

Base de Données

Projet : Développement d'un système de réservation

Ce projet doit être réalisé en binôme, i.e. au plus deux élèves, et remis sur Campus.

Sujet

Il s'agit de développer en Java une application de réservation de places de spectacle s'appuyant sur une base de données Oracle, et de mesurer ses performances dans différents cas de figure.

Les étapes du travail sont les suivantes.

1- Modélisation de la base de données

Une salle de spectacle est identifiée par son nom et caractérisée par son adresse et sa capacité en nombre de places. Chaque salle propose plusieurs classes de tarif, qui sont fonction de sa configuration. Par exemple, un théâtre offrira les classes de tarif « orchestre », « 1^{er} balcon », « 2nd balcon » alors qu'une salle des fêtes ne proposera qu'une classe « tarif unique ». Chaque place d'une salle est identifiée par un numéro relatif à la salle. Elle est associée à une unique classe de tarif, indépendamment du spectacle joué.

Un spectacle est identifié par son nom et caractérisé par la troupe qui le joue ainsi que sa catégorie (« musique », « danse », « humour », etc.). Il peut se jouer dans une ou plusieurs salles, mais sur des plages de temps distinctes. Chaque spectacle donne habituellement lieu à plusieurs représentations, chacune définie par une date et une heure. Pour chaque salle dans laquelle il se joue, un spectacle définit un prix pour chaque classe de tarif de la salle.

Un client effectuant une réservation est identifié par son nom. Lorsqu'il obtient une réservation, celle-ci mentionne le nom de la salle, le nom du spectacle, la date et heure de la représentation, le numéro de place dans la salle et son tarif. Pour pouvoir effectuer des contrôles, on mémorise dans la base le nom du client qui a réservé une place donnée.

En utilisant le modèle Entités/Associations, établissez le schéma conceptuel de la base de données selon le formalisme vu en cours et TD. Vous pouvez utiliser l'outil que vous voulez pour réaliser le schéma. Un fichier d'aide PowerPoint est fourni sur Campus : il définit les briques de base d'un schéma E/A.

2- Création de la base de données relationnelle

Traduisez le modèle-schéma conceptuel de la base en modèle-schéma relationnel. Si votre schéma utilise des identifiants numériques autres que ceux définis dans la description ci-dessus, indiquez dans le rapport :

- qui les génère et comment
- comment est assurée l'unicité des spectacles, salles, etc. dans la base
- l'utilité de ces identifiants numériques

Écrivez un script SQL `creabase.sql` permettant de créer ou recréer la base de données et la peupler. Le script doit définir les contraintes de clé primaire et étrangère nécessaires. Les données insérées dans la base doivent être suffisantes pour réaliser des tests de performance pertinents (voir la section 56).

3- Étude de la forme normale

On considère la table de votre base de données qui comprend le plus grand nombre d'attributs et qui ne dérive pas d'une association du schéma E/A.

Dans le rapport, déterminez les dépendances fonctionnelles et les clés de cette table comme vu en cours et en TD. Indiquez ensuite si la table est en forme normale de Boyce-Codd, en expliquant votre conclusion.

Si la table n'est pas dans la forme de Boyce-Codd, décomposez-la en tables qui sont dans cette forme. Note : il n'est pas nécessaire de modifier le schéma défini dans `creabase.sql`.

4- Définition des contraintes d'intégrité

Définissez toutes les contraintes d'intégrité des données de la base. Par exemple : le nombre de places réservées ne peut excéder la capacité d'une salle, la classe de tarif d'une place doit être une de celles proposées par la salle correspondante, un spectacle ne peut être joué simultanément dans deux salles, etc. Ces contraintes peuvent être satisfaites implicitement par le schéma de votre base, ou devoir être définies explicitement au moyen de contraintes d'attribut, de table ou bien de déclencheurs.

Écrivez un script SQL `contraintes.sql` permettant d'ajouter *a posteriori* ces contraintes au schéma de la base défini par le script `creabase.sql`. (Les contraintes ne doivent pas être définies dans le script `creabase.sql` pour éviter qu'elles gênent le bon fonctionnement de l'application.)

5- Développement de l'application

L'application, développée en Java, permet à un client de réserver une ou plusieurs places pour une représentation donnée. Elle offre les fonctionnalités suivantes :

- lister les représentations d'un spectacle situées entre deux dates, en donnant pour chacune des représentations les différents tarifs définis,
- lister les places d'une représentation, en donnant pour chacune son numéro, son statut (libre/réservé) et son tarif,
- réserver une ou plusieurs places pour une représentation, en spécifiant soit leur numéros, soit leur tarif.

Pour effectuer sa réservation, un client enchaîne typiquement les opérations suivantes. Il consulte d'abord la liste des représentations d'un spectacle qu'il souhaite voir, puis en choisit une. Ensuite, il peut soit réserver des places à un certain tarif si son placement exact dans la salle lui est indifférent, soit consulter la liste des places et choisir celles qu'il préfère (par exemple devant et au centre¹, ou près de places réservées par des amis, etc.). On peut aussi imaginer qu'un client qui n'est pas satisfait des places proposées pour une représentation fasse des recherches sur une ou plusieurs autres représentations.

L'application doit bien entendu prendre en charge le fait que plusieurs clients peuvent effectuer des consultations et réservations de façon concurrente. En particulier, une même place ne doit pas être attribuée à plusieurs clients.

On suppose que les salles, spectacles et représentations sont préalablement créés dans la base par l'administrateur (l'équivalent du script `creabase.sql`) et que ces informations ne sont pas modifiées pendant que les clients effectuent leurs consultations et réservations.

Le détail des interfaces de l'application est donné dans le squelette des classes Java fournies avec le projet. Complétez l'implémentation de ces classes, **sans modifier les interfaces définies**. Il n'est pas demandé de développer une interface graphique ou ligne de commande qui sollicite l'application. C'est le programme de test de performances décrit section suivante qui jouera ce rôle.

6- Mesures de performances

L'application peut implémenter deux stratégies concernant l'enchaînement consultation – réservation des places par un client. Une stratégie consiste à figer le statut libre/réservé des places entre le moment où le client les consulte et le moment où il fait sa réservation. Ainsi, quand le client réserve une place que l'application lui a donnée comme libre, il a la garantie que sa réservation aboutira. L'application peut inversement laisser le statut des places évoluer au gré des réservations concurrentes. Dans ce cas, la réservation d'une place peut échouer car elle a entre-temps été réservée par un autre client. Le client doit alors à nouveau consulter les places, en choisir une nouvelle, tenter de la réserver et répéter ce processus jusqu'à aboutir.

Le paramètre `stable` de la méthode `GestionReservation.listerPlaces()` permet de sélectionner la stratégie choisie. Décrivez textuellement dans le rapport (i) la façon d'implémenter ces deux stratégies, (ii) leur impact prévisible sur les performances et (iii) tous les autres paramètres susceptibles d'influer, et dans quel sens.

Développez ensuite un programme de test de performances qui simule l'activité de plusieurs clients selon la première stratégie, puis selon la seconde. Chaque client exécutera la séquence suivante :

1. choix aléatoire d'un spectacle existant²
2. consultation des représentations de ce spectacle

¹ pour simplifier, la disposition des places dans la salle n'est cependant pas gérée par l'application

² mettre la liste des spectacles existants en dur dans le programme

3. réflexion pendant un délai aléatoire de 10 à 15 secondes, et choix d'une représentation
4. consultation des places de cette représentation
5. réflexion pendant un délai aléatoire de 20 à 30 secondes, et choix aléatoire de deux places parmi celles données comme libres
6. réservation de ces deux places
7. en cas d'échec, reprise à l'étape 2

On peut définir deux métriques de satisfaction pour un client : (i) le temps total nécessaire pour réserver les deux places, i.e. entre le début de l'étape 2 et la fin de l'étape 7, (ii) le nombre d'échecs rencontrés.

Mesurez ces métriques dans les deux stratégies et dans différentes configurations (nombre de clients, etc.) et interprétez les résultats.

67- Rapport

Rédigez un rapport contenant, dans l'ordre :

1. une introduction
2. le modèle E/A de la base
3. le modèle relationnel de la base
4. une étude de la forme normale de la table la plus importante
5. la liste des contraintes d'intégrité définies et leur mode d'implémentation (contrainte d'attribut, contrainte de table, déclencheur), avec explication du choix (pas de code SQL)
6. une description de l'implémentation des deux stratégies de consultation – réservation décrites section 56, de leur impact prévisible sur les performances, et des autres paramètres en jeu
7. une description des mesures de performances effectuées, avec pour chacune : les conditions de la mesure, les résultats obtenus et leur interprétation
8. une conclusion

8- Utilisation des transactions

La classe GestionReservationsImpl doit mettre à jour la base de données au travers de transactions (voir le cours chap. 11 et la correction du TP2). Celles-ci permettent d'exécuter un ensemble d'ordres SQL :

- en mode tout ou rien (propriété d'atomicité). Par exemple, la méthode reserverPlaces() peut regrouper au sein d'une même transaction l'exécution des différentes réservations demandées en paramètre. Si une réservation échoue³, reserverPlaces() peut alors défaire par un rollback() toutes celles qu'elle a déjà engagées. On obtient ainsi le comportement demandé : soit toutes les places demandées sont réservées, soit aucune ne l'est.
- de façon sérielle (propriété d'isolation, avec niveau sérialisable). Par exemple, la méthode reserverPlaces() peut demander l'exécution de sa transaction en mode sérialisable. Si plusieurs de ces transactions sont lancées en même temps, le SGBD les exécutera alors les unes après les

³ parce qu'il y a eu une erreur, ou parce que la place était déjà réservée

autres sans entrelacement (ou en tout cas, tout se passera comme si c'était le cas). Il sera ainsi impossible à deux réserverPlaces() fait simultanément par deux clients distincts de voir libre la même place et de la réserver deux fois.

Les différents ordres SQL exécutés par une méthode comme réserverPlaces() constituent généralement un tout logique, et doivent donc être exécutés au sein d'une même transaction. On peut cependant envisager qu'une méthode soit décomposée en plusieurs transactions indépendantes, ou qu'une transaction s'étende sur plusieurs méthodes.

A noter que si une table est consultée puis mise à jour au sein d'une même transaction, la consultation doit se faire au travers de la construction « select ... **for update** » pour que la transaction soit vraiment sérialisée. Cela consiste simplement à rajouter la clause « for update » à la fin de l'ordre select habituel.

Deux solutions sont possibles pour réaliser des tests mettant en œuvre plusieurs clients simultanés :

- On peut lancer en parallèle plusieurs programmes Java agissant chacun pour le compte d'un client particulier.
- On peut lancer un seul programme qui exécute plusieurs threads en parallèle, chacun agissant pour le compte d'un client particulier. Chaque thread doit alors utiliser un objet GestionReservationsImpl qui lui est propre pour accéder à la base de données.

Remise

Le projet doit être remis sur Campus. Consulter le site pour la date limite de remise.

Il ne faut remettre qu'un seul fichier archive (.tgz ou .zip) par binôme. Ce fichier doit se nommer **NOM1.Prénom1.NOM2.Prénom2.[tgz|zip]** pour identifier les membres du binôme.

L'archive doit contenir les éléments suivants (à plat, sans répertoire intermédiaire) :

- le script `creabase.sql` permettant de créer et peupler la base
- le script `contraintes.sql` permettant d'ajouter les contraintes
- l'ensemble des fichiers sources *.java composant l'application
- le rapport nommé `rapport.doc` ou `rapport.docx` au format MS Word

Évaluation

Le barème indicatif du projet est donné ci-dessous.

Rubriques	Points
Modèle E/A	2
Modèle relationnel	2
Étude forme normale	2
Contraintes d'intégrité	2
Classes de gestion de réservations	45
<u>dont gestion des transactions</u>	<u>1</u>
Programme de mesure des performances	2
Rapport – hors mesures de performances	3
Rapport – partie mesures de performances	32

Important

Les sources doivent compiler et les programmes / scripts s'exécuter sans erreur. Les machines de référence (Java, Oracle) sont les machines Linux de l'EFREI.

L'application Java ~~et les vues seront testées~~ sera testée par un programme de test automatique : les interfaces ~~et schémas~~ définies doivent être respectées à la lettre.