Progettazione Fisica e SQL

Ing. Alessandro Pellegrini, PhD
pellegrini@diag.uniroma1.it

Da qui in poi...

- ...la documentazione ufficiale è vostra amica!

- Per SQL (DBMS e Query):
  - https://dev.mysql.com/doc/refman/8.0/en/

- Per la programmazione dei client in C:
  - https://dev.mysql.com/doc/refman/8.0/en/c-api.html

- Per MySQL Workbench:
  - https://dev.mysql.com/doc/workbench/en/

Stored Procedures e Transazioni

Creazione Stored Procedure

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement
```

Un esempio

```
set autocommit = 0;

create procedure `assign`(in var_matricola varchar(45), out var_token varchar(128))
begin
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

    set transaction isolation level repeatable read;
    start transaction;
    if ... then
        signal sqlstate '45001' set message_text = "Si è verificata una condizione di errore";
    end if;
    commit;
end
```

SQL States

- Alcune procedure SQL hanno la necessità di fornire al DBMS (che eventualmente lo propagherà al client chiamate) un codice che fornisca delle informazioni sul successo o sul fallimento dell'esecuzione—una sorta di valore di ritorno.
- Gli SQL States sono codici di 5 byte (i primi due per la classe)
- Alcuni esempi:
  - 3F000: invalid schema name
  - 30000: invalid SQL statement identifier
  - 23000: integrity constraint violation
  - 22012: data exception: division by zero
  - 45000: unhandled user-defined exception

Livelli di Isolamento

- **READ UNCOMMITTED**
- Gli statement SELECT sono eseguiti in modalità non bloccante, ma è possibile che venga utilizzata una versione precedente di una riga. Pertanto, utilizzando questo livello di isolamento, le letture possono non esere coerenti (dirty read).
- Questo fenomeno si verifica poiché una transazione può leggere dati da una riga aggiornata da un'altra transazione che non ha ancora eseguito l'operazione di commit.

Dirty Reads

T1	T2
SELECT age FROM users WHERE id = 1;	
	UPDATE users SET age = 21 WHERE id = 1;
SELECT age FROM users WHERE id = 1;	
	ROLLBACK;

Livelli di Isolamento

- **READ COMMITTED**
- Il DBMS acqusisce un lock per ogni dato che viene letto o scritto. I lock associati ai dati che sono stati aggiornati in scrittura sono mantenuti fino alla fine della transazione, mentre i lock associati ai dati acceduti in lettura sono rilasciati alla fine della singola lettura.

- Possono verificarsi anomalie di tipo *unrepeatable reads*.

Unrepeatable Reads

T1	T2
SELECT * FROM users WHERE id = 1;	
	UPDATE users SET age = 21 WHERE id = 1;
	COMMIT;
SELECT * FROM users WHERE id = 1;	
COMMIT;	

Livelli di Isolamento

- **REPEATABLE READ**
- Con questo livello di isolamento, vengono mantenuti i lock sia dei dati acceduti in lettura sia in scrittura fino alla fine della transazione. Non vengono però gestiti i *range lock*, pertanto possono verificarsi anomalie di tipo *phantom read*.

- Le *phantom read* sono associate ad inserimenti che avvengono in concorrenza.

- Nelle versioni più nuove di InnoDB, è il livello di isolamento predefinito.

Phantom Reads

T1	T2
SELECT * FROM users WHERE age BETWEEN 10 AND 30;	
	INSERT INTO users(id,name,age) VALUES (3, 'Bob', 27);
	COMMIT;
SELECT * FROM users WHERE age BETWEEN 10 AND 30;	
COMMIT;	

Livelli di Isolamento

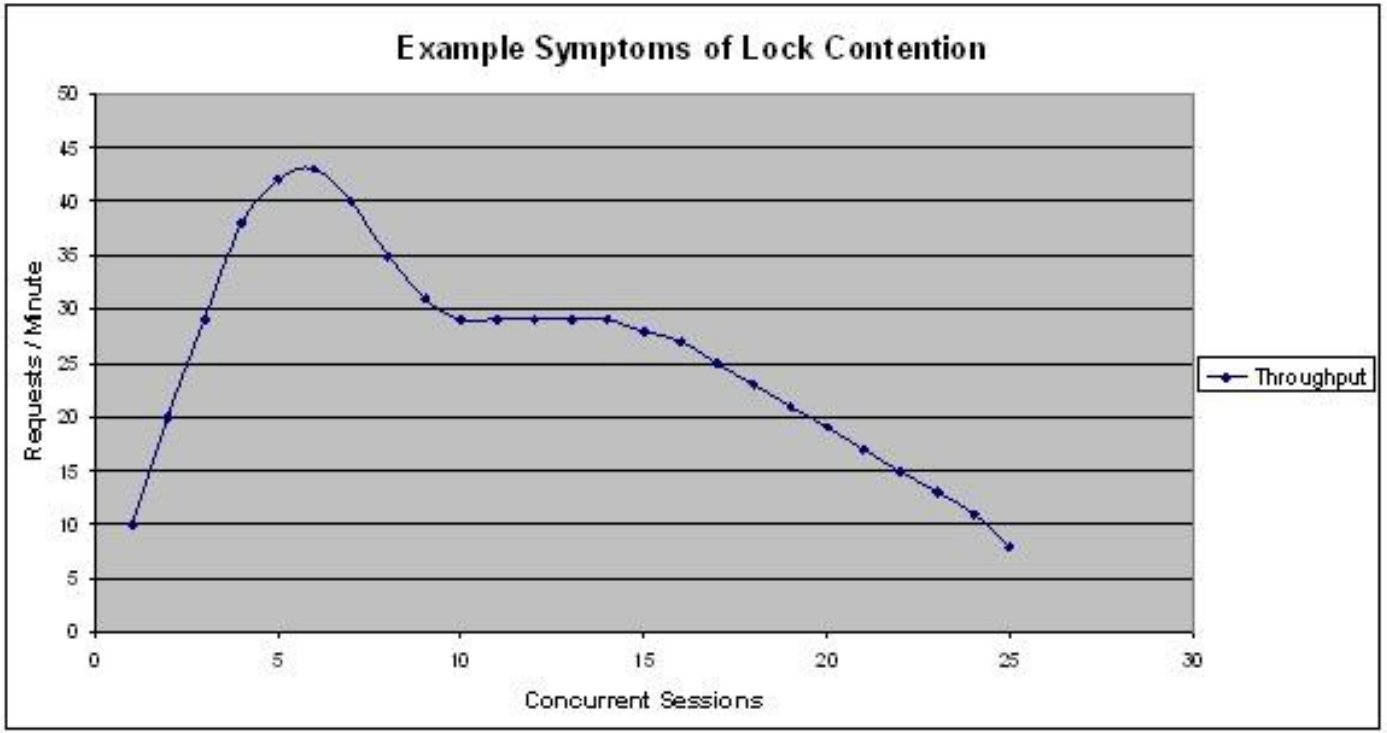
- **SERIALIZABLE**
- Tutti i lock vengono mantenuti fino alla fine della transazione e ogni volta che una SELECT utilizza uno specificatore di tipo WHERE, viene acquisito anche il *range lock*.

- In sistemi non basati su lock, questo livello di isolamento può essere implementato mediante il concetto di *read/write set* o in generale di *multiversion concurrency control*.

Quale scegliere?

1. Per molte applicazioni, la maggior parte delle transazioni possono essere costruite in maniera tale da non richiedere l'utilizzo di livelli di isolamento molto alti (ad esempio SERIALIZABLE), riducendo l'overhead dovuto ai lock
2. Lo sviluppatore deve accertarsi con cautela che le modalità di accesso delle transazioni non causino bug software dovuti alla concorrenza e al rilassamento dei livelli di isolamento.
3. Se si utilizzano unicamente alti livelli di isolamento, la probabilità di *deadlock* cresce notevolmente.

Effetti della contesa sui lock



Modalità di accesso transazionali

- READ ONLY: la transazione si limita alla lettura di dati
- READ WRITE: la transazione legge e scrive dati

- Dare queste informazioni al DBMS consente di generare dei piani di esecuzione ottimizzati dal punto di vista dell'acquisizione dei lock
- Sono informazioni ortogonali al livello di isolamento richiesto

Come marcare le transazioni

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [,transaction_characteristic] ...

transaction_characteristic: {
    ISOLATION LEVEL level
  | access_mode
}
level: {
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
}
```

```
access_mode: {
     READ WRITE
   | READ ONLY
}
```

Nel caso in cui la marcatura non sia globale, le caratteristiche si riferiscono unicamente alla successiva transazione nella sessione. In seguito, verranno ripristinati i valori globali anche per la sessione in corso.