

Base de données

Projet : développement d'un système de
réservation

Maxime Fabre – Florian Lepot

5/17/2013



Sommaire

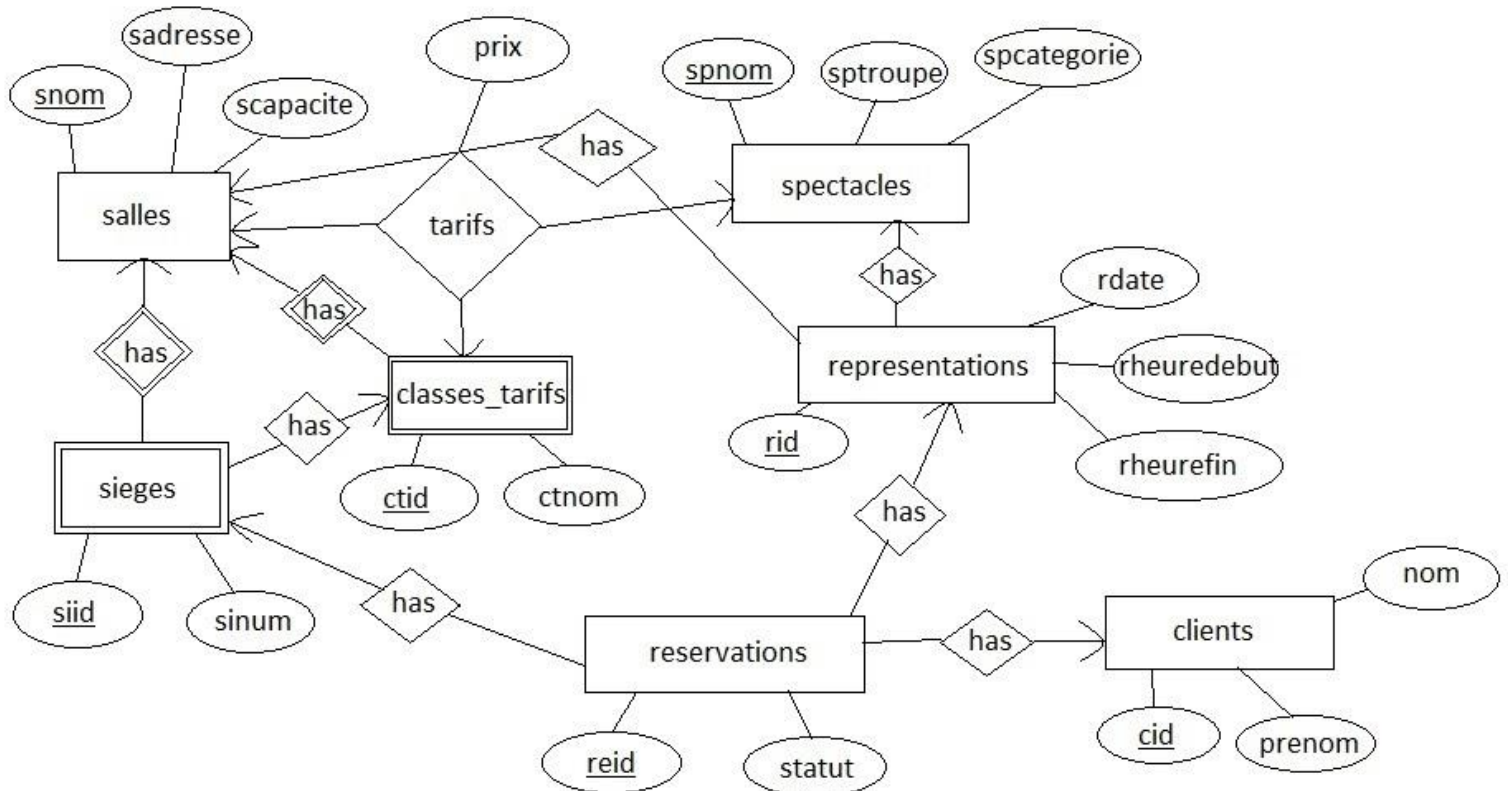
Introduction.....	3
I. Le modèle E /A de la base de données.....	4
II. Le modèle relationnel de la base de données.....	5
III. Liste des contraintes d'intégrités	6
IV. Implémentation des stratégies de consultations	7
V. Description des mesures de performances effectuées.....	8
Résultats	8
Stable.....	8
Instable	8
Synthèse	9
Conclusion	10

Introduction

Pour ce premier projet de base de données, nous devions réaliser une application complète de réservations de places de spectacles. Compte tenu du cahier des charges, nous n'avons pas créé d'interface graphique, et nous sommes focalisés au maximum sur la conception du modèle de notre base de données.

Ce projet nous a beaucoup appris sur la manière à réfléchir et visualiser la structure d'une base de données avant de se lancer dans le code. Bien faire la séparation entre la base de données et la programme java, permet de créer une application la plus optimisée possible. L'enjeu était bien évidemment de finir de se familiariser avec les bases de données, les systèmes de gestion de transactions et de réservations.

I. Le modèle E /A de la base de données



II. Le modèle relationnel de la base de données

salles (snom, sadresse, scapacite)

sieges (siid, #snom, #ctid, sinum)

classes_tarifs (ctid, #snom, ctnom)

tarifs (#snom, #ctid, #spnom, prix)

spectacles (spnom, sptroupe, spcategorie)

representations (rid, #spnom, #snom, rdate, rheuredebut, rheurefin)

reservations (reid, #siid, #rid, #cid, statut)

clients (cid, prenom, nom)

III. Liste des contraintes d'intégrités

Après avoir fini la conception de notre base de données, nous nous sommes attaqués aux contraintes d'intégrités afin d'optimiser au maximum celle-ci. Le script **contraintes.sql** contient l'ensemble des contraintes d'intégrités du projet.

On trouve dans ce fichier les contraintes de type check et trigger. Comme indiqué dans le sujet les contraintes de clé primaire ou secondaire

Les contraintes de valeur par défauts et de validation obligatoire

Les contraintes de validation obligatoire vont concerner toutes les données qui ne pourront pas prendre de valeur null, c'est le cas par exemple de l'adresse d'une salle, ou de l'heure d'une représentation. Bien que ces champs ne soient pas des clés primaires, il est primordial que ces valeurs soient initialisées.

En ce qui concerne les contraintes par défauts, on a effectué trois requêtes :

- La capacité de sièges d'une salle doit être supérieure à 0
- Les numéros des sièges doivent être supérieurs 0
- Le prix d'une place doit être supérieur à 0

Pour ce type de contraintes, un simple CHECK aurait pu suffire dans createbase.sql, cependant par soucis de ne pas tout regrouper dans le même fichier (createbase.sql), les ALTER TABLE convenaient parfaitement.

Triggers

Les autres contraintes, plus compliquées mais toutes aussi importantes ont été créées à partir de triggers. Nous avons respecté les trois triggers énoncés dans le sujet :

- Les réservations d'un spectacle ne dépassent pas la capacité de la salle
- Deux spectacles ne peuvent pas se jouer au même endroit et au même moment

La dernière contrainte « *La classe de tarif d'une place doit être une de celles proposées par la salle correspondante* » est déjà satisfaite grâce au schéma de notre base de données.

Nous avons toutefois rajouté d'autres contraintes afin d'améliorer au maximum la base de données et d'assurer la fiabilité des données.

- Deux salles ne peuvent pas posséder la même adresse
- Deux sièges d'une même salle ne peuvent pas avoir le même numéro
- Un nouveau siège ne peut pas avoir un numéro supérieur à la capacité de la salle
- Une salle ne possède pas deux fois la même classe tarifaire

Pour finir nous avons ajouté une contrainte d'intégrité référentielle sous forme de trigger.

- Quand un spectacle est supprimé toutes les représentations de ce spectacle sont supprimées

Dans une optique d'optimisation, les triggers sont beaucoup plus précis et permettent un contrôle parfait de la contrainte. Ils sont plus larges qu'une simple vérification malgré la perte de simplicité dans l'expression.

IV. Implémentation des stratégies de consultations

Comme cela était demandé dans le sujet, nous avons développé deux stratégies de consultation et réservation de places :

- **Stable** : Si le client consulte les places, celles-ci sont préservées et ne sont plus accessibles pour les autres utilisateurs jusqu'à la réservation (ou l'annulation).
- **Instable** : Les places sont libres jusqu'à la réservation. Il peut donc arriver que des places affichées libres lors de la consultation ne le soient plus au moment de la réservation.

Pour développer la stratégie « stable », nous avons ajouté un champ statut sur les réservations. Il a 2 valeurs possibles : 0 si la place est dans l'état de « pré-réservation » et 1 si la réservation est totalement terminée.

Lorsque l'utilisateur choisit 2 places, ces 2 places sont « pré-réservées », le statut passe donc à 0. Une fois qu'il finit la réservation et la confirme, le statut est passé à 1 pour les places qu'il a réservées.

Cela permet que les utilisateurs qui voudront accéder à la liste des places ne voient pas les places que l'autre utilisateur a choisies et est en train de réserver. Ils ne peuvent donc pas choisir les mêmes et obtenir une erreur au moment de la réservation.

Le principal inconvénient de cette méthode apparaît lorsqu'il ne reste plus beaucoup de places disponibles. En effet, si un utilisateur consulte les places restantes et souhaite les réserver, cela va les rendre inaccessibles aux autres. Mais si l'utilisateur change d'avis et ne confirme finalement pas sa réservation, les visiteurs ne verront pas ces places durant un certain laps de temps et risquent d'aller voir ailleurs.

Nous avons par ailleurs mis en place les transactions pour les réservations. Ainsi il est impossible que seules certaines places ne soient réservées sur celles demandées. Nous l'avons implémenté dans notre code Java et non dans la base de données.

V. Description des mesures de performances effectuées

Pour commencer, nous avons développé la partie programme de test. Il est composé à la fois du fichier Main.java, lequel utilisera un Thread par client de test, et d'un fichier Tests.java qui s'exécutera dans ce Thread et qui simulera la réservation par un client.

Nous avons également ajouté un fichier Stats.java qui nous servira à stocker les statistiques des différents clients pour pouvoir les interpréter par la suite.

Pour les tests, nous avons décidé de les organiser comme ceci :

- Seulement 2 spectacles disponibles
- 4 séries de tests par stratégie
- Chaque série de tests est composée d'une simulation de 5 clients se connectant en même temps au système de réservations.
- Nous faisons la moyenne de ces séries de tests et nous nous basons sur ce résultat pour notre interprétation.

Pour nos mesures de performances, nous avons étudié 2 variables :

- Le temps moyen pour que la réservation soit terminée,
- Le nombre d'échecs que le client rencontre le long de sa réservation.

Un point qu'il est utile de préciser : Lorsque l'utilisateur consulte les places, nous ne bloquons aucune place, il va ensuite en choisir 2, et ces 2 là seulement seront dites « pré-réservées ». Les autres utilisateurs peuvent en conséquence toujours accéder aux places libres sans être bloqués parce-que quelqu'un est en train de réserver. Les 2 places seront totalement réservées quand l'utilisateur confirmera sa réservation.

Résultats

Stable

Série	Temps moyen (en sec)	Erreurs
1	39,6	0
2	50,4	0,2
3	44,6	0,2
4	52,8	0,4

Instable

Série	Temps moyen (en sec)	Erreurs
1	46,0	0,2
2	39,2	0,0
3	50,8	0,4
4	68,0	0,8

Synthèse

Méthode	Temps moyen	Erreurs moyennes
Stable	46,85	0,20
Instable	51,00	0,35

On peut voir que comme prévu, plus nombreuses sont les places réservées, plus le temps moyen pour réserver est long et plus le nombre d'erreurs augmente.

C'est d'autant plus flagrant sur la méthode instable où on arrive presque à 1 erreur par utilisateur pour la 4eme série de tests.

Concernant le temps moyen, il est difficile d'en tirer des conclusions, cependant on peut voir qu'en moyenne le temps est plus élevé pour la méthode instable que pour la méthode stable.

En considérant que plus l'utilisateur attend et se retrouve face à des erreurs, moins il est satisfait, on peut tirer la conclusion évidente que la méthode stable reste la plus efficace et est la méthode à utiliser en priorité.

Conclusion

Pour finir sur ce projet, nous avons essayé de respecter au maximum le cahier des charges. En plus d'avoir réalisé nos objectifs de départ nous avons implémenté d'autres contraintes d'intégrités.

Nous avons aussi beaucoup appris quant à l'utilisation des différentes bases de données (Oracle pour ce projet), la liaison entre la base de données et java, et cela nous a montré qu'une base de données bien pensée offre d'énormes possibilités.

Ce projet n'a pas été des plus simples, en effet habitués à la base de données MySQL, nous avons eu quelques difficultés à passer sur Oracle.