Bases de Données

Cyril Labbé

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

http://membres-lig.imag.fr/labbe/ISN/CoursISN2BD_2.pdf

Table of Contents

Modèle relationnelle

Une relation est définie par un *schéma*

Un schéma : un prédicat, des attributs ayant un domaine, une clé

Exemples

- *Beers(BeerName, BrewerName, Country)*
  $< bn, mn, c > \in Beers \iff$ {the beer bn is manufactured by mn based in c}
- *Drinkers(DrinkName, Drink@, DrinkPhone)*
  $< dn, d@, d > \in Drinkers \iff$ {the drinker dn lives at $d@$ and has phone d}
- *Bars(BarName, Bar@, licence, OpenDate)*
  $< ban, b@, l, op > \in Bars \iff$ ...
- *Sells(BeerName,BarName, price)*
  $< bn, ban, p > \in Sells \iff$ ...
- *Frenquents(BarName,DrinkName)*
  $< ban, dn > \in Frenquents \iff$ ...
- *Dinks(BeerName,DrinkName)*
  $< bn, dn, > \in Dinks \iff$ ...

Deux lectures :

- Ensembliste : une relation est un ensemble de tuples.
- Logique du premier ordre : une relation est une formule logique vérifiée par des variables.

Algèbre relationnelle

Algèbre

Les opérandes sont des relations. Le résultat des opérations est une relation

Les opérateurs de base

- Projection π
- Sélection σ
- Produit cartésien $\times$
- Union $\cup$ et Intersection $\cap$
- Renomage ρ

Exemples

Projection π

$\pi_{(Country, BrewerName)}(Beers)$

Sélection σ

$\sigma_{(Country='France' \vee Country='Belgique')}(Beers)$

Union / Différence

- $\pi_{Bar@}(Bars) \cup \pi_{Drink@}(Drinkers)$
- $\pi_{Bar@}(Bars) - \pi_{Drink@}(Drinkers)$

Produit cartésien

$\pi_{Beers.BarName}(\sigma_{(Beers.BarName=Sells.BarName \wedge BeerName='Bud' \wedge price<3)}(Beers \times Sells))$

Opérateurs dérivés

θ-produit $\bowtie_\theta$

- $Beers \bowtie_{(Beers.BeerName=Sells.BeerName)} Sells \iff \sigma_{(Beers.BeerName=Sells.BeerName)}(Beers \times Sells)$
- $Sells2 \leftarrow \rho_{ban,ben,p}(Sells)$
- $\pi_{Sells.price}(Sells) - \pi_{Sells.price}(Sells \bowtie_{(Sells.price<Sells2.p)} Sells2)$

jointure naturelle $\bowtie$

Egalité des attributs de même nom (θ condition) et projection pour ne garder qu'un fois les attribut de même nom.

- $Sells2 \leftarrow \rho_{ban,ben,price}(Sells)$
- $Sells2 \bowtie Sells$

Division

$\pi_{(BarName,BeerName)}(Sells)/\pi_{BeerName}(Beers)$

Dépendances fonctionnelles et formes normales

Exemples de dépendances fonctionnelles

$BarName \rightarrow Bar@, licence, OpenDate$

Redondances

- $EnVrac(BeerName, BrewerName, Country, BarName, price)$
- $BarName, BeerName \rightarrow price$
- $BeerName \rightarrow BrewerName, Country$

Formes Normales

Mesure de la *r*edondance : 1NF, 2NF, 3NF, BCNF,...

SQL : DML

Interrogation

```
Select <attributs> From <tables> Where <conditions>
Group by <partition condition> Having <partition filter>
Order by <attributs>
```

Exemple 1

```
Select BeerName From Sells Where price > 2
group by BeerName Having count(*) = Select count(*) from
Bars
```

Exemple 2

```
Select BrewerName
From Sells natural join Beers
Where price > 10
```

SQL : DML

Update

```
UPDATE table_name SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

Update : exemple

```
UPDATE Sells SET price=price*1.1 WHERE price < 2;
```

Delete

```
DELETE FROM table_name WHERE some_column=some_value;
```

SQL : DDL

Schéma relationnel

- Création : `CREATE TABLE ... CREATE VIEW ...`
- Modification : `ALTER TABLE ...`
- Droit d'accès : `GRANT privilege ON something TO user`
- Catalogue : `SELECT user_tables FROM all_tables;`

Contraintes d'intégrité

Les plus simples

- Clés *primaires*
- Clés *étrangères*

Exemples

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

Embedded/ECA

Les contraintes les plus complexes

- Event Condition Action (trigger)
- A la charge du programmeur

SQL Embarqué

- Le SQL a vocation à être utilisé dans un programme
- Transférer des information du SGBD vers un programme
- JDBC / Hibernate

Notion de transaction

Table of Contents

Python : Select example

Select and print all beers in the db

```
1
2  #!/usr/bin/python
3
4  import sqlite3
5  from sqlite3 import Error
6
7  def select_all_beers(conn):
8      """
9      :param conn: the Connection object
10     :return:
11     """
12     cur = conn.cursor()
13     cur.execute("SELECT␣*␣FROM␣Beers")
14
15     rows = cur.fetchall()
16
17     for row in rows:
18         print(row)
```

Connection to db

Create connection

```
1  def create_connection(db_file):
2      """ create a database connection to the SQLite database
3          specified by the db_file
4      :param db_file: database file
5      :return: Connection object or None
6      """
7      try:
8          conn = sqlite3.connect(db_file)
9          return conn
10     except Error as e:
11         print(e)
12
13     return None
```

Create db, insert and print

Create, insert and print

```
1  def main():
2      database = "beer.db"
3
4      # create a database connection
5      conn = create_connection(database)
6      print("1. create beers :")
7      conn.execute("drop table beers");
8      conn.execute("create table beers(beerName char(20) primary
9      conn.execute("insert into beers values ('kro','KroBrewer')"
10     conn.execute("insert into beers values ('Leffe','Monast')")
11
12     print("2. Query all beers")
13     select_all_beers(conn)
```

Table of Contents

Pourquoi des transactions

SGBD

Un Système de Gestion de Bases de Données (SGBD) est un système qui fournit un accès efficace, pratique et multi-utilisateur à des ensembles de données (nombreuses) qu'il gère et stocke de manière persistante. 3 fonctions importantes et liées entre elles :

- garantir un état cohérent
- contrôle de la concurrence
- reprise après panne (récupération)

$\Longrightarrow$ transaction

Transaction définition

Une transaction est ...

... une action ou une suite d'actions demandées par un utilisateur (une application) qui lit et/ou met à jour les données de la base.

C'est l'unité *logique* de travail sur la base.
Une transaction $\Longrightarrow$ une ou plusieurs commandes SQl
1 seule commande SQl $\Longrightarrow$ plusieurs opérations.

Update : exemple

```
UPDATE Sells SET price=price*1.1 WHERE price < 2;
```

Transaction exemple

T_1 : modification des prix pour la bière 'La souris brune'

- p $\leftarrow$ Lire(Sells, BeerName='La souris brune', price)
- $p = p * 1.1$
- Ecrire(Sells,BeerName='La souris brune',price,p)

T_2 : supprimer la 'Brasserie des deux Souris' et ses bières

- $\{BeN\}$ $\leftarrow$Lire(Beers,BrewerName='Brasserie des deux Souris',BeerName)
- Supprimer(Beers,BrewerName='Brasserie des deux Souris')
- $\forall B \in \{BeN\}$ Supprimer(Sells,BeerName=B)

En cas d'execution parallèle ?

Transaction notions de bases

Deux fins possibles

- achevée avec succès : elle est dite validée, confirmée (committed).
- si elle échoue : elle est dite annulée, abandonnée (aborted).

Dans le dernier cas la base doit retourner à l'état cohérent précédent le début de la transaction. La transaction est annulée (rolled back).

Notation

- $L_{T_1}(X)$: lecture de l'object X par la transaction T_1
- $E_{T_1}(X)$: écriture de l'object X par la transaction T_1

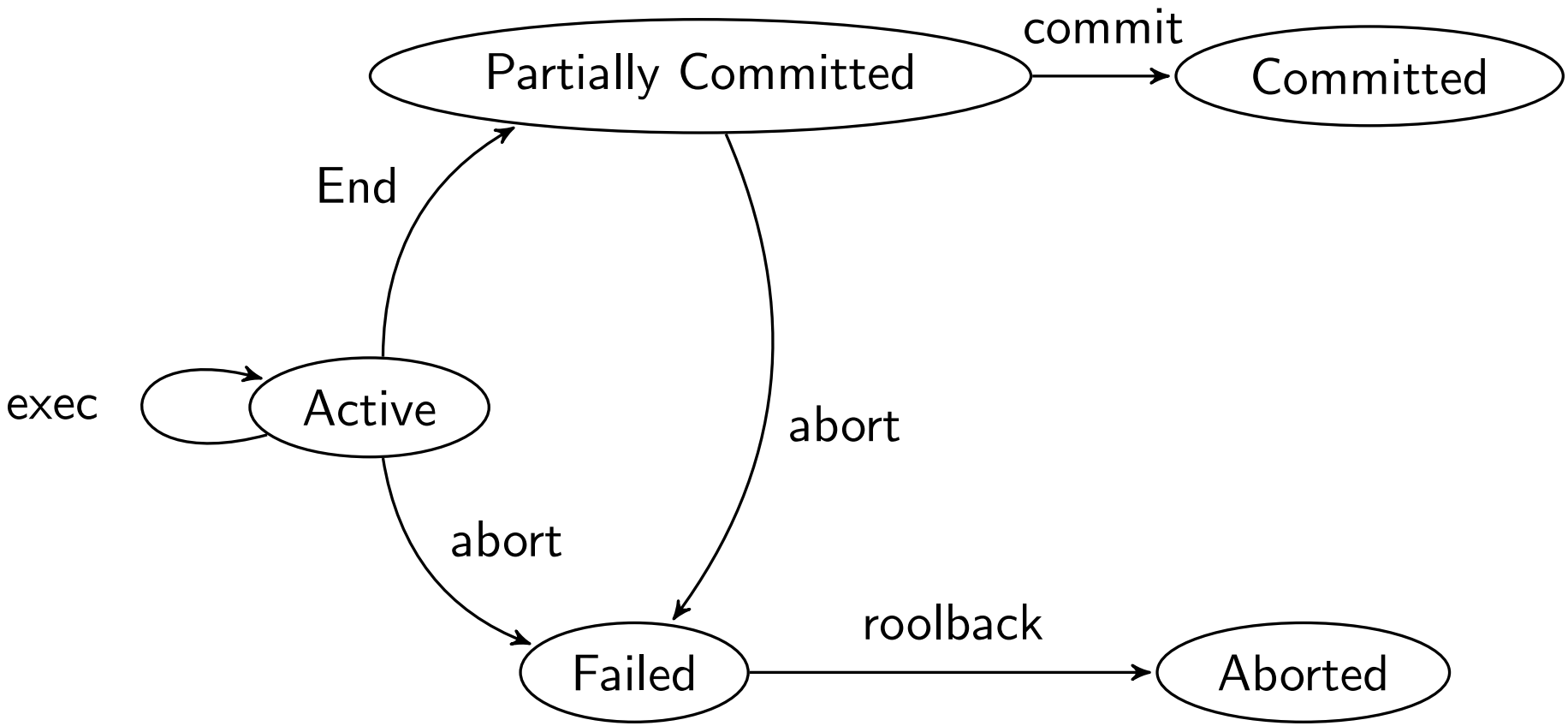
Raisons d'un échec

Il y a échec si

- risque d'incohérences dû aux accès concurrents (violation de sérialisation)
- violation des contraintes d'intégrité
- problème d'accès au support persistant
- annulation par l'utilisateur (une application)

Si un de ces problèmes apparaît la transaction doit être annulée, sinon elle peut être validée.

Graphe d'état d'une transaction



Les propriétés ACID

Le SGBD doit garantir des exécutions qui vérifient les propriétés suivantes :

- Atomicité
- Cohérence
- Isolation
- Durabilité

ACID

Atomicité

Une transaction doit être exécutée de manière atomique (tout ou rien). Elle doit être exécutée dans sa totalité ou annulée en totalité. Elle doit être considérée comme une unique action non divisible.

Cohérence

La transaction transforme la base d'un état cohérent en un autre état cohérent. Cette responsabilité peut être partagée entre le SGBD et les développeurs d'applications.

Isolation

Des transactions doivent pouvoir s'exécuter en parallèle de manière indépendante les unes des autres. ie: les effets partiels des transactions incomplètes ne doivent pas être visibles par les autres transactions.

Durabilité

Les effets d'une transaction validée sont inscrits de manière durable et ne doivent pas subir de perte suite à une défaillance.

Comment garantir ces propriétés

Contraintes d'intégrités

Elles garantissent la cohérence

Mécanisme de reprise

pour la durabilité et l'atomicité

Gestion de la concurrence

pour l'isolation

Table of Contents

Concurrence au sein d'un système

- La concurrence d'accès dans un environnement multi-utilisateurs permet une meilleure utilisation des ressources.
  - Simultanéité des accès
  - Compétition sur les données
- Les opérations d'E/S sont interfoliées pour assurer un accès concurrent aux données.
- Même si les transactions sont correctes l'entrelacement des opérations peut produire des résultats incorrects.

Problèmes liés à la concurrence

4 types de problèmes peuvent être provoqués par la concurrence :

- Pertes de mises à jour.
- Lectures sales
- Lectures non reproductibles
- Lectures fantômes

Pertes de mises à jour

Exécution en série :

		T_1	T_2
t_1		p ← Lire(X)	
t_2		$p = p * 1.1$	
t_3		Ecrire(X,p)	
t_4		Validation	
t_5			p ← Lire(X)
t_6			$p = 100$
t_7			Ecrire(X,p)
t_8			Validation

Perte de mise à jour

		T_1	T_2
t_1		p ← Lire(X)	
t_2			p ← Lire(X)
t_3			$p = 100$
t_4		$p = p * 1.1$	
t_5		Ecrire(X,p)	
t_6			Ecrire(X,p)
t_7			Validation
t_8		Validation	

Lectures sales / Lectures non reproductibles

Lectures sales :

		T_1	T_2
t_1		p ← Lire(X)	
t_2		$p = p * 1.1$	
t_3		Ecrire(X,p)	
t_4			p ← Lire(X)
t_5			$p = 100$
t_6			Ecrire(X,p)
t_7		Annulation / Ecrire(X,p)	
t_8			Validation

Lectures non reproductible :

		T_1	T_2
t_1		p ← Lire(X)	
t_2		$p = p * 1.1$	
t_3		Ecrire(X,p)	
t_4			p ← Lire(X)
t_5			$p = 100$
t_6			Ecrire(X,p)
t_7		p ← Lire(X)	
t_8		Validation	

Lectures fantômes

Lectures d'objets (tuple) qui ne devraient pas exister :

T_1	t_1	$S_1 \leftarrow$ Lire(Sells,price<2)
	t_2	
	t_3	
	t_4	$S_2 \leftarrow$ Lire(Sells,price>2)
	t_5	$S \leftarrow S1 \cap S_2$
	t_6	Si (S=$\emptyset$) alors Validation sinon Annulation

T_2	t_1	
	t_2	Ecrire(Sells,price=price*2)
	t_3	Validation
	t_4	
	t_5	
	t_6	

Calcul d'agrégats (count,avg,min,max,...)

Récapitulatif

Problèmes dû à la concurrence

4 types de problèmes peuvent être provoqués par la concurrence :

- Pertes de mises à jour. $E_{T_1}(x)$ suivit de $E_{T_2}(x)$ avant validation de T_1
- Lectures sales $E_{T_1}(x)$ suivit de $L_{T_2}(x)$ avant validation de T_1
- Lectures non reproductibles $L_{T_1}(x)$ suivit de $E_{T_2}(x)$ avant validation de T_1
- Lectures fantômes : type particulier de lecture sale sur des ensembles d'objets (prédicat, agrégation,...)

Détecter les conflits

- Soit $OLect(T)$ l'ensemble des objets lus par T
- Soit $OEcr(T)$ l'ensemble des objects écrits par T

Une transaction T_i peut être en conflit avec T_j si :

- $OEcr(T_i) \cap OLect(T_j) \neq \emptyset$
- $OEcr(T_i) \cap OEcr(T_j) \neq \emptyset$
- $OLect(T_i) \cap OEcr(T_j) \neq \emptyset$

Notion d'ordonnancement

Ordonnancement

Un ordonnancement est une séquence d'opérations d'un ensemble de transactions concurrentes qui préserve l'ordre des opérations dans chacune des transactions.

Ordonnancement séquentiel

Un ordonnancement où les opérations de chaque transaction sont exécutées de manières consécutives.

Ordonnancement non séquentiel

- t_1 T1 : $S_1 \leftarrow$ Lire(Sells,price¿2)
- t_2 T2 : Ecrire(Sells,price=price*2)
- t_3 T2 : Validation
- t_4 T1 : $S_2 \leftarrow$ Lire(Sells,price¿2)
- t_5 T1 : $S \leftarrow S1 \cap S_2$
- t_6 T1 : Si (S=$\emptyset$) alors Validation sinon Annulation

Ordonnancement séquentiel

- t_1 T2 : Ecrire(Sells,price=price*2)
- t_2 T2 : Validation
- t_3 T1 : $S_1 \leftarrow$ Lire(Sells,price¿2)
- t_4 T1 : $S_2 \leftarrow$ Lire(Sells,price¿2)
- t_5 T1 : $S \leftarrow S1 \cap S_2$
- t_6 T1 : Si (S=$\emptyset$) alors Validation sinon Annulation

Notion d'ordonnancement

Ordonnancement sérialisable

Un ordonnancement est sérialisable si les résultats qu'il produit peuvent être obtenues par un ordonnancement séquentiel

Ordonnancement cohérent

Un ordonnancement est cohérent si il est sérialisable

L'objectif du SGBD

Maximiser la concurrence d'accès en garantissant la cohérence.

Comment savoir si un ordonnancement est cohérent ?

On utilise un graphe de dépendance