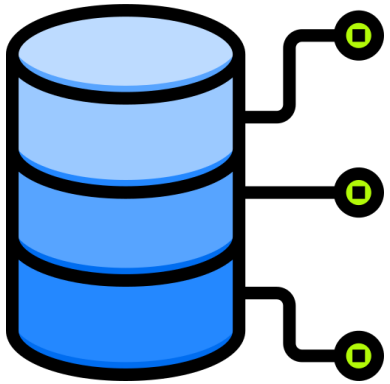
UF2: Tema 7
CONCURRENCIA TRANSACCIONAL



Desenvolupament d'aplicacions Multiplataforma

TRANSACCIÓ

És un **conjunt d'instruccions que han de ser executades com a una única**. Si algun moment una falla, cal poder desfer els canvis i deixar la base de dades al seu estat original.

PROCÉS:

1. *Començar la transacció*
2. *Executar un conjunt de manipulacions i/o consultes*
3. *Si no hi ha cap error, confirmar (commit) la transacció i finalitzar-la*
4. *Si hi ha algun error, desfer (roll back) la transacció i finalitzar-la*

En mySQL hi han dos motors d'emmagatzematge:

- **MYISAM**: No suporta transaccions.
- **INNODB**: Sí suporta transaccions.

ACID

- **Atomicitat**: És l'habilitat d'un SGBD de **garantir** que, o són **executades totes les tasques que intervenen en una transacci**ó, o bé no se n'executa cap.
- **Consistencia**: És refereix al fet que la base de dades ha d'estar en un estat quan acaba una transacció que ha de ser coherent amb l'estat que tenia en començar la transacció.
- **Aïllament:** Cap operació externa a la transacció podrà veure les dades en un estadi intermedi.
- **Durabilitat**: Garantia que un cop que un usuari ha estat notificat de l'èxit d'una transacció, aquesta persistirà, no pot ser desfeta. Molts SGBD implementen la durabilitat escrivint totes les transaccions en un registre que pot ser tornat a executar per recrear l'estat del sistema just abans de la fallada.

Exemple:

- **Atomicitat**: Una transferència de fons es pot completar o bé pot fallar per multitud de raons, però l'atomicitat garanteix que un compte no li seran retirats els fons si l'altre compte no els rep.
- **Consistencia**: Si una restricció d'integritat planteja que tots els comptes han de tenir un balanç positiu, aleshores, qualsevol transacció que doni un balanç negatiu del compte serà avortada.
- **Aillament:**Un treballador d'un banc pot veure els fons transferits d'un compte o de l'altre, però mai quan ja s'havien extret d'un compte i encara no s'havien carregat a l'altre.
- **Durabilitat**: Ha de quedar comitejada.

AUTOCOMMIT

WorkBench per defecte emmagatzema de forma persistent a la base de dades cada instrucció que es fa sempre que no estigui dins d'una transacció.

Té AUTOCOMMIT = 1 (true)

Per assignar valor al autocommit:

SET AUTOCOMMIT *= 0 -- false*

Per veure el valor actual d'autocommit:

SHOW global VARIABLES *like '%commit%'; -- per no veure totes les del sistema*

START TRANSACTION / BEGIN

- Instrucció que serveix per iniciar una transacció.
- Es dona per acabada quan es fa un COMMIT o ROLLBACK.
- Permet l'atomicitat.

```
BEGIN;
UPDATE Cuenta SET balance = balance - 100 WHERE idCliente = 3;
UPDATE Cuenta SET balance = balance + 100 WHERE idCliente =8;
COMMIT
```

NOTA: Cal que la taula hagi estat creada amb motor INNODB;

NOTA: DDL no permet model transaccional.

ROLLBACK / COMMIT

ROLLBACK

Si fem algun canvi en alguna taula a nivell d'estructura o dades i volem tornar a enrere o revocar la última instrucció executada.

insert into client values (34,'Gerard Martínez');

ROLLBACK; ← Retira totes les instruccions fins al COMMIT.

COMMIT

Si volem confirmar per sempre la instrucció executada sense donar opció a fer un ROLLBACK.

UPDATE client SET nom='Marta González' WHERE codi = 34;

COMMIT; ← Confirma les instruccions executades.

SAVEPOINT – ROLLBACK TO SAVEPOINT ‘x’

- SAVEPOINT <nom>
- ROLLBACK TO SAVEPOINT <nom>

Exemple:

```
BEGIN;
insert into viatge (avio, dataV,fila,col) values ('AV1','2020-10-18',1,1);

savepoint a;

insert into viatge (avio, dataV,fila,col) values ('AV1','2020-10-18',1,2);
insert into viatge (avio, dataV,fila,col) values ('AV1','2020-10-18',1,3);
rollback to savepoint a;
```

SAVEPOINT – ROLLBACK TO SAVEPOINT ‘x’

```sql
DROP DATABASE IF EXISTS test;
CREATE DATABASE test CHARACTER SET utf8mb4;
USE test;

CREATE TABLE producto (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  precio DOUBLE
);

INSERT INTO producto (id, nombre) VALUES (1, 'Primero');
INSERT INTO producto (id, nombre) VALUES (2, 'Segundo');
INSERT INTO producto (id, nombre) VALUES (3, 'Tercero');
```

```sql
-- 1. Comprobamos las filas que existen en la tabla
SELECT *
FROM producto;

-- 2. Ejecutamos una transacción que incluye un SAVEPOINT
START TRANSACTION;
INSERT INTO producto (id, nombre) VALUES (4, 'Cuarto');
SAVEPOINT sp1;
INSERT INTO producto (id, nombre) VALUES (5, 'Cinco');
INSERT INTO producto (id, nombre) VALUES (6, 'Seis');
ROLLBACK TO sp1;

-- 3. ¿Qué devolverá esta consulta?
SELECT *
FROM producto;
```

RELEASE SAVEPOINT ‘x’

- SAVEPOINT <nom>
- Elimina un SAVEPOINT creat.

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

InnoDB

- Per crear una taula: **ENGINE = INNOBD**.
- Motor d'emmagatzematge transaccional (segons ACID).
- Pot arribar a realitzar bloquejos de nivell de fila. Aquestes característiques augmenten el rendiment i la capacitat de gestionar diversos usuaris simultanis.
- Si la nostra aplicació fa un alt ús de INSERT i UPDATE notarem un augment del rendiment en comparació amb MyISAM.

MyISAM

- Per crear una taula **ENGINE = MYISAM**.
- No utilitza el log de registre de transaccions.
- No permet transaccions. Rollback no té efectes.
- Permet bloquejos de taula.

¿Quin motor es millor?

- **¿Si la teva taula rep molts *INSERT*, *UPDATE* y *DELETE* comparats amb SELECTS?**
  - InnoDB (alerta accessos a disc)
- **¿Es un problema memòria RAM?**
  - MyISAM (no utilitza el log de transaccions)
- **¿Control de transaccions?**
  - InnoDB

Instruccions varies

- **SHOW ENGINES:** Mostrar els motors d'emmagatzemament disponibles.

Engine	Support	Comment	Transactions	XA	Savepoints
MEMORY	YES	Hash based, stored in memory, useful for temp...	NO	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
FEDERATED	NO	Federated MySQL storage engine	NULL	NULL	NULL
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
InnoDB	DEFAULT	Supports transactions, row-level locking, and fo...	YES	YES	YES
ndbinfo	NO	MySQL Cluster system information storage engine	NULL	NULL	NULL
BLACKHOLE	YES	/dev/null storage engine (anything you write to ...	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
ndbcluster	NO	Clustered, fault-tolerant tables	NULL	NULL	NULL

- **SHOW CREATE TABLE nomTaula**:

Table	Create Table
planeta	CREATE TABLE `planeta` (`nombre` varchar(20) NOT NULL, `massa` decimal(6,2) NOT NULL, `lunas` tinyint DEFAULT NULL, `tempMedia` int DEFAULT NULL, PRIMARY KEY (`nombre`)) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb

ACCES CONCURRENT A LES DADES

TIPUS D'ACCESSOS

DIRTY READ:

Succeeix quan una segona transacció llegeix dades que estan sent modificades per una transacció abans que faci COMMIT

Transacción 1	Transacción 2
`UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;`	
	`SELECT saldo FROM cuentas WHERE id = 1;`
`ROLLBACK`	

TIPUS D'ACCESSOS

NON REPEATABLE READ:

- Es produeix quan una transacció consulta la mateixa dada dues vegades durant l'execució de la transacció i la segona vegada troba que el valor de la dada ha estat modificat per una altra transacció.

Transacción 1	Transacción 2
`SELECT saldo FROM cuentas WHERE id = 1;`	
	`UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;`
`SELECT saldo FROM cuentas WHERE id = 1;`	

TIPUS D'ACCESSOS

Phantom Read:

- Aquest error passa quan una transacció executa dues vegades una consulta que torna un conjunt de files i en la segona execució de la consulta apareixen noves files en el conjunt que no existien quan es va iniciar la transacció.

Transacción 1	Transacción 2
`SELECT SUM(saldo) FROM cuentas;`	
	`INSERT INTO cuentas VALUES (4, 3000);`
`SELECT SUM(saldo) FROM cuentas;`	

LOCK TABLE / UNLOCK TABLE

Hi ha moments en què un tros de codi ha de ser executat simultàniament sense distorsionar els seus resultats i amb concurrència.

Per exemple, suposem un sistema de desplaçament on cada torn ha de tenir ser consecutiu, i les sol·licituds es poden fer des de més d'un punt.

Per a això utilitzem un comptador que es troba en una taula d'una base de dades:

```
UPDATE tbl_contador SET valor = valor + 1; //incrementar el contador
SELECT valor FROM contador; //retornar el contador assignat
```

Però què passa si dos o més usuaris sol·liciten torns al mateix temps?

```
UPDATE tbl_contador SET valor = valor + 1; //pasa a 1
UPDATE tbl_contador SET valor = valor + 1; //pasa a 2
SELECT valor FROM contador; //retorna 2
SELECT valor FROM contador; //retorna 2
```

¿Què podem fer?

BLOQUEIG TIPUS READ: *LOCK TABLE <nombreTb> READ;*

- L'usuari propietari del bloqueig i la resta d'usuaris només podran llegir la taula bloquejada

BLOQUEIG TIPUS WRITE: *LOCK TABLE <nombreTb> WRITE*

- L'usuari propietari del bloqueig serà l'únic que podrà llegir o modificar la taula bloquejada. La resta quedaran a l'espera de que la taula quedi desbloquejada.
- El més restrictiu.

SOLUCIÓ A L' ANTERIOR CAS:

```
LOCK TABLES tbl_contador WRITE;
UPDATE tbl_contador SET valor = valor + 1; //incrementar el contador
SELECT valor FROM contador; //retornar el contador asignado
UNLOCK TABLES;
```

- SHOW TABLE STATUS;

Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free	Auto_increment	Create_time
grupo	InnoDB	10	Dynamic	2	8192	16384	0	0	0	NULL	2020-11-13 19:34:55
tienda	InnoDB	10	Dynamic	5	3276	16384	0	16384	0	NULL	2020-11-13 19:35:04
vehiculos	InnoDB	10	Dynamic	8	2048	16384	0	16384	0	NULL	2020-11-13 19:49:18

- SHOW PROCESSLIST;

Action Output

#	Time	Action	Message	Duration / Fetch
5	21:36:58	insert into vehiculos (1) values ('matricula1')	Error Code: 1064. You have an error in your SQL syntax; check the manual that corres...	0.047 sec
4 6	21:37:03	insert into vehiculos (matricula) values ('matricula1')	Running...	?