TP N°2 : La journalisation des transactions des bases de données

Objectif : Un journal est une séquence d'enregistrements décrivant les mises à jour effectuées par les transactions. C'est l'historique d'une exécution sur fichier séquentiel. Un journal est dit « physique » s'il garde la trace des modifications au niveau octet à l'intérieur des pages

Ex: <Ti, numPg, Depl, Long, img_avant, img_après>

Un journal est dit « logique » s'il garde la trace de la description de haut niveau des opérations de mise à jour/ **Ex: « insérer le tuple x dans la table T et mettre à jour les index »**

Exercice 1 : Quelles données sont validées après les opérations suivantes ?

```
ASET AUTOCOMMIT =0 ;
INSERT INTO R values (5, 6);
SAVEPOINT my_savepoint_1;
INSERT INTO R values (7, 8);
SAVEPOINT my_savepoint_2;
INSERT INTO R values (9, 10);
ROLLBACK TO my_savepoint_1;
INSERT INTO R values (11, 12);
INSERT INTO R values (23, 6);
```

```
SET AUTOCOMMIT=0;
START TRANSACTION;
SAVEPOINT sp1;
INSERT INTO villes(cp, nom, ville) VALUES('14000','TIARET', 'TIARET');
SAVEPOINT sp2;
INSERT INTO villes(cp, nom, ville) VALUES('14002','SOUGEUR', 'TIARET');
ROLLBACK TO SAVEPOINT sp2;
COMMIT;
SELECT * FROM villes;
```

Exercice 2 : Triggers pour enregistrer l'historique des modifications d'une table de BD.
Certaines bases de données proposent des fonctions PITR (Point in time recovery) pour retrouver l'état de la base de donnée à un instant précis, ou bien des systèmes de versions qui permettent de tracer l'historique d'une ligne dans une table. Ici, nous verrons un exemple simple d'utilisation de triggers pour enregistrer un historique des modifications d'une table.

a) Création :

```
CREATE TRIGGER Hist AFTER UPDATE ON Compte
FOR EACH ROW
BEGIN
  INSERT INTO JournalCompte(operation, date, Utilisateur,……)
  VALUES('update', NEW.NumCompte, NOW(), USER, OLD.sole,  NEW.solde);
END;
```

b) Gestion De Triggers :

DROP TRIGGER nomtrigger;
ALTER TABLE nomtable **DISABLE ALL TRIGGERS**;
ALTER TRIGGER nomtrigger **ENABLE**;
ALTER TABLE nomtable **ENABLE ALL TRIGGERS;**
ALTER TRIGGER nomtrigger **DISABLE**;
Show Triggers;
SHOW TRIGGERS FROM BD**;**
SELECT * FROM
INFORMATION_SCHEMA.TRIGGERS WHERE TRIGGER_SCHEMA='Nom BD'**;**

Exercice 02 : Une transaction doit préserver la cohérence de la base de données : (i) les contraintes d'intégrités, (ii) les règles métiers définies et (iii) les règles complexes. La programmation des ces règles c'est la responsabilité du développeur/DBA.

Implémenter la règle métier suivante :

☞ Le montant maximum peut-on retirer au distributeur doit être inférieur à 25000 ,00 DA.

Commandes MySQL :

```
SET AUTOCOMMIT = 0
```

```
select @@autocommit ;
ou
select @@session.autocommit ;
```

```
SELECT ...FOR UPDATE ...
LOCK TABLES table ...{READ | WRITE } ...
```

```
UNLOCK TABLES
```

MySQL: Le choix de moteur

```
mysql> SHOW ENGINES;
+------------+---------+------------------------------------------------------------
| Engine     | Support | Comment
+------------+---------+------------------------------------------------------------
| InnoDB     | YES     | Supports transactions, row-level locking, and foreig
| MRG_MYISAM | YES     | Collection of identical MyISAM tables
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it di
| CSV        | YES     | CSV storage engine
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary ta
| FEDERATED  | NO      | Federated MySQL storage engine
| ARCHIVE    | YES     | Archive storage engine
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performanc
+------------+---------+------------------------------------------------------------
8 rows in set (0.00 sec)
```

```
1 SET storage_engine=NomDuMoteur;
2
```

mysql> SET storage_engine=InnoDB;

MyISAM **Beaucoup lectures / Recherche textuelle**
InnoDB **Read+Write / Transactions / Accès clé primaire**
NDB **Petites Transactions**, traitements parallèles
MEMORY **Uniquement en mémoire**

```
1 /* A la création de la table */
2 CREATE TABLE maTable(
3 ...
4 )ENGINE=MonMoteurDeStockage;
5
6 /* En modifiant une table déjà créée */
7 ALTER TABLE maTable ENGINE=UnAutreMoteur;
8
```

LE VERROUILLAGE DE TABLE

Les commandes LOCK et UNLOCK permettent de verrouiller et de déverrouiller une ou plusieurs tables en lecture ou en lecture/écriture.

Syntaxes

LOCK TABLES nom_de_table verrouillage [, nom_de_table verrouillage];
UNLOCK TABLES

- **Exemple**

Ouvrez deux sessions clients (mysql et MySQL Query Browser par exemple).

Utilisateur	Autre utilisateur
SELECT * FROM villes; SET AUTOCOMMIT=0; START TRANSACTION; **LOCK TABLES** villes **READ**; UPDATE villes SET nom_ville = 'Marsiglia' WHERE cp = '13000'; **UNLOCK TABLES**; COMMIT;	SELECT * FROM villes; -- OK UPDATE villes SET nom_ville = 'Marsilia' WHERE cp = '13000'; -- KO

Utilisateur	Autre utilisateur
SELECT * FROM villes; SET AUTOCOMMIT=0; START TRANSACTION; **LOCK TABLES** villes **WRITE**; UPDATE villes SET nom_ville = 'Marsiglia' WHERE cp = '13000'; **UNLOCK TABLES**; COMMIT;	SELECT * FROM villes; -- KO UPDATE villes SET nom_ville = 'Marsilia' WHERE cp = '13000'; -- KO

- **Syntaxe**

```
SELECT * FROM nomDeTable WHERE condition FOR UPDATE;
```

- **Exemple**

Ouvrez deux sessions clients (mysql et MySQL Query Browser par exemple).

Utilisateur	Autre utilisateur
SET AUTOCOMMIT=0; START TRANSACTION; SELECT * FROM pays WHERE id_pays = '033' **FOR UPDATE**; UPDATE pays SET nom_pays = 'FR' WHERE id_pays = '033'; COMMIT;	 SET AUTOCOMMIT=0; START TRANSACTION; UPDATE pays SET nom_pays = 'fr' WHERE id_pays = '033'; -- Attente bloquante (*) COMMIT;

(*) UPDATE sur un autre pays c'est OK.