Bases de données relationnelles : une application web pour découvrir la concurrence d'accès

Julien Grynberg

Bases de données relationnelles : une application web pour découvrir la concurrence d'accès

Julien Grynberg

TECHNICAL REPORT

Avril 2017

DSI-SEISM / Inria Learning Lab

ISSN 0249-6399

Mooc Bases de données relationnelles : une application web pour découvrir la concurrence d'accès

Julien Grynberg[1]
DSI-SEISM – Inria Learning Lab
Technical Report — Avril 2017 —13 pages.

Résumé : Dans le cadre du MOOC «Bases de données relationnelles : comprendre pour maîtriser[2] » enseigné par Serge Abiteboul (Inria/ENS Cachan), Benjamin Nguyen (INSA CVL) et Philippe Rigaux (CNAM) et lancé en janvier 2016, nous avons développé une application web permettant à chaque apprenant de tester simplement une succession de requêtes SQL. Cette interface proposait d'exécuter des requêtes sur deux transactions parallèles afin de donner aux étudiants la possibilité d'appréhender la concurrence d'accès sur une base de données.
Les auteurs du MOOC ont d'abord défini les spécifications d'une application en ligne à intégrer à leur cours pour que les étudiants puissent appréhender des notions théoriques complexes par la pratique, sans avoir à quitter la plateforme FUN (France Université Numérique) sur laquelle se jouait le cours . Cela permettait, en outre, d'éviter l'installation d'un environnement SQL qui peut être lourd et rebutant pour les étudiants.

Mots clés : Bases de données relationnelles, mysql, python, django, concurrence d'accès, niveau d'isolation, application

[1] Inria – Julien.Grynberg@inria.fr

[2] 2 sessions du MOOC ont été diffusées sur la plateforme FUN : une première session en 2016 et la deuxième session en 2017 https://www.fun-mooc.fr/courses/inria/41008S02/session02/about

RESEARCH CENTRE
GRENOBLE - RHÔNE-ALPES

Inovallée
655 avenue de l'Europe - Montbonnot
38334 Saint Ismier Cedex France

Abstract: As part of an MOOC on relational databases launched in January 2016, we developed a web application allowing each learner to test a succession of SQL queries. This interface proposes to execute requests on two parallel transactions, it allows students to play with database concurrency.
The authors of the MOOC wanted to have a solution integrated into their course so that students could grasp complex theoretical notions by the practice, without having to leave the platform on which they were. This also allowed them not to mention installing an SQL environment that can be unpleasant to students.

Key-words: Database, mysql, python, django, access concurrency, isolation level, application

1 Contexte

Serge Abiteboul, Benjamin Nguyen et Philippe Rigaux ont lancé en janvier 2016 sur France Université Numérique [1] le MOOC "Bases de données relationnelles : comprendre pour maîtriser". Dans le cadre de ce MOOC, ils ont voulu mettre en place une application web permettant à chaque apprenant de tester simplement une succession de requêtes SQL. Plus exactement, ils souhaitaient une interface en ligne où exécuter des requêtes sur deux transactions parallèles afin de donner aux étudiants la possibilité d'appréhender la concurrence d'accès sur une base de données.

Cette application web devait répondre à plusieurs impératifs. Elle devait pouvoir être intégrée à la plateforme FUN et être utilisée pour répondre aux exercices proposés par les auteurs. Chaque apprenant devait pouvoir s'en servir dans une session qui lui serait propre, sans interférer avec celles des autres. Enfin, chaque apprenant devait pouvoir sélectionner le niveau de sécurité dans lequel seraient effectuées ses prochaines requêtes.

Les auteurs du MOOC ont d'abord défini les spécifications d'une application en ligne à intégrer à leur cours pour que les étudiants puissent appréhender des notions théoriques complexes par la pratique, sans avoir à quitter la plateforme FUN (France Université Numérique) sur laquelle se jouait le cours. Cela permettait, en outre, d'éviter l'installation d'un environnement SQL qui peut être lourd et rebutant pour les étudiants.

2 Description du besoin

Chaque utilisateur dispose d'un accès à deux tables :
- Table Client_xx (id, nom, nbPlacesRéservées, solde) ;
- Table Vol_xx (*id, intitulé, placesDispo, placesPrises, tarif*) ;

où xx est l'identifiant de l'utilisateur.

Dans chaque table il y a deux lignes avec les contenus initiaux suivants :

Id	Nom	Solde	nbPlacesRéservées
C1	Serge	1500	0
C2	Philippe	1000	0

Id	Intitulé	placesDispo	placesPrises	Tarif
V1	Mexico	250	0	800

Un bouton permet de réinitialiser ces tables avec le contenu ci-dessus.
Pour tester la concurrence d'accès, l'appli doit montrer deux fenêtres parallèles correspondant à deux connexions (*sessions*) séparées à la base de données. Chaque session correspond à l'un des clients, C1 ou C2.
Dans chacune des sessions, on doit pouvoir dérouler l'exécution, pas à pas, d'une transaction de réservation de *n* places pour San Francisco:
- Select V1 : lecture des places dispos pour le vol V1 ;
  `Select placesDispo, placesPrises, tarif INTO :p1, :p2, :t from Vol where id=V1`
- Select Cx : lecture du compte du client Cx pour vérifier qu'il est suffisant ;
  `Select solde INTO :s from Client where id=Cx`
- Update V1 : modification du vol pour enregistrer les n places prises ;
  `Update Vol SET placesPrises = :p1-n where id=V1`
- Update Cx : modification du client Cx.
  `Update Client SET solde = :s - :n* :t where id=Cx`

L'application propose, dans chacune des deux fenêtres, un bouton permettant d'exécuter successivement ces 4 requêtes ce qui revient à effectuer (lentement) l'exécution de la transaction.
Deux autres boutons permettent d'effectuer respectivement un commit ou un rollback. Aucune autre modification n'est possible.
Par ailleurs, l'application propose un menu déroulant pour choisir le niveau d'isolation :
- READ UNCOMMITTED ;
- READ COMMITTED ;
- REPEATABLE READ ;
- SERIALIZABLE.

Le changement du niveau d'isolation implique une réinitialisation du contenu des tables pour les remettre proprement à niveau.

A chaque étape, l'utilisateur doit pouvoir consulter l'état de la base. Il peut ainsi constater qu'une mise à jour dans une session reste invisible de l'autre session jusqu'au commit, ou qu'un rollback annule toutes les mises à jour.

Le TP consistera à proposer, pour deux exécutions concurrentes des deux transactions, plusieurs entrelacements des instructions.

1. Transaction C1 : réservation de 2 places pour le client C1
2. Transaction C2 : réservation de 5 places pour le client C2

L'utilisateur devra effectuer ces entrelacements avec l'application, et anticiper le comportement du système.

3 Architecture technique

La solution proposée a été développée sur une base Django 1.8, Python 3.4, la librairie mysqlclient pour Python 3 et MySQL/Innodb pour la gestion des bases de données. Elle a été déployée sur une machine sous Ubuntu 14.0, tournant avec Gunicorn comme serveur web et Nginx comme vhost.

Nous avons choisi d'utiliser MySQL plutôt que PostgreSQL comme base de données. Le niveau de sécurité READ_UNCOMMITTED étant simulé par le niveau READ_COMMITTED sur PostgreSQL [2], il n'était en effet pas possible de disposer des 4 niveaux d'isolation. MySQL ne rencontre pas ce problème.

Nous nous sommes tournés vers Python pour disposer rapidement d'un prototype à valider. Nous pouvions utiliser les librairies mysql-python et mysqlclient pour effectuer les opérations SQL qui constituait le coeur de la problématique :

- création d'une base de données par utilisateur ;
- ouverture de deux transactions concurrentes ;
- exécution pas à pas de requêtes de lecture et mise à jour des tables ;
- exécution des requêtes selon le niveau d'isolation désiré par l'utilisateur.

La librairie mysqlclient s'est imposée car elle était compatible avec Python 3 au moment du développement du prototype, ce qui n'était pas le cas de la librairie mysql-python. Le prototype ayant montré ses preuves, nous nous sommes alors tournés vers un framework Python capable de donner satisfaction.

Flask et Django couvraient l'ensemble de nos besoins en la matière, nous avons arbitrairement choisi d'utiliser ce dernier. Les deux disposaient en effet de vues et de templates rapides à mettre en oeuvre, de paramètres de session afin d'identifier simplement l'utilisateur connecté, d'une gestion des utilisateurs intégrée au framework et d'une protection contre le « Cross site request forgery » (CSRF) lors de l'envoi des formulaires contenant les requêtes SQL à exécuter. S'il a été envisagé un instant pour sa facilité à être mis en oeuvre, l'utilisation de l'ORM de Django n'aurait pas été adaptée à la problématique (voir ci-dessus). Django n'est en autre pas conçu pour gérer une base de données par utilisateur. Hormis la gestion des utilisateurs qui a été laissée à sa charge, l'ensemble des requêtes et de la gestion des bases de données utilisateurs a été effectuée en conservant le prototype Python/mysqlclient.
L'utilisation spécifique de Django 1.8 et de Python 3.4 découle du fait qu'il s'agissait des dernières versions disponibles au moment du développement.

4 Cadre d'utilisation

L'interface proposée à l'utilisateur dispose :

- d'un menu pour sélectionner le niveau d'isolation et réinitialiser les données ;
- deux colonnes reprenant les données propres à chaque transaction : état des tables, variables locales utilisées pour la mise à jour des tables, et les 6 requêtes pouvant être exécutées (voir Fig.1) ;
- l'historique des opérations effectuées par les deux transactions (voir Fig. 2 & 3).

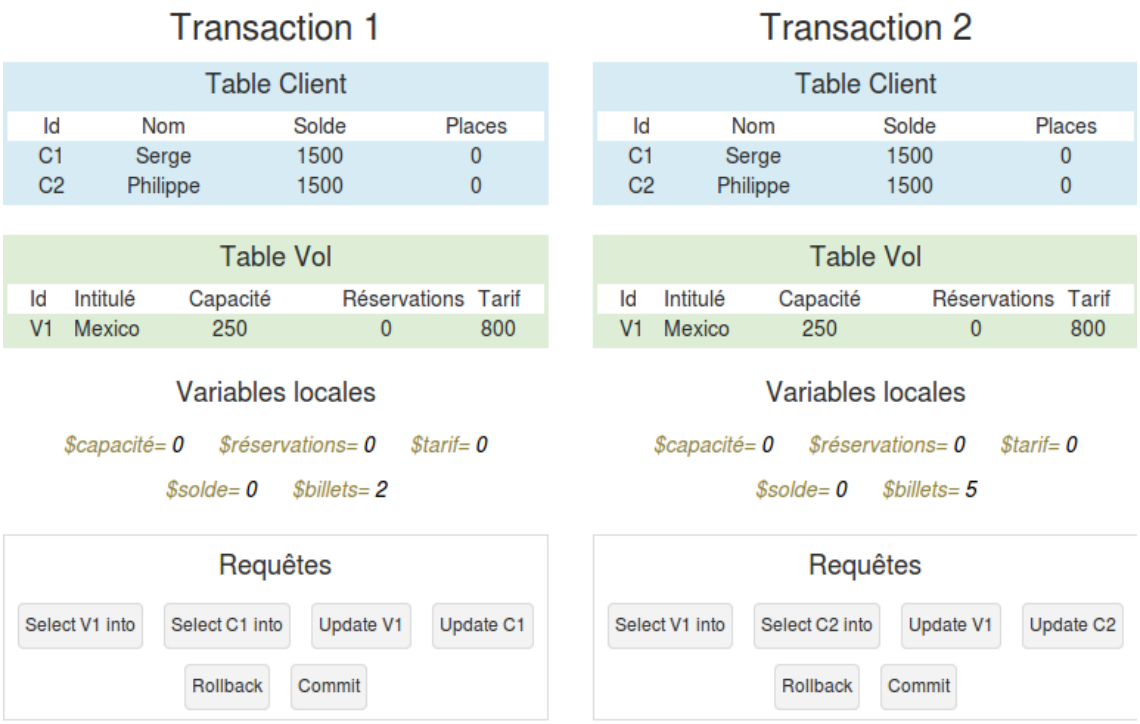

Fig. 1 : L'interface présente à l'utilisateur l'état de la base de données vu par chacune des deux transactions à chaque instant et permet d'effectuer simplement des requêtes sur celle-ci.

Le MOOC a été lancé sur la plateforme France Université Numérique. L'application développée a été intégrée au MOOC sous la forme d'une iframe, qui pointait vers une unique url.
Nous avons utilisé le moteur de session Django afin de pouvoir attribuer un profil unique à chaque utilisateur connecté depuis le MOOC. Si l'utilisateur ne dispose pas d'un identifiant de session, un identifiant est créé côté serveur, avec la création d'une base de données qui lui sera propre pour répondre aux exercices, et un identifiant de session est reçu et stocké côté client dans un cookie.

```
from MySQLdb import connect
from MySQLdb.cursors import DictCursor

try:
   conn = connect(db=self.db, user=self.user, passwd=self.passwd, cursorclass=DictCursor)
except:
   # database does not exist -> create database
   conn = connect(db="init", user=self.user, passwd=self.passwd)
   cursor = conn.cursor()
   cursor.execute("CREATE DATABASE concurrency_%s", [self.etudiant_id])

   # connect to new database and init it
   # table does not exist -> create tables
   # code to create tables and insert values into them...
```

A chaque nouvelle requête envoyée, deux connections à la base de données sont ouvertes :

```
for id in [1, 2]:
   # init transactions and cursors
   setattr(self, "conn_{0}".format(id), connect(db=self.db, user=self.user, passwd=self.passwd,
      cursorclass=DictCursor))
   setattr(self, "cursor_{0}".format(id), getattr(self, "conn_{0}".format(id)).cursor())
```

L'ensemble des requêtes effectuées par l'utilisateur est alors effectué dans ces connections. La requête Select V1 (voir section Description du besoin) est par exemple effectuée sous la forme suivante :

```
   # Select and return vol for connection connection_id
   getattr(self, "cursor_{0}".format(connection_id)).execute('SELECT id, placesDispo, placesPrises,
tarif FROM vol')
   return getattr(self, "cursor_{0}".format(transaction_id)).fetchone()
```

Le résultat est renvoyé sous la forme d'un dictionnaire grace à la classe DictCursor de mysqlclient.

Une fois ces requêtes exécutées, le serveur renvoie en JSON les données à mettre à jour dans le DOM et rend la main à l'utilisateur. Les données mises à jour sont les données des tables et les variables locales de chaque transaction, ainsi que l'historique des opérations effectuées et leur état.
Dans le cas d'une concurrence d'accès, nous avons paramétré le temps d'attente de verrouillage des données à 1 seconde (innodb_lock_wait_timeout), temps minimal autorisé par Innodb. Ainsi, si une des lignes d'une des tables est verrouillée par une première transaction, dans le cas où la seconde ferait une demande d'accès dessus, celle-ci serait rejetée au bout d'une seconde. L'état de la transaction serait alors renseigné comme étant en attente pour l'utilisateur (voir Fig. 2).

Fig. 2 : Les deux transactions veulent réserver au même moment un billet d'avion. Dans le niveau de sécurité Repeatable Read, le Update de la transaction 1 verrouille la table Vol. La transaction 2 est alors en attente.

```
try:
   # Select and return vol for connection connection_id
   getattr(self, "cursor_{0}".format(connection_id)).execute('SELECT id, placesDispo, placesPrises,
      tarif FROM vol')
   return getattr(self, "cursor_{0}".format(transaction_id)).fetchone()
# timeout
except _mysql_exceptions.OperationalError:
   self.set_status_unavailable_for(connection)
```

Seul un COMMIT ou un ROLLBACK permet de sortir de cet état verrouillé. Dans le cas présent, la suite des requêtes de la première transaction est exécutée jusqu'à un COMMIT ou un ROLLBACK, auquel cas la requête rejetée de la deuxième est réeffectuée puisque le verrou est levé (voir Fig.3).

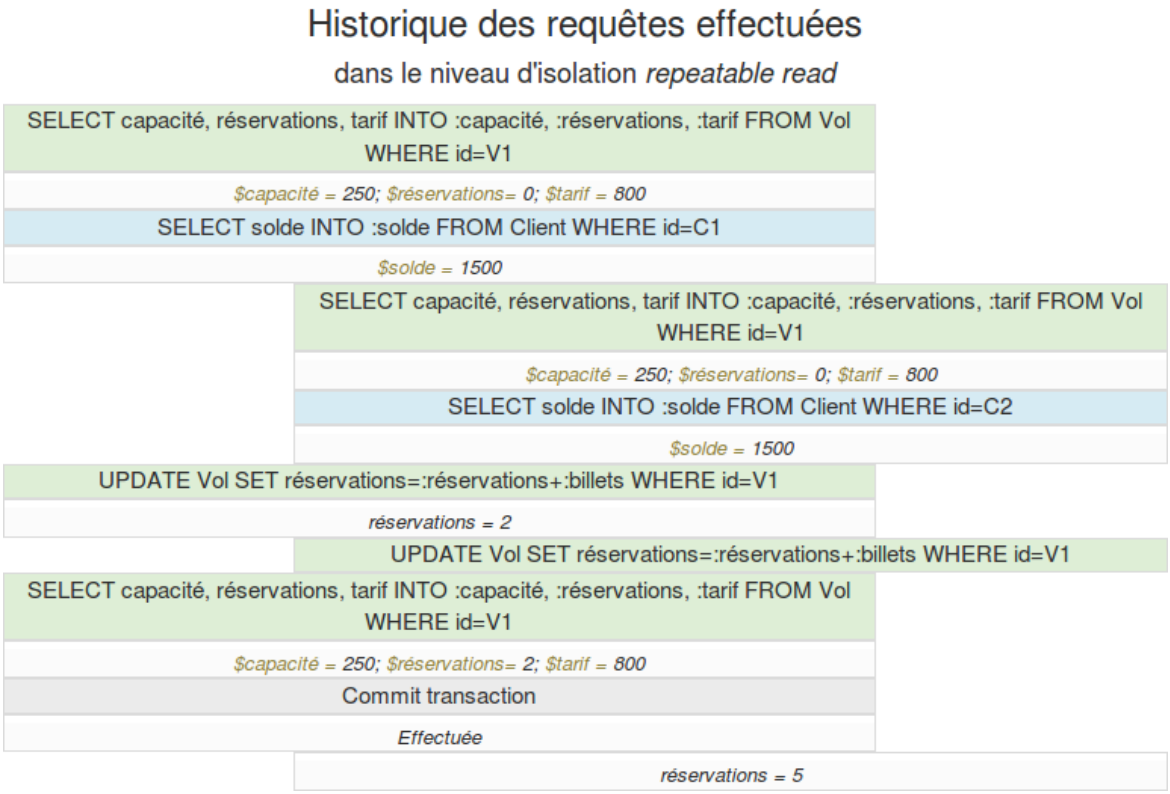

Fig. 3 : Les deux transactions veulent réserver au même moment un billet d'avion. La transaction 2 reste en attente (voir Fig. 2) jusqu'au commit effectué par la transaction 1.

5 Retour d'expérience

9141 étudiants ont suivi la première session du MOOC "Bases de données relationnelles : comprendre pour maîtriser". Pour cette première session, 4324 connexions à l'application ont été créées pendant les deux mois d'ouverture du MOOC pour répondre aux exercices. Le nombre maximum de connexions a été établi la première semaine avec un pic journalier à 330 connexions.

Les apprenants ont très largement apprécié l'utilisation du TP dans le cadre du MOOC qui, selon leurs avis, a bien répondu aux objectifs d'une pratique qui favorisait une meilleure compréhension des concepts théoriques complexes, de la concurrence d'accès à une base de données relationnelle et des niveaux d'isolation.

Dans le cas où des verrous sont posés sur les tables (voir Fig. 2 et section Cadre d'utilisation), le temps de réponse supérieur à 1 seconde de la part du serveur est parfois considéré comme étant long de la part des apprenants. Il serait intéressant de concevoir une solution qui s'affranchisse de ce problème.

[1] https://www.fun-mooc.fr/courses/inria/41008S02/session02/about

[2] https://www.postgresql.org/docs/9.1/static/transaction-iso.html

RESEARCH CENTRE
GRENOBLE - RHÔNE-ALPES

Inovallée
655 avenue de l'Europe - Montbonnot
38334 Saint Ismier Cedex France

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr
ISSN 0249-6399