Databasesystemer, forår 2005
IT Universitetet i København

Forelæsning 11: Transaktioner.

21. april 2005

Forelæser: Rasmus Pagh

Today's lecture

- Serializability and atomicity
- ACID properties and rollbacks
- Dirty reads and isolation levels in SQL

Next: Serializability and atomicity

Databases and concurrency

In most large database systems, many users and application programs will be (and *must* be) accessing the database at the same time.

Groups of updates to a database that "belong together" are grouped into so-called **transactions**.

Having **concurrent** transactions updating the database raises a number of problems that, if not properly dealt with by the DBMS, could leave the database in an inconsistent state, even if all users "did the right thing".

Examples: Booking seats in a plane, transferring money between accounts.

Serializability

In many cases, undesired behavior can occur if two transactions are performed simultaneously.

We would like the DBMS to make transactions satisfy **serializability**:

> Even though many transactions may be performed at the same time, the state of the database should look *as if* transactions were performed one by one (i.e., in a **serial schedule**).

The DBMS is allowed to *choose* the order of transactions:

- It is not necessarily the transaction that is started first, which is first in the serial schedule.
- The order may look different from the viewpoint of different users.

Atomicity

A similar issue is what happens if a transaction, for some reason (e.g., power or hardware failure) is aborted while executing.

We would like the DBMS to make transactions satisfy **atomicity**:

> Even though a transaction may involve many updates, the state of the database should look *as if* either the *whole* transaction or *no part* of the transaction has been carried out.

We already saw the atomicity property in connection with database constraints: A transaction violating a constraint was **rolled back**.

Serializability and atomicity in SQL

As a default, the SQL standard requires the DBMS to process transactions atomically.

As a default, SQL also executes transactions in a serializable manner. (This is not the Oracle default – more on this later.)

However, in some situations the SQL programmer might give the DBMS permission to execute transactions in a non-serializable manner. (More on this later.)

Next: Transactions in SQL and ACID properties

Transactions in SQL

By the SQL standard, transactions consisting of more than one statement are started by the command

```
START TRANSACTION
```

and ended with one of the following commands:

- `COMMIT`: Update the state of the database to reflect the changes performed in the transaction.
- `ROLLBACK`: Discard the transaction.

Transactions in SQL*Plus/TOra

In SQL*Plus/TOra the `START TRANSACTION` command is *implicit*, i.e., statements are by default grouped into transactions.

(NB! This is different from a "generic interface", as described in GUW.)

Transactions are ended when:

- You exit the program.
- A command alters the database schema.
- A `COMMIT` or `ROLLBACK` statement appears.

The ACID properties of transactions

Ideal transactions are said to meet the **ACID** test:

- **A**tomicity – the all-or-nothing execution of transactions.
- **C**onsistency – transactions preserve database constraints.
- **I**solation – the appearance that transactions are executed one by one.
- **D**urability – the effect of a transaction is never lost once it has completed.

Commercial DBMSs tend to fully implement **C** and **D**, while **A** and **I** are only partially implemented (for efficiency reasons).

Durability

We already discussed all ACID properties except **durability**, which is another reason why DBMSs are used for critical applications:

> A good DBMS is able to withstand a power outage, disk or hardware failure, with *little or no loss of data*, returning the database to a recent, *consistent* state.

Next: Dirty reads and isolation levels in SQL

Dirty reads

Suppose that a transaction is in the process of updating a large relation.

Other transactions executing may see:

- Old data, not (yet) modified by the transaction.
- New data, updated by the transaction.

As long as a transaction has not committed, all data that it has changed is referred to as **dirty**.

A read of dirty data (called a **dirty read**) is a main source of trouble when executing transactions concurrently.

To avoid a dirty read, a transaction may have to wait for another (potentially lengthy) transaction to commit before being able to execute.

Read-only transactions

We can tell the DBMS that the current transaction does not update the database, using the SQL statement

```
SET TRANSACTION READ ONLY;
```

It is a good idea to do this whenever we have a read-only transaction:

- Other transactions never need to wait for a read-only transaction to finish, since it does not change anything.
- Thus the DBMS potentially runs faster if it knows that a transaction is read-only.

Read committed

The lowest isolation level in which SQL allows transactions to do updates is `READ COMMITTED` (Oracle's default isolation level).

Transactions running at this isolation level see other transactions as atomic, in the sense that either:

- No changes made by a transaction are seen, or
- all changes made by a transaction are seen.

However, different statements in the transaction may see the database in different states (i.e., some transactions may commit in between).

This is sometimes referred to as the **phantom phenomenon**.

The SERIALIZABLE isolation level

The highest isolation level in SQL is SERIALIZABLE, which gives nearly, but not quite, the behavior of a serial schedule of transactions.

This isolation level gives the following additional guarantee:

- No value read or written by the transaction is changed before the transaction is committed.
- In particular, there is no phantom phenomenon.

It is possible, but not easy, to come up with transactions that, when executed at the SERIALIZABLE isolation level, may give a result different from any serial schedule.

SQL isolation levels

The SQL-92 standard defines four isolation levels, two of which are supported by Oracle.

- SERIALIZABLE (implemented in Oracle). Almost, but not quite, ideal serializable transactions.
- REPEATABLE READ. Allows the result of a query to change during the transaction, in the sense that more tuples may be added.
- READ COMMITTED (Oracle default). The result of any statement reflects some set of committed transactions.
- READ UNCOMMITTED. Allow dirty reads.

Nonstandard isolation levels

Several DBMSs (DB2, SQL Server) have isolation levels that are not in the SQL standard:

- `CURSOR STABILITY`. Like REPEATABLE READ, but additionally prevents "lost updates".
- `SNAPSHOT ISOLATION`. Transactions are guaranteed to only see data as it is "at the time they are started".

A thorough description of these and the SQL isolation levels is provided in the paper *A Critique of ANSI SQL Isolation Levels*, found in the course schedule.

System-generated rollbacks

Sometimes the DBMS must roll back one or more transactions to allow others to go on.

Example: Suppose that transaction A cannot commit before B has committed, and vice versa (i.e., we have a **deadlock**). Then the only way out is to roll back one of the transactions.

Deadlock situations occur more often at higher isolation levels, which is another reason to use the lowest isolation level necessary.

Most important points in this lecture

As a minimum, you should after this week:

- Know and understand the ACID properties of transactions.
- Know how to create transactions in SQL.
- Be able to predict possible transaction behavior at isolation levels `SERIALIZABLE` and `READ COMMITTED`.

Next: Martin Jensen, Oracle Denmark, talks about large multiuser systems (and probably about Oracle's concurrency control technology).