Bases de Données

Cyril Labbé

LIG, Université de Grenoble, France
first.last@imag.fr

http://membres-lig.imag.fr/labbe/M1BD/CoursM1BD_3.pdf

Table of Contents

SQL Isolation level

Les niveaux d'isolation SQL :

Isolation Level	Dirty Reads	Non-Repeatable Reads	Phantom Reads
READ_UNCOMMITTED	A	A	A
READ_COMMITTED	P	A	A
REPEATABLE_READ	P	P	A
SERIALIZABLE	P	P	P

- P: Prevented
- A: Allowed

Choix relatif d'isolation relatif à chaque transaction.

- Début de transaction implicite
- Niveau d'isolation par défaut du SGBD
- Chaque transaction a son propre niveau d'isolation :
  (`Set transaction isolation level ...`)
- Vérification des contraintes d'intégrité (le C de ACID) peut-être soit immédiate, soit différer à la fin (à la validation) de la transaction.
  `SET CONSTRAINTS ALL | name [, ...] DEFERRED | IMMEDIATE`
- En générale, une transaction ne peut démarrer une autre transaction (nested transaction)
- Terminaison explicite `Commit / Abort` ou implicite

Table of Contents

Contraintes d'intégrité

Exemples

- *Beers(BeerName, BrewerName, Country)*
- *Drinkers(DrinkName, Drink@, DrinkPhone)*
- *Bars(BarName, Bar@, licence, OpenDate)*
- *Sells(BeerName#,BarName#, price)*
- *Frenquents(BarName#,DrinkName#)*
- *Dinks(BeerName#,DrinkName#)*

Contraintes d'intégrité référentiels

- Clés *primaires*, Clés *étrangères* ($\pi_{BarName}(Drinks) \subset \pi_{BarName}(Bars)$)

Autres types de contraintes

- Contraintes sur un attribut : $\forall p \in \pi_{price}(Sells), p < 7$
- Contraintes sur un tuple : $\forall t \in \sigma_{BarName='Au\ \ bon\ \ prix'}(Sells), p < 4.5$
- Assertion sur la base. Par exemple : *aucun bar ne doit avoir un prix moyen supérieur à* 5.
- Event Condition Action (trigger)

Contraintes de type *applicatives*

- Certaines vérifications sur *licence*, *@*, *Brewername*.

Clés

Clé sur un attribut

Beers(BeerName, BrewerName)

unique

```
1  CREATE TABLE Beers (
2          BeerName CHAR(20) UN
3          BrewerName CHAR(20)
4  );
```

primary key

```
1  CREATE TABLE Beers (
2          BeerName CHAR(20) PRIMARY K
3          BrewerName CHAR(20)
4  );
```

Clé multi-attribut

Sells(BeerName,BarName, price)

```
1  CREATE TABLE Sells (
2          BarName CHAR(20),
3          BeerName VARCHAR(20),
4          price FLOAT,
5      CONSTRAINT pk_Sells PRIMARY KEY (BarName, BeerName)
6  );
```

Clés étrangères

Clé étrangères

Beers(BeerName, BrewerName) *Sells(BeerName#,BarName#)*

```
1  CREATE TABLE Beers (
2          BeerName    CHAR(20) PRIMARY KEY,
3          BrewerName    CHAR(20)
4  );
5  CREATE TABLE Sells (
6          BarName     CHAR(20),
7          BeerName    CHAR(20),
8          price   REAL,
9      FOREIGN KEY(BarName) REFERENCES Bars(BarName),
10     FOREIGN KEY(BeerName) REFERENCES Beers(BeerName)
11 );
```

Comportement

- Par défaut : rejet de la modification.
- Cascade : les modifications (`Delete`, `Update`, `Set NULL`) dans *Sells* sont propagées dans *Bars* et *Beers*.

CASCADE examples

On Cascade

Beers(BeerName, BrewerName) *Sells(BeerName#,BarName#)*

```
1  CREATE TABLE Sells ( BarName CHAR(20), BeerName CHAR(20),
2          FOREIGN KEY(BeerName) REFERENCES Beers(BeerName)
3          ON DELETE SET NULL
4          ON UPDATE CASCADE
5          FOREIGN KEY(BarName) REFERENCES Bars(BarName)
6          ON DELETE CASCADE
7  );
```

Suppression 'Bud' de la table *Beers*

- `ON DELETE CASCADE`, tous les tuples de *Sells* ayant $beer =' Bud'$ sont supprimés.
- `ON DELETE SET NULL`, $beer = NULL$ pour tous les attributs $beer =' Bud'$ de *Sells*.

Mise à jour de 'Bud' en 'Budweiser' dans *Beers*

- `ON UPDATE CASCADE` tous les tuples de *Sells* ayant $beer =$ *'Bud'* sont modifié en $beer =$ *'Budweiser'*.

Check Attribute

Contrainte sur la valeur d'un attribut

La condition peut utiliser le l'attribut mais tout autre table ou attribut doit être dans la sous-requête.

```
1  CREATE TABLE Sells (
2            BarName CHAR(20),
3            BeerName CHAR(20) CHECK (
4                     BeerName IN (SELECT Beers.BeerName
5                                  FROM Beers)),
6            price REAL CHECK ( price <= 7.0 )
7  );
```

Instant de vérification

La contrainte est vérifiée quand une valeur de l'attribut est modifiée ou insérée dans *Sells*.

Tuple Check

Contrainte sur la valeur d'un tuple

$\forall t \in \sigma_{BarName='Au\ bon\ prix'}(Sells), p < 4.5$

```
1  CREATE TABLE Sells (
2            bar CHAR(20),
3            beer CHAR(20),
4            price REAL,
5  CHECK (bar = "Le c'est pas donné" OR price <= 5.00),
6  CHECK (bar <> "Au bon prix"
7                 OR
8                (bar = "Au bon prix" AND price <= 5.00))
9  );
```

Instant de vérification

La contrainte est vérifiée en cas d'insertion ou de modification.

Assertion

Assertion

Les assertions sont des éléments du schéma (comme les tables et les vues). Une assertion est une condition qui peut faire intervenir n'importe quel tables du schéma de la BD.

```
1  CREATE ASSERTION Moyenne_Bars CHECK (
2        NOT EXISTS (
3              SELECT BarName FROM Sells GROUP BY BarName
4                                  HAVING 5.00 < AVG(price)
5  ));
6  CREATE ASSERTION Bar_Drinkers CHECK (
7    (SELECT COUNT(*) FROM Bars) <=
8    (SELECT COUNT(*) FROM Drinkers)
9  );
```

Instant de vérification

En principe à chaque modification de la BD.

Trigger

Motivation

- Les assertions sont expressive mais, souvent, le SGBD ne sait pas quand les vérifier.
- Les *Attribute* / *tuple checks* sont peut exprésif, mais le SGBD sait quand les vérifier.
- Les *Triggers* sont expressifs et permettent à l'utilisateur de décider l'instant de la vérification.

Event-Condition-Action Rules/Règles actives.

- Event : typiquement une modification de la BD (i.g. *insert/delete/update*)
- Condition : n'importe quel expression booléenne SQL.
- Action : des instructions SQL.

Trigger row-level / statement level.

- Row level triggers : exécuté une fois à chaque modification de tuple (: *New*,: *OLD*).
- Statement-level triggers : exécuté une fois par instruction SQL statement, sans lien avec le nombre de tuples modifiés.

Trigger : exemples

Trigger mode ligne

```
1  CREATE TRIGGER BeerTrig
2  AFTER INSERT ON Sells
3  FOR EACH ROW
4  BEGIN
5      IF (:NEW.price < 1.1 * :OLD.price)
6          raise_application_error('Augmentation trop forte!');
7      END IF;
8  END;
9  /
```

Trigger row-level (Mutating table).

```
1  Create trigger ControlAjout before insert on Beers For each row
2  Declare nb integer;
3  Begin
4          select count(*) into nb from Beers;
5          if nb>1000 then raise_application_error('Trop de Bière!');
6          end if;
7  End;
8  /
Mutating tables (erreur:  ORA-04091:  table Beers en mutation)
```

Trigger : exemples

Trigger row-level => statement level.

```
1  Create trigger ControlAjout2
2  before insert on etudiants
3  Declare
4        nb integer;
5  Begin
6        select count(*) into nb from etudiants;
7        if nb=1000 then raise_application_error('Trop de Bière !');
8        end if;
9  End;
10 /
```

Transactions : résumé

Propriétés ACID

Atomicité, Cohérence, Isolation, Durabilité

Accés à la BD

Savoir quand commencent et finissent les transactions (`commit, abort`)

Contrôle de concurence

Anomalies, verrouillage, estampille

Choisir un niveau d'isolation

Niveau d'isolation SQl

Reprise après panne

Journalisation et point de reprise

Définir des contraintes d'intégrité

SQL, BD actives

Table of Contents

Object vs. Relational

Manipulation de concepts différents

- Classes et Relations
- Tuples et Objects

Modèle relationnel :

- Relation: sous ensemble d'un produit cartésien
- Les relation sont *peuplées* de tuples

Modèle Object :

- Classes : obtenue par *construction* de type *complexes* (tuple, tableau, énumération, heritage..)
- Objects: instances de classes, avec des méthodes.

Quand le SQL...

... est utilisé dans un language de programmation, il faut faire le pont entre les deux mondes

- L'API JDBC permet de faire ce pont

JDBC Features

- Driver management: selection, load
- Database connection management: resource allocation
- Query execution: static and dynamic
- Result processing: SQL types-Java types matching
- Metadata: driver's properties, database's catalog

Driver and connection

Loading Driver in the JVM

```
1  DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
```

```
1  try { Class.forName("oracle.jdbc.OracleDriver").newInstance(); }
2  catch (ClassNotFoundException e) { ... }
```

Connecting to a Database

```
1  String user; String url; String password;
2  Connection conn = DriverManager.getConnection(url, user, password);
```

- url identifies the resource (Uniform Resource Locator). It is different depending on the DBMS. With Oracle it looks like *jdbc:oracle:thin:@serveur:port:database*.
- user is the user's login name
- password is the user's password
- You need to take care of your classpath (e.g. /somepath/oracle/jdbc/lib/ojdbc14.jar where the driver can be found.)
- You may need to set particular environment variables.

A Typical Session: Closing a Connection

When a connection is no longer useful, we need to close it explicitly:

```
1  Connection conn = null;
2  try {
3    conn = DriverManager.getConnection(url, user, passwd);
4     ... // work with the database
5    conn.close(); // close the connection
6  }
7  catch (SQLException e) {
8     ...
9  }
10 finally {
11    try {
12      if (conn != null) conn.close();
13    }
14    catch (SQLException e) {
15      e.printStackTrace();
16    }
17 }
```

Preparing and Submitting Queries and Updates

- Assumption: a connection has been created
- 3 types of queries
  - Statement: basic query
  - PreparedStatement: pre-compiled query
  - CallableStatement: call of an embedded procedure
- 3 types of executions
  - executeQuery to submit a query which returns data
  - executeUpdate for a query which does not return date (e.g. INSERT, UPDATE, DELETE, CREATE TABLE, DROP TABLE, ...)
  - execute to execute an embedded procedure

A Comprehensive Example

```
1  import java.sql.*;
2  public class TestJDBC {
3      public static void main(String[] args) throws Exception {
4          try {
5          DriverManager.registerDriver (new oracle.jdbc.driver.OracleDri
6
7          Connection conn = DriverManager.getConnection();
8          Statement stmt = conn.createStatement();
9          ResultSet rs = stmt.executeQuery("SELECT␣*␣from␣employe");
10         while (rs.next()) {
11                 String nom = rs.getString("nom");
12                 String prenom = rs.getString("prenom");
13                 String email = rs.getString("email");
14         }
15         } catch (...) { }
16         finally { ... }
17     }
18 }
```

Prepared Statements

- compiled and prepared beforehand, so it can be executed faster, and might even be reused,
- can take parameters,
- less error prone with data conversions.
- Use if a query is run multiple times and only the values of the same columns change.

Example : PreparedStatement with ResultSet

```
1       ...
2       Vector<String> res = new Vector<String>();
3       Connection conn = null;
4       PreparedStatement pstmt = null;
5       ResultSet rs = null;
6       int number_of_seats = 500;
7
8       // open the connection to define conn ...
9
10      pstmt = conn.prepareStatement("select␣distinct␣nomS"
11                                    + "from␣LesSpectacles␣"
12                                    + "where␣numS␣>␣?␣order␣by␣nomS");
13      pstmt.setInt(1,number_of_seats); // includes type checking
14      rs = pstmt.executeQuery();
15
16      while (rs.next()){
17          res.addElement(rs.getString(1));
18      }
19      // close the connection: rs, stmt, conn
```

Retrieving Data ResultSet

getXXX() methods to access values of type XXX for a row in a ResultSet:

```
1  ResultSet rs = stmt.executeQuery("SELECT␣a,␣b,␣c␣FROM␣Table1");
2
3  while (rs.next()) {
4
5          int i = rs.getInt("a");         // rs.getInt(1);
6          String s = rs.getString("b"); // rs.getString(2);
7          byte b[~] = rs.getBytes("c"); // rs.getBytes(3);
8          System.out.println("ROW␣=␣" + i + "␣" + s + "␣" + b[0]);
9
10 }
```

Example: ResultSet Meta Data

- number of columns returned,
- column names,
- column types,
- precision and scale of numbers,
- ...

Not all DBMS provide this information, so check for empty or null return values!

```
1  Statement statement = connection.createStatement();
2  ResultSet resultSet = statement.executeQuery("select␣*␣from␣Students"
3  ResultSetMetaData resultSetMetaData = resultSet.getMetaData();
4
5  int nCols = resultSetMetaData.getColumnCount();
6
7  System.out.println("Number␣of␣columns:␣" + nCols);
8  for (int i=1; i<=nCols; i++)
9      {
0          String columnName = resultSetMetaData.getColumnName(i);
1          String columnTypeName = resultSetMetaData.getColumnTypeName(i)
2          System.out.println(columnName + ":␣" +  columnTypeName);
3      }
```

Table of Contents

Commit Mode, start and stop transactions

Commit Mode

- When a connection is created using JDBC, by default it is in auto-commit mode. Each SQL statement is a transaction and is automatically committed immediately after it is executed.
- To allow two or more statements to be grouped into a transaction you need to disable auto-commit mode: `conn.setAutoCommit(false);`
- To allow two or more statements to be grouped into a transaction you need to disable auto-commit mode: no SQL statement will be committed until the commit method is called. The entire set of statements can be rolled back, without committing.

Start / End

- First call of `conn.setAutoCommit(false)`, Each call of `conn.commit()`, implicitly mark the start of a transaction.
- Transactions can be undone before they are committed by calling : `conn.rollback()`

JDBC Isolation level

Transaction isolation levels you can use in JDBC:

Isolation Level	Dirty Reads	Non-Repeatable Reads	Phantom Reads
TRANSACTION_READ_UNCOMMITTED	A	A	A
TRANSACTION_READ_COMMITTED	P	A	A
TRANSACTION_REPEATABLE_READ	P	P	A
TRANSACTION_SERIALIZABLE	P	P	P

- P: Prevented
- A: Allowed

Set JDBC Isolation level

- The default transaction isolation level depends on your DBMS.
- To find out what transaction isolation level your DBMS is set to : `conn.getTransactionIsolation()`
- To set it to another level: `conn.setTransactionIsolation()`

Example

```
1  Connection connection = null;
2  try {
3   connection = DriverManager.getConnection("...");
4   connection.setAutoCommit(false);
5
6   Statement statement = connection.createStatement();
7
8  statement.executeUpdate("UPDATE␣Table1␣SET␣Value=1␣WHERE␣Name='foo'")
9  statement.executeUpdate("UPDATE␣Table2␣SET␣Value=2␣WHERE␣Name='bar'")
10
11  connection.commit();
12
13 } catch (SQLException ex) {
14   connection.rollback();
15 }
```

Going further...

jdbc:

- Managing connections with data sources
- JDBC Data access optimisation
- ...

Mastering multi-tiers architecture:

- Who is in charge of what?
- Frameworks can help: hibernate, struts,

Bibliographie.[?, ?]