Unit 3: Database Management Systems (DBMS)

Bases de Datos y Sistemas de información
Departamento de Sistemas Informáticos y Computación / Universidad Politécnica de Valencia

V. 16.3

Unit 3. Database Management Systems

1.1. Schemas

Original proposal

Proposal of a DBMS architecture by the working group ANSI/SPARC (1977). They propose the database definition with 3 levels of abstraction:

- **Internal level** → Internal schema

  Description of the DB in terms of its physical representation
- **Conceptual level** → Conceptual schema

  Description of the DB independently of the DBMS
- **External level** → External schemas

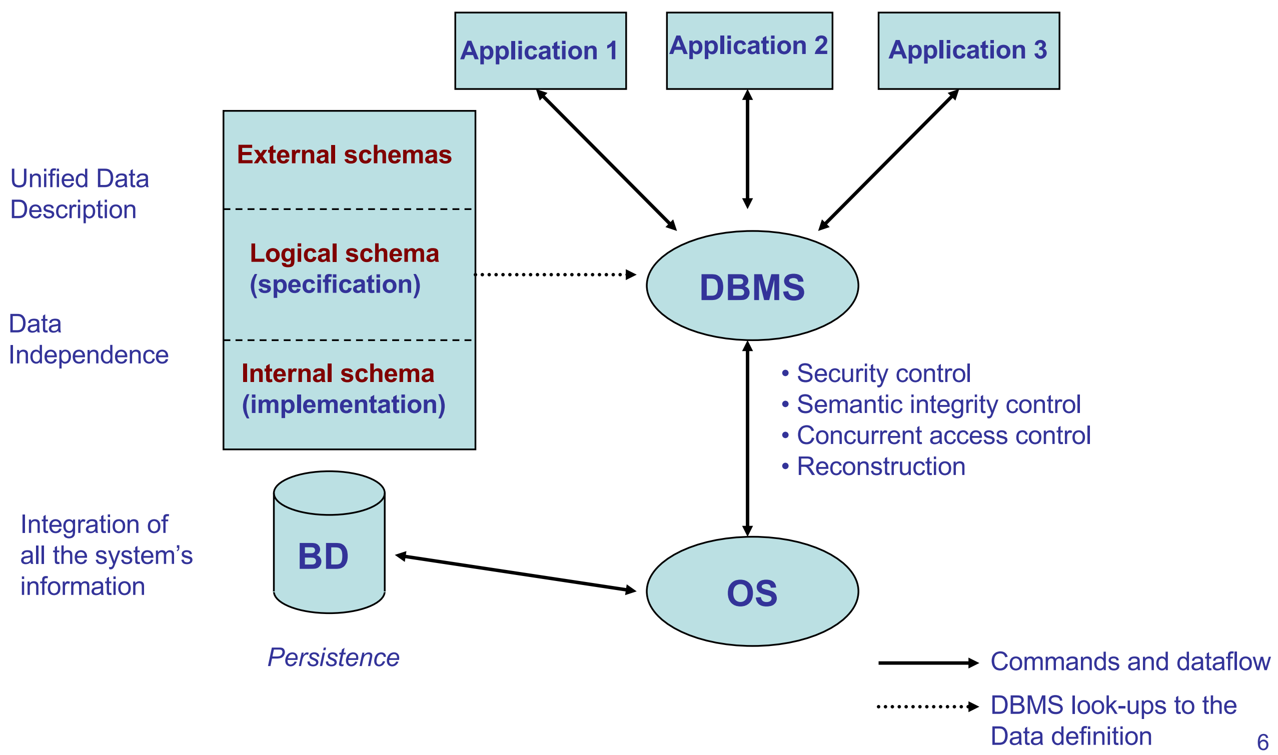
  Description of the users' partial views

Refined proposal

Since there was no generalized conceptual model which was accessible to different kinds of DBMS, a new level was introduced:

- **Logical level** → Logical schema (Unit 2)
  DB description in terms of the DBMS data model
- **Conceptual level** → Conceptual schema (Unit 4)
  Organizational DB description
- **Internal level** → Internal schema *(not in this course)*
  DB description in terms of its physical representation
- **External level** → External schemas (authorizations and views)
  Description of the partial views which the different users have on the DB.

ANSI/SPARC Architecture



Example: Logical schema

Departamento (cod_dep: char(4), nombre: char(50), teléfono: char(8), director: char(9))

PK:{cod_dep} **NNV**:{nombre}

FK:{director} ->Profesor(dni) On delete set to nulls. On update cascade

Asignatura (cod_asg: char(5), nombre: char(50), semestre: char(2), cod_dep: char(4), teoría: real, prácticas: real)

PK:{cod_asg} **NNV**:{nombre, semestre, cod_dep, teoría, prácticas}

Uni:{nombre} **FK**:{cod_dep} -> Departamento(cod_dep)

On delete restrict. On update cascade

$\mathbf{IC_1}$:(teoría <= prácticas) $\mathbf{IC_2}$:(semestre IN {'1A','1B','2A','2B','3A','3B','4A''4B'})

Profesor (dni: char(9), nombre: char(80), teléfono: char(8), cod_dep: char(4), provincia: char(25), edad: entero)

PK :{dni} **NNV** :{nombre, cod_dep}

FK :{cod_dep} -> Departamento(cod_dep)

On delete restrict. On update cascade

Docencia (dni: char(9), cod_asg: char(5), gteo: entero, gpra: entero)

PK :{dni,cod_asg} **NNV** :{gteo,gpra}

FK :{dni} -> Profesor(dni) On delete cascade. On update cascade

FK :{cod_asg} -> Asignatura(cod_asg) On delete restrict. On update cascade

General constraint: GC1: "All teacher must lecture at least one subject".

Example: Internal schema

Depends on the DBMS.

***Asignatura*:**

Hash file by cod_dep

B+ index over (semestre + cod_dep)

Profesor:

Hash file by nombre

Departamento:

Hash file by cod_dep

B+ index over nombre

Docencia:

Disordered file

Example: External schema

External schema for the **DMA Department**

```
CREATE VIEW Profesor-DMA  AS
        SELECT códigodni, nombre, teléfono, categoríaprovincia,edad
        FROM Profesor
        WHERE  dptocod_dep = 'DMA';
CREATE VIEW Asignatura-DMA  AS
        SELECT cod_asg, nombre, semestre, teoría, prácticas
        FROM Asignatura A
        WHERE cod_dep = 'DMA';
CREATE VIEW Docencia-DMA  AS
        SELECT D.dni, D.cod_asg, D.gteo, D.gpra
        FROM Profesor P, Docencia D, Asignatura A
        WHERE   P.cod_dep = 'DMA'
        AND A.cod_dep = 'DMA'
        AND P.dni = D.dni
        AND A.cod_asg = D.cod_asg;
```


A DBMS that supports the 3-level architecture must:

- Allow for the definition of the several schemas for the database (except for the conceptual schema),
- Establish the correspondence between schemas,
- Isolate the schemas: changes in one schema should not affect the schemas at upper levels and neither should they affect the application programs.

DATA INDEPENDENCE

1.2. DBMS Fundamentals

Functions of a DBMS

DBMS: Software which allows the creation and manipulation of databases (DB).

DBMS

↓ **Is based on**

Data model (*relational model*)

↓ **Is composed of**

Data structures and associated operators

Functions of a DBMS

A DBMS allows

- A unified description of the data and independent of the applications
- Application independence with respect to the physical data representation
- Definition of partial data views for different users
- Information management
- Data integrity and security

Functions of a DBMS

Objectives of DB techniques

- Unified and independent data description
- Application independence
- Partial view definition

DBMS Functions

Data definition at several levels

- Logical schema
- Internal schema
- External schema

DMBS Components

Schema definition languages and their associated translators

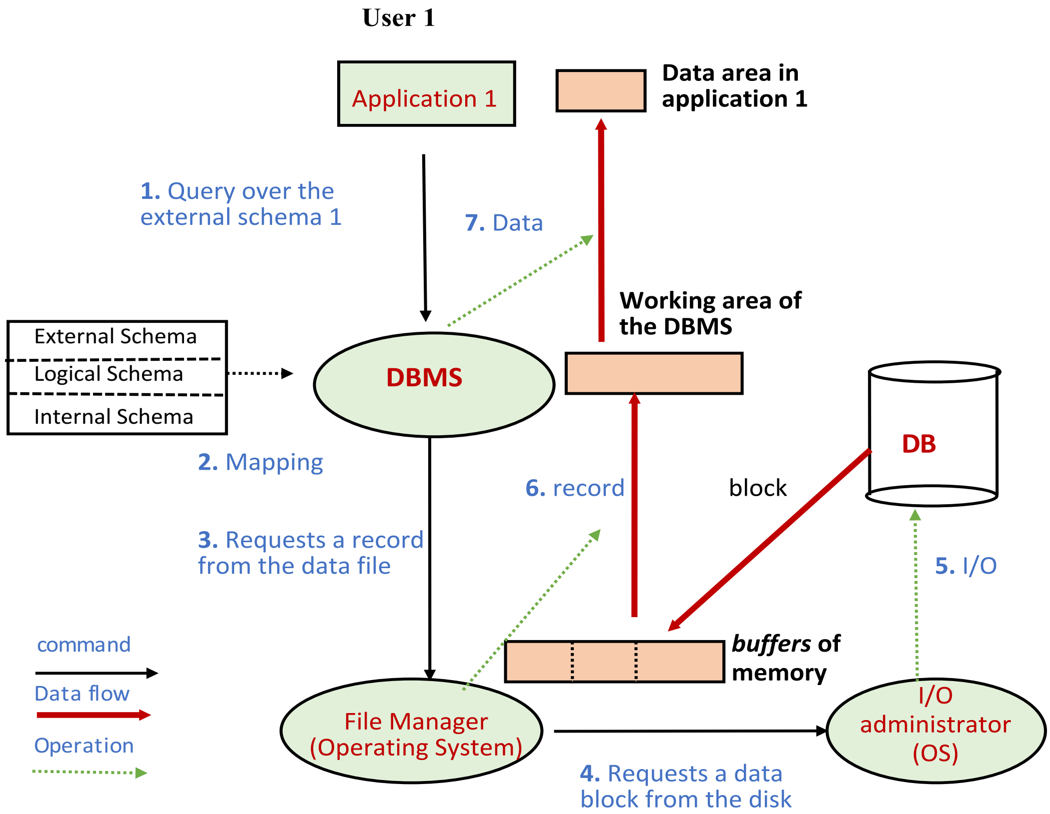
Functions of a DBMS

Objectives of DB techniques	DBMS Functions	DMBS Components
• Information Management	Data manipulation • Query • Update Management and administration of the database	Manipulation languages and their associated translators Tools for: • Restructuring • Simulation • Statistics • Printing and reporting

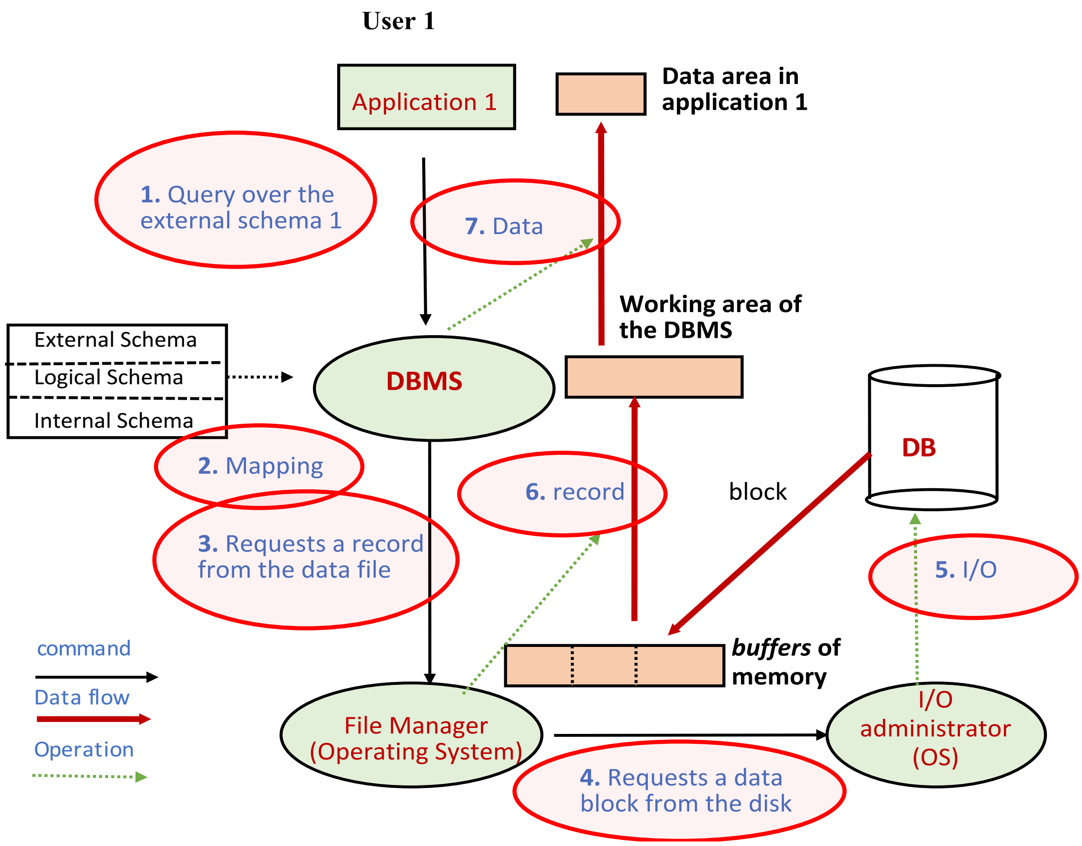
Functions of a DBMS

Objectives of DB techniques	DBMS Functions	DMBS Components
• Data integrity and security	Control of: • Semantic integrity • Concurrent access • Recovery in case of failure • Security (privacy)	Tools for: • Integrity control • Reconstruction • Security control

Accessing the data



Accessing the data



1.3. Data Independence

Data independence

Property which ensures that the application programs are independent of

- the changes which are performed on data which they do not used

or

- the physical representation details of the accessed data

Logical independence

Logical independence between the logical schema and the external schemas:

The external schemas and the application programs cannot be affected by the modifications in the logical schema of data which are not used by these programs

EXAMPLE:

If we add new attributes to the "*Departamento*" table, such as the date in which the department was created, the building,… the external schema of the "*DMA-department*" does not need to be modified

Physical independence

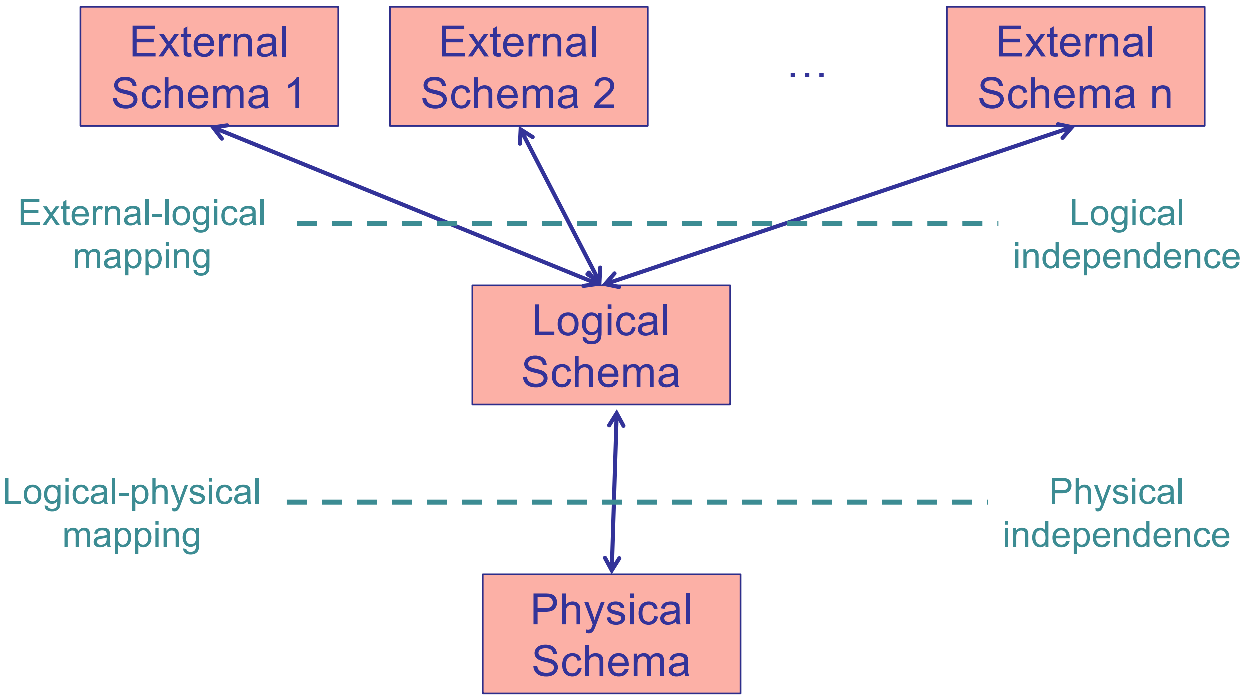
Physical independence between the internal schema and the logical schema:

The logical schema cannot be affected by changes in the internal schema which refer to the implementation of the data structures, access modes, page size, search path, etc.

EXAMPLE:

If the data structures used in the implementation of the "*Asignatura*" table are changed, the logical schema does not need to be modified

Data independence



Unit 3. Database Management Systems

2. Transactions, Integrity, and Concurrency

Objective of DB technology

↓

Information quality

"Data must be structured in such a way as to adequately reflect the objects, relations, and constraints which exist in the part of the real world modeled by the database model."

- Representation of the objects, relations, and constraints in the DB schema.
- Reality changes → User updates
- The information contained in the DB must preserve the schema definition.

2. Transactions, Integrity, and Concurrency

Information quality (integrity perspective):

- The DBMS must ensure that the data are correctly stored
- The DBMS must ensure that user updates over the DB are correctly executed and become permanent.

2. Transactions, Integrity, and Concurrency

DBMS Tools oriented towards integrity:

- Check (when an update is performed) the integrity constraints defined in the schema
- Control the correct execution of the updates (in a concurrent environment)
- Recover (reconstruct) the DB in case of losses or accidents

2.1. Transactions

2.1. Transactions

- DB integrity must be controlled when access operations take place, generally coming from the applications.

- The access operations to a DB are organized in transactions.

TRANSACTION — Sequence of access operations to the DB which constitute a logical execution unit.

Example

Emp (id, name, address, dept)
  PK: {id}
  FK: {dept} $\rightarrow$ Dep(code)

Dep (code, name, location)
  PK: {code}

IC_1: All departments have at least one employee

Insert a new department:
<“d2”, “Human Resources”, “2nd floor”>
whose first employee is the id 20

Example

1st Idea

1) Insert in *Dep*: <d2, “Human Resources”, “2nd floor”>

ERROR: IC_1 *is violated*

2) Modification of *Emp* on the tuple with *id* 20

2nd Idea

1) Modification of *Emp* on the tuple with *id* 20

ERROR: the FK over dept in Emp is violated

2) Insertion in *Dep*: <d2, “Human Resources”, “2nd floor>

Defining transactions

Actions which change transactions states:

begin:

Indicates the beginning of a transaction

end:

Indicates that all the operations in the transaction have been completed.

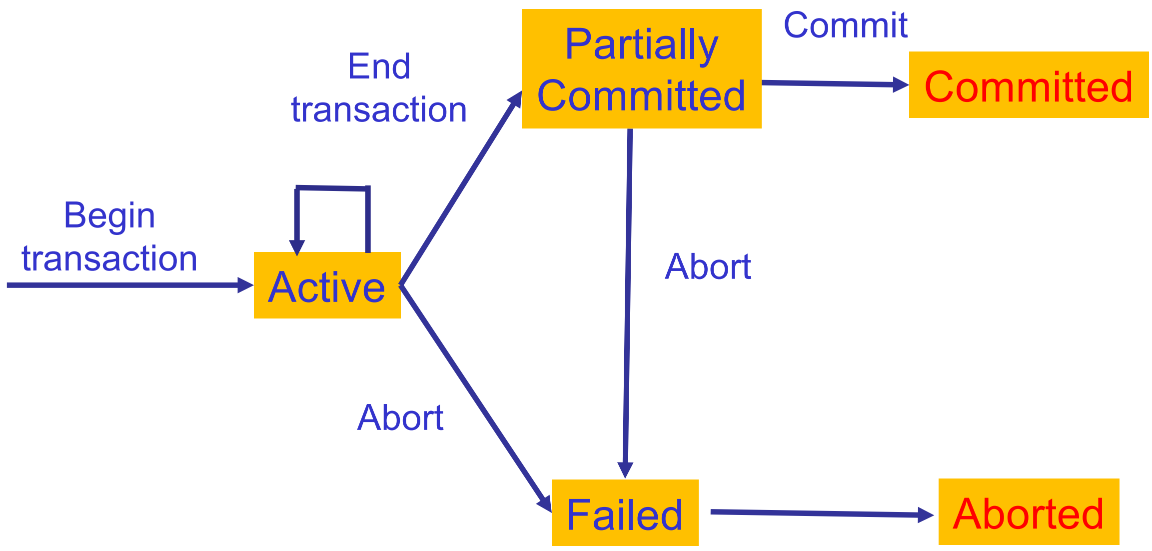
Confirmation *(commit)*:

Indicates the success of the transaction, making the DBMS store the changes performed on the DB

Cancellation *(rollback)*:

Indicates the failure of the transaction due to some reason. The DBMS undoes all the possible changes performed by the transaction.

States of a transaction



Properties of Transactions **(ACID)**

- **Atomicity:** A transaction is an indivisible unit that is either performed in its entirety or is not performed at all ("All or nothing").

- **Consistency:** the transaction must transform the DB from one consistent state to another consistent state (all integrity constraints must be met)

- **Isolation:** Transactions execute independently of one another: All the modifications introduced by a non-confirmed transaction are not visible to other transactions

- **Durability**: The effects of a successfully completed (committed) transaction are permanently recorded in the DB and must not be lost because of a subsequent system or other transaction failure

2.2. Semantic Integrity

2.2. Semantic Integrity

Integrity constraint:

Property of the real world which is modelled by the DB

- Constraints are defined in the logical schema and the DBMS must ensure that they are met.
- Checking is performed whenever the DB changes (an update operation is executed)
- Constraints not included in the DB schema must be maintained by the application programs

  This situation is, in general, inappropriate if the constraints are common to more than one application, since the responsibility to check them is dispersed

Types of integrity constraints

- **Static:** They must be met in each state of the DB (they can be represented by logical expressions)

EXAMPLE:
The *lab credits* in a *subject* cannot be greater than the *class credits*.

- **Transition:** They must be met regarding two consecutive states.

They are not usually implemented by commercial DBMS

EXAMPLE:
The *credits* of a *subject* cannot decrease.

Expressing static constraints in SQL

- Constraints over possible data values. e.g. Domains
- Constraints over attributes. e.g. NNV.
- Constraints over relations. e.g. PK, FK
- General constraints over the DB. e.g: "*All subject must be lectured by at least one teacher.*"

- When are checked: After every command (IMMEDIATE)
At the end of the transaction(DEFERRED)

Expressing transition constraints in SQL

Triggers ("*disparadores*")

- Using triggers, the designer can program the system response when some events are produced.
- This allow us to incorporate complex constraints into the DB.

- Triggers introduce the concept of reactivity and have many other applications (apart from integrity) :
  - Express transition constraints
  - Maintenance of derived information.
  - Implementation of business rules.
  - Database administration (backups, alerts, etc.) and other issues related to security (traceability, logs, ...)

Expressing transition constraints in SQL

A trigger must include:

1. Events: Operations over the DB which trigger it
2. Conditions to determine if the actions must be executed or not.
3. Actions to be executed when an event happen and the conditions are met. They are usually written in a data-oriented high level programming language, that can include SQL commands.

Triggers can be used to express **transition constraints.**

1. The events are the operations over the DB that can violate the constrain.
2. The conditions represent the property of the constraint
3. The actions are the commands that will be executed for rejection or compensation in case of constraint violation.

Expressing transition constraints in SQL

EXAMPLE:

A trigger to implement the integrity constraint: *"A lecturer can only teach a subject assigned to his/her department"*

1. Events: INSERTion of a tuple in the "Asignatura" table
2. Conditions: The lecturer and the subject are not in the same department.
3. Actions: Reject the operation (the insertion).

Note that we have to define more triggers to control:

- The modification of the department of the lecturer or the department of the subject (in the "Profesor" and "Asignatura" tables).
- The insertion of a new tuple in "Docencia"
- The modification of the subject or lecturer of a tuple in "Docencia"

2.3. Concurrent access control

2.3. Concurrent access control

In order to keep the integrity of the database, the DBMS must control concurrent access to the database:

- Avoiding that the results of the execution of several processes (users or programs) can simultaneously lead to incorrect, incoherent or lost results because of the simultaneous execution of other program that accesses the same data

Basic operations

Basic operations in a transaction which are relevant to the DBMS:

read(X):

Reading or access to a piece of data X in the DB over the program variable with the same name.

write(X):

Update (insertion, deletion, or modification) of a piece of data X in the DB by using the program variable with the same name

Reading steps

read(X):

1. Seek the address of the block which contains the datum X
2. Copy the block to a buffer in main memory
3. Copy the datum X from the buffer to the program variable X

Writing steps

write(X):

1. Seek the address of the block containing the datum X
2. Copy the block to a buffer in main memory
3. Copy the datum X from the program variable to the the suitable location in the buffer
4. Copy the updated block from the buffer to the disk

If not Read before

Possible problems

The DBMS must control the concurrent access by the applications.

Problems due to interference of concurrent accesses:

a) Loss of updates

b) Incoherent information corresponding to several valid database states

c) Access to updated data (but still not confirmed) that can still be cancelled

A. Lost of updates

Time	Program 1	Program 2
t1	**read(11548, teoría)** teoría=4,5	
t2		**read(11548, teoría)** teoría=4,5
t3	teoría←teoría+1,5 teoría=6	
t4		teoría←teoría+2 teoría=6,5
t5	**write(11548, teoría)**	
t6		**write(11548, teoría)**

Two programs reading and updating the theory credits of subject 11548

Asignatura

cod_asg	nombre	semestre	cod_dep	teoría	prácticas
11545	Análisis Matemático	1A	DMA	4,5	1,5
11546	Álgebra	1B	DMA	4,5	1,5
11547	Matemática Discreta	1A	DMA	4,5	1,5
11548	Bases de Datos y Sistemas de Información	3A	DSIC	6,5	1,5

4,5 + 1,5 + 2 = 7

B. Incoherent information

Program 1: List for each lecturer his/her number of credits

Docencia

dni	cod_asg	gteo	gpra
111	11545	2	3
123	11545	0	2
123	11547	1	1
564	11545	1	2

Asignatura

cod_asg	...	teoría	prácticas
11545	...	4,5	1,5
11546	...	4,5	1,5
11547	...	4,5	1,5
11548	...	4,5	1,5

Time	Program 1	Program 2
t1	Calculate credits for lecturer 111 **Credits 111= 9** (1×4,5 + 3×1,5)	
t2	Calculate credits for lecturer 123 **Credits 123= 9** (0×4,5 + 2×1,5 + 1×4,5 + 1× 1,5)	
t3		Change a theory group of subject 11545 from lecturer 564 to lecturer 111
t4	Calculate credits for lecturer 453 **Credits 453= 0**	
t5	Calculate credits for lecturer 564 **Credits 564= 7,5** (1×4,5 + 2×1,5)	

Result:

DNI	Credits
111	*9 credits*
123	*9 credits*
453	*0 credits*
564	*7,5 credits*

C. Access to updated (but not confirmed) data

Time	Program 1	Program 2
t1	**read(11548, teoría)**	
t2	teoría=teoría+1,5	
t3	**write(11548,teoría)**	
t4		**read(11548,teoría)** teoría=6
t5		Use this value in its instructions
t6		**confirmation**
t7	**cancellation**	

Techniques

Reserving some data occurrences (Locks)

- Examples **a)** and **c)** must lock a record.
- Examples **b)** must lock all.
- Need for controlling deadlocks

Other solutions (for the example c): Cascade cancellation or transaction isolation

Unit 3. Database Management Systems

Recovery and Security

A database must guarantee both:

- Recovery:

  (Part of integrity, but not from the point of view of consistency. Recovery is focus on durability and persistence)

  A database must always be recovered in front of any kind of failure.

- Security:

  A database cannot have non-authorized access.

3.1. DB Recovery

DB recovery

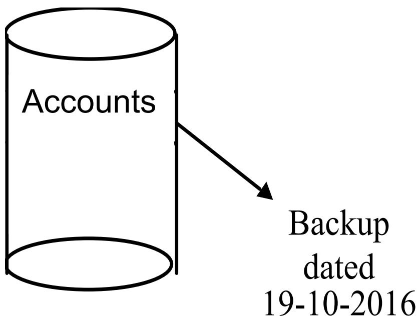
The transaction properties of atomicity and durability force a DBMS to ensure that:

- If confirmed, the changes performed are recorded in the DB to make them persistent.

- If cancelled, the changes performed over the DB are undone.

Database recovery: Example

Backups alone are not sufficient



20-10-2016: Update on accounts
Transaction #1: Update ….

Transaction #51: System failure !

Recovery Procedure:
- Replace the file “Accounts” with its backup

Negative effect:
- The updates of 50 transactions are lost

DB Recovery

- Backups alone are not the solution to the recovery problem.
    - The increase of the backup frequency is not a feasible solution.

- DB technology provides much more efficient and robust techniques for DB recovery.

> Lost of confirmed data is inadmissible with current technology.

Causes of transaction failure

1. **Local to the transaction** (normal system operation)
    - Transaction error (incorrect DB access, failed calculations, etc.)
    - Exceptions (integrity violation, security problems, etc.)
    - Concurrency control (locked state between two transactions)
    - Human decision (inside a program or explicitly).

2. **Extern to the transaction** (system error)
    - A. System failures with loss of main memory.
    - B. Failures in the storage system with loss of secondary memory (disk failure, human errors, virus infection,…)

A. Failures of system main memory

- In the time period between transaction confirmation and recording of the fields in secondary memory.
- The transaction is confirmed and its changes are located in blocks in the memory buffers.
- In this time interval, there is a failure with loss of main memory and the blocks in the buffers are lost.

B. Failures of secondary memory

- A confirmed transaction whose changes have been recorded into the DB
- Then there is a failure in secondary memory and changes are lost.

3.1.1 Recovery from failures of system main memory

Recovery from failures of system main memory

What to do

- Recover confirmed transactions which have not been recorded.
- Cancel transactions which have failed.

Who: Recovery module

How: Most used technique: Use a *journal file (or log, "fichero diario").*

Transaction implementation

Two kinds of transaction implementation (depending on the DBMS):

- Immediate Update
  Updates have an immediate effect on secondary memory.
  In case of cancellation, they have to be undone.

- Deferred Update
  Updates only have immediate effect on main memory.
  The updates are only transferred to secondary memory when confirmed.

- Both need a log file

Logfile

Activities and events recorded in the log file:

- Record the update operations performed by existing transactions.

- The log file is stores in disk to avoid loss after a system failure.

- It is dumped periodically into a massive storage unit (magnetic tape, magnetic or optical disk,…).

Types of entries in a log file

[**start**, T]: A transaction with identifier T has been started.

[**write**, T, X, value_before, value_after]: The T transaction has performed an update instruction on data X.

[**read**, T, X]: The T transaction has read data X.

[**confirm**, T]: The T transaction has been confirmed.

[**cancel**, T]: The T transaction has been cancelled.

We assume ***INMEDIATE UPDATE***

Failure of a transaction $T \rightarrow$ Undo changes performed by T

- Update the data which has been modified by T with its original value (value_before).

- Search for the entries in the logfile

  [write, T, X, value_before, value_after]

System failure with loss of main memory

- Unconfirmed transactions

  [start, T] in the logfile without *[confirm, T]*

  Undo changes performed by *T* (previous process)

- Confirmed transactions

  [confirm, T]

  Execute them again:

  [write, T, X, value_before, value_after]

PROBLEMS:

- Size of logfile can increase very quickly.
- Recovery in case of failure is very expensive (many instructions have to be redone).

SOLUTION:

checkpoints *("puntos de verificación")*

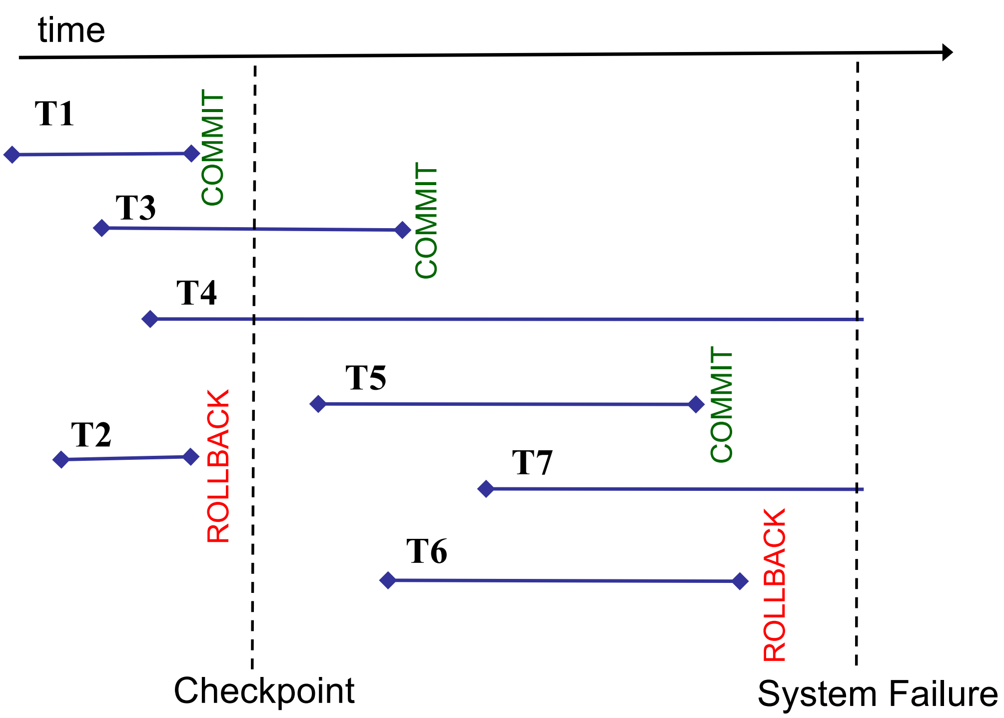
Checkpoints → Are recorded in the logfile periodically

HOW:

- Suspend the execution of transactions temporally.
- Record a checkpoint in the logfile.
- Force the recording of all updates of the confirmed transactions (copy all main memory buffers to disk).
- Resume the execution of the suspended transactions..

DB recovery with **immediate** Update

Recovery the DB from the last checkpoint



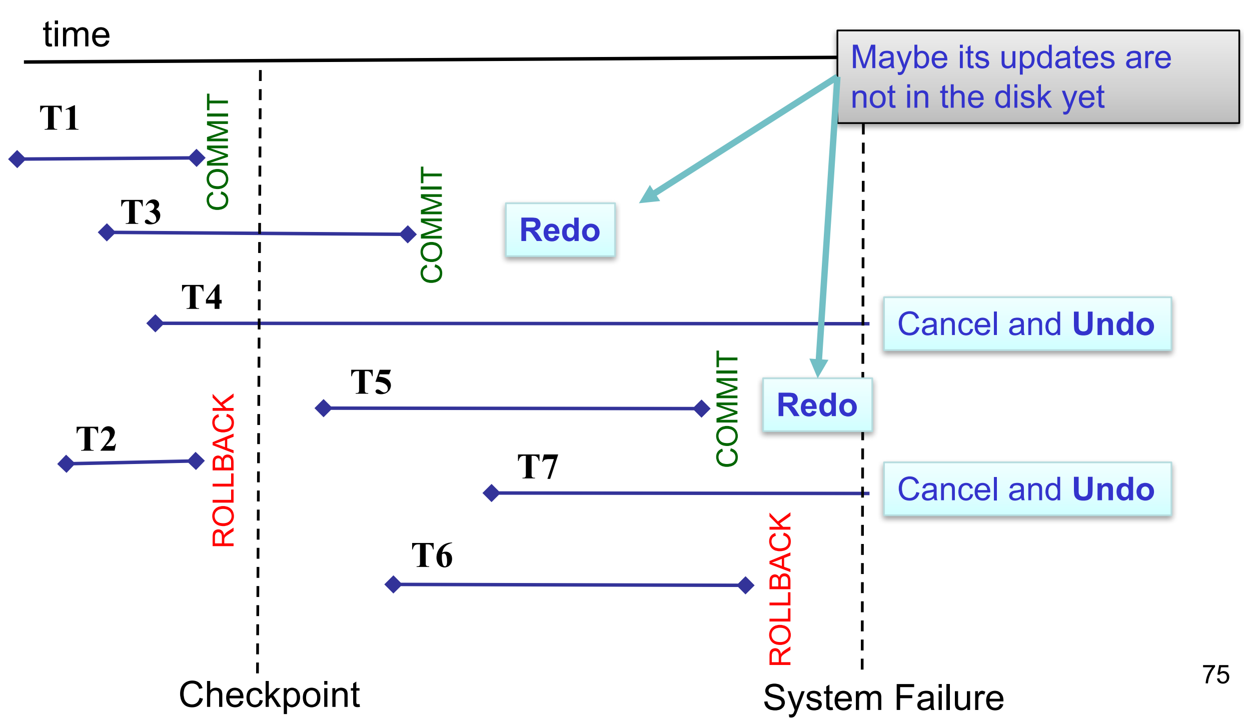
DB recovery with **immediate** Update

Basic considerations:

- Updates performed by confirmed transaction (*a transaction commit appears in the logfile)* could have not been transferred to disk because the buffer block where they are, has not been recorded yet: **Redo**
- Updates performed by non-confirmed transactions (there is no transaction commit in the logfile) could be in disk because their main memory blocks were transferred to disk: **Undo**
- When a checkpoint is recorded, the DBMS records all the updates performed by the confirmed transaction.

DB recovery with **immediate** Update

Recovery the DB from the last checkpoint



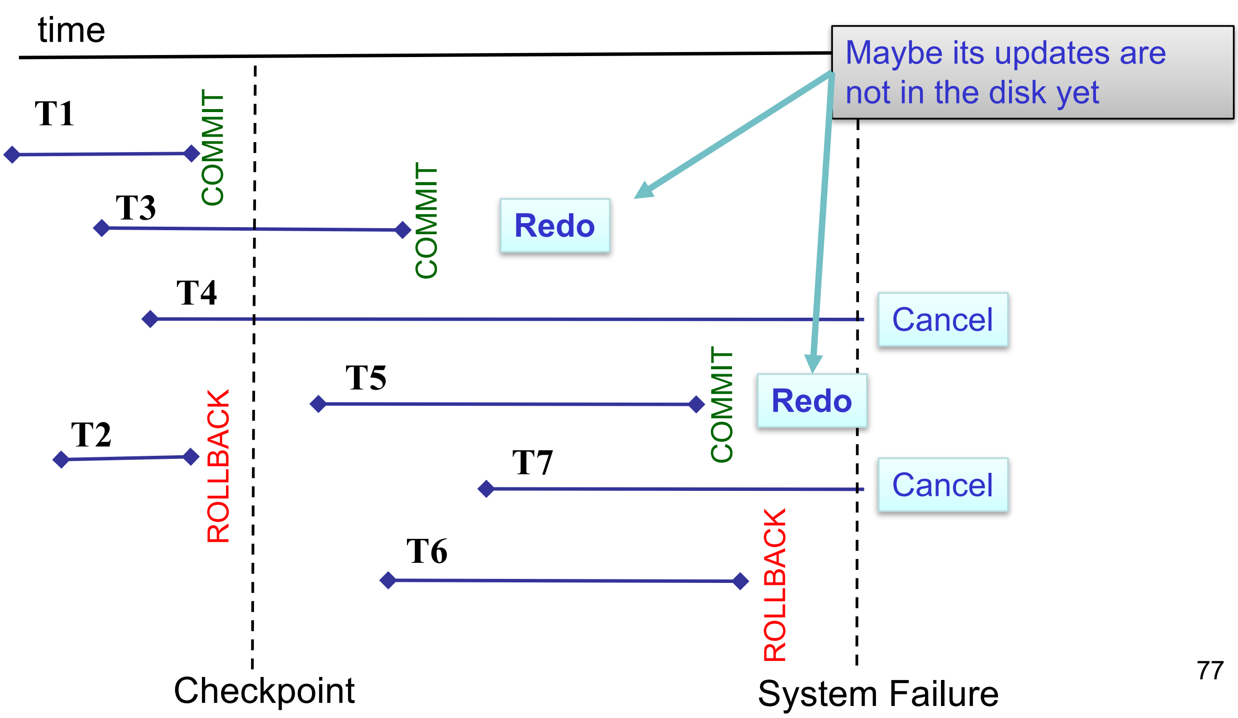
DB recovery with **deferred** Update

Basic considerations:

- Updates performed by confirmed transaction (*a transaction commit appears in the logfile)* could have not been transferred to disk because the buffer block where they are, has not been recorded yet: **Redo**
- Updates performed by non-confirmed transactions (there is no transaction commit in the logfile) are not in disk : **Do nothing**
- When a checkpoint is recorded, the DBMS records all the updates performed by the confirmed transaction.

DB recovery with **deferred** Update

Recovery the DB from the last checkpoint



3.1.2 Recovery from failures of secondary memory

78

Recovery from failures of secondary memory

- Failure of the storage system.
- The database might be damaged totally or partially.
- Technique: reconstruction of the database:
    - Using the most recent backup
    - From the backup moment, the system uses the logfile to redo all the instructions performed by the confirmed transactions.

3.2. Security

3.2. Security

Objective:

Information can only be accessed by the people and processes that are authorized and in the authorized way

Techniques

- User identification
- Establishment of allowed accesses:

Modes
  - Authorization list (objects and allowed operations). *(GRANT)*
  - Level of authorization (less flexible).

- Management of transferrable authorizations:

  Handover of authorizations from one user to another. *(WITH GRANT OPTION)*

Management of authorizations in SQL

Privilege_definition::= **GRANT**
{ ALL |
SELECT |
INSERT [(attribute$_1$,…, attribute)] |
DELETE |
UPDATE [(attribute$_1$,…, attribute)]
}
ON table **TO** {user$_1$,…, user$_m$ | PUBLIC}
[**WITH GRANT OPTION**]

- With the PUBLIC clause all the users have the privilege

3.2.1. Privacy and Security

Ethical and legal implications

Extreme care on:

- Protection against the access or spreading of personal data to non-authorized users.
- Control the flow (to third parties) of information that can contain personal data or information that is apparently aggregated (parameterized queries) but that might reveal particular information for some parameters.
- On occasions, it is even mandatory to communicate the very creation of a single database (if it contains personal information). In Spain, this has to be communicated to the ***"Agencia de Protección de Datos"***.
- Custody of security backups, retired or malfunctioning disks, etc.
- Password precaution (especially the database administrator).
- Small devices (USB disks or sticks, smart phones, tablets, etc.): lost or stolen very easily.

Exercises

1.- The physical schema of a database is modified due to a change of disks. If an application uses a view that in turn uses tables that are stored in any of these disks, what happens to the application?

Justify the answer according to the ANSI/SPARC architecture and the concept of independence.

The application is not affected since **physical independence** prevents a disk change (**physical** schema change) from affecting the **logical** schema. Therefore the external schema is not affected either and the applications will work exactly as they did.

2.- Consider two transactions T1 and T2, which are running concurrently, both working on a piece of data X. Indicate whether one or more properties of transactions are not satisfied here. Briefly justify the answer.

Atomicity
Consistency
Isolation
Durability.

The **isolation**[1] property is not met because the transaction T2 should not see the modification by T1 over X, since both are concurrent.

Time

T1	X in T1	T2	X in T2
read(X)	5		
X=X+1	6		
write(X)	6		
	6	read(X)	6
confirm	6	…	6
…			

Isolation: All the modifications introduced by a non-confirmed transaction are not visible to other transactions

3. How can be recovered a DB from a main memory failure using a logfile and checkpoints ? (Assume immediate updates)

The recovery module must:

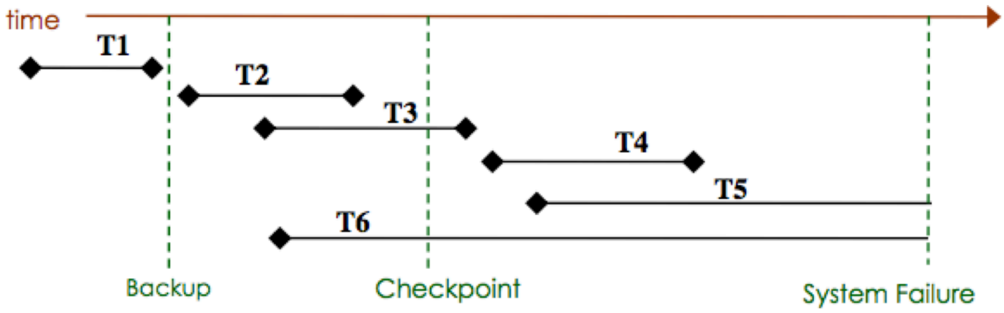
Undo, using the *before_value* in the logfile, all the changes performed by the active non-confirmed transactions since the last checkpoint.

Redo, using the *after_value*, all the changes performed by the transactions that were confirmed after the last checkpoint.

4. Consider the following time diagram, and assuming a DBMS with immediate update. If a system failure occurs, as illustrated in the following figure:

a) What should the DBMS do if the system failure is a main memory loss?

b) What should the DBMS od if the failure is a secondary memory loss ?



a) Using the logfile, the DBMS have to **redo** T3 and T4, and **undo** T5 and T6.
b) Restore the backup and **redo** T2, T3, and T4