Gestion de salles de billard

Université de Nantes

Projet de Base de données 2

BIZOS PIERRE FOUILLET PIERRE THOMAS BRIAN ROUILLIER SIMON

2018

Table des matières

[I. Introduction 2](#_Toc505546409)

[II. Notre sujet 3](#_Toc505546410)

[III. Organisation 4](#_Toc505546411)

[IV. Table initiale : dictionnaire des données 5](#_Toc505546412)

[V. Normalisation 6](#_Toc505546413)

[1. Dépendances fonctionnelles 6](#_Toc505546414)

[2. 3ème Forme normal 6](#_Toc505546415)

[3. Forme normale de Boyce-Cood-Kent 7](#_Toc505546416)

[VI. Diagramme UML de la base de données 8](#_Toc505546417)

[VII. Conclusion 9](#_Toc505546418)

1. Introduction

Il nous est demandé de créer une base de données afin de pouvoir modéliser un sujet de notre choix. L’objectif est de définir la liste des données que l’on a besoin pour définir le sujet choisi. Il fois cette liste réalisée ou plutôt la table avec tous les attributs, il faudra la découper sous plusieurs algorithmes. Ce découpage permettra de créer et d’optimiser une base de données pour le sujet.

Ce document explique les choix que nous avons pris ainsi que la modélisation et les optimisations demandées sur la base de données.

1. Notre sujet

Le sujet concerne la gestion de réservation de tables de billards. Les joueurs doivent s’inscrire et donc renseigner des coordonnes personnelles. Pour réserver une partie, il faut que le client choisisse un lieu ou plutôt une salle de jeu comportant les tables de billards. On y trouve donc les coordonnées et de la salle ainsi que son gérant.

Une fois la salle choisit, le client doit pouvoir sélectionner le type de jeu qu’il souhaite jouer (billard français, américain …) et bien évidemment la date et l’heure qui coordonneront la partie. La réservation se termine par le paiement en ligne (ou à l’accueil avant le début de la partie). Par soucis de gestion, il devra renseigner diverses informations comme le nombre de personnes qui participent à la partie. Un système de fidélisation est également mis en place.

1. Organisation

Pour ce premier rendu nous avions plusieurs étapes à réaliser :

* Définir la table et le sujet
* Insérer des tuples dans la table de manière assez pertinentes
* Identifier les dépendances fonctionnelles
* Décomposer la table à l’aide des deux algorithmes de normalisation
* Puis tester les normalisations obtenues
* Réaliser un premier compte-rendu

Au niveau de l’organisation nous avons défini à quatre le sujet pour nous mettre d’accord sur un sujet qui nous donne tous l’envi de travailler et que l’on trouve intéressant tout en aillant aussi matière à pouvoir avoir un projet complet.

Puis Pierre F. s’est chargé d’insérer des tuples dans la table et de trouver des tuples intéressant à mettre afin de faciliter ensuit le travail. Brian a ensuite identifié les dépendances fonctionnelles à l’aide de la table des décisions voulu et de la table avec les tuples. Puis Pierre B. et Simon se sont partagé les normalisations et se sont eux qui ont donc tester leur normalisation. Puis nous avons tous travaillé sur le compte-rendu.

1. Table initiale : dictionnaire des données

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| idSalle | nomSalle | adrSalle | NomGerant | IdGerant | idTable | typeTable | typeJeu | dateRes | Heure  Deb  Res | heureFinRes | nomPersRes | idPersonne | adrMail  PersRes | coutRes | nb  PersRes | tarifHeure | Membre | Point | prenomPersRes |
| A1 | Millat | 2 Rue jean monnet | Dupont | G1 | T1 | Américain | Le jeu de la 8 | 3/12/17 | 15h | 16h | Thomas | P1 | thomas@gmail  .com | 12€ | 4 | 12€/h | Oui | 12 | Brian |
| A1 | Millat | 2 Rue jean monnet | Dupont | G1 | T1 | Américain | Le jeu de la 9 | 3/12/17 | 10h | 13h | Rouillier | P2 | rouillier@hotmai  .com | 36€ | 3 | 12e/h | Non | 0 | Simon |
| A1 | Millat | 2 Rue jean monnet | Dupont | G1 | T2 | Français | Français | 4/12/17 | 10h | 13h | Rouillier | P2 | rouillier@hotmai  .com | 36€ | 2 | 12e/h | Non | 0 | Simon |
| A2 | Moscou | 3 bd victor hugo | Michel | G2 | T3 | Snooker | Snooker | 1/12/17 | 14h | 16h | Dupont | P3 | dupont@gmail  .com | 36€ | 5 | 16€/h | Non | 0 | Michel |
| A2 | Moscou | 3 bd victor hugo | Michel | G2 | T4 | Anglais | 8 pool | 3/12/17 | 20h | 22h | Garros | P6 | Rollandgarros@gmail.com | 32€ | 2 | 16€/h | Non | 0 | Rolland |
| A2 | Moscou | 3 bd victor hugo | Michel | G2 | T5 | Français | Français | 2/12/17 | 16h | 18h | Desfrites | P4 | desfrites.yvan@yahoo.fr | 24€ | 1 | 12€/h | Oui | 79 | Yvan |
| A2 | Moscou | 3 bd victor hugo | Michel | G2 | T6 | Américain | Le jeu de la 8 | 2/12/17 | 17h | 18h | Null | P5 | pierrenull@laposte.net | 12€ | 2 | 12€/h | Non | 0 | Pierre |
| A3 | Pool Over | 6 bd du port | Leblanc | G3 | T7 | Snooker | Snooker | 2/12/17 | 19h | 22h | Desfrites | P4 | yvan.desfrites@yahoo.fr | 54€ | 3 | 18€/h | Oui | 79 | Yvan |
| A3 | Pool Over | 6 bd du port | Leblanc | G3 | T8 | Américain | Le 14/1 | 2/12/17 | 20h | 21h | Garros | P6 | Rollandgarros@gmail.com | 16€ | 4 | 16€/h | Non | 0 | Rolland |
| A3 | Pool Over | 6 bd du port | Leblanc | G3 | T9 | Français | Français | 2/12/17 | 15h | 16h | Desfrites | P4 | yvan.desfrites@yahoo.fr | 14€ | 2 | 14€/h | Oui | 79 | Yvan |

1. Normalisation

# Dépendances fonctionnelles

idSalle 🡪 nomSalle ,adrSalle, idGerant

idGerant 🡪 nomGerant

typeTable 🡪 tarifHeure

idTable🡪 typeTable

idPersonne 🡪 nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes

dateRes, heureDebRes, idPersonne 🡪 idSalle, idGerant, idTable, typeJeu, coutRes, heureFinRes, nbPersRes

heureDebRes,dateRes,idSalle,idTable🡪idPersonne, coutRes, nbPersRes, heureFinRes, typeJeu, idGerant

idSalle, heureDebRes, idPersonne, dateRes 🡪 idTable, coutRes, nbPersRes, typeJeu, idGerant, nomPersRes, adrMailPersRes, prenomPersRes, Membre, Point, nbRes, heureFinRes, nomSalle, adrSalle, nomGerant, idGerant

# 3ème Forme normal

Résumons les différentes formes normales :

* 1ère Forme normal : Atomique soit une seule valeur pour chaque dépendance
* 2ème Forme normal : Elle interdit qu’une partie de clé détermine un attribut non clé
* 3ème Forme normal : En plus des conditions précédentes, elle interdit en plus qu’un attribut non-clé donne un attribut non-clé

Voici la clé de notre projet : {dateRes, heureDebRes, idPersonne}

Pour décomposer en 3FN on va utiliser l’algorithme de Bernstein :

Tout d’abord on calcul la couverture minimale de DF :

On met d’abord DF sous forme élémentaire c’est-à-dire qu’un seul élément à droite de chaque DF

DF = {

idSalle 🡪 nomSalle ; idSalle 🡪 adrSalle ; idSalle 🡪 idGerant ;

idGerant 🡪 nomGerant ;

typeTable 🡪 tarifHeure ;

idTable 🡪 typeTable ;

idPersonne 🡪 nomPersRes ; idPersonne 🡪 prenomPersRes ; idPersonne 🡪 adrMailPersRes ; idPersonne 🡪 Membre ; idPersonne 🡪 Point ; idPersonne 🡪 nbRes

dateRes, heureDebRes, idPersonne 🡪 idSalle ;

dateRes, heureDebRes, idPersonne 🡪 idGerant ;

dateRes, heureDebRes, idPersonne 🡪 idTable ;

dateRes, heureDebRes, idPersonne 🡪 typeJeu ;

dateRes, heureDebRes, idPersonne 🡪 coutRes ;

dateRes, heureDebRes, idPersonne 🡪 heureFinRes ;

dateRes, heureDebRes, idPersonne 🡪 nbPersRes

heureDebRes, dateRes, idSalle, idTable 🡪 idPersonne ;

heureDebRes, dateRes, idSalle, idTable 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 HeureFinRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 TypeJeu ;

heureDebRes, dateRes, idSalle, idTable 🡪 idGerant ;

heureDebRes, dateRes, idSalle, idTable 🡪 coutRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 idTable ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 coutRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nbPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 typeJeu ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 idGerant ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nomPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 adrMailPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 prenomPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 Membre ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 Point ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nbRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 heureFinRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nomSalle ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 adrSalle ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 idGerant }

Ensuite on calcule la couverture minimale. Pour cela on cherche à éliminer les attributs superflus à gauche donc on a 3 groupes de dépendances fonctionnelles à regarder :

On va calculer la fermeture de dateRes :

dateRes+ = {dateRes}

Puis celle de heureDebRes, idPersonne :

heureDebRes+={heureDebRes} ;

idPersonne+={idPersonne, nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes}

Pour le premier groupe on a pas besoin de retirer d’élément à gauche.

Pour le deuxième groupe il nous reste qu’à calculer la fermeture d’idSalle et idTable :

idSalle+ = {idSalle, nomSalle ,adrSalle, idGerant}

idTable+ = {idTable, typeTable, tarifHeure}

Ici on peut retirer la DF heureDebRes, dateRes, idSalle, idTable🡪idGerant car idGerant n’est déterminer que par idSalle.

On a calculé la fermeture de tous les attributs de gauche du dernier groupe de DF et on remarque ici que l’on peut retirer :

idSalle, heureDebRes, idPersonne, dateRes 🡪 nomSalle ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 adrSalle ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 idGerant ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nomPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 adrMailPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 prenomPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 Membre ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 Point ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nbRes ;

On obtient :

DF = {

idSalle 🡪 nomSalle ; idSalle 🡪 adrSalle ; idSalle 🡪 idGerant ;

idGerant 🡪 nomGerant ;

typeTable 🡪 tarifHeure ;

idTable 🡪 typeTable ;

idPersonne 🡪 nomPersRes ; idPersonne 🡪 prenomPersRes ; idPersonne 🡪 adrMailPersRes ; idPersonne 🡪 Membre ; idPersonne 🡪 Point ; idPersonne 🡪 nbRes

dateRes, heureDebRes, idPersonne 🡪 idSalle ;

dateRes, heureDebRes, idPersonne 🡪 idGerant ;

dateRes, heureDebRes, idPersonne 🡪 idTable ;

dateRes, heureDebRes, idPersonne 🡪 typeJeu ;

dateRes, heureDebRes, idPersonne 🡪 coutRes ;

dateRes, heureDebRes, idPersonne 🡪 heureFinRes ;

dateRes, heureDebRes, idPersonne 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 idPersonne ;

heureDebRes, dateRes, idSalle, idTable 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 HeureFinRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 TypeJeu ;

heureDebRes, dateRes, idSalle, idTable 🡪 coutRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 idTable ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 coutRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nbPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 typeJeu ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 heureFinRes ;}

Maintenant on va retirer les DF qui peuvent être obtenu par transitivité :

On fait la fermeture transitive du côté gauche sans les DF qui ont pour côté gauche : « dateRes, heureDebRes, idSalle, idTable » soit les DF suivantes :

heureDebRes, dateRes, idSalle, idTable 🡪 idPersonne ;

heureDebRes, dateRes, idSalle, idTable 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 heureFinRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 typeJeu ;

heureDebRes, dateRes, idSalle, idTable 🡪 coutRes ;

On calcul la fermeture de {heureDebRes, dateRes, idSalle, idTable}+ = {heureDebRes, dateRes, idSalle, idTable, nomSalle, adrSalle, idGerant, nomGerant, typeTable, tarifHeure}

On ne peut supprimer aucune des DF du dessus car il n’y a pas de dépendances entre idPersonne, nbPersRes, heureFinRes, typeJeu et coutRes.

Faisons de même avec le groupe suivant :

dateRes, heureDebRes, idPersonne 🡪 idSalle ;

dateRes, heureDebRes, idPersonne 🡪 idGerant ;

dateRes, heureDebRes, idPersonne 🡪 idTable ;

dateRes, heureDebRes, idPersonne 🡪 typeJeu ;

dateRes, heureDebRes, idPersonne 🡪 coutRes ;

dateRes, heureDebRes, idPersonne 🡪 heureFinRes ;

dateRes, heureDebRes, idPersonne 🡪 nbPersRes ;

On remarque que idSalle 🡪 idGerant donc on peut supprimer la dépendance

dateRes, heureDebRes, idPersonne 🡪 idGerant

On obtient :

{dateRes, heureDebRes, idPersonne, nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes}

Nous avons ensuite le dernier groupe :

idSalle, heureDebRes, idPersonne, dateRes 🡪 idTable ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 coutRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 nbPersRes ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 typeJeu ;

idSalle, heureDebRes, idPersonne, dateRes 🡪 heureFinRes

Il n’y aucune dépendance directe entre : idTable, coutRes, nbPersRes, typeJeu, heureFinRes.

On n’a même pas besoin de la fermeture car la clé est un sous-groupe de la partie gauche de ces dépendances fonctionnelles. Donc on peut supprimer ce dernier groupe.

On a donc CV(DF)={

idSalle 🡪 nomSalle ; idSalle 🡪 adrSalle ; idSalle 🡪 idGerant ;

idGerant 🡪 nomGerant ;

typeTable 🡪 tarifHeure ;

idTable 🡪 typeTable ;

idPersonne 🡪 nomPersRes ; idPersonne 🡪 prenomPersRes ; idPersonne 🡪 adrMailPersRes ; idPersonne 🡪 Membre ; idPersonne 🡪 Point ; idPersonne 🡪 nbRes

dateRes, heureDebRes, idPersonne 🡪 idSalle ;

dateRes, heureDebRes, idPersonne 🡪 idTable ;

dateRes, heureDebRes, idPersonne 🡪 typeJeu ;

dateRes, heureDebRes, idPersonne 🡪 coutRes ;

dateRes, heureDebRes, idPersonne 🡪 heureFinRes ;

dateRes, heureDebRes, idPersonne 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 idPersonne ;

heureDebRes, dateRes, idSalle, idTable 🡪 nbPersRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 heureFinRes ;

heureDebRes, dateRes, idSalle, idTable 🡪 typeJeu ;

heureDebRes, dateRes, idSalle, idTable 🡪 coutRes ;

}

On peut donc dégager sept nouvelles tables en regroupant les dépendances fonctionnelles en fonction des attributs de gauche :

R1(idSalle, nomSalle, adrSalle, idGerant) muni de DF1 = {idSalle 🡪 nomSalle, adrSalle, idGerant}

R2(idGerant, nomGerant) muni de DF2 = {idGerant 🡪 nomGerant}

R3(typeTable, tarifHeure) muni de DF3 = {typeTable 🡪 tarifHeure}

R4(idTable, typeTable) muni de DF4 = {idTable 🡪 typeTable}

R5(idPersonne, nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes) muni de

DF5 = {idPersonne 🡪 nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes}

R6(dateRes, heureDebRes, idPersonne, idSalle, idTable, typeJeu, coutRes, heureFinRes, nbPersRes)

Muni de DF6 = {dateRes, heureDebRes, idPersonne 🡪 idSalle, idTable, typeJeu, coutRes, heureFinRes, nbPersRes}

R7(heureDebRes, dateRes, idSalle, idTable, idPersonne, nbPersRes, heureFinRes, typeJeu, coutRes) muni de DF7 = {heureDebRes, dateRes, idSalle, idTable 🡪 idPersonne, nbPersRes, heureFinRes, typeJeu, coutRes}

La table R6 contient la clé de notre table globale donc nous n’avons pas besoin de rajouter une table

# Forme normale de Boyce-Cood par décomposition

En plus des conditions des autres formes normales, la FNBC interdit qu’une partie de clé donne une autre partie de clé et qu’une non-clé donne une partie de clé.

R avec les DF trouvées dans la partie 1

R1(idSalle, nomSalle, adreSalle, idGerant) avec DF { idSalle 🡪 nomSalle ,adrSalle, idGerant}

R2(idGerant, nomGerant) avec DF {idGerant 🡪 nomGerant}

R3(typeTable, tarifHeure) avec DF { typeTable 🡪 tarifHeure}

R4(idPersonne, nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes avec DF { idPersonne 🡪 nomPersRes, prenomPersRes, adrMailPersRes, Membre, Point, nbRes}

R5(dateRes, heureDebRes, idPersonne, idSalle, idGerant, idTable, typeJeu, coutRes, heureFinRes, nbPersRes) avec DF {dateRes, heureDebRes, idPersonne 🡪 idSalle, idGerant, idTable, typeJeu, coutRes, heureFinRes, nbPersRes}

R6(heureDebRes, dateRes, idSalle, idTable, idPersonne, coutRes, nbPersRes, heureFinRes, typeJeu, idGerant) avec DF { heureDebRes,dateRes,idSalle,idTable 🡪 idPersonne, coutRes, nbPersRes, heureFinRes, typeJeu, idGerant}

R7(idSalle, heureDebRes, idPersonne, dateRes, idTable, coutRes, nbPersRes, typeJeu, idGerant, nomPersRes, adrMailPersRes, prenomPersRes, Membre, Point, nbRes, heureFinRes, nomSalle, adrSalle, nomGerant, idGerant) avec DF { idSalle, heureDebRes, idPersonne, dateRes 🡪 idTable, coutRes, nbPersRes, typeJeu, idGerant, nomPersRes, adrMailPersRes, prenomPersRes, Membre, Point, nbRes, heureFinRes, nomSalle, adrSalle, nomGerant, idGerant}

1. Diagramme UML de la base de données

Pour cela nous avons utilisé le logiciel JMerise.

1. Conclusion

Pour l’instant il ne s’agit que d’une proposition de sujet. Le document nous permet d’appliquer les algorithmes vus en cours.