R3.07
Bases de Données - Transactions

Département Informatique
IUT de Lannion
Université de Rennes 1

BUT Informatique

Plan du cours

Définitions

Définition (Transaction)

Une interaction atomique avec la base de données qui préserve la cohérence des données.

Atomique : **tous les effets ou aucun** d'une transaction doi(ven)t être visible(s) pour les autres utilisateurs. Une transaction doit faire passer une base d'un état cohérent à un autre état cohérent.

Exemple

```
debut-transaction
    update compte set
    balanceC=balanceC+delta
    where #compte = compteId ;

    insert into history values(compteId,
        agenceID, caisseId, delta, sysdate) ;

    update caisse set balance=balance + delta
    where #caisse = caisseId ;

    update agence set balanceA = balanceA + delta
    where #agence = agenceId ;
fin_transaction
```

Menaces et Vie d'une transaction

Problèmes de concurrence

- pertes d'opérations
- introduction d'incohérences
- verrous mortels (deadlock)

Panne de transaction

- erreur en cours d'exécution du programme applicatif
- nécessité de défaire les mises à jour effectuées

Panne système

- reprise avec perte de la mémoire centrale
- toutes les transactions en cours doivent être défaites

Panne disque

- perte de données de la base

Vie d'une transaction

- **vie sans histoire** : aucun événement ne vient perturber l'exécution de T même si cette exécution est temporairement suspendue. En fin de vie, toutes les actions prévues ont été exécutées.
- **"assassinat"** : un événement extérieur vient interrompre l'exécution de façon irrémédiable (panne, action du SGBD).
- **"suicide"** : au cours de son exécution la transaction détecte une condition qui fait que la poursuite de l'exécution devient impossible, elle arrête alors son exécution.

Propriétés d'une transaction : ACID

- **A**tomicité : toutes les mises à jour doivent être effectuées ou aucune
- **C**ohérence : la transaction doit faire passer la base de données d'un état cohérent à un autre
- **I**solation : les résultats d'une transaction ne sont visibles aux autres transactions qu'une fois la transaction validée
- **D**urabilité : les modifications dues à une transaction validée ne seront jamais perdues

Transaction : COMMIT et ABORT

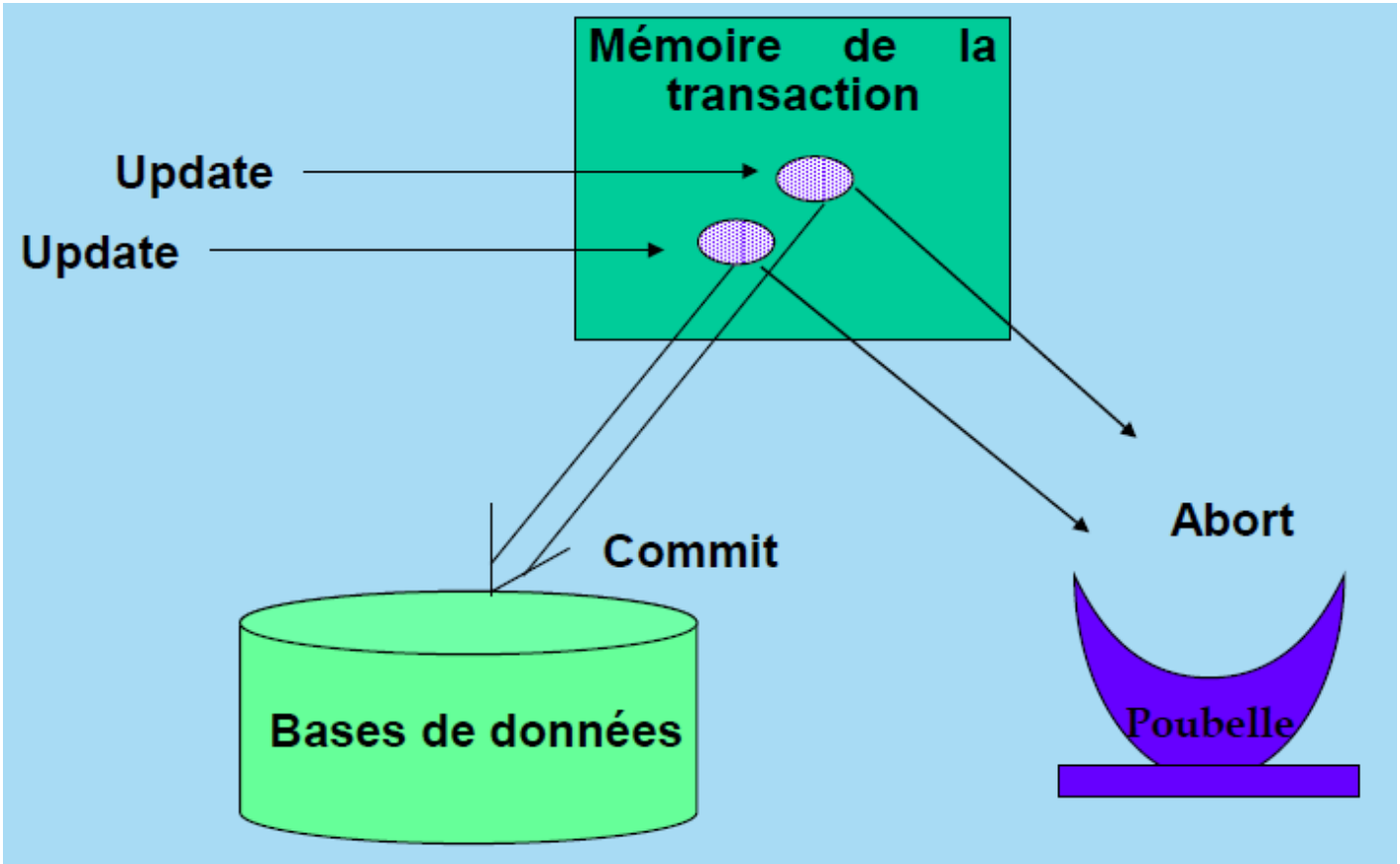
COMMIT :

- Provoque l'intégration réelle des mises à jour dans la base
- Relâche les verrous

ABORT :

- Provoque l'annulation des mises à jour
- Relâche les verrous
- Reprend la transaction

Transaction : COMMIT et ABORT



Plan : 2 - Concurrence

Position du problème

L'exploitation multi-utilisateurs d'une base de données doit être telle que :

- les utilisateurs puissent accéder aux données simultanément ;
- les résultats des traitements (requêtes, programmes) doivent être cohérents ;
- les pertes d'informations doivent être inexistantes.

Les transactions peuvent rentrer en interférence et, si aucune précaution n'est prise, des problèmes qui mettent en jeu l'intégrité des données peuvent se produire et doivent être évités.

Position du problème

Définition

Deux transactions sont concurrentes si elles accèdent en même temps aux mêmes données.

Des niveaux d'isolation sont définis en fonction de phénomènes, résultant de l'interaction entre les transactions concurrentes, qui ne doivent pas se produire à chaque niveau.

- lecture sale (dirty read) ou lecture impropre
- lecture non reproductible (non repeatable read)
- lecture fantôme (phantom read)

Position du problème

Un exemple de perte d'opération : une opération (en t5 et t7) annule de façon inappropriée une opération antérieure (t3).

temps	transac T1	état base	transac T2
t1	Lire(A)	A = 10	–
t2	–		Lire(A)
t3	A := A + 10		
t4	–		–
t5	–		A := A + 50
t6	Ecrire(A)	A=20	–
t7	–	A=60	Ecrire(A)

Table – Perte d'opération : perte en t3

Position du problème

temps	transac T1	état base	transac T2
t1	Lire(A)	A=10	–
t2	A :=A+20		–
t3	Ecrire(A)	A=30	–
t4	–		Lire(A) : 30
t5	**annulation**		

Table – Lecture impropre (incohérence) - donnée salie : donnée mise à jour mais non validée

Définition (Dirty read)

Une transaction lit des données écrites par une transaction concurrente non validée (non confirmée).

Position du problème

temps	transac T1	état base	transac T2
t1	Lire(A)	A=10	–
t2	–		Lire(A)
t3	A :=A+10		–
t4	Ecrire(A)	A=20	–
t5	–	–	Lire(A)

Table – Lecture non reproductible (incohérence)

Définition (lecture non reproductible)

Une transaction relit des données qu'elle a lu précédemment et trouve que les données ont été modifiées ou détruites par une autre transaction (validée depuis la lecture initiale) (non repeatable read)

Position du problème

temps	transac T1	état base	transac T2
t1		R1 : count(*) = 10	
t2	R1 : count(*) = 10		–
t3	–		insert into R1
t4	R1 : count(*) = 11		–

Table – Lecture fantôme

Définition (lecture fantôme)

Une transaction ré-exécute une requête renvoyant un ensemble de lignes satisfaisant une condition de recherche et trouve que l'ensemble des lignes satisfaisant la condition a changé du fait d'une autre transaction récemment validée (phantom read).

Plan : 2 - Concurrence

Solutions

On cherche à garantir la sérialisation [1] dans l'idéal.
Deux types de solutions ont été trouvées pour tenter de faire face à ces phénomènes :

- niveau d'isolement (isolation) d'une transaction : propriété d'une transaction qui voudrait que chaque transaction à tout instant perçoive la base dans un état cohérent ; quatre niveaux dans la norme SQL2 ont été définis pour répondre aux différents phénomènes.
- verrous posés sur les différentes ressources (n-uplets, tables, ...) et différentes politiques pour les utiliser (verrouillage à une phase, deux phases).

1. quand l'exécution concurrente des transactions fournit un résultat équivalent à **une** exécution correcte séquentielle.

Solutions I

Niveau (degré) d'isolation en SQL2

Ces niveaux d'isolation sont définis pour garantir ce qui **ne doit pas** arriver.

- degré 0 : pose de verrou court exclusif lors des écritures (*pas de perte de mise à jour*).
  Une transaction ne modifie pas de données salies par une autre transaction.
- degré 1 : pose de verrous longs exclusifs en écriture ; *cohérence des mises à jour*
  degré 0 + la transaction ne confirme pas ses changements avant sa fin.
- degré 2 : *cohérence des lectures individuelles* ; pose de verrous courts partagés
  degré 1 + la transaction ne lit pas de données salies.

Solutions II

Niveau (degré) d'isolation en SQL2

- degré 3 : *reproductibilité des lectures* ; pose de verrous longs en lecture ; assure la sérialisation
  degré 2 + d'autres transactions ne salissent pas de données lues par la transaction avant qu'elle ne soit terminée (*sérialisable*).

Solutions

Niveau d'isolement (degré croissant)	**Lecture impropre**	**Lecture non-reproductible**	**lecture fantôme**
Uncommited Read (Lecture de données non validées)	possible	possible	possible
Commited Read (Lecture de données validées)	impossible	possible	possible
Repeatable Read (Lecture reproductible)	impossible	impossible	possible
Sérialisable	impossible	impossible	impossible

Table – Incohérences possibles en fonction des niveaux d'isolement

Plan : 2 - Concurrence

Verrou mortel (deadlock)

Définition (Verrou mortel)

Situation dans laquelle des transactions sont bloquées, chacune attendant qu'une autre relâche une ressource pour pouvoir continuer.

temps	transac T1	transac T2
t1	Lock(T1 , G1, écriture)	–
t2	–	Lock(T2, G2, écriture)
t3	Lock(T1 , G2, écriture)	–
t4		Lock(T2, G1, lecture)
t5	T1 attend	T2 attend

Table – Verrou mortel (deadlock)

Verrou mortel (deadlock)

Résolution

- par ignorance
- par évitement : on détermine les transactions que l'on peut lancer sans danger d'interblocage ;
  - graphe d'état ou liste de ressources
  - performance
  - difficile si bdd répartie
- détection /résolution : à l'exécution on détecte les situations de blocage
  - graphe d'attente et cycle dans le graphe
  - si blocage : assassinat
- prévention : définir des critères de priorité

Plan : 3 - Une mise en œuvre (PostgreSQL)

Une mise en œuvre (PostgreSQL)

Généralités

Sous PostgreSQL, la mise en œuvre comprend (tiré de la documentation) :

- trois niveaux d'isolation ;
- un principe qui consiste à s'assurer que les données visibles d'une requête proviennent du même instant d'observation : toute table modifiée est recopiée préalablement dans un *fichier image avant*[2] qui permet de reconstituer les données si nécessaire.
- un système de verrouillage qui assure des mises à jour non destructives ; différents types de verrou existent.

Le niveau d'isolation par défaut de PostgreSQL est **Read Committed** et assure : **pas de lecture impropre, pas de pertes de mise à jour**, mais laisse la possibilité de lecture fantôme et de lecture non reproductible.

2. C'est une image de la base de données datant du moment où l'exécution de la requête commence.

Une mise en oeuvre (PostgreSQL)

Généralités

Transaction :

- Début explicite : `START TRANSACTION ;` / `BEGIN;`
- Fin explicite : `COMMIT WORK` , `ROLLBACK;`
- Début implicite : n'existe pas, contrairement à ORACLE par exemple.
- Fin implicite : fin de session (et annulation).
- Découpage des transactions :
  `SAVEPOINT <nom_point_repère>,`
  `ROLLBACK TO [SAVEPOINT] <nom_point_repère>.`

Plan : 3 - Une mise en œuvre (PostgreSQL)

Une mise en oeuvre (PostgreSQL)

Niveaux d'isolation

Fichier image avant : avant toute modification sur une page, la page est sauvegardée

- `ROLLBACK` facile à mettre en œuvre ;
- Les concurrents accèdent à l'*image avant*

Une mise en oeuvre (PostgreSQL)

Niveaux d'isolation

Niveaux d'isolation : `SET TRANSACTION` initialise les caractéristiques de la transaction actuelle

```
SET TRANSACTION mode_transaction [, ...]
SET SESSION CHARACTERISTICS
    AS TRANSACTION mode_transaction [, ...]
```

où `mode_transaction` est composé de :

```
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ |
                      READ COMMITTED |
                      READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```

Une mise en oeuvre (PostgreSQL) I

Mise en place des niveaux d'isolation

- `SET TRANSACTION ISOLATION LEVEL READ ONLY` :
  Assure la cohérence au niveau transaction : dans la transaction les lectures sont reproductibles.
  La **transaction** ne voit que les changements qui ont été validés au moment où elle démarre ; dans la transaction on ne peut pas utiliser `INSERT`, `DELETE`, `UPDATE`.
- `SET TRANSACTION ISOLATION LEVEL READ COMMITTED` :
  C'est l'option par défaut. Chaque requête (SELECT, SELECT imbriqué, clause WHERE) exécutée par la transaction ne voit que les données qui ont été validées avant que la **requête** commence. Cette option ne résoud pas les problèmes de lecture non reproductible ni de lecture fantôme.

Une mise en oeuvre (PostgreSQL)

Niveaux d'isolation

- `REPEATABLE READ`
  - Toute instruction de la transaction en cours ne peut voir que les lignes validées avant que la première requête ou instruction de modification de données soit exécutée dans cette transaction.
- `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` :
  - Toutes les requêtes de la transaction en cours peuvent seulement voir les lignes validées avant l'exécution de la première requête ou instruction de modification de données de cette transaction.
  - Si un ensemble de lectures et écritures parmi les transactions sérialisables concurrentes créait une situation impossible à obtenir avec une exécution en série (une à la fois) de ces transactions, l'une d'entre elles sera annulée avec un état `SQLSTATE` à `serialization_failure`.

Une mise en oeuvre (PostgreSQL)

Niveaux d'isolation

- Le standard SQL définit un niveau supplémentaire, `READ UNCOMMITTED`. Dans PostgreSQL, `READ UNCOMMITTED` est traité comme `READ COMMITTED`.

Plan : 3 - Une mise en œuvre (PostgreSQL)

Une mise en oeuvre (PostgreSQL)

Système de verrouillage

Il existe des verrous gérés par les commandes LDD et des verrous associés aux données (n-uplets et tables).

- Un verrou n'empêche pas les lectures : un écrivain ne peut empêcher un lecteur et un lecteur ne peut empêcher un écrivain ; un écrivain peut attendre un écrivain.
  Remarque : il ne s'agit pas forcément des mêmes données ou valeurs !
- Le verrou existe tant que la transaction n'est pas terminée.

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

Voici les modes de **verrous au niveau table** :

- **ACCESS SHARE (AS)**
  En conflit avec le mode verrou ACCESS EXCLUSIVE.
  Les commandes SELECT acquièrent un verrou de ce mode avec les tables référencées. En général, toute requête lisant seulement une table et ne la modifiant pas obtient ce mode de verrou.
- **ROW SHARE (RS)**
  En conflit avec les modes de verrous EXCLUSIVE et ACCESS EXCLUSIVE.
  La commande SELECT FOR UPDATE et SELECT FOR SHARE acquièrent un verrou de ce mode avec la table cible (en plus des verrous ACCESS SHARE des autres tables référencées mais pas sélectionnées FOR UPDATE/FOR SHARE).

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

- **ROW EXCLUSIVE (RX)**
  En conflit avec les modes de verrous SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE.
  Les commandes UPDATE, DELETE et INSERT acquièrent ce mode de verrou sur la table cible (en plus des verrous ACCESS SHARE sur toutes les autres tables référencées). En général, ce mode de verrouillage sera acquis par toute commande modifiant des données de la table.

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

- **SHARE UPDATE EXCLUSIVE (SUX)**
  En conflit avec les modes de verrous SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE. Ce mode protège une table contre les modifications simultanées de schéma et l'exécution d'un VACUUM.
  Acquis par VACUUM (sans FULL), ANALYZE, CREATE INDEX CONCURRENTLY, et quelques formes de ALTER TABLE.
- **SHARE (S)**
  En conflit avec les modes de verrous ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE. Ce mode protège une table contre les modifications simultanées des données.
  Acquis par CREATE INDEX (sans CONCURRENTLY).

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

- **SHARE ROW EXCLUSIVE (SRX)**
  En conflit avec les modes de verrous ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE. Ce mode protège une table contre les modifications concurrentes de données, et est en conflit avec elle-même, afin qu'une seule session puisse le posséder à un moment donné.
  Ce mode de verrouillage **n'est acquis automatiquement par aucune commande** PostgreSQL.

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

- **EXCLUSIVE (X)**
  En conflit avec les modes de verrous ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE. Ce mode autorise uniquement les verrous ACCESS SHARE concurrents, c'est-à-dire que seules les lectures à partir de la table peuvent être effectuées en parallèle avec une transaction contenant ce mode de verrouillage.
  Ce mode de verrouillage **n'est acquis automatiquement sur des tables par aucune commande** PostgreSQL.

Une mise en oeuvre (PostgreSQL) I

Modes de verrous au niveau table

- **ACCESS EXCLUSIVE (AX)**
  Entre en conflit avec tous les modes (ACCESS SHARE, ROW SHARE, ROW EXCLUSIVE, SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE et ACCESS EXCLUSIVE). Ce mode garantit que le détenteur est la seule transaction à accéder à la table de quelque façon que ce soit. Acquis par les commandes ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER et VACUUM FULL. C'est aussi le mode de verrou par défaut des instructions LOCK TABLE qui ne spécifient pas explicitement de mode de verrouillage.

Une mise en oeuvre (PostgreSQL)

Verrou table

La commande LOCK TABLE permet de déclarer explicitement un verrou table.

```
LOCK [ TABLE ] [ ONLY ] nom [ * ] [, ...]
     [ IN mode_verrou MODE ] [ NOWAIT ]

<mode_verrou> = ACCESS SHARE | ROW SHARE |
  ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE |
  SHARE | SHARE ROW EXCLUSIVE | EXCLUSIVE |
  ACCESS EXCLUSIVE
```

Avec l'option NOWAIT, LOCK TABLE n'attend pas que les verrous conflictuels soient relâchés : si le verrou indiqué ne peut être acquis immédiatement sans attente, la transaction est annulée.

Si l'option est omise, PostgreSQL attend que l'autre verrou soit relâché, pose le verrou demandé et rend le contrôle.

Une mise en oeuvre (PostgreSQL)

Modes de verrou conflictuels

Verrou demandé	Commande	Verrou déjà détenu							
		AS	RS	RX	SUX	S	SRX	X	AX
AS	SELECT								X
RS	SELECT FOR UPDATE/FOR SHARE							X	X
RX	UPDATE, DELETE, INSERT					X	X	X	X
SUX	VACCUM, ANALYZE , CREATE INDEX CONCURRENTLY, ALTER TABLE (qques formes)				X	X	X	X	X
S	CREATE INDEX			X	X		X	X	X
SRX	LOCK TABLE IN SHARE ROW EXCUSIVE MODE			X	X	X	X	X	X
X	LOCK TABLE IN EXCUSIVE MODE		X	X	X	X	X	X	X
AX	ALTER TABLE, DROP TABLE, TRUNCATE, REINDEX, CLUSTER, VACUUM FULL	X	X	X	X	X	X	X	X

NOTA : Tous les verrous de niveau table peuvent être acquis par la commande `LOCK ... TABLE IN ...`

Une mise en oeuvre (PostgreSQL) I

Verrous n-uplet

En plus des verrous au niveau table, il existe des **verrous au niveau n-uplet**, qui peuvent être des verrous **exclusifs** ou **partagés**. C'est en réalité un peu plus compliqué que cela, mais on se contentera de ce niveau pour ce cours.

Un **verrou exclusif** est acquis **automatiquement** en faveur d'une transaction pour **chaque** n-uplet qui est mis en cause dans l'une des commandes : INSERT, UPDATE, DELETE, SELECT . . .FOR UPDATE.

Verrou Ligne Exclusif : pas de modification par une autre transaction tant que la transaction qui possède le verrou n'est pas terminée.

Une mise en oeuvre (PostgreSQL) I

Verrou n-uplet

Verrou Ligne Partagé : obtenu par un SELECT . . .FOR SHARE, il n'empêche pas les autres transactions d'obtenir le même verrou partagé.

Néanmoins, aucune transaction n'est autorisée à mettre à jour, supprimer ou verrouiller exclusivement une ligne dont une autre transaction a obtenu un verrou partagé. Toute tentative de le faire bloque tant que les verrous partagés n'ont pas été enlevés.

La durée de vie d'un verrou n-uplet est celle de la transaction.

Verrous sur les pages

En plus des verrous tables et lignes, les verrous partagés/exclusifs sur les pages sont utilisés pour contrôler la lecture et l'écriture des pages de table dans l'ensemble des tampons partagées.

Ces verrous sont immédiatement relâchés une fois la ligne récupérée ou mise à jour. Les développeurs d'applications ne sont normalement pas concernés par les verrous au niveau page mais nous les mentionnons dans un souci d'exhaustivité.

Transactions depuis un langage de programmation d'applications

Exemple de mise en œuvre avec ADOdb (Abstraction SGBD sous PHP)

- Début de transaction : `BeginTrans` ou `StartTrans`;
- Fin : `RollbackTrans` , `CommitTrans` , `CompleteTrans`;
- Niveau d'isolation : `SetTransactionMode`.

Bibliographie I


Pierre DELMAL.
SQL2-SQL3, Applications à ORACLE, 3ème Edition.
De Boeck Université, 2001, ISBN 2-8041-3561-6.


S. Miranda, J.-M. Busta
L'art des bases de données, tome 2 : les bases de données relationnelles
Eyrolles, 1986.


Jacques SIROUX.
Cours de Bases de Données - DUT Informatique - IUT Lannion


Remi EYRAUD.
Base de Données.
https://pageperso.lis-lab.fr/~remi.eyraud/WP/?page_id=129