gradeIntru
13     FROM personne p
14     WHERE p.idPers = idIntru;
```

15

```

16
17
18     SELECT a.idPers INTO proprio
19     FROM appartient a
```

```

20 WHERE a.idP = pieceIntru;
21
22
23 IF (gradeIntru = 'PE') THEN
24     RETURN 0;
25 ELSE
26     SELECT count(p.idP) INTO passageProprio
27     FROM passePar p
28     WHERE p.date = dateIntru
29         AND p.idP = pieceIntru
30         AND p.idPers = proprio;
31     IF (passageProprio > 0) THEN
32         RETURN 0;
33     END IF;
34 END IF;
35
36 RETURN 1;
37 END $$ LANGUAGE 'plpgsql';

```

Les résultats que nous étions censés obtenir avec la première version du fichier d'exécution étaient :

2014-01-20	s3	Amanda Maire
2014-01-20	s3	Mousse Line
2014-01-20	s2	Mousse Line
2014-01-13	s1	Louis Dort
2014-05-10	s7	Geo Graff
2014-01-22	s1	Albert Gamotte
2014-01-20	s3	Amanda Maire
2014-01-20	s3	Mousse Line

Mais nous obtenions les résultats suivant :

2014-01-13	s1	Laure Aidubois
2014-01-22	s1	Albert Gamotte
2014-05-10	s7	Geo Graff
2014-01-13	s1	Louis Dort
2014-01-20	s3	Mousse Line
2014-01-20	s3	Amanda Maire

Le rapport d'intrusion résultant d'une fonction assez compliquée, il nous était très difficile de trouver la source de ces différences et d'expliquer nos erreurs en regardant simplement les tables. Nous avons donc été amenés à écrire une requête SQL nous permettant d'avoir une vue d'ensemble et de mieux voir quels tuples nous manquaient, quels tuples nous avions en trop et pourquoi.

Nous avons donc utilisé la requête :

```

1 SELECT p.nom, p.idPers, p.grade, pa.date, pa.idP, a.idPers Proprietaire
2 FROM personne p, passepar pa, appartient a
3 WHERE p.idPers = pa.idPers AND a.idP = pa.idP;

```

qui nous a affiché les résultats suivants :

nom	idpers	grade	date	idp	proprietaire
Lance Pierre	p1	MCF	2014-01-12	s1	p1
Lance Pierre	p1	MCF	2014-01-12	s1	p2
Laure Aidubois	p2	MCF	2014-01-13	s1	p1
Laure Aidubois	p2	MCF	2014-01-13	s1	p2
Louis Dort	p7	Etudiant	2014-01-13	s1	p1
Louis Dort	p7	Etudiant	2014-01-13	s1	p2
Bruno Zieuvar	p5	PE	2014-01-14	s1	p1
Bruno Zieuvar	p5	PE	2014-01-14	s1	p2
Mousse Line	p9	Etudiant	2014-01-20	s2	p3
Albert Gamotte	p3	PU	2014-01-20	s2	p3
Amanda Maire	p10	Etudiant	2014-01-20	s3	p4
Mousse Line	p9	Etudiant	2014-01-20	s3	p4
Albert Gamotte	p3	PU	2014-01-22	s1	p1
Albert Gamotte	p3	PU	2014-01-22	s1	p2

Ainsi nous avons pu voir que *2014-01-13 / s1 / Laure Aidubois*, considérée comme une intruse par notre fonction, était la deuxième propriétaire de la salle s1 dans laquelle elle se trouve et ne devait donc pas être considérée comme intruse. De ce fait *2014-01-13 / s1 / Louis Dort*, présent au même moment, ne devait pas être considéré comme un intru non plus.

Nous en avons conclu que notre fonction ne gérait pas les propriétaires multiples pour un même bureau.

De même, nous avons constaté que le tuple *2014-05-10 / s7 / Geo Graff*, présent dans notre rapport d'intrusion, n'apparaissait même pas dans les résultats que nous obtenions, et ceci pour une raison simple : la salle s7 étant une salle de cours, elle ne possède pas de propriétaire.

Nous avons donc pu voir que notre fonction ne gérait pas les pièces autres que les bureaux, qui ne possèdent pas de propriétaires et qui ne devaient donc pas être prises en compte dans le rapport d'intrusion.

À l'aide de cette requête nous avons donc pu repérer nos erreurs et effectuer les modifications nécessaires pour obtenir des résultats cohérents. Nous avons ainsi pu corriger notre fonction `estIntru` dont le code est le suivant :

```

1 CREATE OR REPLACE FUNCTION estIntru(dateIntru date, pieceIntru VARCHAR(10), idIntru VARCHAR(10))
2 RETURNS Integer AS $$
3 DECLARE
4     gradeIntru VARCHAR;
5     proprio VARCHAR;
6     typePiece VARCHAR;
7     passageProprio Integer;
8
9 BEGIN
10    SELECT p.grade INTO gradeIntru
11    FROM personne p
12    WHERE p.idPers = idIntru;
13
14    SELECT pi.type INTO typePiece
15    FROM piece pi
16    WHERE pi.idP = pieceIntru;
17
18
19    IF (gradeIntru = 'PE' OR typePiece != 'Bureau') THEN

```

```

20      RETURN 0;
21  ELSE
22      SELECT count(p.idP) INTO passageProprio
23      FROM passePar p
24      WHERE p.date = dateIntru
25      AND p.idP = pieceIntru
26      AND p.idPers IN
27          (SELECT a.idPers
28          FROM appartient a
29          WHERE a.idP = pieceIntru);
30      IF (passageProprio > 0) THEN
31          RETURN 0;
32      END IF;
33  END IF;
34
35  RETURN 1;
36
37 END $$ LANGUAGE 'plpgsql',

```

### 3.3 Date de réservation dans l'application

Dans l'application nous devions être en mesure de pouvoir créer une réservation d'une salle par une personne pour une date. Nous avons rencontré des problèmes car la fonction PGTYPESdate\_from\_asc ne comprend pas toujours les dates rentrées au format français classique (c'est-à-dire JJ/MM/AAAA).

Pour être certains que la date soit bien interprétée par la fonction, nous avons choisi de demander la date au format *JJ/MM/AAAA* à l'utilisateur que nous transformons ensuite au format *AAAA-MM-JJ* qui est un format accepté comme indiqué dans la documentation <http://www.postgresql.org/docs/9.3/static/ecpg-pgtypes.html>, partie 33.6.2.

Par ailleurs nous avons vérifié que la date entrée par l'utilisateur était bien d'un format valide : le jour existe bien dans le mois demandé et dans l'année demandée. En cas de problème, nous demandons une nouvelle saisie de la date. Nous n'avons pas eu de problème particulier pour cette validation qui est un algorithme classique.

Au final le code écrit est le suivant :

```

1 // On indique que la date n'est pas valide initialement
2 int dateValide = 0;
3 do {
4     printf("Jour (JJ) :\n");
5     scanf("%s", jourChaine);
6     printf("Mois (MM) :\n");
7     scanf("%s", moisChaine);
8     printf("Année (AAAA) :\n");
9     scanf("%s", anneeChaine);
10
11    // Conversion en entiers pour vérifier la date
12    int jour, mois, annee;
13    jour = atoi(jourChaine);
14    mois = atoi(moisChaine);
15    annee = atoi(anneeChaine);

```

```

16
17 // Les différents jours max dans les mois
18 int joursDansLesMois[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
19
20 // Année bissextile ?
21 if (annee % 400 == 0 || (annee % 100 != 0 && annee % 4 == 0))
22     joursDansLesMois[1] = 29;
23
24 // Le jour correspond bien au mois ?
25 if (mois <= 12 && mois >= 1) {
26     if (jour <= joursDansLesMois[mois-1])
27         dateValide = 1;
28 }
29
30 if (dateValide == 0)
31     printf("Merci d'entrer une date valide.\n");
32 } while(dateValide == 0);
33
34 // Si l'utilisateur a rentré une date sur 1 seul caractère on est gentil avec lui
35 if (strlen(moisChaine) == 1) {
36     moisChaine[1] = moisChaine[0];
37     moisChaine[0] = '0';
38 }
39
40 // Si l'utilisateur a rentré une date sur 1 seul caractère on est gentil avec lui
41 if (strlen(jourChaine) == 1) {
42     jourChaine[1] = jourChaine[0];
43     jourChaine[0] = '0';
44 }
45
46 // Passage au format AAAA-MM-DD pour PostgreSQL
47 strcpy(dateChaine, anneeChaine);
48 strcat(dateChaine, "-");
49 strcat(dateChaine, moisChaine);
50 strcat(dateChaine, "-");
51 strcat(dateChaine, jourChaine);
52
53 dateSQL = PGTYPEsdate_from_asc(dateChaine, NULL);

```

## 3.4 Report des exceptions dans l'application

Nous avons mis en place plusieurs triggers de vérification de contraintes d'intégrité complexes lors de la création de notre base de données. Ces triggers lèvent des exceptions lorsque l'on tente de faire une mise à jour qui viole les contraintes d'intégrité. Lorsque l'on tente de faire une mise à jour qui viole une contrainte d'intégrité en ligne de commande dans le terminal, l'exception levée par PostgreSQL est affichée directement dans le terminal.

Nous voulions pouvoir reproduire ce comportement pour les mises à jour ne respectant pas les contraintes d'intégrité effectuées depuis notre application. Après une longue recherche, nous avons réalisé que la directive `EXEC SQL WHENEVER sqlerror sqlprint` ne s'appliquait qu'à la fonction où elle se trouvait. Nous avons donc déplacé cette directive au début du programme, en dehors d'une fonction pour avoir une portée globale.

# Chapitre 4

## Ce qui aurait pu être fait

### 4.1 Constraintes supplémentaires

#### 4.1.1 Constraintes sur les dates

Il aurait été bien de pouvoir planifier des tâches ou des réservations de manière plus précise que pour une journée. En effet il est assez irréaliste qu'une personne ne fasse qu'une tâche pendant toute une journée. Il aurait été bien de pouvoir planifier des tâches ou des réservations pour une intervalle de temps, avec une date de début et une date de fin.

#### 4.1.2 Constraintes sur les réservations

Actuellement il est possible de réserver une salle même si on n'a pas défini de tâche, ce qui n'est pas optimal pour le suivi du planning des personnes de l'université.

#### 4.1.3 Utilisation d'incrémentation automatique pour les clés primaires

Actuellement les pièces sont identifiés par une clé primaire  $s_i$  et les personnes par  $p_i$  où  $i$  est un entier naturel non nul. Il aurait peut-être été préférable d'utiliser une incrémentation automatique pour les clés primaires de ces deux tables pour ne pas devoir renseigner la valeur de la clé primaire lors d'insertions dans celles-ci.

# Chapitre 5

## Répartition du travail et conclusion

### 5.1 Répartition du travail

Retrouvez dans le tableau ci-dessous comment nous avons réparti entre nous les différentes fonctionnalités à réaliser pour mener ce projet :

Tâche	Manon Ansart	Antoine Augusti
<b>Conception de la base de données</b>		
Passage au modèle relationnel	✓	✓
Création des tables		✓
Insertion des données	✓	
Création des droits	✓	
Création des index		✓
Création des fonctions	✓	
Création des vues		✓
Création des triggers		✓
Script de suppression	✓	
<b>Application de gestion</b>		
Ajout d'une personne	✓	
Suppression d'une personne	✓	
Réservation d'une salle		✓
Lieux visités par une personne	✓	
Rapport d'activité d'une personne		✓
Rapport d'intrusion		✓
<b>Rapport</b>		
Rédaction du rapport	✓	✓
<b>Gestionnaire de versions</b>		
Mise en place de Git		✓

Bien évidemment ce tableau de répartition des tâches peut être vérifié à l'aide de nos commits sur notre répertoire de travail Git, se trouvant sur GitHub.

### 5.2 Conclusion

Nous avons été heureux de réaliser ce mini-projet de bases de données tout d'abord car il a très bien repris toutes les notions que nous avons pu aborder cette année dans les cours de bases de données. Bien que la base de données à concevoir n'était pas très conséquente, nous avons pu revoir une

grande partie des concepts que nous avons appris cette année.

De plus, nous avons été heureux de nous replonger dans nos précédents cours et travaux pratiques afin de savoir comment appliquer précisément la notion dont nous avions besoin au fur et à mesure de l'avancement de notre projet. Nous avons dû également nous référer plusieurs fois à la documentation en ligne de PostgreSQL pour accomplir des tâches que nous n'avions pas pu aborder en cours. Ce travail de recherche (qui mène à un succès !) a été stimulant.

Pour conclure, ce projet a été pour nous l'occasion d'appliquer toutes les notions apprises lors de séances de travaux pratiques. Nous sommes fiers et satisfaits au terme de notre projet de pouvoir interagir à l'aide de notre application avec la base de données que nous avons conçu. Toutefois nous n'oublions pas que ceci est un bien maigre résultat par rapport à ce que doit être une interaction efficace et agréable pour un utilisateur entre une base de données et des applications. Nous attendons avec impatience de pouvoir faire mieux !

# Chapitre 6

## Annexes

### 6.1 Code source SQL

#### 6.1.1 creation.sql

```

1 CREATE TABLE piece(
2     idP VARCHAR(10) PRIMARY KEY NOT NULL,
3     type VARCHAR(15),
4     gps_lat REAL,
5     gps_long REAL,
6     CHECK (type IN ('Bureau', 'Salle de Cours', 'Autre')),
7     CHECK (gps_lat >= 0 AND gps_lat <= 90),
8     CHECK (gps_long >= -180 AND gps_long <= 180)
9 );
10
11 CREATE TABLE personne(
12     idPers VARCHAR(10) PRIMARY KEY NOT NULL,
13     nom VARCHAR(20),
14     grade VARCHAR(10),
15     CHECK (grade IN ('Etudiant', 'MCF', 'PU', 'BIATOSS', 'PE'))
16 );
17
18 CREATE TABLE tache(
19     date date,
20     tache VARCHAR(15),
21     idPers VARCHAR(10) REFERENCES personne(idPers) ON DELETE CASCADE,
22     PRIMARY KEY (date, tache, idPers),
23     CHECK (tache IN ('Enseignement', 'Recherche', 'Réunion'))
24 );
25
26 CREATE TABLE appartient(
27     idP VARCHAR(10) REFERENCES piece(idP) ON DELETE CASCADE,
28     idPers VARCHAR(10) REFERENCES personne(idPers) ON DELETE CASCADE,
29     PRIMARY KEY(idP, idPers)
30 );
31
32 CREATE TABLE passePar(
33     idP VARCHAR(10) REFERENCES piece(idP) ON DELETE CASCADE,
34     idPers VARCHAR(10) REFERENCES personne(idPers) ON DELETE CASCADE,
35     date date,

```

```

36     PRIMARY KEY(idP, idPers, date)
37 );
38
39 CREATE TABLE reservation(
40     idP VARCHAR(10) references piece(idP) ON DELETE CASCADE,
41     idPers VARCHAR(10) references personne(idPers) ON DELETE CASCADE,
42     date date,
43     PRIMARY KEY(idP, idPers, date)
44 );
45
46 -- Les données
47 INSERT INTO personne VALUES ('p1', 'Lance Pierre', 'MCF');
48 INSERT INTO personne VALUES ('p2', 'Laure Aidubois', 'MCF');
49 INSERT INTO personne VALUES ('p3', 'Albert Gamotte', 'PU');
50 INSERT INTO personne VALUES ('p4', 'Tina Pachange', 'BIATOSS');
51 INSERT INTO personne VALUES ('p5', 'Bruno Zieuvaire', 'PE');
52 INSERT INTO personne VALUES ('p6', 'Geo Graff', 'Etudiant');
53 INSERT INTO personne VALUES ('p7', 'Louis Dort', 'Etudiant');
54 INSERT INTO personne VALUES ('p8', 'Marc Assin', 'Etudiant');
55 INSERT INTO personne VALUES ('p9', 'Mousse Line', 'Etudiant');
56 INSERT INTO personne VALUES ('p10', 'Amanda Maire', 'Etudiant');
57
58 INSERT INTO piece VALUES ('s1', 'Bureau', 49.38, 1.06);
59 INSERT INTO piece VALUES ('s2', 'Bureau', 49.38, 1.06);
60 INSERT INTO piece VALUES ('s3', 'Bureau', 49.38, 1.06);
61 INSERT INTO piece VALUES ('s4', 'Autre', 49.45, 1.06);
62 INSERT INTO piece VALUES ('s5', 'Autre', 49.38, 1.06);
63 INSERT INTO piece VALUES ('s6', 'Salle de Cours', 49.38, 1.06);
64 INSERT INTO piece VALUES ('s7', 'Salle de Cours', 49.45, 1.06);
65
66 INSERT INTO appartient (idPers, idP) VALUES('p1', 's1');
67 INSERT INTO appartient (idPers, idP) VALUES('p2', 's1');
68 INSERT INTO appartient (idPers, idP) VALUES('p3', 's2');
69 INSERT INTO appartient (idPers, idP) VALUES('p4', 's3');
70
71 INSERT INTO tache (date, idPers, tache) VALUES ('01/02/2014', 'p1', 'Enseignement');
72 INSERT INTO tache (date, idPers, tache) VALUES ('02/02/2014', 'p1', 'Enseignement');
73 INSERT INTO tache (date, idPers, tache) VALUES ('12/03/2014', 'p1', 'Recherche');
74 INSERT INTO tache (date, idPers, tache) VALUES ('01/02/2014', 'p2', 'Recherche');
75 INSERT INTO tache (date, idPers, tache) VALUES ('05/02/2014', 'p2', 'Enseignement');
76 INSERT INTO tache (date, idPers, tache) VALUES ('10/01/2014', 'p3', 'Recherche');
77 INSERT INTO tache (date, idPers, tache) VALUES ('11/01/2014', 'p3', 'Recherche');
78 INSERT INTO tache (date, idPers, tache) VALUES ('12/01/2014', 'p3', 'Recherche');
79 INSERT INTO tache (date, idPers, tache) VALUES ('13/01/2014', 'p3', 'Recherche');
80 INSERT INTO tache (date, idPers, tache) VALUES ('15/03/2014', 'p4', 'Réunion');
81 INSERT INTO tache (date, idPers, tache) VALUES ('12/03/2014', 'p4', 'Réunion');
82 INSERT INTO tache (date, idPers, tache) VALUES ('10/02/2014', 'p6', 'Enseignement');
83 INSERT INTO tache (date, idPers, tache) VALUES ('11/02/2014', 'p6', 'Enseignement');
84 INSERT INTO tache (date, idPers, tache) VALUES ('12/02/2014', 'p6', 'Enseignement');
85 INSERT INTO tache (date, idPers, tache) VALUES ('10/02/2014', 'p7', 'Enseignement');
86 INSERT INTO tache (date, idPers, tache) VALUES ('11/02/2014', 'p7', 'Enseignement');
87 INSERT INTO tache (date, idPers, tache) VALUES ('12/02/2014', 'p7', 'Enseignement');
88

```

```

89 INSERT INTO reservation(date, idPers, idP) VALUES('12/03/2014', 'p1', 's4');
90 INSERT INTO reservation(date, idPers, idP) VALUES('13/03/2014', 'p2', 's6');
91 INSERT INTO reservation(date, idPers, idP) VALUES('05/02/2014', 'p2', 's6');
92 INSERT INTO reservation(date, idPers, idP) VALUES('01/02/2014', 'p2', 's5');
93 INSERT INTO reservation(date, idPers, idP) VALUES('12/01/2014', 'p3', 's6');
94 INSERT INTO reservation(date, idPers, idP) VALUES('11/01/2014', 'p3', 's5');
95 INSERT INTO reservation(date, idPers, idP) VALUES('15/03/2014', 'p4', 's7');

96
97 INSERT INTO passePar(idPers, idP, date) VALUES('p1', 's1', '12/01/2014');
98 INSERT INTO passePar(idPers, idP, date) VALUES('p2', 's1', '13/01/2014');
99 INSERT INTO passePar(idPers, idP, date) VALUES('p7', 's1', '13/01/2014');
100 INSERT INTO passePar(idPers, idP, date) VALUES('p5', 's1', '14/01/2014');
101 INSERT INTO passePar(idPers, idP, date) VALUES('p9', 's2', '20/01/2014');
102 INSERT INTO passePar(idPers, idP, date) VALUES('p3', 's2', '20/01/2014');
103 INSERT INTO passePar(idPers, idP, date) VALUES('p10', 's3', '20/01/2014');
104 INSERT INTO passePar(idPers, idP, date) VALUES('p9', 's3', '20/01/2014');
105 INSERT INTO passePar(idPers, idP, date) VALUES('p3', 's1', '22/01/2014');
106 INSERT INTO passePar(idPers, idP, date) VALUES('p6', 's7', '10/05/2014');

107
108 -- Les index
109 CREATE INDEX personne_nom ON personne (nom varchar_pattern_ops);
110 CREATE INDEX reservation_date ON reservation (date);

111
112 -- Les fonctions
113 CREATE OR REPLACE FUNCTION tacheCoherente(dateTache date, pers VARCHAR(10)) RETURNS VARCHAR AS $$
114 DECLARE
115     nbReserv Integer;
116     salleTache VARCHAR;
117     typeTache VARCHAR;
118
119 BEGIN
120     SELECT COUNT(r.idP) INTO nbReserv
121     FROM reservation r
122     WHERE r.idPers = pers AND r.date = dateTache;
123
124     IF (nbReserv = 0) THEN
125         RETURN 'vrai';
126     ELSE
127         SELECT p.type INTO salleTache
128         FROM reservation r, piece p
129         WHERE r.idPers = pers
130             AND r.date = dateTache
131             AND r.idP = p.idP;
132
133         SELECT t.tache INTO typeTache
134         FROM tache t
135         WHERE t.idPers = pers AND t.date = dateTache;
136
137         IF (salleTache = 'Bureau' AND (typeTache = 'Recherche' OR typeTache = 'Réunion')) THEN
138             RETURN 'vrai';
139         END IF;
140
141         IF (salleTache = 'Salle de Cours' AND typeTache = 'Enseignement') THEN

```

```

142         RETURN 'vrai';
143     END IF;

144
145     IF (salleTache = 'Autre' AND typeTache = 'Réunion') THEN
146         RETURN 'vrai';
147     END IF;

148
149     RETURN 'faux';
150 END IF;

151
152 END $$ LANGUAGE 'plpgsql';

153
154 CREATE OR REPLACE FUNCTION estIntru(dateIntru date, pieceIntru VARCHAR(10), idIntru VARCHAR(10) ) R
155 DECLARE
156     gradeIntru VARCHAR;
157     proprio VARCHAR;
158     typePiece VARCHAR;
159     passageProprio Integer;

160
161 BEGIN
162     SELECT p.grade INTO gradeIntru
163     FROM personne p
164     WHERE p.idPers = idIntru;

165
166     SELECT pi.type INTO typePiece
167     FROM piece pi
168     WHERE pi.idP = pieceIntru;

169
170     IF (gradeIntru = 'PE' OR typePiece != 'Bureau') THEN
171         RETURN 0;
172     ELSE
173         SELECT count(p.idP) INTO passageProprio
174         FROM passePar p
175         WHERE p.date = dateIntru
176             AND p.idP = pieceIntru
177             AND p.idPers IN
178                 (SELECT a.idPers
179                  FROM appartient a
180                  WHERE a.idP = pieceIntru);
181         IF (passageProprio > 0) THEN
182             RETURN 0;
183         END IF;
184     END IF;

185
186     RETURN 1;
187
188 END $$ LANGUAGE 'plpgsql';

189
190 -- Vérifications pour les propriétaires de pièce
191 -- - La personne ne peut être un PE ou un Etudiant
192 -- - La personne n'est pas déjà propriétaire d'une autre pièce
193 -- - La pièce est bien un bureau
194 -- - La pièce ne compte pas plus de 2 propriétaires

```

```

195 CREATE OR REPLACE FUNCTION testAppartient() RETURNS trigger AS $$  

196 DECLARE  

197     gradePer VARCHAR;  

198     typePiece VARCHAR;  

199     piecePossedee VARCHAR;  

200     possedeDejaPiece Integer;  

201     nbProprietaires Integer;  

202  

203 BEGIN  

204     SELECT p.grade INTO gradePer  

205     FROM personne p  

206     WHERE p.idPers = NEW.idPers;  

207  

208     SELECT type INTO typePiece  

209     FROM piece  

210     WHERE idP = NEW.idP;  

211  

212     SELECT COUNT(idP) INTO possedeDejaPiece  

213     FROM appartient  

214     WHERE idPers = NEW.idPers;  

215  

216     SELECT COUNT(idP) INTO nbProprietaires  

217     FROM appartient  

218     WHERE idP = NEW.idP;  

219  

220     -- La personne ne peut être un PE ou un Etudiant  

221     IF (gradePer = 'PE' OR gradePer = 'Etudiant') THEN  

222         RAISE EXCEPTION '% ne peut être un propriétaire car c''est un %.', NEW.idPers, gradePer;  

223     END IF;  

224  

225     -- La personne n'est pas déjà propriétaire d'une autre pièce  

226     IF (possedeDejaPiece = 1) THEN  

227         SELECT idP INTO piecePossedee  

228         FROM appartient  

229         WHERE idPers = NEW.idPers;  

230  

231         RAISE EXCEPTION 'La personne % est déjà propriétaire de la pièce % et ne peut donc pas être %.',  

232     END IF;  

233  

234     -- La pièce est bien un bureau  

235     IF (typePiece <> 'Bureau') THEN  

236         RAISE EXCEPTION 'La pièce % n''est pas un bureau et ne peut donc appartenir à personne.', N  

237     END IF;  

238  

239     -- La pièce ne compte pas plus de 2 propriétaires  

240     IF (nbProprietaires >= 2) THEN  

241         RAISE EXCEPTION 'La pièce % possède déjà 2 propriétaires et ne peut donc en avoir un troisième.', N  

242     END IF;  

243  

244     RETURN NEW;  

245 END $$ LANGUAGE 'plpgsql';  

246  

247 -- Vérifications pour une réservation

```

```

248 -- - La personne doit être un MCF, un PU ou un BIATOSS
249 -- - La salle ne doit pas déjà être réservée
250 -- - Les BIATOSS ne peuvent réserver que des salles de type 'Autre'
251 -- - La personne doit réserver une salle cohérente avec la tâche annoncée
252 CREATE OR REPLACE FUNCTION testReservation() RETURNS trigger AS $$
```

**DECLARE**

```

254     nbReserv Integer;
255     salleTache VARCHAR;
256     typeTache VARCHAR;
257     gradePer VARCHAR;
258     nbTaches Integer;
```

**BEGIN**

```

261     -- Vérification du grade
262     SELECT p.grade INTO gradePer
263     FROM personne p
264     WHERE p.idPers = NEW.idPers;
```

**IF** (gradePer <> 'BIATOSS' AND gradePer <> 'MCF' AND gradePer <> 'PU') **THEN**

```

267         RAISE EXCEPTION '% ne peut réserver une salle car c''est un %', NEW.idPers, gradePer;
```

**END IF;**

**-- Pas de réservation dans le passé**

```

271     IF (NEW.date < current_date) THEN
272         RAISE EXCEPTION 'Vous ne pouvez pas réserver une salle pour une date passée.';
```

**END IF;**

```

275     SELECT COUNT(r.idP) INTO nbReserv
276     FROM reservation r
277     WHERE r.idP = NEW.idP AND r.date = NEW.date;
```

**-- Erreur si la salle a déjà été réservée**

```

281     IF (nbReserv <> 0) THEN
282         RAISE EXCEPTION 'La salle % ne peut être réservée pour le % car elle a déjà été réservée.',
```

**ELSE**

```

283         SELECT p.type INTO salleTache
284         FROM piece p
285         WHERE p.idP = NEW.idP;
```

```

287         SELECT COUNT(t.tache) INTO nbTaches
288         FROM tache t
289         WHERE t.idPers = NEW.idPers AND t.date = NEW.date;
```

```

291         SELECT t.tache INTO typeTache
292         FROM tache t
293         WHERE t.idPers = NEW.idPers AND t.date = NEW.date;
```

**-- Si aucune tâche n'a été réservée c'est ok**

```

296     IF (nbTaches = 0) THEN
297         RETURN NEW;
```

**END IF;**

**-- Les BIATOSS ne peuvent réserver que des salles de type 'Autre'**

```

301 IF (gradePer = 'BIATOSS' AND salleTache <> 'Autre') THEN
302   RAISE EXCEPTION 'Les BIATOSS ne peuvent réserver que des salles de type Autre';
303 END IF;

304

305 -- On vérifie que la salle réservée est cohérente avec la tâche
306 IF (salleTache = 'Bureau' AND (typeTache = 'Recherche' OR typeTache = 'Réunion')) THEN
307   RETURN NEW;
308 END IF;

309

310 IF (salleTache = 'Salle de Cours' AND typeTache = 'Enseignement') THEN
311   RETURN NEW;
312 END IF;

313

314 IF (salleTache = 'Autre' AND typeTache = 'Réunion') THEN
315   RETURN NEW;
316 END IF;

317

318 RAISE EXCEPTION 'La salle % (réservée à %) ne peut être réservée pour le % par % car elle n
319 END IF;

320

321 END $$ LANGUAGE 'plpgsql';

322

323 CREATE OR REPLACE FUNCTION testTache()
324 RETURNS trigger AS $$
325 DECLARE
326   gradePersonne VARCHAR;
327   typePiece VARCHAR;
328 BEGIN

329

330   SELECT p.grade INTO gradePersonne
331   FROM personne p
332   WHERE NEW.idPers = p.idPers;

333

334   SELECT p.type INTO typePiece
335   FROM piece p, reservation r
336   WHERE p.idP = r.idP
337   AND NEW.date = r.date;

338

339 -- Vérification du grade
340 IF (gradePersonne = 'PE') THEN
341   RAISE EXCEPTION 'Les personnels d''entretien ne sont pas concernés par cette gestion de pla
342 END IF;

343

344 IF (gradePersonne = 'Etudiant' AND typeDeLaTache <> 'Enseignement') THEN
345   RAISE EXCEPTION 'Un BIATOSS ne peut pas effectuer une tâche de type %.', typeDeLaTache;
346 END IF;

347

348 IF (gradePersonne = 'BIATOSS' AND typeDeLaTache <> 'Réunion') THEN
349   RAISE EXCEPTION 'Un BIATOSS ne peut pas effectuer une tâche de type %.', typeDeLaTache;
350 END IF;

351

352 -- Vérification entre pièce et tâche
353 IF (typePiece = 'Bureau' AND NEW.tache = 'Enseignement') THEN

```

```

354      RAISE EXCEPTION 'Un bureau ne peut pas être réservé pour faire de l''enseignement.';
355  END IF;

356
357  IF (typePiece = 'Salle de Cours' AND NEW.tache <> 'Enseignement') THEN
358      RAISE EXCEPTION 'Une salle de cours ne peut être réservée que pour faire de l''enseignement';
359  END IF;

360
361  IF (typePiece = 'Autre' AND NEW.tache <> 'Réunion') THEN
362      RAISE EXCEPTION 'Une salle de type autre ne peut être réservée que pour faire une réunion.';
363  END IF;

364
365 RETURN NEW;
366 END $$ LANGUAGE 'plpgsql';

367
368 -- Les vues
369 CREATE VIEW rapport_activite AS
370     SELECT t.date date, p.nom nom, tacheCoherente(t.date, p.idPers) ok
371     FROM tache t, personne p
372     WHERE t.idPers = p.idPers;

373
374 CREATE VIEW intrusion AS
375     SELECT p.date date, p.idP salle, pe.nom intru
376     FROM passePar p, personne pe
377     WHERE p.idPers = pe.idPers
378     AND estIntru(p.date, p.idP, p.idPers) = 1;

379
380 -- Les triggers
381 CREATE TRIGGER triggerAppartient
382 BEFORE INSERT OR UPDATE ON appartient
383 FOR EACH ROW EXECUTE PROCEDURE testAppartient();

384
385 CREATE TRIGGER triggerReservation
386 BEFORE INSERT OR UPDATE ON reservation
387 FOR EACH ROW EXECUTE PROCEDURE testReservation();

388
389 CREATE TRIGGER triggerTache
390 BEFORE INSERT OR UPDATE ON tache
391 FOR EACH ROW EXECUTE PROCEDURE testTache();

392
393 -- Les droits
394 GRANT SELECT, UPDATE ON tache TO grtt42;
395 GRANT SELECT, UPDATE ON passepar TO grtt42;
396 GRANT SELECT, UPDATE ON piece TO grtt42;
397 GRANT SELECT, UPDATE ON personne TO grtt42;
398 GRANT SELECT, UPDATE ON rapport_activite TO grtt42;
399 GRANT SELECT, UPDATE ON intrusion TO grtt42;
400 GRANT SELECT, UPDATE ON appartient TO grtt42;
401 GRANT SELECT, UPDATE ON reservation TO grtt42;
402 GRANT SELECT ON tache TO grtt11;
403 GRANT SELECT ON passepar TO grtt11;
404 GRANT SELECT ON piece TO grtt11;
405 GRANT SELECT ON personne TO grtt11;
406 GRANT SELECT ON rapport_activite TO grtt11;

```

```

407 GRANT SELECT ON intrusion TO grtt11;
408 GRANT SELECT ON appartient TO grtt11;
409 GRANT SELECT ON reservation TO grtt11;
```

### 6.1.2 suppression.sql

```

1 DROP VIEW IF EXISTS rapport_activite;
2 DROP VIEW IF EXISTS intrusion;
3 DROP TRIGGER IF EXISTS triggerAppartient ON appartient;
4 DROP TRIGGER IF EXISTS triggerReservation ON reservation;
5 DROP TRIGGER IF EXISTS triggerTache ON tache;
6 DROP TABLE IF EXISTS reservation;
7 DROP TABLE IF EXISTS passePar;
8 DROP TABLE IF EXISTS appartient;
9 DROP TABLE IF EXISTS tache;
10 DROP TABLE IF EXISTS personne;
11 DROP TABLE IF EXISTS piece;
12 DROP FUNCTION IF EXISTS tacheCoherente(DATE, VARCHAR(10));
13 DROP FUNCTION IF EXISTS estIntru(DATE, VARCHAR(10), VARCHAR(10));
14 DROP FUNCTION IF EXISTS testAppartient();
15 DROP FUNCTION IF EXISTS testReservation();
16 DROP FUNCTION IF EXISTS testTache();
```

## 6.2 Code source C

### 6.2.1 appliUniv.pgc

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <sqlca.h>
4 #include <stdlib.h>
5 #include <string.h>
6 /* Pour le type relatif aux noms dans la table pg_tables (NAMEDATALEN)*/
7 #include <postgres.h>
8 /* Pour ajouter le type date SQL */
9 #include <pgtypes_date.h>
10
11 // Affichage des erreurs
12 EXEC SQL WHENEVER sqlerror sqlprint;
13
14 void insererReservation() {
15     EXEC SQL BEGIN DECLARE SECTION;
16     char idPers[10];
17     char idPiece[10];
18     char jourChaine[5];
19     char moisChaine[5];
20     char anneeChaine[5];
21     char dateChaine[20];
22     date dateSQL;
23     const char* requeteInsertionReservation = "INSERT INTO reservation (idP, idPers, date) VALUES ";
24     EXEC SQL END DECLARE SECTION;
25 }
```

```

26 printf("Insertion dans RESERVATION(idPers, date, idPiece)\n");
27 printf("idPers : \n");
28 scanf("%s", idPers);
29 printf("Date de la réservation\n");
30
31 // On indique que la date n'est pas valide initialement
32 int dateValide = 0;
33 do {
34     printf("Jour (JJ) :\n");
35     scanf("%s", jourChaine);
36     printf("Mois (MM) :\n");
37     scanf("%s", moisChaine);
38     printf("Année (AAAA) :\n");
39     scanf("%s", anneeChaine);
40
41     // Conversion en entiers pour vérifier la date
42     int jour, mois, annee;
43     jour = atoi(jourChaine);
44     mois = atoi(moisChaine);
45     annee = atoi(anneeChaine);
46
47     // Les différents jours max dans les mois
48     int joursDansLesMois[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
49
50     // Année bissextile ?
51     if (annee % 400 == 0 || (annee % 100 != 0 && annee % 4 == 0))
52         joursDansLesMois[1] = 29;
53
54     // Le jour correspond bien au mois ?
55     if (mois <= 12 && mois >= 1) {
56         if (jour <= joursDansLesMois[mois-1])
57             dateValide = 1;
58     }
59
60     if (dateValide == 0)
61         printf("Merci d'entrer une date valide.\n");
62 } while(dateValide == 0);
63
64 // Si l'utilisateur a rentré une date sur 1 seul caractère on est gentil avec lui
65 if (strlen(moisChaine) == 1) {
66     moisChaine[1] = moisChaine[0];
67     moisChaine[0] = '0';
68 }
69
70 // Si l'utilisateur a rentré une date sur 1 seul caractère on est gentil avec lui
71 if (strlen(jourChaine) == 1) {
72     jourChaine[1] = jourChaine[0];
73     jourChaine[0] = '0';
74 }
75
76 // Passage au format AAAA-MM-DD pour PostgreSQL
77 strcpy(dateChaine, anneeChaine);
78     strcat(dateChaine, "-");

```

```

79     strcat(dateChaine, moisChaine);
80     strcat(dateChaine, "-");
81     strcat(dateChaine, jourChaine);
82
83     printf("idPiece : \n");
84     scanf("%s", idPiece);
85
86     dateSQL = PGTYPEStdate_from_asc(dateChaine, NULL);
87     EXEC SQL PREPARE requetePreparee FROM :requeteInsertionReservation;
88     EXEC SQL EXECUTE requetePreparee USING :idPiece, :idPers, :dateSQL;
89 }
90
91 void afficherRapportIntrusion() {
92     int arreter = 0;
93     EXEC SQL BEGIN DECLARE SECTION;
94         char nomPersonne[20];
95         char idPiece[10];
96         char* dateChaine;
97         date dateSQL;
98     EXEC SQL END DECLARE SECTION;
99
100    EXEC SQL DECLARE curseurIntrusion CURSOR FOR SELECT * FROM intrusion;
101    EXEC SQL OPEN curseurIntrusion;
102
103    printf("RAPPORT D'INTRUSION date - salle - intru\n");
104    printf("----- \n");
105    do {
106        EXEC SQL FETCH curseurIntrusion INTO :dateSQL, :idPiece, :nomPersonne;
107        if (sqlca.sqlerrd[2] == 1) {
108            dateChaine = PGTYPEStdate_to_asc(dateSQL);
109            printf("%s %s %s\n", dateChaine, idPiece, nomPersonne);
110        }
111        else
112            arreter = 1;
113    } while (arreter != 1);
114    printf("----- \n");
115
116    EXEC SQL CLOSE curseurIntrusion;
117 }
118
119 void rapportActivite() {
120     int arreter = 0;
121     EXEC SQL BEGIN DECLARE SECTION;
122         char nomPersonne[20];
123         char ok[10];
124         char* dateChaine;
125         date dateSQL;
126     EXEC SQL END DECLARE SECTION;
127
128     printf("Nom de la personne ?\n");
129     // Attention, on attend une chaîne avec des espaces !
130     scanf(" %[^\n]s", nomPersonne);
131

```

```

132 EXEC SQL DECLARE curseurActivite CURSOR FOR SELECT date, ok FROM rapport_activite WHERE nom = :nomPersonne;
133 EXEC SQL OPEN curseurActivite;
134
135 printf("RAPPORT D'ACTIVITE DE %s --- date - cohérent (vrai/faux)\n", nomPersonne);
136 printf("-----\n");
137 do {
138     EXEC SQL FETCH curseurActivite INTO :dateSQL, :ok;
139     if (sqlca.sqlerrd[2] == 1) {
140         dateChaine = PGTYPESto_date(dateSQL);
141         printf("(%s, %s)\n", dateChaine, ok);
142     }
143     else
144         arreter = 1;
145 } while (arreter != 1);
146 printf("-----\n");
147
148 EXEC SQL CLOSE curseurActivite;
149 }

150
151 void lieuxVisites() {
152     int arreter = 0;
153     EXEC SQL BEGIN DECLARE SECTION;
154     char nomPersonne[20];
155     char idPiece[10];
156     char* dateChaine;
157     date dateSQL;
158     EXEC SQL END DECLARE SECTION;
159
160     printf("Nom de la personne ?\n");
161     // Attention, on attend une chaîne avec des espaces !
162     scanf(" %[^\n]s", nomPersonne);
163
164     EXEC SQL DECLARE curseurTache CURSOR FOR SELECT pa.date, pa.idP FROM passePar pa , personne p WHERE pa.nomPersonne = '%s';
165     EXEC SQL OPEN curseurTache;
166
167     printf("LIEUX VISITES PAR %s --- date - piece\n", nomPersonne);
168     printf("-----\n");
169     do {
170         EXEC SQL FETCH curseurTache INTO :dateSQL, :idPiece;
171         if (sqlca.sqlerrd[2] == 1) {
172             dateChaine = PGTYPESto_date(dateSQL);
173             printf("(%s, %s)\n", dateChaine, idPiece);
174         }
175         else
176             arreter = 1;
177     } while (arreter != 1);
178     printf("-----\n");
179
180     EXEC SQL CLOSE curseurTache;
181 }

182
183 void supprimerPersonne() {
184     EXEC SQL BEGIN DECLARE SECTION;

```

```

185     char idPers[10];
186     const char* requeteSuppression = "DELETE FROM personne WHERE idPers = ?";
187 EXEC SQL END DECLARE SECTION;
188
189 printf("Valeur de l'attribut idPers du tuple à supprimer : ");
190 scanf("%s", idPers);
191
192 EXEC SQL PREPARE requetePreparee FROM :requeteSuppression;
193 EXEC SQL EXECUTE requetePreparee USING :idPers;
194 }
195
196 void insererPersonne() {
197     EXEC SQL BEGIN DECLARE SECTION;
198     char idPers[10];
199     char nom[20];
200     char grade[10];
201     const char* requeteInsertion = "INSERT INTO personne(idPers, nom, grade) VALUES (?, ?, ?);"
202 EXEC SQL END DECLARE SECTION;
203
204     printf("Insertion dans PERSONNE(idPers, nom, grade)\n");
205     printf("idPers : ");
206     scanf("%s", idPers);
207     printf("nom : ");
208     scanf("%s", nom);
209     printf("grade : ");
210     scanf("%s", grade);
211
212     while (strcmp(grade, "PE") != 0 && strcmp(grade, "BIATOSS") != 0 && strcmp(grade, "Etudiant") != 0)
213         printf("Les grades pris en compte sont PE, BIATOSS, Etudiant, PU et MCF. Veuillez entrer un autre grade : ");
214         scanf("%s", grade);
215 }
216
217
218 EXEC SQL PREPARE requetePreparee FROM :requeteInsertion;
219 EXEC SQL EXECUTE requetePreparee USING :idPers, :nom, :grade;
220 }
221
222 int main (int argc, char *argv[]) {
223     int entree;
224     int sortir = 0;
225
226     /* Les variables de connexion à la base */
227     EXEC SQL BEGIN DECLARE SECTION;
228     char base[100];
229     const char* user = "grtt6";
230     const char* motDePasse = "grtt6";
231     EXEC SQL END DECLARE SECTION;
232
233     EXEC SQL INCLUDE sqlca;
234
235     /* --- DEBUT CONNEXION A LA BASE --- */
236     strcpy(base, "tcp:postgresql://asi-pg.insa-rouen.fr:5432/grtt6");
237

```

```

238 printf ("Connexion à la base : %s\n", base);
239 EXEC SQL CONNECT to :base USER :user USING :motDePasse;
240 printf ("Connexion à la base effectuée\n");
241 /* --- FIN CONNEXION A LA BASE --- */

242
243 do {
244     printf("Que voulez-vous faire ?\n");
245     printf("1 - Ajouter une personne\n");
246     printf("2 - Supprimer une personne\n");
247     printf("3 - Faire une réservation de salle\n");
248     printf("4 - Afficher la liste des lieux visités par une personne\n");
249     printf("5 - Afficher le rapport d'activité d'une personne\n");
250     printf("6 - Afficher le rapport d'intrusion\n");
251     printf("7 - SORTIR\n");

252     scanf("%i", &entree);

253     switch(entree) {
254         case 1 :
255             printf("1 - Ajouter une personne\n");
256             insererPersonne();
257             break;
258         case 2 :
259             printf("2 - Supprimer une personne\n");
260             supprimerPersonne();
261             break;
262         case 3 :
263             printf("3 - Faire une réservation de salle\n");
264             insererReservation();
265             break;
266         case 4 :
267             printf("4 - Afficher la liste des lieux visités par une personne\n");
268             lieuxVisites();
269             break;
270         case 5 :
271             printf("5 - Afficher le rapport d'activité d'une personne\n");
272             rapportActivite();
273             break;
274         case 6 :
275             printf("6 - Afficher le rapport d'intrusion\n");
276             afficherRapportIntrusion();
277             break;
278         case 7 :
279             sortir = 1;
280             break;
281     }
282 }
283 EXEC SQL COMMIT;

284
285 }
286 } while (sortir != 1);

287
288 printf ("Déconnexion\n");
289 EXEC SQL DISCONNECT;
290 printf ("Déconnexion effectuée\n");

```

```
291  
292     return EXIT_SUCCESS;  
293 }
```