PERCONA

A.C.I.D., Transactions and deadlocks for programmers

Marcos Albe – Support Engineer - Percona

Percona University Buenos Aires - February 2013

About myself

- Marcos Albe
- Support engineer at Percona
- Beer connoisseur

Agenda

- A.C.I.D. fundamentals
- Transactions fundamentals
- Why programmers should care
- What to do then...
- Questions

A.C.I.D. fundamentals

A

Atomicity

Transactions succeed or fail **as a whole**

Atomicity requires that each transaction is "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes.

A.C.I.D. fundamentals

C

Consistency

Transactions move from **one valid state to the next**

The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers, and any combination thereof.

A.C.I.D. fundamentals

I

Isolation

Changes in one transaction **don't affect other transactions**

The isolation property ensures that the concurrent execution of transactions results in a system state that could have been obtained if transactions are executed serially, i.e. one after the other.

A.C.I.D. fundamentals

D

Durability

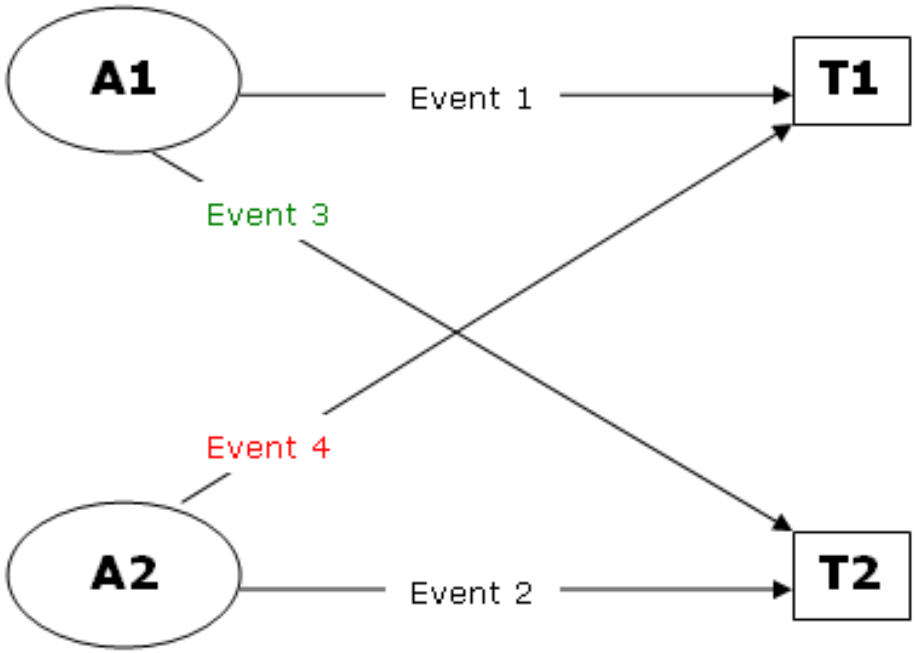
Survival of data is guaranteed after transaction is committed

Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter).

Transaction fundamentals

- BEGIN
- Do work needing A.C.I.D. compliance
- COMMIT (or ROLLBACK)

Why programmers should care



DEADLOCKS: one of A1 or A2 **MUST** rollback and programmers should handle properly

What to do then

```
$done=false;  $retries=10;
while (!$done && $retries > 0) {    // start the transaction...
    $db->beginTransaction();
    try {
        $db->insertSome($data);
        $db->insertMore($data2);
        $db->runPossibleDeadlockingStuff($data3);
        // if the last one does not deadlock, then transaction will COMMIT here
        $db->commit();
        $done = true;
    } catch (Zend_Db_Exception $e) {
        // ROLLBACK the transaction. Deadlock resolution would actually have already rolled back
        $db->rollBack(); // this is just to make things clear

        // something went wrong above, check if it's a deadlock and if it is, decrease retries left
        if ($e->getCode() == 1205 || $e->getCode() == 1213) { // 1205 and 1213 = deadlocks in MySQL
            $retries--;
        } else {
            // other non-recoverable error happened, exit loop
            print($e->getMessage());
            $done = true;
            // throw($e); we could re-throw the error here.
        }
    }
}

if ($retries == 0) {  throw new Exception("Too much concurrency, try again later",1205);   }
```

Questions

PERCONA

marcos.albe@percona.com

We're Hiring! www.percona.com/about-us/careers/