

ENIB semestre S6o :

Informatique - Bases de données relationnelles

HOPITAL

Document de conception

Younes el amrani lias Maazouz



Version : 8 juin 2024

**Table des matières**

1. [Cahier des charges](#_bookmark0) 2
2. [Tout dans une table](#_bookmark1) 2
   1. [Création de la base](#_bookmark2) 2
      1. [Modélisation](#_bookmark3) 2
      2. [Structuration](#_bookmark4) 2
      3. [Création](#_bookmark6) 3
   2. [Manipulation de données](#_bookmark7) 3
      1. [Chargement de données](#_bookmark8) 4
   3. [Cas d’usages](#_bookmark9) 4
      1. [Recherche d’informations](#_bookmark10) 4
      2. [Traitement d’informations](#_bookmark11) 5
   4. [Bilan](#_bookmark14) 7
3. [Modélisation](#_bookmark15) 7
   1. [Entité-Association](#_bookmark16) 7
   2. [Structuration](#_bookmark17) 9
   3. [Création de la base](#_bookmark19) 9
   4. [Insertion d’informations](#_bookmark20) 10
   5. [État de la base de données](#_bookmark21) 10
4. [Cas d’usages](#_bookmark22) 11
   1. [Recherche d’information](#_bookmark23) 11
      1. [Projection et restriction](#_bookmark24) 12
      2. [Jointures entre tables](#_bookmark25) 12
      3. [Opérations ensemblistes](#_bookmark26) 13
      4. [Division relationnelle](#_bookmark27) 14
      5. [Fonctions d’agrégat](#_bookmark28) 14
      6. [Groupements](#_bookmark29) 15
5. [conclusion](#_bookmark30) 16
6. [Annexes](#_bookmark31) 18
   1. [Langage SQL](#_bookmark32) 18
      1. [Création de tables](#_bookmark33) 18
      2. [Insertion d’information](#_bookmark34) 20

# Cahier des charges

On souhaite mettre en œuvre une base de données qui permettra de gérer les informations concernant les services hospitaliers sur le territoire national. Cette base de données devra permettre de gérer les informations sur les hôpitaux, y compris leur nom et la ville où ils se trouvent. Chaque hôpital sera composé de plusieurs services et laboratoires, chacun étant identifié distinctement, même s'ils partagent le même type (par exemple, deux maternités ou deux laboratoires de cancérologie). Pour chaque service et laboratoire, on enregistrera leur nom. Un médecin ne pourra exercer que dans un seul service et effectuer ses recherches dans un seul laboratoire. On devra enregistrer pour chaque médecin son nom, la ville où il exerce et sa spécialité. Un service et un laboratoire seront constitués de plusieurs médecins. Un patient pourra consulter plusieurs médecins d'un même hôpital et un médecin pourra recevoir plusieurs patients. Pour chaque patient, on enregistrera son nom, prénom, date de naissance et la ville où il habite. Pour manipuler les informations, il faudra pouvoir utiliser un outil (type tableur) accessible à un utilisateur, permettant de charger et sauvegarder les informations sur chaque entité du système hospitalier.

# Tout dans une table

Dans une première version de notre projet, nous allons simplifier ce cahier des charges pour créer une base de données dans laquelle nous stockerons les informations concernant les consultations dans une seule table que nous pourrons ensuite manipuler dans un fichier au format CSV. Notre cahier des charges simplifié pourrait être:

En tant qu'utilisateur de la base de données, je veux récupérer les consultations (date, heure, nom du patient, prénom du patient, nom du médecin, prénom du médecin, service, laboratoire) en connaissant le nom de l'hôpital et la ville où il se trouve.

## Création de la base

A partir de ce cahier des charges nous allons créer une première version de la base de données.

### Modélisation

Pour modéliser notre base de données, à partir du cahier des charges, nous relevons les entités et les informations les caractérisant. Dans un premier temps, nous décidons de définir une seule entité **consultations** : consultation\_id, hôpital, ville\_hôpital, date, heure, nom\_patient, prénom\_patient, date\_naissance\_patient, ville\_patient, nom\_médecin, prénom\_médecin, spécialité\_médecin, ville\_médecin, service, laboratoire

En utilisant le formalisme UML, on peut représenter graphiquement l'entité **consultations** qui intègre ses propriétés.

### Structuration

AÀ partir de la modélisation précédente, nous pouvons proposer une structuration de la base contenant une seule table avec des noms de colonnes correspondant au modèle UML.

1. CSV : Comma Separated Values

consultations(

consultation\_id, nom\_hopital,

ville\_hopital,

service,

laboratoire,

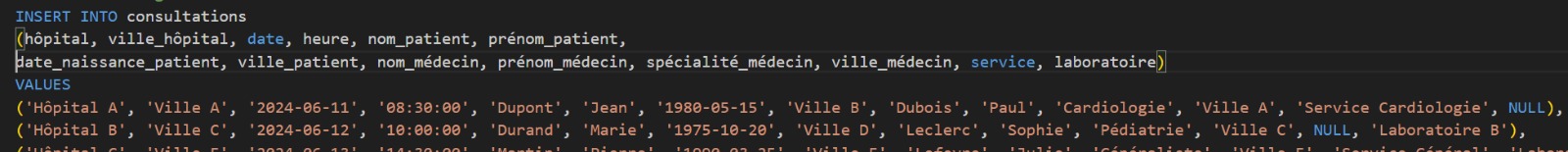
nom\_medecin, ville\_medecin, specialite\_medecin, nom\_patient, prenom\_patient, date\_naissance\_patient, ville\_patient)

L’attribut souligné (consultation\_id) représentera la clé primaire (PRIMARY KEY) pour identifier de manière unique un enregistrement (élément) dans une table (consultations).

### Création

À partir de la modélisation précédente, nous pouvons proposer une structuration de la base contenant une seule table avec des noms de colonnes correspondant au modèle UML.

Une fois la création de table réalisée, nous pourrons alimenter la base de données en utilisant les commandes d'insertion du langage SQL.

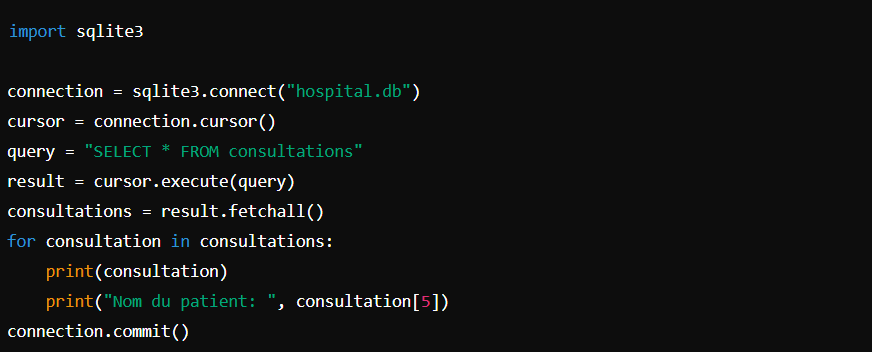


## Manipulation de données

On doit pouvoir traiter les données d'une base dans un programme d'application. La première application va nous permettre de charger les données de la base et vérifier que l'on récupère bien les données stockées.

### Chargement de données

Le programme python ci-dessous montre simplement comment se connecter à une base de données SQLite pour faire une recherche d’information en SQL et afficher les données récupérées dans la table des matchs.

consultation.py

## Cas d’usages

Nous allons distinguer deux types de cas d’usage :

1. recherche d’informations
2. traitement d’informations

La première catégorie nous permettra de mettre en place des interrogations en langage SQL sur la base de données.

La deuxième catégorie nous permettra de manipuler dans un programme d’application des données stockées dans une base relationnelle.

### Recherche d’informations

Pour tous nos cas d’usages de recherche d’informations nous exprimerons ces "use-case" par :

* + - * une phrase en français pour exprimer la recherche d’information
      * l’expression algèbrique correspondant à cette recherche
      * la requête SQL permettant de mettre en œuvre la recherche sur la base de données du projet.

Sur une seule table, les cas d’usages à mettre en place devront servir à tester les projections (Π), restrictions (σ) et utilisation de connecteurs logiques (∧, ∨).

**Cas d'usage** :

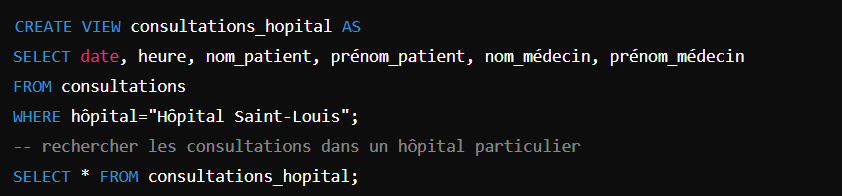
Récupérer, dans une vue, l’identifiant des équipes (à domicile et visiteurs) lors de la première journée du tournoi : Récupérer, dans une vue, les consultations ayant eu lieu dans un hôpital particulier.

**Algèbre relationnelle**

* 1. R = σ[hôpital="Hôpital Saint-Louis"](consultations\_hopital)
  2. P = Π(date, heure, nom\_patient, prénom\_patient, nom\_médecin, prénom\_médecin)(R)
  3. matchs\_J1 = P

Explication :

1. R : rechercher les consultation de l’ Hôpital Saint-Louis
2. P : récupérer la date et l’heure et nom de patient et de medcin
3. consultations\_hopital: créer une vue sur les consultations dans un hôpital particulier

**Requête SQL**

Le deuxième cas d’usage sert à vérifier la prise en compte de la priorité du connecteur de conjonction (∧) sur le connecteur de disjonction (∨) en langage SQL.

**Cas d’usage**

Récupérer les équipes (à domicile et visiteurs) des matchs de la première journée ou du match contre la France lors de la troisième journée

**Algèbre relationnelle**

1. R = σ[hôpital="Hôpital Saint-Louis"  ∨ hôpital="Hôpital brest" ∧ *prénom\_patient*=”*YOUNES*”](consultation)
2. P =Π(*nom\_medecin,heure,date*)(R) Explication :
   1. R : les consultation de l’hopital de  " Saint-Louis"  ou de l’hopital de  " brest" de patient younes
   2. P : le nom de médecin et l’heure et la date de consultation

**Requête SQL**

SELECT nom\_médecin,heure,date FROM consultation

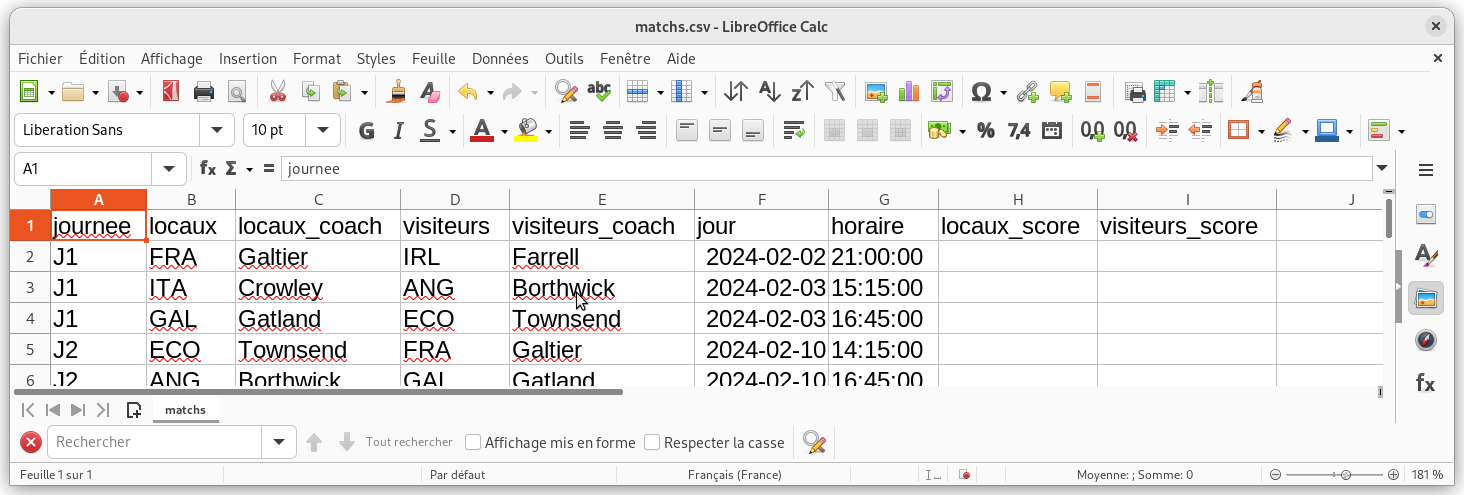
WHERE hopital=" Hôpital Saint-Louis " OR hopital=" Hôpital brest " AND nom\_patient="younes";

### Traitement d’informations

Pour traiter l’information, nous allons proposer un moyen de charger les données de la base dans une application connues des utilisateurs et, réciproquement, de sauvegarder les données manipulées par cette application dans une base de données.

Pour ce type de cas d’usage nous allons utiliser un outil de type tableur (office, libreoffice ...) pour charger ou sauvegarder dans des tables de la base de données le informations manipulables par cet outil.

La figure [1](#_bookmark12) illustre la représentation de données au format CSV sous libreoffice. L’utilisateur peut alors, à l’aide de cet outil, modifier les informations contenues dans ce fichier afin de les sauvegarder dans une base de données pour pouvoir les rechercher ultérieurement.

Figure 1 – Chargement de fichier matchs.csv sous libreoffice Pour notre cas d’usage nous allons mettre en place deux applications python pour :

1. charger une table de base de données et générer les données un fichier au format CSV
2. charger un fichier au format CSV, le modifier et sauvegarder les modifications dans une base de données La première application python ci-dessous, permet de :
   * se connecter sur une base de données (sixnations.db)
   * ouvrir un fichier CSV (open("matchs.csv","w")) en écriture
   * rechercher les données de la table des matchs (SELECT \* from matchs)
   * charger les données dans l’application (cursor.execute(select\_match)
   * pour chaque enregistrement récupéré (for match in matchs)
   * écrire pour chaque colonne de l’enregistrement (for i in range(len(columns))
   * les données de l’enregistrement (string) dans le fichier CSV (writerow(eval(""+string+""))).

matchs\_to\_csv.py

**import** sqlite3

**import** csv

connection = sqlite3 . connect(" sixnations. db") connection . row\_factory = sqlite3 . Row cursor= connection . cursor () f= **open** (" matchs. csv "," w") select\_match =" SELECT \* from matchs" matchs= cursor. execute ( select\_match ) columns =[]

**for** match **in** matchs. description : columns. append ( match [0])

with f : writer= csv . DictWriter(f, fieldnames= columns) writer. writeheader ()

**for** match **in** matchs : string =""

**for** i **in range** ( **len** ( columns )) : string = string +" columns[" + **str** ( i)+"]:" +" match [ columns[" + **str** ( i)+"]],"

writer. writerow ( **eval** ("{" + string +"}"))

)

connection . commit ()

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

A l’exécution de ce script python, on aura sauvegardé les données de la base dans un fichier au format CSV qui pourra être utilisé par la suite dans une application reconnaissant ce format, comme illustré sur la figure **??**. Réciproquement, dans un programme d’application nous pourrons récupérer les données d’un fichier CSV, les modifier dans l’application et les stocker dans une base de données SQL, comme illustré par le code ci-dessous. Dans ce deuxième programme d’application :

* + on se connecte sur une base de données (sixnations.db)
  + on ouvre le fichier CSV en lecture (open("matchs.csv","r"))
  + on définit la requête de mise à jour (UPDATE matchs SET locaux\_score=? ...) qui utilisera des données (les ?) à mettre à jour dans la base.
  + pour chaque ligne du fichier CSV (for row in reader )
  + s’il y a des données à mettre à jour (if row["locaux\_score"] and row["visiteurs\_score"] : )
  + on exécute la requête de mise à jour sur ces données (cursor.execute())
  + pour le match concerné (row["match\_id"])
  + sinon il n’y a pas de mise à jour à faire

csv\_to\_matchs.py

**import** sqlite3

**import** csv

connection = sqlite3 . connect(" sixnations. db") cursor= connection . cursor () f= **open** (" matchs. csv "," r")

update\_match =

" UPDATE matchs SET locaux\_score =?, visiteurs\_score = ? WHERE match\_id = ?" with f :

reader = csv . DictReader(f, delimiter=",")

**for** row **in** reader :

**if** row [" locaux\_score "] **and** row [" visiteurs\_score "] : cursor. execute (

update\_match , ( row [" locaux\_score "], row [" visiteurs\_score "], row [" match\_id "])

)

**else** :

**print** (" no score for match between "\

+ row [" locaux "]+" and " + row [" visiteurs"]\

+" for match\_id : " + row [" match\_id "]) connect. commit ()

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

## Bilan

L'intégration des données des hôpitaux dans une seule table présente des avantages en termes de simplicité et de gestion. Cependant, cette approche peut poser des défis pour l'intégrité et la cohérence des données, en particulier lorsque les informations deviennent volumineuses et complexes. Le passage à un modèle de base de données relationnelle plus structuré, avec des tables séparées pour les hôpitaux, services, laboratoires, médecins, et patients, est recommandé pour les futures versions du projet.

# Modélisation

Pour modéliser notre base de données hospitalière, nous utilisons le modèle Entité-Association (EA) pour représenter les relations entre les différentes entités : hôpitaux, services, laboratoires, médecins et patients.

## Entité-Association

On définit plus précisement le cahier des charges initial en faisant ressortir (en gras) les entités identifiées.

On souhaite mettre en place un système d’information pour gérer les services hospitalier sur le territoire national. Un hopital est composé de plusieurs services et laboratoires dans lesquels travaillent des médecins qui consultent des patients. On connait le nom de l’hopital et la ville où il se trouve . On distingue chaque service et chaque laboratoire quelque soit son hopital d’appartenance. Ainsi deux maternités, deux laboratoires de cancérologie etc …, seront bien distincts mais seront rattachés à un seul hopital. Pour chaque service et laboratoire on connait leur nom. Un médecin ne peut exercer que dans un seul service et fait sa recherche dans un seul laboratoire. On connait le nom du médecin, la ville où il exerce et sa spécialité. Un service et un laboratoire sont, bien évidemment, constitués de plusieurs médecins. Un patient peut consulter plusieurs médecins d’un hôpital et un médecin peut recevoir plusieurs patients. Du patient on connait ses nom, prénom date de naissance et ville où il habite.

Une fois les entités identifiées on relève dans le cahier des charges les informations qui leur sont liées :

* **Hôpital** : nom, ville
* **Service** : nom, hôpital\_id
* **Laboratoire** : nom, hôpital\_id
* **Médecin** : nom, ville, spécialité, service\_id, laboratoire\_id
* **Patient** : nom, prénom, date de naissance, ville

En fonction de ce recensement d’entités et des informations qui les caractérisent, et à partir du cahier des

charges, on peut établir les associations entre les différentes entités en explicitant certaines phrases sous la forme :

* Sujet-Verbe-Complément

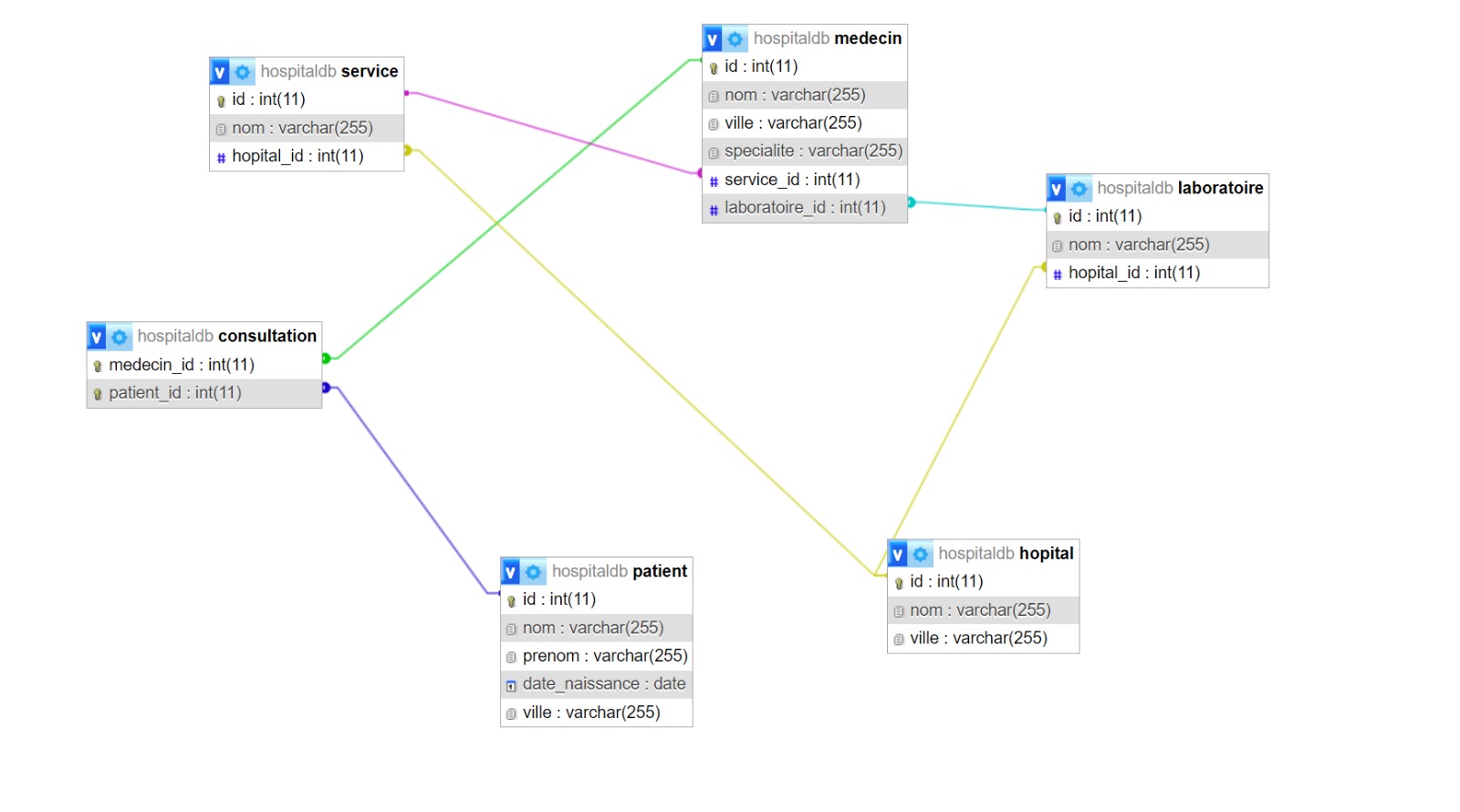
où les *sujet,complément* représenteront des entités et le *verbe* l’**association** entre ces entités.

L’inversion des *sujet,complément* devra permettre d’expliciter les **cardinalités** des associations entre les deux entités. Les cardinalités définissent le nombre de représentants de chaque entité des deux côtés de l’association. On peut reformuler les phrases du cahier des charges pour extraire les phrases suivantes :

* Un hôpital contient plusieurs services et laboratoires.
* Un service appartient à un seul hôpital et contient plusieurs médecins.
* Un laboratoire appartient à un seul hôpital et contient plusieurs médecins.
* Un médecin travaille dans un seul service et fait sa recherche dans un seul laboratoire.
* Un patient peut consulter plusieurs médecins et chaque médecin peut recevoir plusieurs patients.

En fonction des entités, associations et cardinalités relevées à partir de ce cahier des charges nous pouvons

proposer le modèle UML suivant :





1.. \*

1.. \*

1

1

1.. \*

1

1

1.. \*

1.. \*

## Structuration

Figure 2 – Tournoi des Six Nations

A partir de la proposition du modèle de données décrite par le diagramme UML de la figure [2](#_bookmark18) nous pouvons traduire ce modèle conceptuel par la structuration de la base de données suivante :

hopital(hopital\_id,nom,ville)

service(service\_id,nom,#hopital\_id)

laboratoire(laboratoire\_id, nom, #hopital\_id)

medecin(medecin\_id, nom, ville,specialite,#service\_id,#laboratoire\_id)

patient(patient\_id, nom, prenom, date\_naissance, ville)

consultation(#consultation\_id, #medecin\_id, #patient\_id, date)

Les attributs *soulignés* symbolisent les clés primaires des tables à créer.

Les attributs *hashtagués* symbolisent les références (associations) sur d’autres entités (tables). Les références seront représentées par des clés étrangères sur ces tables.

## Création de la base

A partir de cette structuration de la base de données nous pouvons utiliser le langage SQL pour créer les tables correspondantes.

Dans ce document nous représentons simplement la création des tables hopital, service, laboratoire, medecin, patient, consultation qui illustre l’uti- lisation de clés primaires et étrangères. La création de autres tables se trouvera dans le code SQL (fichier hopital\_create.sql) associé à ce document.

-- Table des hôpitaux ------------------------------------------------------------------------------------------

CREATE TABLE hopitaux (

hopital\_id INTEGER PRIMARY KEY,

nom VARCHAR(100),

ville VARCHAR(100) );

-- Table des service--------------------------------------------------------------------------------------------

CREATE TABLE service (

service\_id INTEGER PRIMARY KEY,

nom VARCHAR(100),

hopital\_id INTEGER,

FOREIGN KEY (hopital\_id) REFERENCES hopital(hopital\_id) );

-- Table des laboratoire---------------------------------------------------------------------------------------

CREATE TABLE laboratoire (

laboratoire\_id INTEGER PRIMARY KEY,

nom VARCHAR(100),

hopital\_id INTEGER,

FOREIGN KEY (hopital\_id) REFERENCES hopital(hopital\_id) );

-- Table des médecins -----------------------------------------------------------------------------------------

CREATE TABLE medecins (

medecin\_id INTEGER PRIMARY KEY,

nom VARCHAR(100),

ville VARCHAR(100),

specialite VARCHAR(100),

service\_id INTEGER,

laboratoire\_id INTEGER,

FOREIGN KEY (service\_id) REFERENCES services(service\_id), FOREIGN KEY (laboratoire\_id) REFERENCES laboratoires(laboratoire\_id) );

-- Table des patient-----------------------------------------------------------------------------------------

CREATE TABLE patient(

patient\_id INTEGER PRIMARY KEY,

nom VARCHAR(100),

prenom VARCHAR(100),

date\_naissance DATE,

ville VARCHAR(100) );

-- Table des consultations

CREATE TABLE consultations (

consultation\_id INTEGER PRIMARY KEY,

medecin\_id INTEGER,

patient\_id INTEGER,

date DATE,

FOREIGN KEY (medecin\_id) REFERENCES medecin(medecin\_id),

FOREIGN KEY (patient\_id) REFERENCES patient(patient\_id) );

## Insertion d’informations

Afin de pouvoir faire des tests sur des cas d’usage, quel que soit le système de gestion hospitalière à étudier, il est inutile d’insérer une quantité importante d’informations (trop d’informations tuent l’information !). Dans un premier temps, nous alimenterons notre base de données en considérant simplement le cas d'un seul hôpital avec deux services (Maternité, Urgences) et deux laboratoires (Laboratoire de Biologie, Laboratoire de Cancérologie) où travailleront trois médecins (Dr. Dupont, Dr. Martin, Dr. Bernard).

## État de la base de données

On trouvera les commandes d’insertion d’information dans le fichier sixnations\_insert.sql représenté en annexe [p.6.1.2.](#_bookmark34)

Nous insèrerons volontairement un nombre minimal d’informations dans la base de manière à pouvoir tester et vérifier facilement les résultats des requêtes SQL correspondant aux cas d’usages que nous mettrons en œuvre dans la section suivante.

À partir de l’état de cette base de données nous pouvons déjà vérifier, en faisant une jointure entre les différentes tables, que nous récupérons l’ensemble des informations utiles de la base de données.

1

CREATE VIEW sixnations AS SELECT

m.journee,m.jour,m.horaire AS heure, dom.equipe\_id AS domicile,m.locaux\_score, ext.equipe\_id AS exterieur,m.visiteurs\_score, p.nom,p.prenom,p.naissance,p.nationalite, j.equipe\_id AS equipe,j.poste,j.taille,j.poids, pt.horaire,pt.genre

FROM matchs m

INNER JOIN equipes dom ON m.locaux\_id=dom.equipe\_id INNER JOIN equipes ext ON m.visiteurs\_id=ext.equipe\_id

INNER JOIN joueurs j ON (j.equipe\_id=dom.equipe\_id OR j.equipe\_id=ext.equipe\_id) INNER JOIN personnes p ON p.personne\_id=j.joueur\_id

INNER JOIN points pt ON (pt.joueur\_id=j.joueur\_id AND pt.match\_id= m.match\_id);

SELECT \* FROM sixnations;

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

Nous pouvons réutiliser le script python (section [2.3.2)](#_bookmark13) de sauvegarde de données d’une table (matchs) dans un fichier CSV (matchs.csv) pour l’adapter au chargement des données de la vue (sixnations) dans un fichier CSV (sixnation.csv).

Nous pourrons exploiter les données de ce fichier dans une application de type tableur comme représenté ci- dessous.

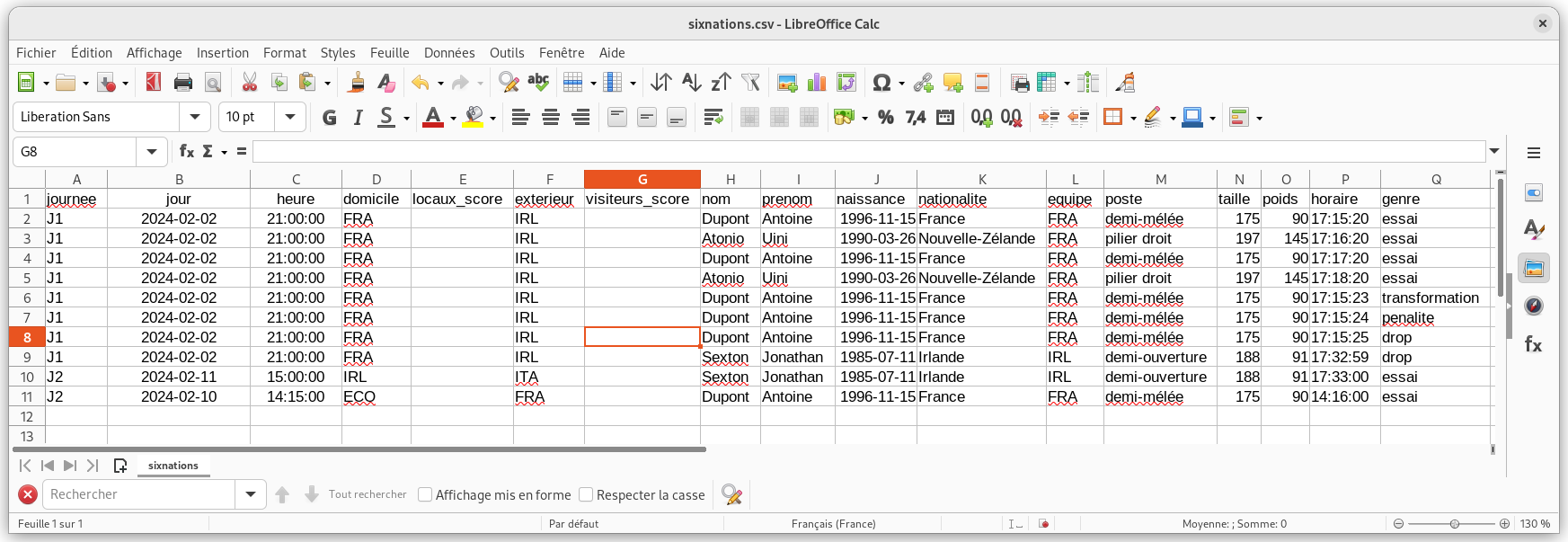


Figure 3 – Chargement du fichier sixnations.csv sous libreoffice

L’utilisation de la vue correspondant à ces jointures permet de faciliter la recherche d’informations. En effet, nous pouvons simplement rechercher :

* toutes les informations sur les matchs de l’equipe de France

SELECT \* FROM sixnations WHERE equipe=’FRA’;

Ou encore :

* toutes les informations sur ’Antoine’ ’Dupont’ durant ce tournoi

SELECT \* FROM sixnations WHERE nom=’Dupont’ AND prenom=’Antoine’;

Cependant lorsque la recherche d’informations ne nécessite pas l’utilisation de toutes les tables il ne faudra pas avoir recours à cette vue qui ne représente finalement que l’encapsulation d’une requête qui exécute les jointures à chaque utilisation.

De plus nous constatons la redondance d’informations représentées dans cette vue. Redondance qu’il faut éviter d’avoir à chaque recherche d’informations si les tables impliquées dans une jointure ne sont pas utiles à la recherche.

A partir de notre structuration de la base de données, nous pourrons mettre en place différents cas d’usages de recherche d’informations sur différentes tables en testant les différents opérateurs de l’algèbre relationnelle, en vérifiant que le résultat des requêtes est obuste aux changement d’état de la base de données.

# Cas d’usages

Dans cette version de notre projet, nous ne mettrons en place que les cas d’usage de recherche d’informations.

## Recherche d’information

Les cas d’usage que nous allons définir vont nous permettre de mettre en œuvre les différents opérateurs de l’algèbre relationnelle ainsi que les calculs sur les résultats de requêtes (fonctions d’agrégat) et les regroupements d’informations dans une requête.

Selon le modèle de données de notre projet, nous formulerons en langage SQL les questions suivantes afin de tester les différents opérateurs de l’algèbre relationnelle.

1. sur une seule table (Π, σ, ∧, ∨)
   * récupérer l’identifiant des équipes (à domicile et visiteurs) lors de la première journée du tournoi.
   * récupérer les équipes (à domicile et visiteurs) des matchs de la première journée ou du match contre la France lors de la troisième journée
2. sur plusieurs tables (×, ▷◁, \)
   * pour chaque match, récupérer l’identifiant de l’équipe qui joue à domicile et son score, suivi de l’identifiant de l’équipe qui joue à l’extérieur et de son score.
   * trouver les joueurs dont la nationalité est différente de celle de l’équipe dans laquelle ils jouent.
3. pour représenter l’opérateur de division relationnelle (𝚵)
   * trouver les joueurs de l’équipe de France qui ont été sélectionnés à tous les matchs de leur équipe.
   * trouver les joueurs des équipes du tournoi qui ont été sélectionnés à tous les matchs de leur équipe.
4. en appliquant des fonctions d’agrégat (avg(),count(),sum() ...) sur les résultats de requête SQL
   * nombre d’essais marqués lors du tournoi
   * le nom et le poids des joueurs les plus lourds du tournoi.
5. en appliquant des fonctions d’agrégat (avg(),count(),sum() ...) sur les résultats de groupements
   * récupérer, pour un match (le premier par exemple), le nombre d’essais,transformations,pénalités et drop marqués pour l’équipe jouant à domicile. (GROUP BY)
   * récupérer, pour chaque match des équipes jouant à domicile, le nombre d’essais marqués lorsque ce nombre est au moins égal à quatre. (GROUP BY,HAVING)

Comme dans le cadre de la prmeière version de notre projet nous exprimerons nos cas d’usages de recherche d’information par :

* une phrase en français pour exprimer la recherche d’information
* l’expression algèbrique correspondant à cette recherche
* la requête SQL permettant de mettre en œuvre la recherche sur la base de données du projet.

Il sera nécessaire de tester et vérifier les cas d’usage en exécutant les requêtes SQL sur une base contenant

suffisamment (mais pas trop) d’informations.

### Projection et restriction

Ces deux cas d’usages ont déjà été abordés lors de la première version de notre projet [(p.4)](#_bookmark9)

### Jointures entre tables

Lorsqu’une base de données a été structurée en plusieurs tables suivant le modèle Entité-Association, il faut évidemment vérifier que nous puissions récupérer les informations liées entre elles par ces associations comme si toutes ces informations ne formaient qu’une seule entité.

**Cas d’usage**

Pour chaque match, récupérer le nom de l’équipe qui joue à domicile et son score, suivi du nom de l’équipe qui joue à l’extérieur et de son score.

**Algèbre relationnelle**

1. J*locaux* =▷◁[*matchs.locaux*\_*id*=*equipes.equipe*\_*id*] (matchs, equipes)
2. P*locaux* = Π(*match*\_*id,nom,locaux*\_*score*)(J*locaux*)
3. P*locaux* = ρ*nom*→*locaux*(P*locaux*)
4. J*visiteurs* =▷◁[*matchs.visiteurs*\_*id*=*equipes.equipe*\_*id*] (matchs, equipes)
5. P*visiteurs* = Π(*match*\_*id,nom,visiteurs*\_*score*)(J*visiteurs*)
6. P*visiteurs* = ρ*nom*→*visiteurs*(P*visiteurs*)
7. J*matchs* =▷◁[*Plocaux.match*\_*id*=*Pvisiteurs.match*\_*id*] (P*locaux*, P*visiteurs*) Explication
8. J*locaux* : les équipes qui jouent à domicile (les locaux)
9. P*locaux* : les journées des matchs, le nom des équipes qui jouent à domicile et leur score
10. P*locaux* : on renomme le nom de l’équipe qui joue à domicile en locaux
11. J*visiteurs* : les équipes qui jouent à l’extérieur (les visiteurs)
12. P*visiteurs* : les journées des matchs, le nom des équipes qui jouent à l’extérieur et leur score
13. P*visiteurs* : on renomme le nom de l’équipe qui joue à l’extérieur en visiteurs
14. J*matchs* : les informations sur les équipes qui jouent à domicile et à l’extérieur.

**Requête SQL**

SELECT m.match\_id, dom.nom AS locaux, m.locaux\_score, ext.nom AS visiteurs ,m.visiteurs\_score FROM matchs m

INNER JOIN equipes dom ON m.locaux\_id=dom.equipe\_id INNER JOIN equipes ext ON m.visiteurs\_id=ext.equipe\_id;

Cette formulation de requête correspond bien au résultat attendu en faisant une double jointure entre la table des matchs et celle des deux equipes jouant à domicile et à l’extérieur. Cependant, si nous voulons établir une correspondance entre les expressions algébriques et les requêtes SQL nous pouvons décomposer la requête initiale en créant des vues (v*locaux*, v*visiteurs*) :

* pour chaque match, récupérer l’équipe qui joue à domicile et son score :
* J*locaux* =▷◁[*matchs.locaux*\_*id*=*equipes.equipe*\_*id*] (matchs, equipes)
* P*locaux* = Π(*match*\_*id,nom,locaux*\_*score*)(J*locaux*)
* P*locaux* = ρ*nom*→*locaux*(P*locaux*)
* V*locaux* = P*locaux*
* pour chaque match, récupérer l’équipe qui joue à l’extérieur et son score :
* J*visiteurs* =▷◁[*matchs.visiteurs*\_*id*=*equipes.equipe*\_*id*] (matchs, equipes)
* P*visiteurs* = Π(*match*\_*id,nom,visiteurs*\_*score*)(J*visiteurs*)
* P*visiteurs* = ρ*nom*→*visiteurs*(P*visiteurs*)
* V*visiteurs* = P*visiteurs*
* faire une jointure (naturelle) sur l’identifiant des matchs des deux vues (V*locaux*, V*visiteurs*) :
* J*match* =▷◁[*V .match*\_*id*=*V .match*\_*id*] (V*locaux*, V*visiteurs*)

*locaux visiteurs*

Le code SQL ci-dessous représente cette décomposition en vues et la jointure (naturelle) entre ces vues.

CREATE VIEW v\_locaux AS

SELECT m.match\_id,e.nom AS locaux,m.locaux\_score AS score

FROM matchs m INNER JOIN equipes e ON m.locaux\_id=e.equipe\_id;

CREATE VIEW v\_visiteurs AS

SELECT m.match\_id,e.nom AS visiteurs ,m.visiteurs\_score AS score FROM matchs m INNER JOIN equipes e ON m.visiteurs\_id=e.equipe\_id;

SELECT \*

FROM v\_locaux d NATURAL JOIN v\_visiteurs;

### Opérations ensemblistes

Savoir formuler une requête avec des projections,restrictions et jointures entre tables (les ensembles contenant l’information des bases de données) permet de générer un nouvel ensemble constitué des éléments retournés par la requête.

L’étape suivante consiste à pouvoir faire des opérations ensemblistes entre deux requêtes afin de trouver :

* + - * les informations contenues dans ces deux ensembles (union,∪)
      * seulement les informations communes à ces deux ensembles (intersection,∩)
      * les informations qui sont dans un ensemble mais pas dans l’autre (différence,\)

**Cas d’usage**

trouver les joueurs dont la nationalité est différente de celle de l’équipe dans laquelle ils jouent.

**Algèbre relationnelle**

1. J*jp* =▷◁[*j.joueur*\_*id*=*p.personne*\_*id*] (joueurs, personnes)
2. P*nationalite* = Π(*joueur*\_*id,nationalite*)(J*jp*)
3. J*je* =▷◁[*j.equipe*\_*id*=*e.equipe*\_*id*] (joueurs, equipes)
4. P*pays* = Π(*joueur*\_*id,pays*)(J*je*)
5. E*n*\_*p* = \(P*nationalite*, P*pays*)

**Requête SQL**

SELECT joueur\_id,nationalite FROM v\_joueurs\_personnes EXCEPT

SELECT joueur\_id,pays FROM v\_joueurs\_equipes;



**Utilisation des vues**

Nous avons créé précédemment (section [6.1.1)](#_bookmark33) les vues (v\_joueurs\_personnes, v\_joueurs\_equipes) correspondant aux jointures J*jp*, J*je*. C’est le moment de savoir les réutiliser !

### Division relationnelle

Dans le cadre de notre projet, nous pouvons définir un cas d’usage permettant la mise en place d’une division relationnelle.

**Cas d’usage**

trouver les joueurs de l’équipe de France qui ont été sélectionnés à tous les matchs de leur équipe.

**Algèbre relationnelle**

1. R*france* = σ[*matchs.locaux*\_*id*=*′F RA′* ∨ *matchs.visiteurs*\_*id*=*′F RA′*](matchs)
2. P*matchs* = Π(*match*\_*id*)(R*f rance*)
3. P*selections* = Π(*joueur*\_*id,match*\_*id*)(selections)
4. D = 𝚵(P*selections*, P*matchs*)

**Requête SQL**

SELECT DISTINCT joueur\_id FROM selections se

WHERE NOT EXISTS (

SELECT match\_id FROM matchs m

WHERE m.locaux\_id=’FRA’ OR m.visiteurs\_id=’FRA’ EXCEPT

SELECT match\_id FROM selections si WHERE si.joueur\_id=se.joueur\_id

);

### Fonctions d’agrégat

Les fonctions, dites d’agrégat, servent à appliquer des calculs sur les ensembles produits lors de requêtes SQL. Elles permettent de faire des "statistiques" (sum(),count(),avg() ...) sur les résultats de requêtes.

On peut utiliser ces fonctions sur une requête sans partitionnement (regroupement) :

* + - * pour appliquer un calcul sur le résultat de requête externe
      * pour appliquer un calcul sur le résultat de requête interne

**Cas d’usage :** sur une requête externe nombre d’essais marqués lors du tournoi

**Algèbre relationnelle**

1. R*essai* = σ[*genre*=”*essai*”](points)
2. P*nb*\_*essais* = Π(*count*(*genre*))(R*essai*)

**Requête SQL**

SELECT count(genre) FROM points

WHERE genre="essai";

**Cas d’usage :** sur une requête interne

le nom et le poids des joueurs les plus lourds du tournoi.

**Algèbre relationnelle**

1. P*poids*\_*max* = Π(*max*(*poids*))(joueurs)
2. R*j*\_*pmax* = σ(*poids*=*P*

*poids*\_*max*

)(v\_joueurs\_personnes)

1. P*nom*\_*pmax* = Π(*nom,poids*)(R*j*\_*pmax*)

**Requête SQL**

SELECT nom,poids

FROM v\_joueurs\_personnes

WHERE poids = (SELECT max(poids) FROM joueurs);

### Groupements

Les fonctions d’agrégats pourront également être appliquées sur les partitionnements (sous-ensembles) produits lors d’un regroupement (GROUP BY) sur des valeurs d’attributs, en appliquant éventuellement des restrictions (HAVING) sur les éléments des sous-ensembles.

Pour représenter les groupements en algèbre relationnelle, on s’appuiera sur la formalisation suivante :

(*a*1*,...an*)*,f* (*p*) [*having*(*p*)]

G (E)

Où :

* E : ensemble sur lequel portera le regroupement (les partitions de E)
* (a1, ...a*n*) : attributs de regroupement des éléments de E
* p : élément des partitions de E suivant les attributs de regroupement (p ∈ P(*a*1*,...an*)(E))
* f(p) : fonction d’agrégat à appliquer sur les éléments (p) des partitions de l’ensemble E
* having(p) : proposition logique pour faire des restrictions sur les éléments (p) des partitions

Cette formalisation correspondra à la requête SQL :

SELECT a1,...,an,f(p) FROM E

GROUP BY a1,...,an

HAVING having(p);

On peut également appliquer une restriction sur l’ensemble E avant de faire un groupement ce qui correspondrait à une requête SQL :

SELECT a1,...,an,f(p) FROM E

WHERE where(e) GROUP BY a1,...,an

HAVING having(p);

Où :

* + e : élément de l’ensemble E (e ∈ E)
  + where(e) : proposition logique pour faire des restrictions sur les éléments de l’ensemble E

A partir de cette formalisation des regroupements sur une requête nous pouvons représenter les cas d’usages

retenus pour notre projet.

**Cas d’usage :** regroupement (GROUP BY)

Récupérer, pour un match (le premier par exemple), le nombre d’essais,transformations,pénalités et drop marqués pour l’équipe jouant à domicile.

**Algèbre relationnelle**

1. R*m*1 = σ[*match*\_*id*=1](matchs)
2. J*locaux* =▷◁[*Rm*1*.locaux*\_*id*=*joueurs.equipe*\_*id*] (R*m*1, joueurs)
3. J*locaux*\_*pts* =▷◁[*Jlocaux.match*\_*id*=*points.match*\_*id*∧*Jlocaux.joueur*\_*id*=*points.joueur*\_*id*] (J*locaux*, points)
4. G*points* = G(*genre*)*,count*(*genre*)(J*locaux*\_*pts*)

[]

1. G*points* = ρ(*count*(*genre*)→*nb*\_*points*)(G*points*)

**Requête SQL**

SELECT p.genre,COUNT(p.genre) AS nb\_points FROM matchs m

INNER JOIN points p ON (m.match\_id=p.match\_id)

INNER JOIN joueurs j ON (p.joueur\_id=j.joueur\_id AND m.locaux\_id=j.equipe\_id) WHERE m.match\_id=1

GROUP BY p.genre;

Décomposition de la recherche :

1. R*m*1 : les informations sur le premier match
2. J*locaux* : les informations sur les joueurs de l’équipe qui joue à domicile lors du premier match
3. J*locaux*\_*pts* : les informations sur les joueurs qui ont marqué lors du premier match à domicile
4. G*points* : nombre d’essais,transformations,pénalités et drop marqués par l’équipe qui joue à domicile lors du premier match

Lorsque l’on fait des groupements on peut également faire des restrictions sur les sous-ensembles générés par ces regroupements en appliquant des fonction d’agrégat.

**Cas d’usage :** regroupement (GROUP BY) avec restriction (HAVING)

Récupérer, pour chaque match des équipes jouant à domicile, le nombre d’essais marqués lorsque ce nombre est au moins égal à quatre.

**Algèbre relationnelle**

* 1. R*essais* = σ[*genre*=”*essai*”](points)
  2. J*locaux* =▷◁[*v*\_*locaux.match*\_*id*=*Ressais.match*\_*id*] (v\_locaux, R*essais*)
  3. J*locaux*\_*joueurs* =▷◁[*Jlocaux.joueur*\_*id*=*joueurs.joueur*\_*id*∧*Jlocaux.locaux*=*joueurs.equipe*\_*id*] (J*locaux*, joueurs)
  4. G*points* = G(*match*\_*id*)*,count*(*genre*)(J*locaux*\_*joueurs*)

[*count*(*genre*)*>*3]

* 1. G*points* = ρ(*count*(*genre*)→*nb*\_*points*)(G*points*)

**Requête SQL**

SELECT v.match\_id, COUNT(p.genre) AS nb\_points FROM v\_locaux v

INNER JOIN points p ON (v.match\_id=p.match\_id)

INNER JOIN joueurs j ON (p.joueur\_id=j.joueur\_id AND v.locaux=j.equipe\_id) WHERE p.genre="essai"

GROUP BY v.match\_id

HAVING COUNT(p.genre) > 3;

# conclusion

Dans ce projet nous avons réussi à proposer un modéle de données du "Tournoi des 6 Nations" à partir d’un simple cahier des charges.

A partir de ce modèle de données nous avons créé une base de données relationnelles et établi des cas d’usages pour interroger la base à l’aide de requêtes SQL. Nous avons aussi mis en place des cas d’usages pour permettre à un utilisateur d’alimenter, modifier et récupérer les informations de la base de données à partir de fichier au format CSV.

Nous avons également mis en place des programmes d’application en python pour effectuer des traitements sur la base de données afin de calculer les scores de chaque équipe lors des matchs ainsi que les points de classements des équipes à partir des règles du rugby.

Pour aller plus loin, nous pourrions également faire évoluer le modèle de données pour pouvoir prendre en compte :

* les sanctions sur les joueurs (cartons jaunes,rouges ...)
* les blessures, protocoles commotion ...
* les erreurs d’arbitrage ...
* ...

Nous pourrions également mettre en place d’autres cas d’usages pour faire des statistiques sur :

* les points marqués par les joueurs
* les sélections de joueurs
* les sanctions, erreurs d’arbitrage
* ...

Si seulement nous avions eu plus de temps pour développer notre projet.

# Annexes

## Langage SQL

* + 1. **Création de tables** sixnations\_create.sql

Entité Personne DROP TABLE IF EXISTS personnes;

CREATE TABLE personnes ( personne\_id INTEGER PRIMARY KEY, nom VARCHAR(20),

prenom VARCHAR(20), naissance DATE, nationalite VARCHAR(20)

);

Entité Equipe

DROP TABLE IF EXISTS equipes;

CREATE TABLE equipes (

equipe\_id CHAR(3) PRIMARY KEY, nom VARCHAR(20),

pays VARCHAR(20),

coach\_id INTEGER,

FOREIGN KEY (coach\_id) REFERENCES personnes(personne\_id)

);

Entité Joueur

DROP TABLE IF EXISTS joueurs;

CREATE TABLE joueurs (

joueur\_id INTEGER PRIMARY KEY, poste VARCHAR(20),

taille SMALLINT, poids SMALLINT, equipe\_id CHAR(3),

FOREIGN KEY (joueur\_id) REFERENCES personnes(personne\_id), FOREIGN KEY (equipe\_id) REFERENCES equipes(equipe\_id)

);

------------ Vue : Informations sur l’identité des Joueurs ------- DROP VIEW IF EXISTS v\_joueurs\_personnes;

CREATE VIEW v\_joueurs\_personnes AS SELECT \*

FROM joueurs INNER JOIN personnes ON (joueur\_id=personne\_id);

------------ Vue : Informations sur les joueurs des equipes ------- DROP VIEW IF EXISTS v\_joueurs\_equipes;

CREATE VIEW v\_joueurs\_equipes AS

SELECT \* FROM joueurs j INNER JOIN equipes e ON (j.equipe\_id=e.equipe\_id);

Entité Match DROP TABLE IF EXISTS matchs;

CREATE TABLE matchs (

match\_id INTEGER PRIMARY KEY, journee CHAR(2),

locaux\_id CHAR(3), visiteurs\_id CHAR(3),

jour DATE DEFAULT current\_date, horaire TIME DEFAULT current\_time, locaux\_score INTEGER DEFAULT NULL,

visiteurs\_score INTEGER DEFAULT NULL,

FOREIGN KEY(locaux\_id) REFERENCES equipes(equipe\_id), FOREIGN KEY(visiteurs\_id) REFERENCES equipes(equipe\_id)

);

Entité Point

DROP TABLE IF EXISTS points;

CREATE TABLE points ( match\_id INTEGER,

horaire TIME DEFAULT current\_time, joueur\_id INTEGER,

genre VARCHAR(15) NOT NULL,

CHECK (genre IN (’essai’,’transformation’,’penalite’,’drop’)), PRIMARY KEY(match\_id,horaire),

FOREIGN KEY(match\_id) REFERENCES match(match\_id), FOREIGN KEY(joueur\_id) REFERENCES joueur(joueur\_id)

);

Entité Selection

DROP TABLE IF EXISTS selections; CREATE TABLE selections (

joueur\_id INTEGER, match\_id INTEGER,

PRIMARY KEY(joueur\_id,match\_id),

FOREIGN KEY(match\_id) REFERENCES matchs(match\_id), FOREIGN KEY(joueur\_id) REFERENCES joueurs(joueur\_id)

);

Entité Classement

DROP TABLE IF EXISTS classements; CREATE TABLE classements (

equipe\_id CHAR(3) PRIMARY KEY, points SMALLINT,

FOREIGN KEY(equipe\_id) REFERENCES equipes

);



CREATE VIEW

La création des vues se justifie lorsque qu’une requête est à réutiliser.

Comme nous avons fait le choix de séparer les informations sur l’identité d’une personne et les caractéris- tiques des joueurs, qui sont malgré tout des personnes comme les autres, lorsque nous ferons des requêtes nous aurons fréquemment l’occasion de récupérer l’identité des joueurs. La création d’une vue sur ces informations se justifie totalement.

De la même manière un joueur est nécessairement lié à une seule équipe et nous aurons fréquemment l’occasion de récupérer les informations sur les joueurs évoluant dans une équipe. On peut donc également créer une vue sur la requête de jointure entre les joueurs et les équipes dans lesquelles ils évoluent.

* + 1. **Insertion d’information** sixnations\_insert.sql Entité Personne : les coachs

INSERT INTO personnes (nom,prenom,naissance,nationalite) VALUES (’Galthié’,’Fabien’,’1969-03-20’,’France’); INSERT INTO personnes (nom,prenom,naissance,nationalite) VALUES (’Farrell’,’Andy’,’1969-03-20’,’Angleterre’);

-- Entité Personne : les joueurs --------------------------- INSERT INTO personnes (nom,prenom,naissance,nationalite) VALUES (’Dupont’,’Antoine’,’1996-11-15’,’France’);

INSERT INTO personnes (nom,prenom,naissance,nationalite) VALUES (’Sexton’,’Jonathan’,’1985-07-11’,’Irlande’); INSERT INTO personnes (nom,prenom,naissance,nationalite) VALUES (’Atonio’,’Uini’,’1990-03-26’,’Nouvelle-Zélande’);

Entité Equipe

INSERT INTO equipes VALUES (

’FRA’,’XV de France’,’France’,

(SELECT personne\_id FROM personnes WHERE nom=’Galthié’)

);

INSERT INTO equipes VALUES (

’IRL’,’XV du Trèfle’,’Irlande’,

(SELECT personne\_id FROM personnes WHERE nom=’Farrell’)

);

- Entité Joueur : de l’équipe de France ------------------ INSERT INTO joueurs VALUES (

(SELECT personne\_id FROM personnes WHERE nom=’Dupont’ AND prenom=’Antoine’), ’demi-mélée’,90,175,

(SELECT equipe\_id FROM equipes WHERE pays=’France’)

);

INSERT INTO joueurs VALUES (

(SELECT personne\_id FROM personnes WHERE nom=’Atonio’ AND prenom=’Uini’), ’pilier droit’,145,197,

(SELECT equipe\_id FROM equipes WHERE pays=’France’)

);

-- Entité Joueur : de l’équipe d’Irlande ------------------ INSERT INTO joueurs VALUES (

(SELECT personne\_id FROM personnes WHERE nom=’Sexton’ AND prenom=’Jonathan’), ’demi-ouverture’,91,188,

(SELECT equipe\_id FROM equipes WHERE pays=’Irlande’)

);

Entité Match : premiere journee

INSERT INTO matchs (journee,locaux\_id,visiteurs\_id,jour,horaire) VALUES ( ’J1’,

(SELECT equipe\_id FROM equipes WHERE pays=’France’), (SELECT equipe\_id FROM equipes WHERE pays=’Irlande’), ’2024-02-02’,

’21:00:00’

);

-- Entité Point : equipe de France, premiere journee -------------------- INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES (

(

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:15:20’, (

SELECT joueur\_id FROM v\_joueurs\_personnes

WHERE nom=’Dupont’ AND prenom=’Antoine’

),

’essai’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:16:20’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Atonio’ AND prenom=’Uini’

),

’essai’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:17:20’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND prenom=’Antoine’

),

’essai’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:18:20’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Atonio’ AND prenom=’Uini’

),

’essai’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:15:23’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND prenom=’Antoine’

),

’transformation’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

),

’17:15:24’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND prenom=’Antoine’

),

’penalite’

);

INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES ( (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’Fra’)

), ’17:15:25’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND prenom=’Antoine’

),

’drop’

);

-- Entité Point : equipe d’Irlande, premiere journee -------------------- INSERT INTO points (match\_id,horaire,joueur\_id,genre) VALUES (

(

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’IRL’ OR visiteurs\_id=’IRL’)

), ’17:32:59’, (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Sexton’ AND prenom=’Jonathan’

),

’drop’

);

-- Entité Selection : joueurs de l’equipe de France, premiere journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

INSERT INTO selections VALUES( (

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Atonio’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J1’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

-- Entité Selection : joueurs de l’equipe de France, deuxieme journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes

WHERE nom=’Dupont’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J2’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

-- Entité Selection : joueurs de l’equipe de France, troisieme journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J3’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

-- Entité Selection : joueurs de l’equipe de France, quatrieme journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J4’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

-- Entité Selection : joueurs de l’equipe de France, cinquieme journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Dupont’ AND equipe\_id=’FRA’

), (

SELECT match\_id FROM matchs

WHERE journee=’J5’ AND (locaux\_id=’FRA’ OR visiteurs\_id=’FRA’)

)

);

-- Entité Selection : joueurs de l’equipe d’Irlande lors de la premiere journee ----- INSERT INTO selections VALUES(

(

SELECT joueur\_id FROM v\_joueurs\_personnes WHERE nom=’Sexton’ AND equipe\_id=’IRL’

), (

SELECT match\_id FROM matchs

WHERE journee=’J5’ AND (locaux\_id=’IRL’ OR visiteurs\_id=’IRL’)

)

);

-- Entité Classement : le classement (non connu) des equipes --- INSERT INTO classements VALUES(’ANG’,NULL);

INSERT INTO classements VALUES(’ECO’,NULL); INSERT INTO classements VALUES(’FRA’,NULL); INSERT INTO classements VALUES(’GAL’,NULL); INSERT INTO classements VALUES(’IRL’,NULL);

INSERT INTO classements VALUES(’ITA’,NULL);