Transactions

Databases 2018-2019

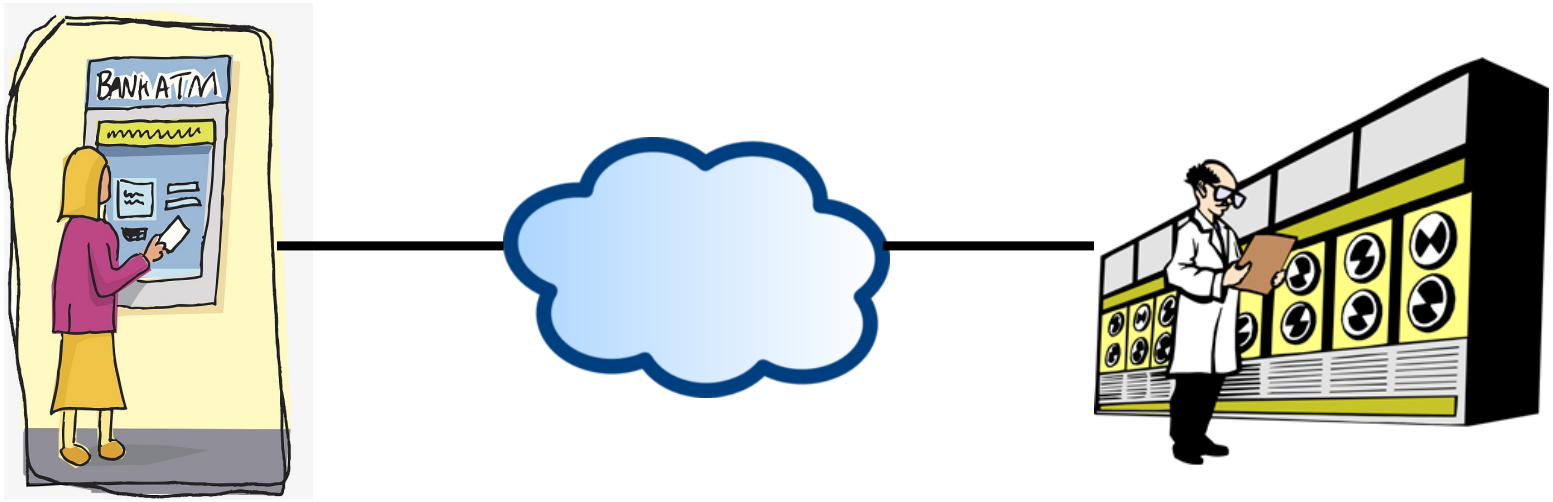
Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

.

Introduction to Transactions

- There are many cases in which **a complex operation** on a database (DB) comprises **several basic operations** (SQL sentences).
- **Example: Cash withdrawal from a bank account.**



Motivating example

- A bank DB usually keeps separate information about the **balance** of an account and its **entries.**
- A cash withdrawal of 400,00€ involves the following operations:

1. **Get the balance** of the account and compute the result of subtracting 400,00 € to its current balance.
2. If this value is negative, show an error message and finish.
3. **Add an entry** to the account: **withdrawal of** 400,00 **€.**
4. **Set the balance** of the account to the value computed in Step 1.

- DB operations are **in red.**
- The DB management system **must guarantee** that the state of the DB **is consistent.**

Motivating example

- The DB management system **must guarantee** that the state of the DB **is consistent.**

table **AccountBalance**

Acc	**Account Holder**	**Balance**
37	Alice & Bob	1500,00
...	...	...

table **AccountEntries**

Acc	**Order**	**Date**	**Amount**
37	1	25/08/2018	850,00
37	2	05/09/2018	-350,00
37	3	13/09/2018	1000,00
...	...	...	...

Motivating example

- The DB management system **must guarantee** that the state of the DB **is consistent.**

table **AccountBalance**

Acc	**Account Holder**	**Balance**
37	Alice & Bob	**1100,00**
...	...	...

table **AccountEntries**

Acc	**Order**	**Date**	**Amount**
37	1	25/08/2018	850,00
37	2	05/09/2018	-350,00
37	3	13/09/2018	1000,00
37	**4**	**08/10/2018**	**-400,00**
...	...	...	...

Motivating example

- But complex operations may behave in strange ways...

> "A good programmer is someone who always looks both ways before crossing a one-way street"
>
> –Doug Linder, Systems administrator*

Motivating example

- But complex operations may behave in strange ways...

> "A good programmer is someone who always looks both ways before crossing a one-way street"
>
> –Doug Linder, Systems administrator*

- **At what steps may the DB be in an inconsistent state?**

1. **Get the balance** of the account and compute the result of subtracting 400,00 € to its current balance.
2. If this value is negative, show an error message and finish.
3. **Add an entry** to the account: **withdrawal of** 400,00 **€.**
4. **Set the balance** of the account to the value computed in Step 1.

Motivating example

- But complex operations may behave in strange ways...

> "A good programmer is someone who always looks both ways before crossing a one-way street"
>
> –Doug Linder, Systems administrator*

- **At what steps may the DB be in an inconsistent state?**

DB is consistent if either all operations are performed, or no operation is performed at all.

Motivating example

- But complex operations may behave in strange ways...

> "A good programmer is someone who always looks both ways before crossing a one-way street"
>
> –Doug Linder, Systems administrator*

- **At what steps may the DB be in an inconsistent state?**

DB is consistent if either all operations are performed, or no operation is performed at all.

- **What can happen to leave the DB in an inconsistent state?**

Motivating example

- But complex operations may behave in strange ways...

> "A good programmer is someone who always looks both ways before crossing a one-way street"
>
> –Doug Linder, Systems administrator*

- **At what steps may the DB be in an inconsistent state?**

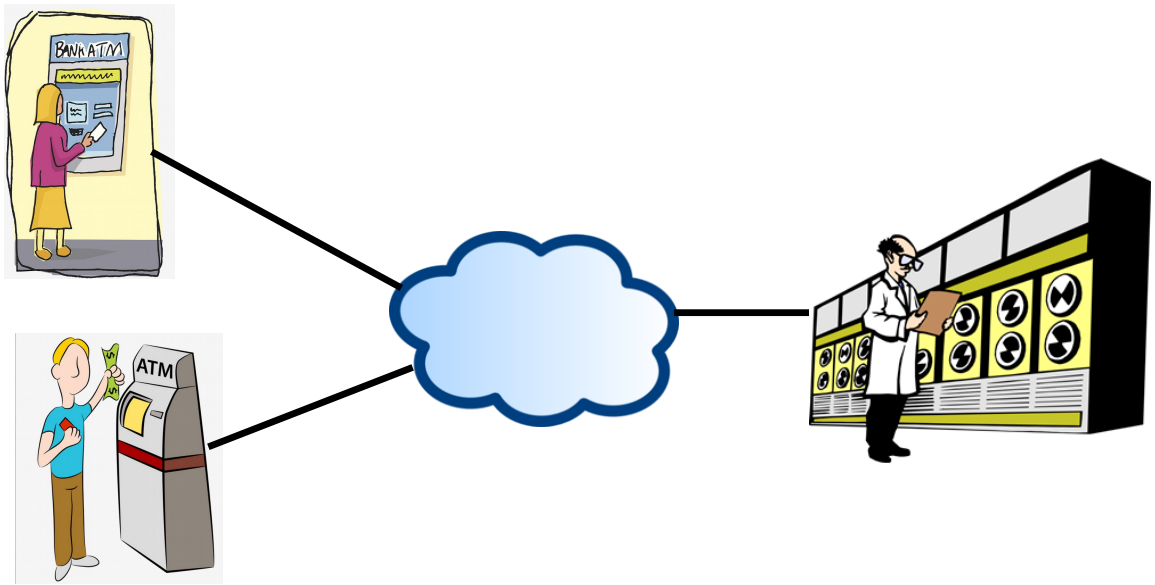
DB is consistent if either all operations are performed, or no operation is performed at all.

- **What can happen to leave the DB in an inconsistent state?**
- Many events might leave the DB inconsistent:
  - Power outage in the servers room.
  - Failure in ATM (or client computer), network connection.
  - An earthquake...

k⋆ Quote taken from: Gómez, M.A.; Gómez, J. *Tecnología de la programación: apuntes de*

Motivating example

- There is a particularly relevant event... that happens very frequently.
- Let us suppose that just when Alice operates the ATM, her partner, Bob, withdraws some more money from the same bank account...
- Basic DB operations may **interleave in any way.**



Motivating example

- There is a particularly relevant event... that happens very frequently.
- Let us suppose that just when Alice operates the ATM, her partner, Bob, withdraws some more money from the same bank account...
- Basic DB operations may **interleave in any way.** For example:

Op	Cust.	Operation	Balance
			1500.00
1a	**Alice**	Get the balance of the account and compute **1500.00 - 400.00**	1500.00
2a	**Alice**	Add an entry of -400.00 € to the account	1500.00
1b	**Bob**	Get the balance of the account and compute **1500.00 - 150.00**	1500.00
2b	**Bob**	Add an entry of -150.00 € to the account	1500.00
3a	**Alice**	Set the balance of the account to the value computed in Op 1a	1100.00
3b	**Bob**	Set the balance of the account to the value computed in Op 1b	1350.00

Motivating example

- There is a particularly relevant event... that happens very frequently.
- Let us suppose that just when Alice operates the ATM, her partner, Bob, withdraws some more money from the same bank account...
- Basic DB operations may **interleave in any way.** For example:

Op	Cust.	Operation	Balance
			1500.00
1a	**Alice**	Get the balance of the account and compute **1500.00** - **400.00**	1500.00
2a	**Alice**	Add an entry of -400.00 € to the account	1500.00
1b	**Bob**	Get the balance of the account and compute **1500.00** - **150.00**	1500.00
2b	**Bob**	Add an entry of -150.00 € to the account	1500.00
3a	**Alice**	Set the balance of the account to the value computed in Op 1a	1100.00
3b	**Bob**	Set the balance of the account to the value computed in Op 1b	1350.00

The DB state is inconsistent!

Motivating example

- This example is simplistic and rather contrived, but

> **DB programs should be protected against any interleaving of transactions without having to explicitely program all cases.**

- Situations like this one happen quite frequently as large DBs may have hundreds of users **concurrently** accessing to data.
- DB management systems provide a mechanism to encapsulate complex operations as if they were atomic: **transactions.**
  - A transaction is composed of several (at least one) DML data modification sentences, or a single DDL sentence.
  - A transaction is composed of SQL sentences executed **from a single DB connection.**

Definition of transaction – Properties

A **transaction** is a set of **one or several SQL sentences that are a logical indivisible unit.**

- Properties of transactions[1]:
  - **Atomicity:** Either all operations are carried out or none are.
  - **Consistency:** Each transaction must preserve the consistency of the database (in most cases it is responsibility of the programmer).
  - **Isolation:** Transactions are protected from the effects of concurrently scheduling other transactions.
  - **Durability:** When a transaction has been successfully completed, its effects should persist even if the system crashes.
- These properties are so important that they have a name: **ACID properties.**

[1] Ramakrishnan, R.; Gehrke, J. *Database Management Systems*, 3rd ed.

Transaction control: **COMMIT** and **ROLLBACK**

- There exists a set of SQL sentences that allow the implementation of transaction control techniques.
- **Starting a transaction:** There is no specific sentence.
  - A transaction starts if there is no active transaction and a DML or DDL sentence executes.
  - Or when a **SET TRANSACTION** executes.
- **Finishing a transaction:** There are two basic sentences:
  - **COMMIT confirms** the modifications performed to DB data and makes them permanent and visible to other active sessions.
  - **ROLLBACK cancels all changes** made to DB data from the beginning of the transaction.
  - Moreover, any DDL sentence **implicitly terminates any previous transaction** (committing changes made by previous DML sentences, just as if a `COMMIT` were executed).
- When a transaction finishes, next executable SQL DML sentence automatically starts **the next transaction** (a DDL sentence forms a transaction on its own).

Example

- **example1.sql: What are the contents of `empl` after executing the following sentences?**

```
CREATE TABLE empl (
  Id VARCHAR2(9) PRIMARY KEY,
  Name VARCHAR2(20),
  Salary NUMBER(6,2)
);
INSERT INTO empl VALUES ('10A','Jorge Perez',3000.11);
ROLLBACK;
INSERT INTO empl VALUES ('30C','Javier Sala',2000.22);
INSERT INTO empl VALUES ('30C','Soledad Lopez',2000.33);
INSERT INTO empl VALUES ('40D','Sonia Moldes',1800.44);
INSERT INTO empl VALUES ('50E','Antonio Lopez',1800.44);
COMMIT;
INSERT INTO empl VALUES ('70C','Soledad Martin',2000.33);
```

- **What are the contents of `empl` visible from other sessions?**

Transaction control: **SAVEPOINT**

- In addition to sentences **COMMIT** and **ROLLBACK**, we can set **intermediate points:**
  - Sentence **SAVEPOINT *identifier*** sets a point in a transaction in order to **partially cancel** all changes after that savepoint.
  - To cancel any later change after a savepoint we use the following sentence:

    ROLLBACK TO SAVEPOINT savepointIdentifier

- Note that **intermediate SAVEPOINT sentences do not finish the current transaction:** later DB changes are just undone without finishing the transaction.
- We can use **several SAVEPOINT** sentences in the same transaction:
  - When we rollback to a `SAVEPOINT`, all later savepoints are removed (but not the previous ones).
  - We can **shift** a `SAVEPOINT` using the same identifier in different savpoint sentences: in this case we will rollback to the last savepoint with that identifier.

Another example

- **example2.sql: What are the contents of `empl` after executing the following sentences?**

```
SET TRANSACTION NAME 'sal_update';
UPDATE empl SET Salary = 7000 WHERE Id= '40D';

SAVEPOINT after_salary_update;
UPDATE empl SET Salary = Salary + 100 WHERE Id= '40D';

ROLLBACK TO SAVEPOINT after_salary_update;
UPDATE empl SET Salary = Salary + 250 WHERE Id= '40D';
COMMIT;
```

Locks

- The ACID **isolation property** requires that two unfinished transactions **do not mutually interfere.**
- In order to provide isolation, DBMS use **locking** techniques for some DB elements.
  - A lock **restricts the access to these elements for some specific DB operations.**
- Roughly speaking, there are **two types of locks:**
  - **Shared locks:** are produced by query sentences (`SELECT`). Other sessions can modify locked data, but those changes **are not visible** from the `SELECT` sentence.
  - **Exclusive locks:** are produced by DDL sentences and DML modification sentences (`INSERT`, `UPDATE`, `DELETE`). Other sessions cannot modify locked data and they have to wait until the lock is released.
  - We consider as **exclusive locks** the ones generated by `SELECT...FOR UPDATE` sentences (although they are different.)
- Other sessions **can query data locked by an exclusive lock.**

Lock conflicts

- There is a **lock conflict** when two sessions connected to the DB **try to modify the same data simultaneously**
- The DBMS must **serialize** concurrent accesses, by means of **row or table locking mechanisms.**
- Oracle tries to lock the minimum amount of data to achieve the maximum level of concurrency:
  - **DDL** sentences (`CREATE TABLE`, `ALTER TABLE`, etc.) lock **the whole table.**
  - **DML** data modification sentences (`INSERT`, `UPDATE`, `DELETE`, `SELECT...FOR UPDATE`) lock just **the rows affected by the modification.**
- The isolation level can be configured with a **`SET TRANSACTION`** sentence.

Lock conflicts behaviour

1. **DDL sentences:** If a DDL sentence cannot get the lock on a DB object, it **terminates immediately with an error condition.**
2. **DML query sentences:** These sentences make shared locks and **they are not affected by exclusive locks**, although they use **the data as it was before the locking transaction started:**
   - To this end, the DBMS must keep **the data state of any unfinished transaction.**
   - A DBMS that allows querying uncommitted data is said that it allows *dirty read;* Oracle does not allow it.
3. **DML data modification sentences:** If any of these sentences cannot get the lock on a table, **it waits in a queue sorted in chronological order.**

- This delay in the execution of DML sentences is called **lock contention.**
  - The delay might be very long if there are many transactions and each one is composed of many basic operations.

Lock conflicts solving

- We can control the **waiting time in a lock** using a specific `SELECT` sentence:

```
SELECT ... FOR UPDATE NOWAIT
SELECT ... FOR UPDATE WAIT n
```

- This is a query sentence **but locks the retrieved rows as if it were to modify their contents.**
- It finishes immediately (or after `n` seconds) if selected rows are locked in an exclusive lock by another session.
- Some contention level is normal in any real database, but **the system performance can be dramatically reduced if there are design or programming errors.**

Lock contention

Design/programming issues that increase the contention level:

- **Long transactions.** Example: if a table is locked and a user action is expected before the `commit` is executed.
- **Batch processes.** Example: monthly closing processes: *conceptually* is a single transaction, although in practise may entail locking thousands of rows in several tables for hours.
- **External applications.** The DBMS may have other applications installed that involve high contention levels.
- **Locks for repeated queries.** Some applications lock table rows to execute several queries *consistently*. This can be solved by *read-only* transactions, using

  ```
  SET TRANSACTION READ ONLY
  ```

  No row is locked, but guarantees that the table data is not affected by other transactions.

Deadlocks

- A particular case of lock is a **deadlock.**
- A deadlock occurs when two or more sessions are waiting for data locked by each other, resulting in all the sessions being blocked.
- **Example:** Two transactions executing in different sessions:

SESSION A

```
-- A locks account 37
UPDATE Account SET ...
WHERE AccId = '37';
```

SESSION B

```
-- B locks account 44
UPDATE Account SET ...
WHERE AccId = '44';
```

```
-- A waits for account 44.
UPDATE Account SET ...
WHERE AccId = '44';
```

```
-- B waits for account 37.
UPDATE Account SET ...
WHERE AccId = '37';
```

- Oracle detects deadlocks **and automatically cancels one of the transactions raising an exception.**

Transaction control: **SET TRANSACTION**

- In a transaction A the rows in affected tables **are locked as the DML sentences execute:**
  - If other sessions finish their transactions with `COMMIT` during the execution of A, their changes might be visible by the remaining sentences **if the affected rows have not been locked in advance.**
- This behaviour may produce **inconsistencies.**
- We can avoid them using the following sentence:

  `SET TRANSACTION [READ ONLY | READ WRITE] NAME` ***`nombre`***
  - **It explicitely starts a transaction.**
  - If **`READ ONLY`**, **all sentences** included in the transaction access the data in the state in which they are in the DB **at the very instant the transaction starts:** changes made by other transactions **are not visible** during the execution of the read-only transaction.
- Option **`READ WRITE`** is the default option: it sets the beginning of a transaction that modifies data.
- **`READ ONLY`** starts a transaction that **does not modify any data.**
  - It is used for making all queries in the transaction **consistent with the same DB state:** the state at the beginning of the transaction.