Projet Bases de Données - Documentation

SIALA Rafik, JEDRA Yassir, TRIMECH Abdellatif, MÉLOUX Maxime

April 2016

1 Introduction

2 Analyse statique

2.1 Propriétés

Déterminées lors de la lecture du sujet.
{id_Joueur, Nom, Prénom, Adresse, Naissance,
id_Partie, Niveau, Couleur, id_J_Blanc, id_J_Noir, Vainqueur,
id_Pièce, Roi, Reine, Tour, Cavalier, Fou, Pion, Couleur, Ligne, Colonne,
id_Coup, O_Ligne, N_Ligne, O_Colonne, N_Colonne }

2.2 Contraintes

a) Dépendances fonctionnelles:
id_Joueur → Nom, Prénom, Adresse, Naissance
id_Partie, Niveau, couleur → id_Joueur
id_Partie, Niveau → id_J_Blanc, id_J_Noir, Vainqueur
id_Partie, Niveau, id_Pièce → Couleur, Ligne, Colonne
id_Partie, Niveau, id_Coup → O_Ligne, N_Ligne, O_Colonne, N_Colonne

b) Contraintes de Valeur:
Ext(id_Partie) $\subseteq$ {1..16}
Ext(Niveau) $\subseteq$ {1..5}
Ext(id_J_Blanc) $\subseteq$ Ext(id_Joueur)
Ext(id_J_Noir) $\subseteq$ Ext(id_Joueur)
Ext(Vainqueur) $\subseteq$ Ext(id_Joueur)
id_Piece $\subseteq$ {1..32}
Ligne, O_Ligne, N_Ligne $\in$ {1..8}
Colonne, O_Colonne, N_Colonne $\in \{A..H\}$

c) Contraintes de multiplicités:
Une partie se joue entre deux joueurs.
Dans une partie il y a un vainqueur.
Dans une partie, un joueur dispose de 16 pièces.
Dans une partie, un joueur dispose de 8 pions.
Dans une partie, un joueur dispose de 2 tours.
Dans une partie, un joueur dispose de 2 cavaliers.
Dans une partie, un joueur dispose de 2 fous.
Dans une partie, un joueur dispose de 1 reine.
Dans une partie, un joueur dispose de 1 roi.
Dans une partie, plusieurs coups peuvent être jouées.

d) Autres contraintes:
Les joueurs jouants dans la même partie doivent avoir des couleurs différentes.

Le vainqueur est l'un des deux joueurs d'une partie.

3 Passage au relationnel

Partie((Niveau, NumPartie){*pk*}**, NumVainqueur)**
- Partie est une entité simple. Possède un et un seul vainqueur(cf. shèma entité association cardinalité 1..1).

Joueur(NumJoueur{*pk*}**, Nom, Prénom, Adresse, Naissance)**
- Joueur est une entité simple.

Pièce((Niveau, NumPartie ,NumPièce){*pk*}**, estSortie, Type, Ligne, Colonne, couleur)**
- Pièce est une entité faible de partie. Elle possède des sous-types, une position et une couleur (cf. shéma entité association cardinalité 1..1).

Coup((Niveau, NumPartie, NumCoup){*pk*}**, LigneDep, ColonneDep, LigneArr, ColonneArr)**
- Coup est une entité faible de partie. Elle est définie par deux positions (cd. shèma entité association cardinalité 1..1 et 2..2).

Participe((Niveau, NumPartie, NumJoueur, Couleur){*pk*}**)**
- Participe est une association ternaire entre partie, joueur et couleur.

Couleur(couleur {*pk*} **)**
- couleur est une entité simple.

On n'a pas rajouté la table position car on peut s'en passer sans pour autant affecter la cohérence de la base.

4 Forme normale

id_Joueur → Nom, Prénom, Adresse, Naissance
id_Partie, Niveau, couleur → id_Joueur
id_Partie, Niveau → id_J_Blanc, id_J_Noir, Vainqueur
id_Partie, Niveau, id_Pièce → Couleur, Ligne, Colonne
id_Partie, Niveau, id_Coup → O_Ligne, N_Ligne, O_Colonne, N_Colonne

Partie((Niveau, NumPartie){*pk*}, NumVainqueur)

- 1FN: OK,
- 2FN: OK, car FN1 et numVainqueur dépend pleinement de Niveau et NumPartie
- 3FN: OK, car FN2 et numVainqueur dépend directement de Niveau et NumPartie

Joueur(NumJoueur{*pk*}, Nom, Prénom, Adresse, Naissance)

- 1FN: OK,
- 2FN: OK, car FN1 et car tous les attributs non clés dependent pleinement de NumJoueur
- 3FN: OK, car FN2 et tous les attributs non clés dépendent directement de NumJoueur

Pièce((Niveau, NumPartie ,NumPièce){*pk*}, estSortie, Type, Ligne, Colonne, couleur)

- 1FN : OK,

- 2FN : OK, car FN1 et tous les attributs non clés dependent pleinement de NumPartie, Niveau et NumPièce
- 3FN : OK, car FN2 et tous les attributs non clés dependent directement de NumPartie, Niveau et NumPièce

Coup((Niveau, NumPartie, NumCoup){pk}, LigneDep, ColonneDep, LigneArr, ColonneArr)

- 1FN : OK,
- 2FN : OK, car tous les attributs non clés dépendent pleinement des attributs clés
- 3FN : OK, car tous les attributs non clés dépendent directement des attributs clés

Participe((Niveau, NumPartie, NumJoueur, Couleur){pk})

- 1FN : OK,
- 2FN : OK, car pas d'attributs non clés
- 3FN : OK, car pas d'attributs non clés

Couleur(couleur{pk})

- 1FN : OK,
- 2FN : OK, car pas d'attributs non clés
- 3FN : OK, car pas d'attributs non clés

5 Bilan du projet - Difficultés rencontrées

Les vérifications effectuées dans `movePiece` ont pris beaucoup de temps mais étaient assez faciles à implémenter.
Nous avons également eu des soucis avec la base de données Oracle (problèmes d'accès et de verrouillage fréquents).
Une partie longue et difficile à faire était le débuggage des vérifications (ordre des paramètres, cas limites...).
Un problème remarqué tardivement nous a contraint à modifier légèrement la structure de la base de données.

En revanche, on note que les requêtes SQL ont été très faciles à implémenter, et avec très peu d'erreurs; en effet, la structure sous-jacente était bien conçue de manière à faciliter l'implémentation des fonctionnalités.

6 Mode d'emploi

Le script `exec.sql` initialise la base de données; elle supprime les tables si celles-ci existent déjà, après quoi elle les recrée et initialise les tables `Couleur` et `Joueur`. On peut notamment modifier le script `initData.sql` afin de changer le nombre de joueurs automatiquement ajoutés à la table, les autres pouvant être ajoutés par le biais du programme.

Le programme en lui-même est assez simple d'utilisation. Une boucle principale permet de sélectionner différentes options d'un menu. Attention, cependant, aucun changement n'est sauvegardé dans la base de données à moins de choisir l'option `Sauvegarder` ou `Sauvegarder et Quitter` du menu. Ce choix

a été fait pour faciliter le déboguage du programme, ainsi que pour pouvoir effectuer des démonstrations et tests sans nécessairement devoir réinitialiser la base de données en permanence.

Seules les contraintes assez simples (types et intervalles) sont implémentées en SQL, le reste d'entre elles étant vérifiées en Java. Les fonctionnalités du programme sont les suivantes :

- Enregistrer un joueur : Demande à l'utilisateur les nom, prénom, adresse et date de naissance du joueur à ajouter. On vérifie que ces champs sont tous valides, que le joueur n'existe pas déjà et que le tournoi n'est pas déjà plein avant d'effectuer l'insertion.
- Créer une nouvelle partie : Demande à l'utilisateur les ids des joueurs, le niveau de la partie et son numéro. On vérifie ici que tous les joueurs ont bien rejoint le tournoi, que les deux joueurs sont différents, qu'ils ont bien gagné toutes leurs éventuelles parties précédentes, qu'ils sont de même niveau dans le tournoi (même nombre de matchs effectués), et que la partie possède bien le prochain numéro assigné dans le niveau courant. On crée ensuite la partie et les pièces, qu'on place à leurs positions initiales.
- Afficher une partie en cours : On demande à l'utilisateur un niveau et numéro de partie. On vérifie que la partie existe, puis on affiche la position de toutes les pièces.
- Jouer un mouvement : On demande à l'utilisateur un niveau et numéro de partie, un id de pièce, ainsi qu'un numéro de ligne et de colonne. Les vérifications effectuées sont les suivantes : Existence de la partie, tour du bon joueur, partie non encore terminée, mouvement valide, pièce toujours sur le plateau, cases intermédiaires vides pour la tour, le fou ou la reine, arrivée sur une case vide ou occupée par une pièce adverse. On met ensuite à jour la position de la pièce, ainsi que le vainqueur de la partie et la pièce prise si nécessaire.