VECCHIO ESERCIZIO RIPASSO

A university owns many campus (each one has a name and an address). Each campus has one or more parking areas with several parking lots. Within a parking area, parking lots are numbered starting from 1. Each parking area has a unique name and is described also by the number of parking lots.

The university employees and students can rent a parking lot in the campus where they work or study. The rental contract starts on a given date and ends on another date. These dates can vary depending on the person.

One person can rent from the university only one parking lot at a time but we have to consider that upon a contract renewal the specific parking lot might change. The system must keep also the data of the old contracts.

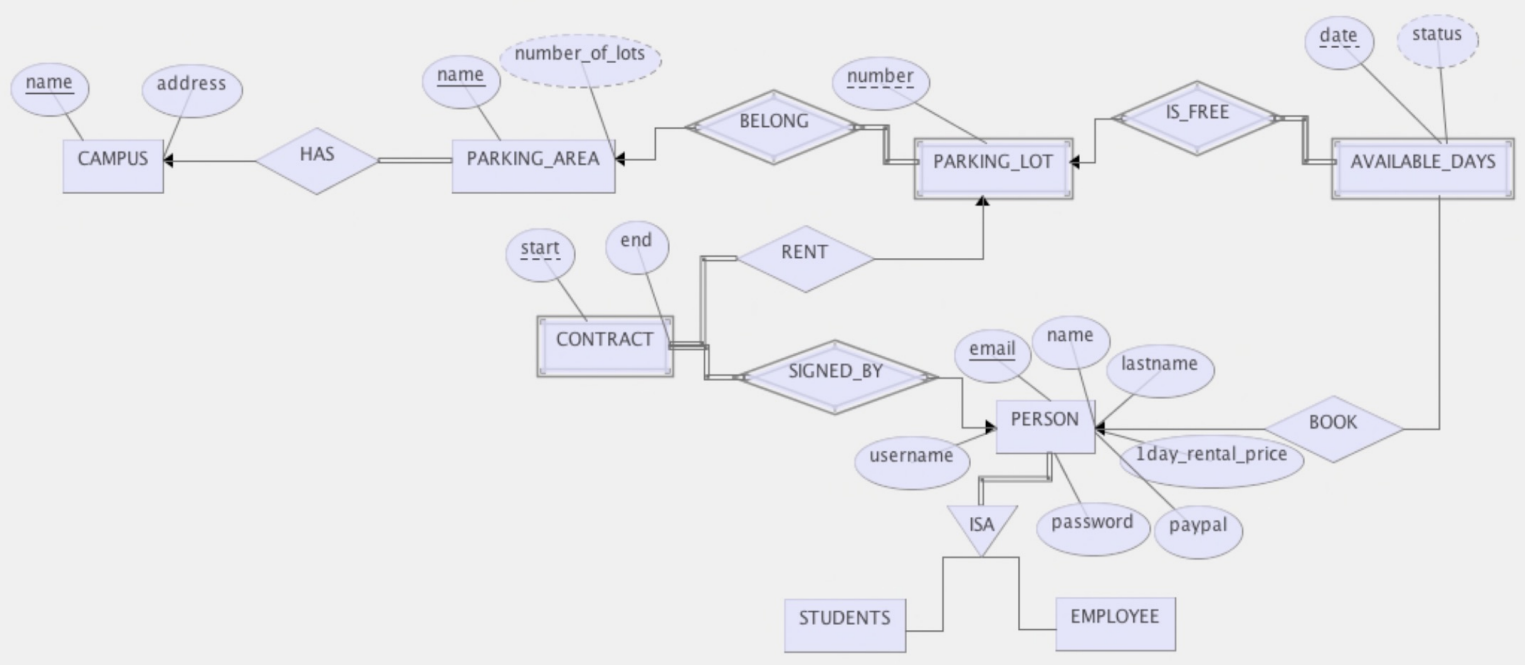
The system we need to implement must allow a renter to make its parking lot available to other users when she/he is absent (vacation, illness, etc.). Other users can book the parking lot when it is available. Therefore, it is necessary to: 1) manage the users personal info (name, lastname, email), 2) know which parking lots are occupied or free.

Each renter can freely set the price for renting out her/his parking lot for one day.

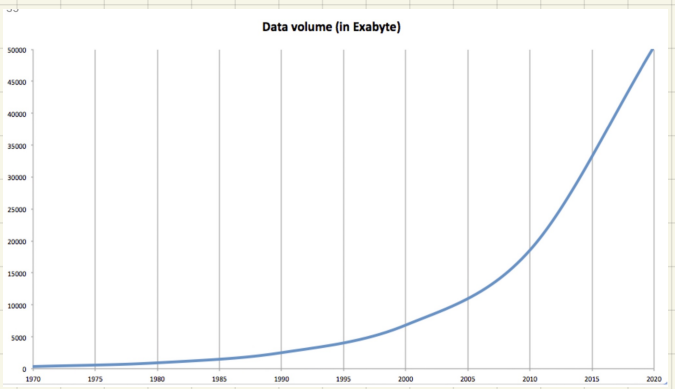
The users of the parking system must register to the website, they are given a username and a password and they are requested to provide the Paypal account id used for transferring money.

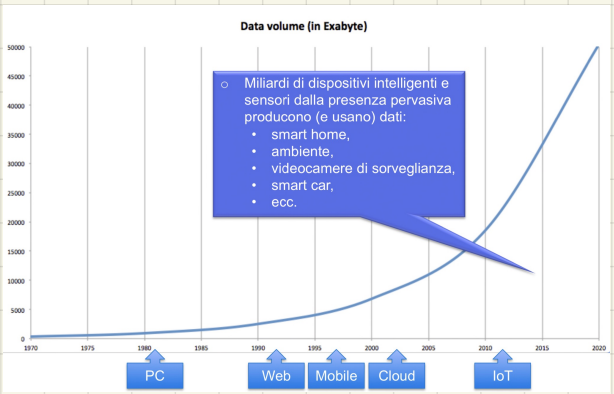
Design the database conceptual structure using the Entity-Relationship model

Schema DB

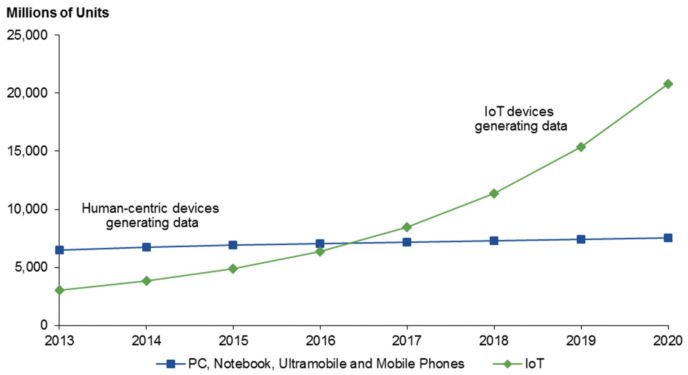


DATA EVOLUTION, PREDICTIVE ANALITYCS



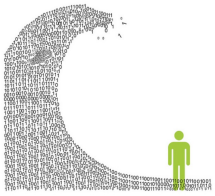


Chi produce i dati ?



Le sfide dei BigData

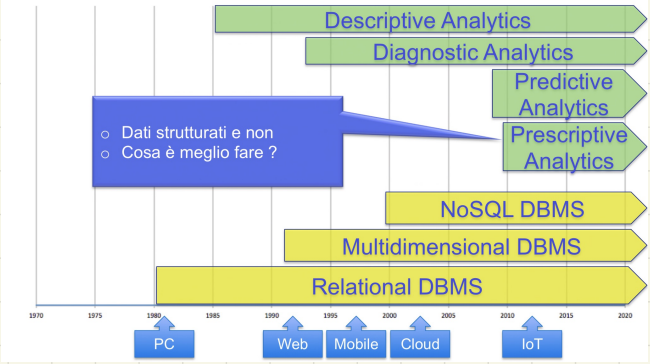
- Quantità di dati spropositata
  - Volume dei dati nel cloud cresce del 32% all'anno
  - Entro il 2018 il 76% dei dati sarà nel cloud
- Possedere e gestire grandi moli di dati costa:
  - Infrastruttura e personale
  - Acquisire, filtrare, pulire, integrare i dati
  - Protezione:
    - Dalla perdita
    - Dal furto / accesso non autorizzato
    - Per questioni legali (privacy, compliance)
- Difficile farne uso in modo semplice e utile per i propri scopi
  - Fonti e formati eterogenei
  - Molto "rumore"
  - Servono strumenti
  - Servono nuove competenze



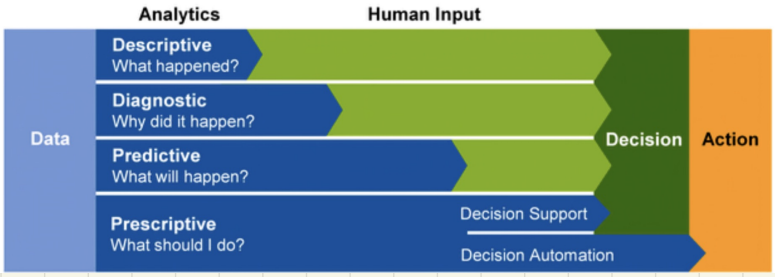
Le opportunità che offrono i BigData

- Estrarre informazioni e conoscenza (varia a seconda del contesto):
  - Comprensione di fenomeni
  - Supporto nel prendere decisioni
  - Analizzare gli effetti di una decisione
  - Miglioramento di prodotti e servizi
  - Miglioramento di processi
  - Conoscenza di interessi e comportamenti dei potenziali clienti
  - Ecc.
- Nuovi business model basati sulla vendita di dati/conoscenza

Evoluzione sistemi per la gestione dei dati e data analysis



L'Analytics in sintesi



Cosa si può fare con l'Advanced Analytics ?

- Scoperta di relazioni tra i dati
  - Es.: chi gradisce X di solito gradisce anche Y
- Riconoscimento di pattern
  - Es.: riconoscimento di oggetti simili in immagini diverse
- Previsione di trend e comportamenti (behaviors)
  - Es.: clienti che è più probabile perdere

Strumenti per l'Advanced Analytics (1)

- Statistica (medie, deviazione standard, regressione lineare)
  - Inferire caratteristiche generali da un campione di dati
    - es. Identificazione eccessi di carico della rete
- Clustering
  - Raggruppa elementi che hanno certe caratteristiche comuni
    - es. Segmentare la clientela in base alle abitudini di acquisto
  - Identifica elementi fuori dalla norma (outliers)
    - Es. Comportamenti anomali: fraud detection
- Association rules
  - Identifica associazione tra caratteristiche
    - es. chi compra la birra di solito compra anche le patatine

Strumenti per l'Advanced Analytics (2)

Machine learning: una volta "allenato" il modello, consente di classificare i nuovi dati sfruttando la similitudine con quelli conosciuti

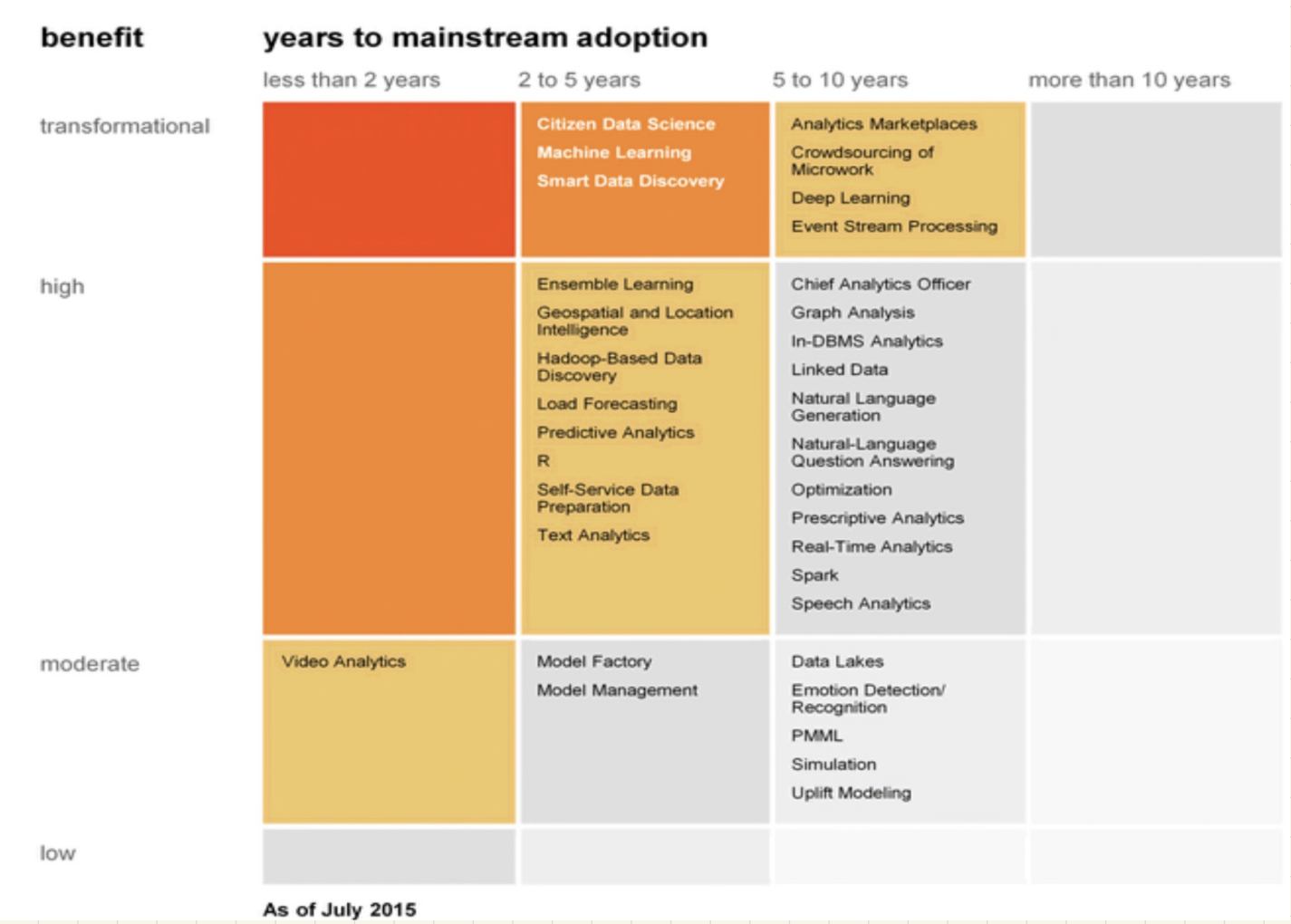
- Decision Trees e Rule induction
- Neural networks
- Naive Bayes

- Es. Classificazione sintomi di una malattia a fini diagnostici

Le Tecnologie per i BigData

- Sistemi NoSQL
  - schemaless: scalabilità e flessibilità
- Hadoop
  - File system distribuito e relativo ecosistema (MapReduce, H-base, Mahout, Hive, Pig, Scoop, etc.)
- Spark
  - MapReduce in memory, high performance, streaming, machine learning
  - Supporta Java, Scala, Python, R
- R
  - Linguaggio e ambiente di sviuppo con funzioni statistiche avanzate

Analisi impatto delle tecnologie (Gartner Priority)



DBMS (DataBase Management System)

What is a DBMS?

- ✓ **A database management system is a software system able to manage collections of data that are large, shared, persistent and to ensure their reliability[Ceri]**
- ✓ **A database management system**
  - ✓ allows you to define the structures for data storage
  - ✓ provides mechanisms for data manipulation
  - ✓ provides services that allow concurrent access to data
  - ✓ provides mechanisms to ensure data integrity and consistency
- ✓ **A DBMS is a SW product, usually commercially available, that installs on a system and is ready to provide a series of services related to data management**
- ✓ **DBMS are designed to handle large amounts of data**

What is a database?

- ✓ **A database is a collection of data, used to represent information of interest to an information system, more technically, a collection of data managed by a DBMS [Ceri]**
- ✓ **A database is therefore a set of data that represents a certain reality and that are**
  - ✓ properly structured
  - ✓ managed by a DBMS

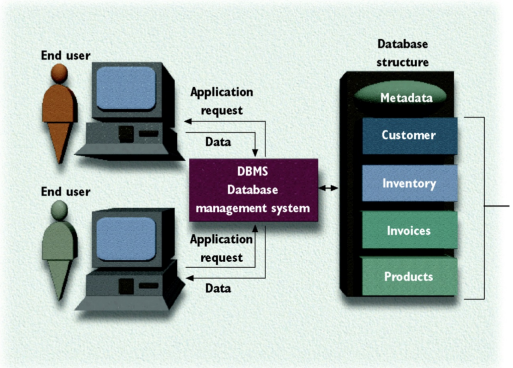
DBMS and databases

- ✓ You can access the data stored in the database only through the DBMS services
- ✓ You can startup and shutdown a DBMS
- ✓ When the DBMS is stopped
  - ✓ the database is inaccessible
  - ✓ no system resources are used (except for files containing the database)
- ✓ When the DBMS is running
  - ✓ the database is accessible
  - ✓ there are, on the system that hosts the DBMS, the processes that implement the DBMS itself

DBMS and databases

- ✓ **The purpose of the DBMS is to make sure that the data contained in the database are**
  - ✓ available for users
  - ✓ consistent
  - ✓ protected (safety, security)

DBMS and database



The DBMS manages the interaction between the applications and the databases

DBMSs were developed to handle a number of recurring problems that occurred during the development of every application using file based systems supported by conventional operating systems;

DBMS fa tutto da solo automaticamente:
- check dei problemi e risoluzione

↑

DBMS vs file system

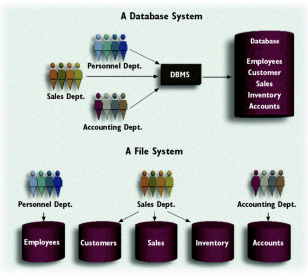
Problems we were having:

- ✓ inconsistency and redundancy of data
  - ✓ the programs may have been written by various programmers in different languages at different times
  - ✓ the same information can be duplicated, redundancy can lead to inconsistency
- ✓ difficulties in accessing data
  - ✓ it is difficult to manage custom queries, they require writing a program
- ✓ data isolation
  - ✓ data can be stored in different files with different formats
  - ✓ it is complex to write new programs that access this data

Other problems …

- ✓ integrity issues
  - ✓ you cannot define and/or modify integrity constraints globally without writing or editing programs
- ✓ atomicity of the changes
  - ✓ a sequence of changes may introduce inconsistencies if it is not fully executed (e.g. HW problem)
  - ✓ the sequence must be managed as a single indivisible operation.
- ✓ concurrent access by multiple users
  - ✓ multiple users who concurrently modify data may introduce inconsistencies if changes are not properly sequenced
- ✓ security
  - ✓ not all users must have access to all data

DBMS vs file system (cont.)



Comparison between database and file system based solutions

DBMS CARATTERISTICS

Main functions of a DBMS

Data dictionary management
- ✓ the data dictionary contains the definitions of the data and the relationships between them (metadata)
- ✓ provides data abstraction and removes structural dependencies from the system

Data storage management
- ✓ The DBMS creates data storage facilities and relieves the programmer from dealing with the physical characteristics of the data.

Data management
- ✓ the DBMS avoids the programmer having to manage the distinction between physical and logical format of the data

Security management
- ✓ the DBMS provides data security and privacy within the database
- ✓ data security is particularly important in a multi-user database

Multi-user access control
- ✓ The DBMS ensures that multiple users can access the data simultaneously while maintaining its consistency.

Backup and recovery management
- ✓ DBMS provides backup and recovery procedures to ensure data security and integrity

Data integrity management
- ✓ the DBMS allows the definition of rules (constraints) of integrity that then takes care of applying to the data

What are DBMS for?

To avoid
- ✓ redundancies
- ✓ inconsistencies
- ✓ anomalies due to concurrent access by several users

To provide
- ✓ security
- ✓ data integrity
- ✓ ease of data management and transfer
- ✓ standard

Pros and Cons of DBMS

Pros
- ✓ data can be managed as a common resource at the company level and represent its reality
- ✓ centralised management
- ✓ availability of integrated services
- ✓ reduction of redundancies and inconsistencies
- ✓ data independence

Cons
- ✓ cost of the product and associated tools
- ✓ cost and complexity of migration

When is a DBMS useful?

- ✓ Large and/or complex databases
- ✓ Need to ensure data persistence
- ✓ Multi users
- ✓ Frequent data changes
- ✓ Ad hoc queries
- ✓ Access through various applications
- ✓ Security and access protection
- ✓ Integrity Constraints
- ✓ Backup and Recovery

When you don't need a DBMS?

- ✓ Costs and benefits must be assessed on a case-by-case basis
  - ✓ investment in hardware, software and training
- ✓ A DBMS is not of great advantage when
  - ✓ database + applications are simple and do not evolve
  - ✓ small database
  - ✓ real-time application
  - ✓ Concurrent access not required

Types of use of a DBMS

Architecture
- ✓ centralized
- ✓ distributed

Types of application
- ✓ OLTP: OnLine Transaction Processing
  - ✓ Transactional: many users performing short transactions
- ✓ OLAP: OnLine Analytical Processing
  - ✓ Few users who perform very large transactions
  - ✓ decision support / business intelligence (BI)
  - ✓ data warehouse

Main Relational DBMS

Commercial Products
- ✓ Oracle
- ✓ Microsoft SQL Server
- ✓ IBM DB2

Open Source
- ✓ Oracle MySQL
- ✓ PostgreSQL

https://db-engines.com/en/ranking

TRANSACTION, CONCURRENCY AND CONSISTENCY

Transaction

- A *transaction* is the part of the execution of a program that accesses and/or updates a certain amount of data. It can be composed of several SQL commands.
- In order for a transaction to be successful, all the operations that are part of it must complete succesfully
- Otherwise the transaction fails (and all changes are cancelled)
- A transaction must "see" a consistent *database*
- During the execution of a transaction the *database* may not be consistent, from the point of view of the transaction in question
- When the transaction is completed (*committed*), the *database* must be consistent
- It is the responsibility of the application *designer to* ensure that transactions are properly defined.
- Two main problems to be addressed
  - various types of failures, hardware failures, system crashes, application crashes
  - concurrent access to the same data by multiple transactions

ACID properties

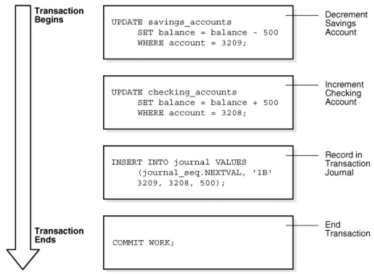
To preserve data integrity, the *DBMS* must ensure:

- ***Atomicity***: either all the operations of a transaction are reflected in the database or none are.
- ***Consistency***: The DBMS ensures that none of the DB's integrity constraints are violated. Executing a transaction in isolation preserves the consistency of the database.
- ***Isolation***: Although several transactions may be carried out in a concurrent manner, each transaction must be unaware of the others. That is, for each pair of transactions Ti and Tj, it must appear to Ti that Tj finished execution before Ti started, or that Tj started execution after Ti finished.
- **Durability (*durability*)**. After a transaction has been successfully completed, changes made to the database persist, even in the event of system failure.

↳ after committed the data must be there

Example of fund transfer

- Transaction T1 for the transfer of $500 from account 3209 (A) to account 3208 (B):



- ***Consistency requirement*** - the sum of A and B remains unchanged after the execution of the transaction

↳ nessun soldo deve apparire o sparire

Example of fund transfer (cont.)

- **Atomicity requirement** - if the transaction fails in step 1 and before step 4, the system must ensure that updates are not reported in the database, otherwise it would be inconsistent. -> tutto fatto o no
- **Durability requirement** as soon as the user is notified that the transaction has been completed (i.e., the transfer of $500 has taken place), database updates by the transaction must persist despite possible failures -> deve durare per sempre
- **Isolation requirement**
  - if between the first UPDATE and the last command a further transaction had access to the database (at that point still only partially updated), it would see a non consistent image (the sum of A+B would be less than it should). This should not happen.
  - The requirement could be ensured in a trivial way by executing transactions in serial mode, that is, one after the other. But this is not feasible because of performance (throughput).

↳ se funziona senza sapere cosa l'altra transaction stia facendo

Serial execution of transactions

- A DBMS, having to support the execution of several transactions that access shared data, could execute these transactions sequentially. ("serial execution")
- For example, the T1 and T2 transactions could be executed in this way:

T1	T2
R(X)	
W(X)	
COMMIT	
	R(Y)
	W(Y)
	COMMIT

Concurrent execution of transactions

- Alternatively, the DBMS can execute multiple transactions in concurrently, alternating the execution of operations of one transaction with those of other transactions ("interleaved execution").
- Performing multiple transactions concurrently is necessary to ensure good performance:
  - It takes advantage of the fact that, while one transaction is waiting for the completion of an I/O operation, another can use the CPU, which leads to an increase in system throughput (no. of transactions processed in the time unit).
  - If you have a "short" and a "long" transaction, concurrent execution leads to reduced average system response time

T1	T2
R(X)	
	R(Y)
	W(Y)
	COMMIT
W(X)	
COMMIT	

-> tutte le modifiche effettuate ad una tabella sono visibili solo alla transazione in questione, gli altri non vedono nulla.

-> i livelli di isolamento impediscono una modifica ai dati, così il risultato della transaction non cambia e il DB rimane consistent.

Concurrency issues

- The Transaction Manager must ensure that competing transactions do not interfere with each other. If this does not happen, you may have 4 basic types of problems, exemplified by the following scenarios:
  - **Lost Update**: two people, in two different agencies, both buy the last ticket for the U2 concert in Zurich (!?)
  - **Dirty Read**: in the program of the concerts of U2 there is a stop in Lausanne on 15/07/21, but when you try to buy a ticket for that date you are told that in reality it hasn't been fixed yet (!?)
  - **Unrepeatable Read**: for the U2 concert (finally the date has been fixed!) you see that the price is 40 Fr., think about it for 5 minutes, but the price in the meantime has risen to 50 Fr (!?)
  - **Phantom Row**: you want to buy tickets for all the U2 stages in Switzerland, but when you buy the tickets you find out that the stages are no longer two but have become 3 (!?)

Lost Update

- The following schedule shows a typical case of lost update, in which for convenience we highlight also the operations that modify the value of the data X and we show how the value of X in the DB varies

This update is lost

T1	X	T2
R(X)	1	
X=X-1	1	
	1	R(X)
	1	X=X-1
W(X)	0	
COMMIT	0	
	0	W(X)
	0	COMMIT

- The problem arises because T2 reads the value of X before T1 (who has already read it) changes it.

Dirty Read

- In this case, the problem is that a transaction reads a value that's not there:

This read is "dirty"

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
ROLLBACK	0	
	0	.....
	0	COMMIT

- The work done by T2 is based on an 'intermediate' value of X, and therefore not stable ('the final date is not 15/07/21').
- The consequences are unpredictable (depends on what T2 does) and would occur even if T1 did not abort.

Unrepeatable Read

- Now the problem is that a transaction reads a data twice and finds different values ("the price has increased in the meantime"):

The 2 queries are inconsistent with each other

T1	X	T2
R(X)	0	
	0	R(X)
	0	X=X+1
	1	W(X)
	1	COMMIT
R(X)	1	
COMMIT	1	

- Even in this case there can be serious consequences
- The same problem arises for "analysis" transactions
  - For example, T1 adds up the amount of 2 current accounts while T2 carries out a transfer of funds from one to the other (T1 may therefore give an incorrect total).

il dato viene cambiato durante la transaction.

Phantom Row

- This case may arise when rows that another transaction should logically consider are entered or deleted
  - In the example the tuple t4 is a "phantom", as T1 "does not see it".

prog

CodProg	Sede	
P01	Zurigo	t1
P01	Lugano	t2
P02	Lugano	t3
P03	Lugano	t4

T1:

```
UPDATE Prog SET Sede = 'Basel'
WHERE Sede = 'Lugano
```

T1 does not see this tuple

T2:

```
INSERT INTO Prog VALUES ('P03', 'Lugano')
```

T1	T2
R(t2)	
R(t3)	
....	
W(t2)	
W(t3)	
	INSERT(t4)
.....	
COMMIT	
	COMMIT

la riga P03 viene aggiunta e committata dopo il commit finale di T1, quindi non viene vista.

-> Se viene effettuato un Rollback dopo il commit non succede nulla, perché sta già iniziando un'altra transazione.

SQL commands for transactions

- A data manipulation language must provide a construct that allows you to specify the set of actions that form a transaction
- In SQL, a transaction starts implicitly
- As it ends with:
  - COMMIT WORK - makes the current transaction final and starts a new one
  - ROLLBACK WORK - cancel the current transaction
- A DDL command executed by the transaction causes it to end with an implicit COMMIT.

- The transaction model used by DBMS is actually more articulated; in particular it is possible to define so-called "savepoints", which are used by a transaction to only partially undo the work done
  - An intermediate step in a transaction can be defined as a *roll back* point:
    - SAVEPOINT label_name
  - You can *roll back to* a midpoint with:
    - ROLLBACK WORK TO SAVEPOINT label_name

- In certain cases, in which very long queries have to be executed (preparation of reports, statistics, ...), it is necessary that the data on which the operation runs are not changed during the process. That is, update, insert, delete executed by other transactions must be invisible to the transaction in question, which, if the DBMS isolation level does not already provide this, we can use the command
  - SET TRANSACTION READONLY -> anche se altre transactions fanno commit, non succede nulla perché solo readonly.
- The COMMIT command returns the transaction to its normal behavior

↳ non vedo i cambiamenti di altre transactions

Failure of a transaction

- A transaction may fail due to the following reasons:
  - Transaction failure:
    - logical errors: the transaction cannot be completed due to internal error conditions
    - system errors: the DBMS must abort an active transaction due to an error condition
  - System Failure: power failure, hardware failure or software errors cause the system to crash
  - Media Failure: A failure of the head or similar *device* can destroy a *storage* system in part or in full.
- **When a transaction fails, it is cancelled** (*rollback*) -> to savepoint

SQL commands for transactions


```
Example:
     SELECT * FROM Department
     INSERT INTO Department(DeptNo, DeptName, AdmrDept) VALUES ('X00','new dept 1',
        'A00')
     SAVEPOINT foo
     SELECT * FROM Department
     INSERT INTO Department(DeptNo, DeptName, AdmrDept)VALUES ('Y00', 'new dept 2',
        'A00')
     SELECT * FROM Department
     ROLLBACK WORK TO SAVEPOINT foo
     SELECT * FROM Department
     COMMIT WORK
```

The management of concurrency

- ✓ Locks
- ✓ Lock Types (Share and Exclusive)
- ✓ Granularity of Locks (table, page, row)
- ✓ Implicit Locks
- ✓ Explicit Locks and SQL
- ✓ Deadlocks

The Locks

- ✓ **A common technique used by DBMS (and not only) to manage concurrent access to data is the use of locks:**
  - ✓ Locks are a mechanism commonly used by operating systems to regulate access to shared resources. -> come Java
  - ✓ To perform an operation it is first necessary to "acquire" a lock on the resource (e.g. a row in a table).
- ✓ **The purpose of the locks is to insure:**
  - ✓ *consistency* - assure users that the data they are manipulating will not be modified by others until they have finished it
  - ✓ *integrity* - ensure that the data and database structures reflect changes in the correct sequence

Lock Types: Shared and Exclusive Locks

- ✓ The locks can be of 2 types:
  - ✓ **Exclusive lock** mode prevents the associated resource from being shared. This lock mode is obtained to modify data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released.
  - ✓ **Share lock** mode allows the associated resource to be shared, depending on the operations involved. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer (who needs an exclusive lock). Several transactions can acquire share locks on the same resource.

Duration of Locks

- ✓ All locks acquired by commands that are part of a transaction are maintained until the end of the transaction, i.e. they are released when a COMMIT or ROLLBACK is executed.

Granularity of the Locks

- ✓ Each type of DBMS provides *locks* with different granularity. Usually at the level of:
  - ✓ table (*table level*), block (*page or block level*) or *row level*
- ✓ A page-level lock causes all rows that physically share the same page to be locked, similarly for table-level locks.
- ✓ Each system has its own way of working, for example:
  - ✓ SQL server supports (and also chooses independently from) the 3 levels. If you put the lock on a line use the row level. If on more than one, it switches to the *page level* (*lock escalation*)
  - ✓ Oracle supports *row level* and *table level* and always operates in *row level* mode for DML commands.

Implicit Locks

- ✓ "Because the locking mechanisms of Oracle Database are tied closely to transaction control, application designers need only define transactions properly, and Oracle Database automatically manages locking."
- ✓ "Oracle Database locking is fully automatic and requires no user action. Implicit locking occurs for all SQL statements so that database users never need to lock any resource explicitly"
- ✓ When a DML command is executed to modify the contents of the database (INSERT, UPDATE, DELETE), two-level locks are automatically set:
  - ✓ Row (exclusive, to protect data from competing changes)
  - ✓ Table (to prevent the table from being modified / deleted during the operation)
- ✓ A transaction puts exclusive locks on the lines modified, inserted or deleted by the transaction itself

Explicit Locks

- ✓ Although the problem of handling competing transactions is the responsibility of the DBMS, which simplifies application code, SQL provides two basic mechanisms to influence the way a transaction is performed
  - ✓ Explicitly request an exclusive lock on rows of a table, e.g.

```
SELECT ... FROM ... WHERE ... FOR UPDATE;
```

  - ✓ If you know you have to process many rows of a table, you can explicitly request a lock on the whole table with the command LOCK TABLE

Explicit Locks (cont.)

- ✓ To create an exclusive lock on the lines you think you want to modify, you need to use the "FOR UPDATE" clause in the SELECT command

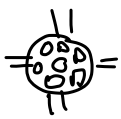
```
SELECT a,b FROM tab1 WHERE a>3 FOR UPDATE
```

- ✓ What happens if the line is already blocked?
  - ✓ Normally, the SELECT does not return until the lock is removed.
  - ✓ If you want to return the SELECT without waiting, you must add the NOWAIT keyword at the end of the command. If the SELECT encounters blocked lines it will return immediately with an error
  - ✓ You can also limit the waiting time with the WAIT N clause (N= time in seconds).
- ✓ Normally a SELECT ... FOR UPDATE is then followed by an UPDATE command
- ✓ The *lock* is removed after COMMIT or ROLLBACK
- ✓ Both the behaviour in case of already locked resource and the removal of the lock work in a similar way in case of lock on the whole table (LOCK TABLE).

Deadlock

-> sono bloccati tutti insieme, perché tutti bloccano qualcosa che blocca tutti

es. rotonda piena di auto:



- ✓ Let's consider the following partial *schedule*:

T3	T4
Lock-X(B)	
Read(B)	
B=B-50	
Write(B)	
	Lock-S(A)
	Read(A)
	Lock-S(B)
Lock-X(A)	

- ✓ Neither T3 nor T4 can advance - by executing lock-S(B), T4 must wait for T3 to release its *lock* on B, while by executing lock-X(A), T3 must wait for T4 to release its *lock* on A.
- ✓ This situation is called *deadlock*. To resolve the *deadlock*, one between T3 and T4 must be (partially) *rolled back so* that the *locks* are released.

Deadlock (cont.)

- ✓ A system is deadlocked when in one set of transactions each transaction awaits the end of another.
- ✓ DBMS implement protocols in an attempt to limit the possibility of finding yourself in *deadlock* situations
- ✓ Sequentially executing one transaction at a time would ensure a *deadlock-free* system, but this is not feasible for *performance reasons* and a compromise must be found.

Deadlock recovery

- ✓ Each system implements *deadlock* prevention techniques
- ✓ However, when a deadlock cannot be avoided, and it happens, the system must be able to handle it:
  - ✓ Some transactions will have to be (partially) cancelled (*roll back*) to break the *deadlock*. rollback della transaction meno costosa
  - ✓ You will have to select the transactions that lead to the least sacrifice
  - ✓ You will have to determine how many steps back
    - ✓ Total *rollback*: abort the transaction and then start it again.
    - ✓ Partial *Rollback*: Abortion only takes the necessary steps to break the *deadlock*.
  - ✓ Phenomena called *starvation* you can always have the same transaction is selected as the victim
  - ✓ It is necessary for this to include cost factors in the algorithm to avoid the *starvation* phenomenon

ogni trans può decidere il suo isolation level, se non lo decide viene settato quello di default.

The levels of isolation

- On the most advanced systems it is also possible to explicitly choose the "level of isolation" in which you want to execute the transaction
- The SQL standard defines 4 possible isolation levels:
  - **READ UNCOMMITTED** non emette
    - When it's used, the DBMS does not issue shared locks while reading data. So, you can read an uncommitted transaction that might get rolled back later. This isolation level is also called dirty read. This is the lowest isolation level.
  - **READ COMMITTED**
    - This is the default isolation level in SQL Server. When it's used, SQL Server will use shared locks while reading data. It ensures that a physically corrupt data will not be read and will never read data that another application has changed and not yet committed, but it not ensures that the data will not be changed before the end of the transaction.
  - **REPEATABLE READ** -> quando run, select,.. hanno sempre stesso risultato
    - When it's used, then dirty reads and nonrepeatable reads cannot occur. It means that locks will be placed on all data that is used in a query, and another transactions cannot update the data.
  - **SERIALIZABLE** -> DB sempre nella stessa situazione
    - Most restrictive isolation level. When it's used, then phantom values cannot occur. It prevents other users from updating or inserting rows into the data set until the transaction is complete.

Problems related to the various levels of isolation

	Phantom	Urepeatable Read	Dirty Read	Lost Update
Serializable	NO	NO	NO	NO
Repeatable Read	YES	NO	NO	NO
Read Committed	YES	YES	NO	NO
Read Uncommitted	YES	YES	YES	NO

	DB2	ORACLE	SQL SERVER 7.0	PostgreSQL	MYSQL
Serializable	YES	YES	YES	YES	YES
Repeatable Read	YES	NO	YES	NO	YES
Read Committed	YES	YES	YES	YES	YES
Read Uncommitted	YES	NO	YES	NO	YES

To change the isolation level: **SET TRANSACTION ISOLATION LEVEL <MODE>.**

Implementation of atomicity and transaction isolation

- The technique used which consists in replicating in a special memory space the lines that have been modified by the various transactions in progress.
- In this way the system can show at each transaction the data that its level of isolation and its activity require
- Even in the case of rollback of a transaction you therefore have the old copy of the data that can be restored.
- Like this:
  - Each transaction can be isolated from the others at the level that requires
  - Atomicity is implemented because in case of transaction interruption you can restore the previous state

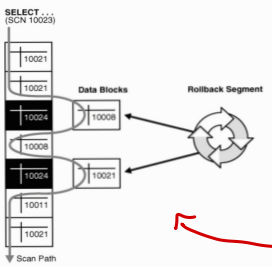
Undo segments of Oracle

- The DBMS must have a space to save the data while it is being modified by a transaction:
  - must be able to provide the transaction that is modifying the data with the updated but not final (uncommitted) copy of the data and must be able to provide the copy of the data prior to the start of the update to all other transactions
  - must be able to do this for multiple and competing transactions
- The structure to host the updated data is the segment UNDO
- UNDO *segments* are physical structures and their number, size and position can be defined by the database administrator (DBA).

Undo (or Rollback) segments of Oracle (cont.)

- In fact, the modified data are kept in the original table and the original lines are copied into the UNDO *segment*
- Access to the data by other transactions is routed to the UNDO segment
- When the changes are final (committed), and there are no serialized transactions in progress, the *UNDO segments* with the old transaction data are cleaned up.
- The sizing of the UNDO *segment* depends very much on the typical load of the database and it is the task of DBA
- The greater the number of simultaneous changes and the greater the amount of data changed, the larger the size of the *UNDO segments* must be
- To reduce conflicts (*contention*), you should set up multiple UNDO segments, which are useful when multiple transactions are executed at the same time.

Multiversion Concurrency Control



"Oracle Database automatically provides read consistency to a query so that all the data that the query sees comes from a single point in time **(statement-level read consistency)**.
Oracle Database can also provide read consistency to all of the queries in a transaction **(transaction-level read consistency)**.
The rollback segments contain the old values of data that have been changed by uncommitted or recently committed transactions.
The picture on the left, shows how Oracle Database provides statement-level read consistency using data in rollback segments."

Implementation of durability

- The transaction log
- Recovery of a database
- DBA and durability
- Disasters

Durability and Recovery

- Let's consider the Ti transaction transferring $50 from account A to account B
- The goal is to ensure that data is not lost after the user has been informed of the successful execution of the operation.
- A lot of output operations (writes to storage) may be required for Ti (to update A and B).
- For performance reasons these entries may occur after the user has received confirmation.
- A fault, or an error, may occur before writes can be made
- To ensure the durability of the transaction, despite possible failures, a description of the changes (of the steps to be taken) is first saved on the *storage* without modifying the *database.*
- The most common approach is the one called *log-based recovery* (or *redo log* in Oracle)

Transaction Log

- A *log* is maintained on the storage
- The *log* is a sequence of records (*log record*) that keeps the history of updates on the *database*, i.e. transactions, of all transactions while they are executed
- When a T transaction starts, it is recorded by writing a *log record of* the type <T start>
- Before a DB P page is modified by the T transaction, the Log will contain a record of the type

(LSN, T, PID, before(P), after(P), prevLSN)

LSN = Log Sequence Number (progressive number of the record in the Log)
T = transaction identifier
PID = modified page identifier
before(P) = is the so-called "before image" of P, i.e. the content of P before modification
after(P) = is the "after image" of P, which is the content of P after the modification
prevLSN = LSN of the previous log record related to T

- When T finishes its last step, the <T commit> *log record* is written

Media Failure

- What happens if you lose the database due to, for example, a hard drive crash?
  - In the case of a media failure, a database *restore* using a recently made backup is done
  - At that point the restored content will be the one present at the time of the backup, so it will be old
  - By inputting the Log to the database during recovery, all transactions that have executed COMMIT since the time of the backup that has been restored are repeated.
  - At the end of the log there will be some transactions not yet committed that will be ignored but all the others will be reproduced.
  - It is obvious that if the Log were to be corrupted, the restore operations could not work properly
  - For this reason it is common to keep the Log at least in duplicate and on different media!

Disasters

- What happens if I haven't backed up?
- What if I put the database and the Logs on the same disk that broke?
- What happens if I have the backup but don't keep the Logs?

I'll probably have to look for another job! 😨

Disasters

- What can I do to take cover in case a disaster happens? (the server, the data center, the building, a cataclysm happens, etc.).
- To think of the appropriate solution there must be clear 2 parameters:
  - RPO - Recovery Point Objective
    - How much time in terms of data work I can afford to lose (e.g. 1 hour of company work)
  - RTO - Recovery Time Objective
    - How long at the most do I have to go back to work
  - The smaller these values, the more expensive will be the solution to achieve them.
- From the trivial transport of backups and logs to a remote location, to the replication of infrastructure and data at a distance

Summary: life of a transaction completed with a commit

- Before you can commit a transaction that has changed data, the following happened:
  - The DBMS has generated rollback *records* in memory
  - The DBMS has generated log records (*log records*) in memory
  - (these records can be put on storage before the transaction is final)
- When the transaction is final, the following happens:
  - The transaction is marked "complete"
  - *Log* records are put on disk, if they are not already on disk.
  - *Locks* kept on rows and tables are released
  - *Rollback* recordings do not need to be put on disk for a difinitive transaction and are deleted.
  - Data still residing in memory *buffers* are saved to *storage*
- Writing *log* records to disks is sufficient as a permanent history of transactions, as a *database* can always be reconstructed from this information at that point so the user can be confirmed the success of the operation

Esercizio TRANSACTION

Transactions 1 (Taken from old test)

Given the STUDENTS table having the following structure and content:

STUDENT_ID	NAME	LASTNAME	ADDRESS	CITY	BIRTH_DATE	CLASS
23	Mario	Rossi	Via dei Pioppi	Manno	19.3.1980	I2A
34	Marco	Bianchi	Via dei Tigli	Bioggio	27.2.1982	I2A
45	Tom	Quadri	Via del Re	Agno	12.9.1980	I2B
32	Vasco	Vogel	Via Pioda	Comano	13.1.1979	I2A
48	Carlo	Lunghi	Via Balestra	Chiasso	27.5.1980	I2B
12	Alan	Nisi	Via Castello	Bellinzona	15.12.1982	I1A

12 B

12 A

And the following 2 transactions which operate in READ COMMITTED isolation level:

T1	T2	Res. T2
	Select count(*) from students where class ='I2B'	?
Update students set class='I2B' where lastname='Vogel';		-----
	Select count(*) from students where class ='I2B'	?
Commit;		----
	Select count(*) from students where class ='I2B'	?
	Set transaction isolation level serializable;	----
	Select count(*) from students where class ='I2A'	?
Update students set class='I2A' where class='I1A';		----
	Select count(*) from students where class ='I2A'	?
Commit;		----
	Select count(*) from students where class ='I2A'	?

45
48

45
48

vedo cambiam. dopo il commit!

45
48
32

23
34

23
34

23
34

sono 2 perché
-> i cambiam. in serializable li vede solo T1.
-> Se committo T2 e faccio ancora select, me vedo 3 perché ritorno a Read committed.

Write the results of the SQL commands included in T2 (where in the 3rd column there is a '?')

Es. Transaction 2

Transactions 2 (idem)

Given the 2 tables having the structure and content below:

ACCOUNTS table:

ACCOUNT_ID	NAME	LASTNAME	BIRTHDATE	BALANCE
23	Mario	Rox	19.3.1980	11'000
34	Bob	Bianchi	20.2.1982	3'400
45	Luigi	Quadri	12.9.1980	34'450
32	Vasco	Junghi	13.1.1979	20'310
48	Alex	Lunghi	27.5.1980	6'700
12	Jacopo	Nisi	10.2.1982	2'590

And the OPERATIONS table, in relationship with the ACCOUNTS table via the ACCOUNT_ID FK:

ACCOUNT_ID	OPERATION	DATE
23	+1000	10.2.2005
32	-550	12.4.2005
45	+200	23.10.2005
12	+5'000	17.11.2005
12	-500	19.11.2005
45	-1000	13.11.2005
23	+3'000	17.11.2005

1)
write the SQL commands for a T transaction that is intended to generate the average of the operations for each account **excluding** changes performed by other transactions that will complete before the T transaction

2)
We intend to modify the personal data of Mr. Jacopo Nisi in the ACCOUNTS table.
2.1) write the SQL command that we must use in order to lock the data preventing other users to modify them in the meantime.
2.2) what is going to happen to the transaction that would try to lock or modify the data that we have locked at 2.1?
2.3) while data are locked by our transaction, can other users perform SELECT commands that access the data in question?
2.4) How can we free (unlock) the data and make them available for updates to other users/transactions?

1)

set transaction isolation level repeatable read (or serializable)

```
select account_id, avg(operation) from accounts, operations where accounts.account_id=operations.account_id group by account_id;
```

2)

2.1 `select * from accounts where name='Jacopo' and lastname='Nisi' for update;`

2.2 the command would wait till the release of the lock

2.3 yes

2.4 commit or rollback

tipo <map> di java <- # INDEXES -> rende l'accesso più veloce ad una certa tabella e migliora le performance. E ci sono più tipi di indici.

Introduction

- The DBMS keeps data by on hard disks.
- Hard disks are "slow" devices
- The data access time is strongly influenced by the number of disk accesses the DBMS has to do in order to retrieve the required data.
- So, given a query:

The fewer disk accesses the DBMS does, the better the performance

- DBMSs have tools that allow intelligent access to data and minimize block reads from the disk.
  - Data buffering (keeps in memory the most recently/frequently used data)
  - Indexes (structures that allow access to data in "direct" mode)

Fundamental concept

- Indexes have existed long before computer science, they are designed to speed up access to the desired data.
- e.g:
  - Author's catalogue in the library,
  - Index chapters or topics in a book
- I find a book more quickly by searching "the address" in the library catalogue than by sequentially going through all the shelves.
- Search key - attribute or set of attributes to search records in an index
  - E.g. Book title
- An index is a set of records (called index entries) of the form

(search key, pointer)

- E.g. The book list in a library is an index!
- When you have to perform searches according to the key used in an index (e.g. Author name), the DBMS uses the index to get the pointer to the table rows and access them directly without having to go through the whole table sequentially.

Fundamental concepts

- Logically, an index can be seen as a set of pairs of the type (k_i ,p_i) where:
  - k_i is a key value of the field on which the index is constructed
  - p_i is a pointer to the record (possibly the only one) with key value k_i .
- The advantage of using an index is that the key is only part of the information contained in a table row. Therefore, **the index uses less space than the data**
- The different types of indexes differ essentially in the way they organize the set of pairs (k_i ,p_i)

Table Classes

Address
20
40
60
80

Class	Students	Room
11A	21	220
12A	15	230
11B	23	113
12B	17	117

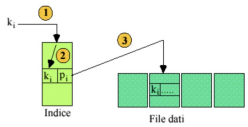
Simple Index

Key	Address
11A	20
12A	40
11B	60
12B	80

ki pi

Data access via index

- Consider an index on a primary key used to search for the tuple with k_i key .
- The steps to be taken are:
  1. Accedere all'indice
  2. Ricercare la coppia (k_i, p_i)
  3. Accedere alla pagina dati relativa



- Although it takes up less space than data, an index can reach considerable dimensions that lead to management problems similar to data.
- Indexes must therefore have structures that allow quick access to pairs (K_i, P_i) even in the case of large indexes.

Index types

- The indexes may have different structures and/or organisations. The most common types/structures are the following:
  - trees, binary trees (B-Tree)
    - tree structures with a predictable access time
  - hash indexes:
    - search keys are evenly distributed on nodes using a hash function.
  - bitmap indexes (bitmap indexes)
    - Used primarily in Datawarehouse contexts

Each DBMS can implement diffent types of indexes

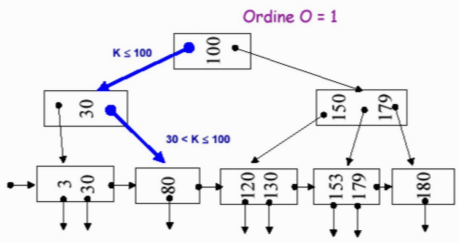
B+-Tree

-> come algoritmi!

- The B+-tree is the structure commonly used in DBMS to create dynamic multi-level indexes, i.e. that do not require periodic reorganization.
- The main features of a (primary) B+-tree are:
- The pairs (ki ,pi) are all contained in pages (or "nodes") leaf and the leaves are linked to list by pointers to facilitate the resolution of interval queries
- Each leaf contains a number of pairs that varies between O and 2 * O, where O is called the "order" of B+-tree
- Each path from the root node to a leaf has length h (height of the B+-tree), so the tree is perfectly balanced
- Each intermediate node (neither root nor leaf) has at least O + 1 pointers to as many nodes children; the root, if not a leaf, has at least 2 children
- A node with F + 1 child contains F key values, also called " separators ".
- Each node has at most 2 * O + 1 node children

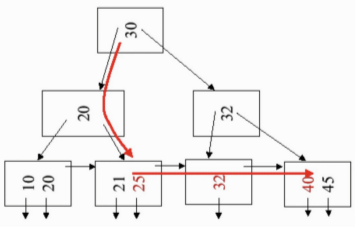
B+Tree example

- Example of B+tree height h = 3



B+-Tree: interval search

- To search for all key values in the [L, H] range, first look for the L value, then continue in sequence on the leaves
- Example: looking for keys in the range [23,42].



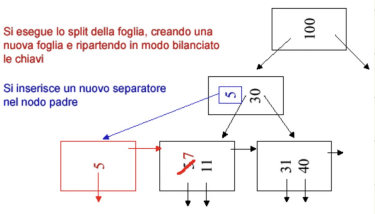
B+-Tree: insert

- The insert procedure proceeds first by searching for the leaf where the new key value is to be entered.

1. If there is room, the new pair (k_i ,p_i) is inserted in the leaf and the procedure ends
2. If there is no more room, a recursive "split" procedure is activated which, at the limit, propagates up to the root
3. The delete procedure follows a similar logic

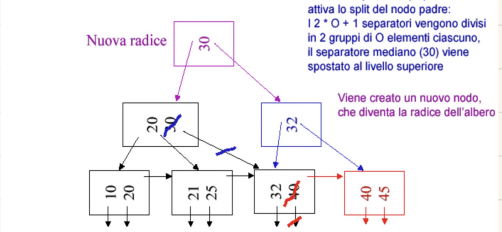
B+-Tree: insert (example)

- insert the key 7



B+-Tree: insert (example)

- Si inserisce la chiave 45



Bitmap indexes (Oracle)

- In a bitmap index, instead of storing a list of pointers for a certain key in the index, a bitmap is stored for each possible value of the key.
- In the bitmap there is a bit for each possible value in a row
- If the bit is set, it means that the line to which the pointer refers contains the key.
- Each bit of the bitmap therefore corresponds to a possible pointer.
- A mapping function converts the position of the bit to the actual pointer. So the bitmap index forms the same type of functionality as a B-tree although it uses a different internal representation.
- If the number of possible key values is small, the bitmap index is very space efficient.
- Bitmap indexes are efficient in case of WHERE with multiple conditions.

Bitmap index (example)

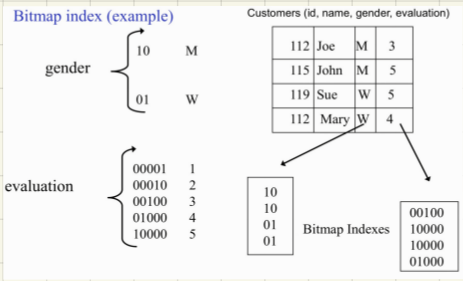
cust_id	marital status	Region	Gender	Monthly expense
101	Single	East	Man	2500
102	Married	Central	Woman	1200
103	Married	West	Woman	1345
104	Divorced	West	Man	2340
105	Single	Central	Woman	1560
106	Married	Central	Woman	3450

- Region, marital_status, gender are columns with low cardinality of values, so they are good to be indexed with bitmap indexes.
- Example of bitmap index on 'region' column

East	1	0	0	0	0	0
Central	0	1	0	0	1	1
West	0	0	1	1	0	0

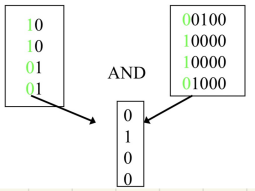
- The execution of logical operations (AND,OR, NOT) between columns indexed with bitmap indexes is very fast.

Bitmap index (example)



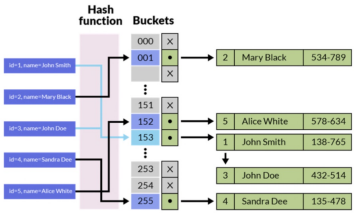
Bitmap indexes

- Advantages:
  - easily compressible
  - you can use bit-by-bit operations to execute queries
    - Example: how many men have a rating of 5?



HASH indexes

- A hash function distributes the pointer to the rows among buckets
- It works only for exact searches (=)



Defining indexes in SQL

- Create an index:

CREATE INDEX index_name ON table_name (attribute-list);

E.g. CREATE INDEX index_bn ON branch (branch_name)

- Use CREATE UNIQUE INDEX to specify and force the condition for which the search key can only have unique values
- Delete an index:

DROP INDEX index_name;

Create an index with Oracle DBMS

- B+-tree (default)

```
CREATE INDEX prod_category ON
Products(Category);
```

- Bitmap index

```
CREATE BITMAP INDEX prod_type ON
Products(Type);
```

Index characteristics

- Indexes are optional structures associated with tables or clusters
- they use storage space
- Indexes speed up data search operations
- They are used optionally to ensure uniqueness
- As a side effect, indexes can slow down insertion (insert), update and delete because extra structures need to be updated.
- The presence or absence of an index has no effect on the functionality of the SQL statement, but on the uniqueness controls, the execution mode and therefore on performance.
- Indexes are created manually through SQL DDL statements
- Indexes are automatically updated by the DBMS without explicit user intervention

Which columns to index

- Indexes should be created if it happens frequently to read less than 10 or 15% of a large table
- Small tables usually do not need indexes
- Columns that are candidates for indexing have the following characteristics:
  - each value is unique or
  - there is a large *range* (*range*) of values or
  - there are many null values (null value) and *queries* often return all lines with a value (e.g.: WHERE COL_X > 0)
- Columns less suitable for indexing have instead:
  - little distinction of values
  - many null values and searches for non-zero values will not be made

Indices on multiple attributes

- Suppose we have an index on combined keys (branch_name, balance)
- Example:

SELECT ... WHERE branch_name='Perryridge' AND balance=1000

- The index on the combined key will only return *records* that meet both conditions
- There are efficiency aspects to consider

Access with multiple keys

- Example:

SELECT account_number FROM account WHERE branch_name='Perryridge' AND balance=1000

- Possible DBMS strategies to fulfill the request using indexes on individual attributes:
  - use an index on 'branch_name' to find records with 'Perryridge'; test 'balance=1000'.
  - use an index on 'balance' to find records with '1000'; test 'branch_name= Perryridge'.
  - use an index on 'branch_name' to find the records with 'Perryridge', then use an index on 'balance' to find the records with '1000' and then take the intersection of the pointer sets as follows

Indexes on multiple attributes

- A multi-attribute index built on A1,A2,... An is effective if you specify values for all attributes or for the first i < n, i.e.

the order of definition is relevant!

- For example, the ExamIDX index:

CREATE INDEX EXAMS_INDEX ON EXAMINATIONS(Code, Date, Rating)

- is only useful for queries that specify values for
  - all three attributes
  - for Code and Date
  - only for Course Codes
- Therefore, if the WHERE clause only contains:
  Date = '28- 06- 2019' AND Rating = 6 (senza Code)
- the index would not be used by the DBMS!

Other uses of indexes

- In addition to solving queries, an index can be used to access records in a certain order, and so it can also be useful if there are ORDER BY and GROUP BY clauses in the query.

```
SELECT *FROM Students ORDER BY datanascita
```

- For some queries the presence of an index can even avoid having to access the table!

```
SELECT COUNT( DISTINCT datanascita)FROM Students
```

- if there is an index on (DeptNo, Salary), even the following query does not need to access the table:

```
SELECT DeptNo, MAX( Salary)
FROM Employee GROUP BY DeptNo
```

Example of Indexes in other DBMSs

- PostgreSQL:
  - B-tree
  - Hash
  - GiST (Generalized Search Tree)
  - SP-GiST (space-partitioned GiST)
  - GIN (**g**eneralized **in**verted index)
  - BRIN (**b**lock **r**ange **in**dexes)

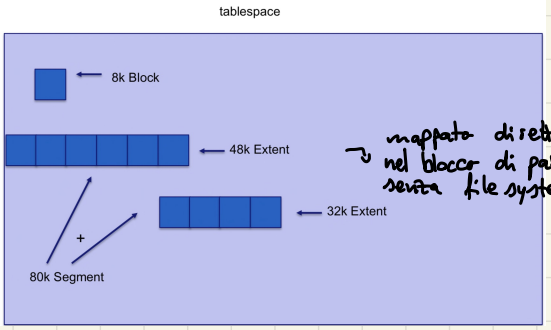
PHYSICAL DESIGN

Database storage concepts

- ✓ We have to keep in mind that every DBMS was designed with specifics internals
- ✓ What we see is the Oracle case

- ✓ The minimun storage unit that is managed by the DBMS is the **block** (8k)
- ✓ A set of contiguous blocks that can be allocated is called an **extent**
- ✓ The space allocated to an object is called a **segment**
- ✓ An object may be composed by multiple segments sparsely allocated
- ✓ Segments are allocated within **tablespaces**

Database storage concepts (oracle)



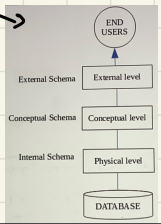
mappato direttam. nel blocco di partizione senza file system.

Tablespaces → logical level

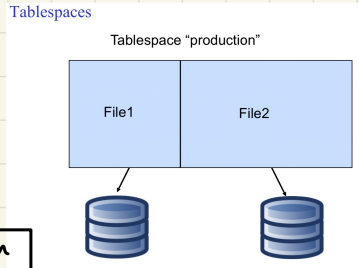
- ✓ Tablespaces are objects, consisting of 1 or more files, within which database objects that require storage space can be created. For example, tables and indexes.
- ✓ Example:

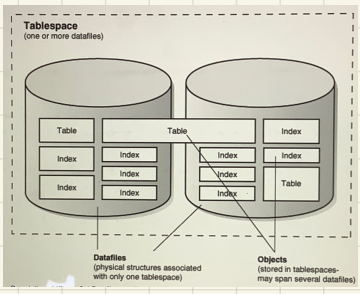
per organizz. gli oggetti

- ✓ The objects created in the "production" tablespace will actually be written in file1 or file2 or divided between both of them according to the size and the DBMS storage management policies.



operative system ≠ file system





Checking tablespaces and datafiles

- ✓ **How can I know which tablespaces do exist?**

```
select tablespace_name from user_tablespaces;
```

- ✓ **How can I know which are the datafiles?**

```
select  name, bytes, TS# from V$DATAFILE;
```

Checking storage usage

```
SELECT segment_name, segment_type, tablespace_name, bytes, blocks,
extents FROM user_segments;
```

	SEGMENT_NAME	SEGMENT_TYPE	TABLESPACE_NAME	BYTES	BLOCKS	EXTENTS
13	PRODOTTI	TABLE	USERS	65536	8	1
14	CONSEGNE	TABLE	USERS	65536	8	1
15	INCLUDONO	TABLE	USERS	65536	8	1
16	STUDENTI	TABLE	USERS	65536	8	1
17	CORSI_DI_LAUREA	TABLE	USERS	65536	8	1
18	CORSI	TABLE	USERS	65536	8	1
19	VALUTAZIONI	TABLE	USERS	65536	8	1
20	SYS_C0010992	INDEX	USERS	65536	8	1
21	SYS_C0011001	INDEX	USERS	65536	8	1
22	SYS_C0010994	INDEX	USERS	65536	8	1
23	SYS_C0010995	INDEX	USERS	65536	8	1

differenze nel dare al DB un raw device o un file system:

- organizzati in extends (per tabelle) DBMS
- buffering data (file system). lo organizza, è un software
  - ↳ prende un pezzo di raw e mette lì dei pezzi di file, come fa il DB con le tabelle.
- Sono 2 software che gestiscono i dati diversamente
- interrogazione → portati su raw file system → su raw DBMS
- meglio non avere un file system, meno memoria e più performante, ma un suo vantaggio: avendo i dati su 2 parti posso spostare i dati da un disco ad un'altro.
- Con raw device non ho un file system!

What is Phisical Database Design → creazione delle strutture del database in un DBMS su un certo sistema.

- ✓ Definition/structuring of available storage space for the database
  - ✓ Positioning of tables and indexes
  - ✓ Defining table sizes and storage parameters
- ✓ Considerations on the use of advanced features of some DBMS
  - ✓ Cluster
  - ✓ Index organized tables
  - ✓ Partitioning
- ✓ Physical design requires knowledge of
  - ✓ database structure
  - ✓ of the data and the use that will be made of it
  - ✓ the configuration of the system on which the deployment takes place
  - ✓ of the DBMS chosen and the possibilities it offers

Physical design – table sizing and positioning

- ✓ The objective of the physical design is to implement the schema resulting from the logical design on a certain DBMS that has certain physical resources.
- ✓ The positioning of the tables depends on various factors including:
  - ✓ system configuration
  - ✓ available resources
  - ✓ expected growth in database size
  - ✓ performance requirement

Complessità:
- n° oggetti
- n° operaz.

Physical design – table sizing and positioning

- ✓ At the time of creation, each table must be sized by estimating the initial size and expected growth.
- ✓ Usually the DBMS administration manual helps you understand how the space is actually used by the DBMS for each row and column type
- ✓ By combining this information with the information that emerged from the initial analysis of the requirements (volumes), it is possible to correctly size a table
- ✓ The same applies to other structures that require physical space (e.g. indexes).

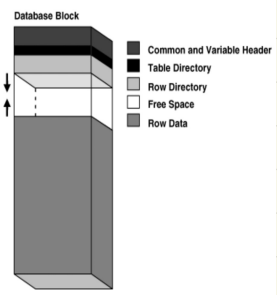
The STORAGE clause → il valore di default senza la clausola usa

- ✓ The DBMS manages the block-based physical space
- ✓ The physical characteristics of tablespaces, tables, indexes, undo segments can be defined at the time of creation and modified afterwards.
- ✓ Each type of structure requiring physical space can be dimensioned through the STORAGE clause of the respective CREATE command

The STORAGE clause

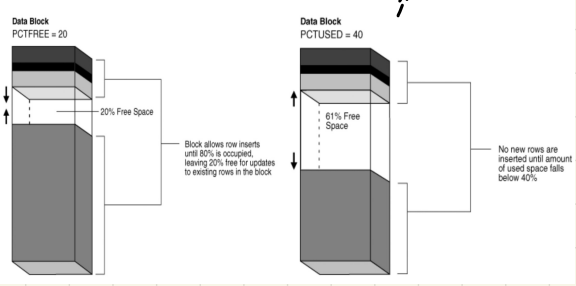
- ✓ The STORAGE clause allows you to specify various parameters, including:
  - ✓ INITIAL - space allocated at the time of creation
  - ✓ NEXT - additional space in case the allocated space is exhausted
  - ✓ MAXEXTENTS - maximum number of extensions (default: UNLIMITED)
  - ✓ PCTINCREASE - percentage increase in extension size
- ✓ Two other parameters are available to control the way space is used within a block:
  - ✓ PCTUSED - usage limit under which new lines are added to a block
  - ✓ PCTFREE - space in the block will remain free for updates of the lines contained in it

Block structure



Block usage

dati distribuiti più casualmente



The STORAGE clause

```
CREATE TABLESPACE bank …;

CREATE TABLE account (
  …
) STORAGE (
      INITIAL     100M
      NEXT        10M
      MAXEXTENTS  UNLIMITED
      PCTINCREASE 5
) TABLESPACE bank;
```

ESERCIZIO STORAGE CONCEPT

Create a table with the following structure:

person(id (pk), office, salary) //use integer type for all columns

adding the STORAGE clause defining INITIAL, NEXT and PCTINCREASE (do use small values for INITIAL and NEXT)

1) check the user segments in the data dictionary

2) create the sequence id_seq;

3) insert a row in the table using as PK the value returned by the sequence // remember to commit che change

4) check again the user segments in the data dictionary

5) insert 50'000 rows in the table

insert into PERSON select id_seq.nextval, level, 1 from dual connect by level<=50000;

commit;

6) check again the user_segments and note the differences

Foglio di lavoro | Query Builder

```
create table person (
    id integer primary key,
    office INTEGER,
    salary integer
) STORAGE (
    INITIAL 100
    NEXT 10
    PCTINCREASE 5
);

1)
SELECT segment_name, segment_type, tablespace_name, bytes, blocks,
4) extents FROM user_segments;
6)

2) create SEQUENCE id_seq
    start with 1
    increment by 1
    nocache
    nocycle;

3) insert into person(id, office, salary) values (1, 10, 99999);

5) insert into PERSON select id_seq.nextval, level, 1 from dual connect by level<=50000;

commit;
```

1	DIAGNOSI	TABLE	USERS	65536	8	1
2	MEDICO_ESTERNO	TABLE	USERS	65536	8	1
3	MEDICO_INTERNO	TABLE	USERS	65536	8	1
4	PAZIENTE	TABLE	USERS	65536	8	1
5	PERSON	TABLE	USERS	2097152	256	17
6	SPECIALIZZAZIONE	TABLE	USERS	65536	8	1
7	STUDENTS	TABLE	USERS	65536	8	1

Advanced Features: clusters (raggruppamenti)

- Some DBMS allow you to create clusters of tables: that is, to create tables that, although logically separate, are physically integrated. That is, they share the physical blocks according to certain criteria to be specified at the time of creation.
- They make sense if the tables have one or more columns in common
- They are useful for performance issues of large table joins.
- The rows of the various tables that have, for the defined columns, equal values, are stored in the same blocks in order to facilitate the joins.

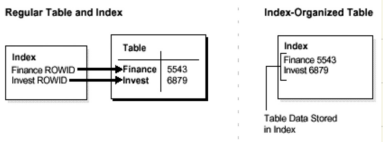
How to create a cluster (example)

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
    PCTUSED 80
    PCTFREE 5
    TABLESPACE users
    STORAGE (INITIAL 200K
             NEXT 300K
             MINEXTENTS 2
             MAXEXTENTS 20
             PCTINCREASE 33);
CREATE TABLE dept (
    deptno NUMBER(3) PRIMARY KEY, . . . )
    CLUSTER emp_dept (deptno);
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
    deptno NUMBER(3),
    foreign key (deptno) REFERENCES dept)
    CLUSTER emp_dept (deptno);
```

Advanced Features: Index Organised Tables

- You can make sure that a table is physically managed as if it were a B-Tree index, i.e. all the attributes of the table are in the index and the table, understood as a tuple sequence, does not actually exist.
- IOTs provide very quick access in case of queries using the primary key.
- Table updates are resolved in the index update only.
- Storage needs are lower than the classic implementation (table + index) because the key values are not repeated in the two structures.
- There are no effects on the DML part included in the programs accessing the table.
- Note: called clustered tables in Microsoft SQL server

Table vs IOT



How to create an IOT (example)

```
CREATE TABLE docindex(
    token char(20),
    doc_id NUMBER,
    token_frequency NUMBER,
    token_offsets VARCHAR2(512),
    CONSTRAINT pk_docindex PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX TABLESPACE ind_tbs;
```

Advanced Features: partitioning

- Partitioning:
  - tables and indexes can be partitioned (i.e. divided) to improve their management
  - each partition becomes an object partially independent from the others
  - query on the whole table
  - query only on one partition
- The partitions can be put in different tablespaces, possibly on different systems (distributed database)

Partitioning with Oracle

- Partitioning on tables and indexes
- We see only tables
  - You must choose:
    - partitioning key: field to partition against
    - partitioning method: approach used to define partitions

Partitioning with Oracle

- Range partitioning:
  - each partition is associated with a range
  - rows are inserted into a partition if the value of the partitioning key falls within that range

Partitioning with Oracle: Range partitioning

```
CREATE TABLE sales
(product_id int,
    income number,
…)
PARTITION BY RANGE (income)
(PARTITION p1 VALUES LESS THAN (200),
 PARTITION p2 VALUES LESS THAN (400),
 PARTITION p3 VALUES LESS THAN (600),
 PARTITION p4 VALUES LESS THAN (maxvalue));
```

Partitioning with Oracle

- Hash partitioning:
  - a hashing algorithm is applied to the partitioning key
  - rows with the same value returned by the hashing algorithm are inserted in the same partition

Partitioning with Oracle: Hash partitioning

```
CREATE TABLE sales
(product_id int,
    income number,
…)
PARTITION BY HASH (income)
PARTITIONS 4;
```

Partitioning with Oracle

- List partitioning:
  - each partition is defined by a list of values
  - rows are inserted into a partition if the value of the partition key matches one of the values associated with the partition
  - useful for enumeration type fields (only a few values allowed)

Partitioning with Oracle: List partitioning

```
CREATE TABLE Products
(product_id NUMBER(3),
category varchar(20),
…)
PARTITION BY LIST (category)
(PARTITION p1 VALUES ('shoes','clothing'),
 PARTITION p2 VALUES ('food'),
 PARTITION p3 VALUES (DEFAULT));
```

Checking partitions

- Where do I see partitions:

```
SELECT * FROM user_tab_partitions;
```

Partition pruning

- During optimization, the system determines which partitions are not significant for the execution of a given query and restricts the execution to the partitions of interest only

Why partitioning

- Performance
  - query on a subset of data
    - can help where indexes cannot
- Management
  - backup only active partitions
  - easy drop of partitions (without lenthy delete of rows)
- Availability
  - if a partition is unavailable the system works anyway (as long as it does not need the data in the unavailable partition)

Assignment 2: partitioning (20 min.)

0) Create the table PERSON structured as follows:

person(id (pk), office, salary) //use all integer types

Having the following 3 partitions:

- 1 with salary values less than 50000
- 1 with salary values less than 80000
- 1 with salary values less than 150000

1) check the user segments and partitions in the data dictionary

2) insert some rows in the table (a few rows are enough)

3) try to add a person with salary > 150000

4) check the user segments

5) write a query that selects rows based on a salary value and observe that the execution plan will show the partition pruning

0)

```
create table person (
    id integer primary key,
    office INTEGER,
    salary integer
) STORAGE (
    INITIAL 100
    NEXT 10
    PCTINCREASE 5
) PARTITION BY range(salary) (
    partition p1 values less than (50000),
    partition p2 values less than (80000),
    partition p3 values less than (150000)
);
```

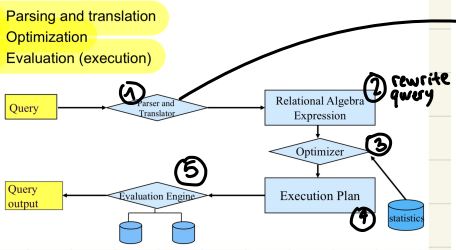
4)

4	PAZIENTE	TABLE	USERS	65536	8	1
5	PERSON	TABLE PARTITION	USERS	65536	8	1
6	PERSON	TABLE PARTITION	USERS	65536	8	1
7	PERSON	TABLE PARTITION	USERS	65536	8	1

QUERY PROCESSING

Basic Steps in Query Processing

- Parsing and translation
- Optimization
- Evaluation (execution)



Query Processing Steps (1)

1. Parsing and Translation
   - It translates the query into its internal form. This is then translated into relational algebra
   - The parser verifies that:
     - the syntax is correct
     - operators are applied to data of an appropriate type
     - the referenced objects (columns, tables, etc.) exist
     - the user has the necessary privileges to perform the operation
   - The parser finds the necessary information in the data dictionary
   - Before proceeding to the actual phase of query optimization, the DBMS performs a "rewriting" step of the query.

Query Rewrite (2)

- The purpose of the rewriting phase is to simplify the query and get to a form that is easier to analyze and thus optimize. Typical operations that take place in this phase include
  - **View resolution**: merge the input query with the queries that define the referenced views
  - **Unnesting**: if the query includes subqueries it tries to transform it into a form without them
  - **Use of Constraints**: the constraints defined on the schema are used to simplify the query
- The way these operations are performed varies from system to system, so it is not possible to provide general solutions (the way also changes for the same DBMS if different "optimization levels" are chosen!).

View resolution

- For a simple example, consider the view:

```
CREATE VIEW EmpSalaries( EmpNo, Last, First, Salary)
AS SELECT EmpNo, LastName, FirstName, Salary
FROM Employee WHERE Salary > 20000
```

vista autom. trasformata in ...

and the query:

```
SELECT Last, First
FROM EmpSalaries WHERE Last LIKE 'B%' EmpSalaries
```

- The resolution of the view leads to rewrite the query as:

```
SELECT LastName, FirstName
FROM Employee WHERE Salary > 20000
AND LastName LIKE 'B% '
```

Unnesting

- Switching to a form without subquery is sometimes immediately understandable; for example, the following query:

```
SELECT EmpNo, PhoneNo
FROM Employee WHERE WorkDept IN ( SELECT DeptNo FROM Department WHERE
    DeptName = 'Operations' )
```

is rewritten as:

```
SELECT EmpNo, PhoneNo
FROM Employee E, Department D WHERE E. WorkDept = D. DeptNo AND D.
    DeptName = 'Operations
```

- Note that you do not need the DISTINCT option because WorkDept is a foreign key

Unnesting

- Other times the transformation is less easy to understand. For example, the query:

```
SELECT E. EmpNo FROM Employee E WHERE NOT EXISTS ( SELECT *
        FROM Emp_ Act EA
        WHERE E. EmpNo = EA. EmpNo )
```

- is rewritten using an outer join:

```
SELECT Q. EmpNo FROM ( SELECT E. EmpNo, EA. EmpNo
        FROM Emp_ Act EA RIGHT OUTER JOIN Employee E ON (E.
    EmpNo = EA. EmpNo) ) AS Q (EmpNo, EmpNo1)
    WHERE Q. EmpNo1 IS NULL
```

Use of constraints

- The DBMS uses constraints to simplify queries.
- For example, if EmpNo is a key:

```
SELECT DISTINCT EmpNo
FROM Employee
```

- it's more simply rewritten as:

```
SELECT EmpNo
FROM Employee
```

- which has the advantage of not involving a (useless!) operation of removing duplicate tuples from the result

Use of constraints

- The Foreign Key constraint can also be efficiently exploited:

```
SELECT EA. EmpNo -- EA. EmpNo REFERENCES Employee( EmpNo)
FROM Emp_ Act EA WHERE EA. EmpNo IN (SELECT EmpNo FROM Employee)
```

is rewritten as:

```
SELECT EA. EmpNo FROM Emp_ Act EA
```

- If EA. EmpNo admits null values, you should add WHERE EA.EmpNo IS NOT NULL to respect the semantics of the query
- In this example the user himself could write the query in a simplified form; in the next example this is not possible, but only the DBMS can do it

Query Processing Steps (2) (3)

At the end of the rewrite, the actual optimization is performed.

- Optimization
  - Finds *the* cheapest *execution plan* for the query, i.e. the way that overall presents a minimum cost among all possible alternatives.
  - The cost estimate can be based on statistical information contained in the DBMS data dictionary (**cost-based optimisation**):
    - number of rows per table,
    - number of blocks in the table,
    - row size,
    - number of distinct values for the involved attributes,
    - existing indexes on the columns involved in the query,
    - etc.
  - Or based on predefined "**rule-based optimisation**" rules

Determining the execution plan (4)

- The optimizer, in order to be able to determine the access plan at minimum cost, has a strategy of enumeration (generation) of *execution plans*, whose main tasks are:
  - Enumerate all plans that potentially could be optimal
  - Do not generate access plans that are certainly not optimal
- The number of access plans that are generated to optimize a query can be very high
  - For example, if a query performs N relations join, there are at least N! *execution plans* potentially optimal that the optimizer should consider
- Several DBMS allow to explicitly control the enumeration strategy, so as to allow a finer "tuning" of the performance

Query Processing Steps (3) (5)

- Evaluation
  - The query execution engine takes the execution plan, executes it, and returns the query result.

Cost of a Query

- There are several possible ways to estimate the cost of a query, for example: cost of disk access, cpu time, communication time in a distributed system.
- Typically disk access is the dominant cost and is also relatively easy to estimate. So usually the number of *block transfers* from disk is used as a measure of execution cost. It's assumed that all block transfers have the same cost.
- The algorithms have to consider the size of the buffers, since finding a block in memory is much faster than retrieving it from disk.

Esempi:

Example: ORACLE EXECUTION PLAN

The EXPLAIN PLAN command is used to save the execution plan in a table.
This is useful to check how the DBMS performs the query.
The PLAN_TABLE table, in which the plan is saved, must be created in advance and with the required structure.

To save the execution plan in the PLAN_TABLE table for a certain *statement*, it must be executed as follows:

EXPLAIN PLAN FOR *statement*

You must now query the PLAN_TABLE table to see the execution plan.
In PLAN_TABLE there are many attributes, the most relevant to understand the system behavior are: :

- operation
- options
- object_name

The PLAN_TABLE table must be emptied before each run.

Example

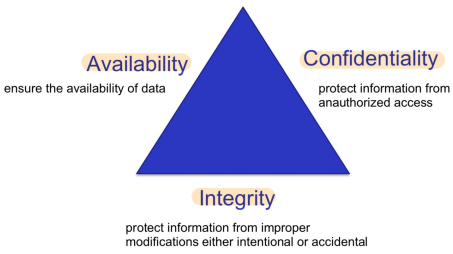
```
SQL > explain plan for select nome from clienti where telefono='2121212';
SQL > select operation, options, object_name from plan_table;

OPERATION          OPTIONS          OBJECT_NAME
TABLE ACCESS       FULL             CLIENTI
SELECT STATEMEMT

SQL> create index inum on clienti(telefono);
SQL> delete from PLAN_TABLE;
SQL > explain plan for select nome from clienti where telefono='2121212';
SQL > select operation, options, object_name from plan_table;

OPERATION          OPTIONS          OBJECT_NAME
TABLE ACCESS       BY INDEX ROWID   CLIENTI
INDEX              RANGE SCAN       INUM
SELECT STATEMENT
```

Example (cont.)

```
SQL > explain plan for select * from a where a2=1;
SQL > select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
-----------------

-------------------------------------------------------------------
| Id | Operation          | Name |      Rows | Bytes | Cost |
-------------------------------------------------------------------
|  0 | SELECT STATEMENT   |      |           |       |      |
|* 1 | TABLE ACCESS FULL  |    A |           |       |      |
-------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

PLAN_TABLE_OUTPUT
-----------------
    1 - filter("A"."A2"=1)

Note: rule based optimization
```

DATABASE SECURITY

The database security triad



DBMS security

- Assumes security at network, operating system, human and physical level
- Specific aspects of DBMS
  - Each user can be granted the permission to read a part of the data and write a part of the data
  - The authorization may apply to an entire table or to a part of it (rows or columns).

The Database Administrator (DBA) and security

the one that insall the system

- The DBA is the central authority in charge of managing the whole system and also the DBMS security
- The DBA responsibility include
  - classifiying users and data according to the company policy
  - creating users and granting/revoking privileges to users
  - monitoring the access and usage of the system / data (audit)
- Data security classifies into various stages i.e.
  1) Authentication,
  2) Authorization,
  3) Access control
  4) Auditing

Authentication 1

- Authentication means verifying the identity of users.
- The Initial step in accessing a database for a user is to authenticate himself.
- The next step after identifying the user is to check the user privileges i.e. authorization and access rights.

- Users can be created using the CREATE USER statement.
- CREATE USER is a system privilege, the database administrator has this privilege to create database users identified by a password.

```
CREATE USER <username>
IDENTIFIED BY <password>
DEFAULT TABLESPACE <permanent tablespace name>
QUOTA <size> ON <default tablespace name>
TEMPORARY TABLESPACE <tablespace name>
```

- A newly created user **cannot connect to the database** until he is granted the CREATE SESSION system privilege.

↳ l'amministratore del DB deve concedere il permesso all'utente di essere su quel DB, e gli assegna una password.

- Alternatively to the CREATE USER command, we can grant the privilege to connect to the DBMS

GRANT CONNECT TO user_name IDENTIFIED BY password

- The password is kept encrypted in the data dictionary.
- Behind the scenes a user is created like with the CREATE USER command.
- The CREATE SESSION privilege is automatically granted.
- The password can later be changed with the **ALTER USER** command
- The ALTER USER command can also be used to change other user characteristics (quota, role, ...).
- The user can be removed from the system (DBMS) using the **DROP USER** command

- With authorization commands we gives permissions to users to access the database, modify the data or display the information.
- It also controls the user to access another user's object schemas i.e. tables, rows or columns.
- There are two methods to grant privileges to a user: one is grant privileges to user explicitly and another method is grant privileges to a role
- And then grant a role to a user or to a group of database users.
- The privileges are classified into two types: System privileges (DDL operations) and Object Privileges (DML operations on objects).

- Authorizations can be defined at different levels
- Typically
  - Type of action
  - user
  - Type of resource (or specific resource)
- This can be done with the appropriate command to grant authorisations, the GRANT command
- Example:

GRANT select ON students TO roberto;

Privilegi possibili per un utente

The CONNECT privilege

- A user with CONNECT authorization can:
  - Connect to the DBMS
  - Query the tables for which he has been authorised
  - Make changes to the tables for which he has been authorized (insert, delete, update)
- But he can't change the database schema
  - create/drop/alter tables
  - create/drop/alter indexes
- Evidently a user with only CONNECT authorization can only work on data structures provided by other users.

The RESOURCE privilege

- A user with RESOURCE authorization can modify the scheme:
  - create/drop/alter tables
  - create/drop/alter indexes
  - create/drop/alter sequences
  - enable/disable auditing...
- Since RESOURCE allows the creation of structures, there must be a way to define where it can create them and whether it has a limit on the amount of space it can use (quota).

GRANT RESOURCE ON DEVELOPER (25000) TO ALICE

- developer is a tablespace in the example above.

The DBA privilege

- A user with DBA permissions can perform all administrative actions including startup and shutdown of the DBMS.
- There may be more than one in a system...
- For security issues, DBA authorisation should be given to a very limited number of users with appropriate expertise.
- A DBMS usually has its own default user that is created at the time of installation (ORACLE=system)

Grant authorizations on objects

→ accesso ad un oggetto o ad una sua parte

- The GRANT command is used to assign permissions:

GRANT <privilege list> ON <relation or view name> TO <user list>

- user list is
  - a user-id
  - public, which assigns permission to all valid users
- The user who assigns a certain authorization must possess it in order to do so.

Authorizations and SQL

- SELECT: allows read access to the table and the possibility to query it through a view.
- Example: Assign to the users U1, U2 and U3 the SELECT permission on the BRANCH table:

permette a U1, U2, U3 di eseguire una select.

GRANT select ON branch TO U1, U2, U3

- INSERT: possibility to insert lines
- UPDATE: possibility to edit lines
- DELETE: possibility to delete lines
- REFERENCES: possibility to define foreign keys in tables
- ALL PRIVILEGES: alias for all possible authorisations

Intra-object authorizations

- Permissions can be assigned not only at object level (table), but also at column level.
- So a column could be read-only for one user and read-write for another.

GRANT update (balance) ON account TO alice;

Assignment of authorizations

- The ability to assign permissions can be switched from one user to another.
- When a "normal" user receives a certain authorization (e.g. delete tables) he can assign it in turn to another user, if he has been authorized to do so.
- It is a sort of graph where nodes are users and the root of the graph is the default DBA user, i.e. the only user who has DBA permissions when the DBMS is installed.

Authorizations

- the clause WITH GRANT OPTION allows the user to whom a certain authorization is assigned to pass it on to another user.

a cui do i "miei" privilegi a U1

  - Example:

GRANT select ON branch TO U1 WITH GRANT OPTION

  - Gives U1 SELECT authorization on branch and allows U1 to assign it to others
- A user who is given access to a structure owned by another user can access it using the form:

USER_NAME.STRUCTURE_NAME

- E.G.

SELECT * FROM ALICE.BRANCH

Revocation of Authorizations

- Permissions can be revoked using the REVOKE command

REVOKE <privilege list> ON <relation or view name> FROM <user list>

- Example:

REVOKE select ON branch from U1 CASCADE

- Revocation of an authorization to a user may cause the authorization to be revoked from other users to whom the user in question had granted it; REVOKE is *cascading*
- <privilege list> can be ALL to revoke all permissions of a certain user
- If <user list> includes PUBLIC, all users lose permission except those who have been explicitly granted permission
- All authorisations that depend on the one revoked are also revoked.

Roles

- A ROLES is a group of privileges that are suitable for a user category
- Permissions can be granted or revoked from roles, in the same way as for individual users.
- Roles can be assigned to users and other roles
- E.g.
  - CREATE ROLE teller
  - CREATE ROLE manager
  - GRANT SELECT ON branch TO teller
  - GRANT teller TO alice
  - GRANT teller TO manager

Authorizations and views

- We have already seen that views are a way to restrict access to data, so they can be used to protect data from access by certain users.
- A combination of views with the security provided by the permissions mechanism can be used to allow each user to access only the data they need.
- The permissions on the views depend on the permissions on the tables behind the view.

Audit

- Some systems offer the possibility of auditing
- It's a way of monitoring activity to figure out who's doing what.
- Auditing can be enabled or disabled
- The actions normally monitored are:
  - Failed attempts to access the database
  - Attempts to access tables
  - User use of the GRANT and REVOKE command
  - Attempts to modify the scheme (create, drop, alter)
- Example:
- AUDIT CONNECT, RESOURCE WHENEVER NOT SUCCESSFUL

SQL injection attacks

- An SQL Injection attack is an exploit where a user is able to insert malicious code into an SQL query, resulting in an entirely new query
- It is common for user input to be read, and form part of an SQL query. For example, in PHP:

```
$query = "SELECT * FROM Products
WHERE name LIKE '%" . $searchterm. "%'";
```

- If a user is able to pass the application malicious information, this information may be combined with regular SQL queries
- The resulting query may have a very different effect

- An application or website is vulnerable to an injection attack if the programmer hasn't added code to check for special characters in the input:
  - ' represents the beginning or end of a string
  - ; represents the end of a command
  - /*...*/ represent comments
  - -- represents a comment for the rest of a line

- Imagine a user login webpage that requests a user ID and password. These are passed from a form to PHP via $_POST

  $_POST['id'] = 'Roberto'
  $_POST['pass'] = 'password'

- The ID is later used for a query:

  SELECT user_pass FROM users WHERE uID= 'Roberto';

- In PHP the code might look like this:

```
$query = "SELECT user_pass FROM users WHERE uID= '" .
$_POST['id'] . "'";
$result = mysql_query($query);
$row = mysql_fetch_row($result);
$pass = row['user_pass'];
```

- The password would then be compared with the other field the user entered

- If the user enters *Roberto*, the command becomes:

  SELECT user_pass FROM users WHERE uID= 'Roberto';

- But what about if the user entered this string as name?

  ';DROP TABLE Users;--

- With the malicious code inserted, the meaning of the SQL changes into two queries and a comment:

  SELECT user_pass FROM users WHERE uID= '';DROP TABLE Users; --'

- Sometimes the goal isn't sabotage, but getting information
- Consider an online banking system:

  SELECT No, SortCode FROM Accounts WHERE No = '11244102'

- This attack is aimed at listing all accounts at a bank. The SQL becomes a single, altered query:

  SELECT No, SortCode FROM Accounts WHERE No = '1' OR 'a' = 'a'

  This is particularly effective with weakly typed languages like PHP

Defending against SQL injection attacks

- Defending against SQL injection attacks is not difficult, but a lot of people still don't
- There are numerous ways you can improve security. You should be doing most of these at any time where a user inputs variables that will be used in an SQL statement
- In essence, don't trust that all users will do what you expect them to do
  - Restrict DBMS access priviliges to the minimum
  - Validate input
  - Encrypt sensitive data

Other tools for database security

- Data encryption
- Data redaction
- Data masking
- Database firewalls

Data Encryption

- Storing sensitive data inside your database can always lead to problems with security
- If in doubt, encrypt sensitive information so that if any breaches occur, damage is minimal
- Another reason to encrypt data is the majority of commercial security breaches are inside jobs by trusted employees
- **Never** store unencrypted passwords
- DBMS provide data encryption

- All companies have sensitive data that needs to be secured (salaries, credit card data, health data).
- Redact: to edit, or prepare for publishing.
- Redaction isn't simply removing the ability to see sensitive data, it is removing all traces of the content.
- Typically used for Employee Data, Customer Data, or Company Data

- Policies can be defined at DBMS level

- Data redaction applies also to reports contaning sensitive data
- Text based example
  - *Before: John plans to buy a new car next year.*
  - *After: [Redacted] plans to buy a new car next year.*
- How it works
  - page region
  - pattern matching
  - manual

- data masking, also known as data obfuscation, is a process enterprises use to hide data when exporting entire datasets.
- Generally, real data is obscured by random characters or other fake data.
- Typically used when trasfering data to an external system (development system, for example)
- data masking techniques
  - encryption
  - character scrambling
  - nulling out or deletion
  - shuffling