POLITECNICO
DI TORINO

Recap Database Management System (01NVVOV)

Jacopo Nasi
Computer Engineer
Politecnico di Torino

I Period - 2017/2018

20 dicembre 2017

Indice

License

Acknowledgments

Questo breve riepilogo non ha alcuno scopo se non quello di agevolare lo studio di me stesso, se vi fosse di aiuto siete liberi di usarlo.
Le fonti su cui mi sono basato sono quelle relative al corso offerto (**Database Management System (01NVVOV)**) dal Politecnico di Torino durante l'anno accademico 2017/2018.
Non mi assumo nessuna responsabilità in merito ad errori o qualsiasi altra cosa. Fatene buon uso!

1 Database Management System

1.1 Introduction

The DataBase Management System **DBMS** is a software package designed to store and manage databases. The architecture of the system is similar to the one in the figure 1. Since the DB data part can be really big it can't fit

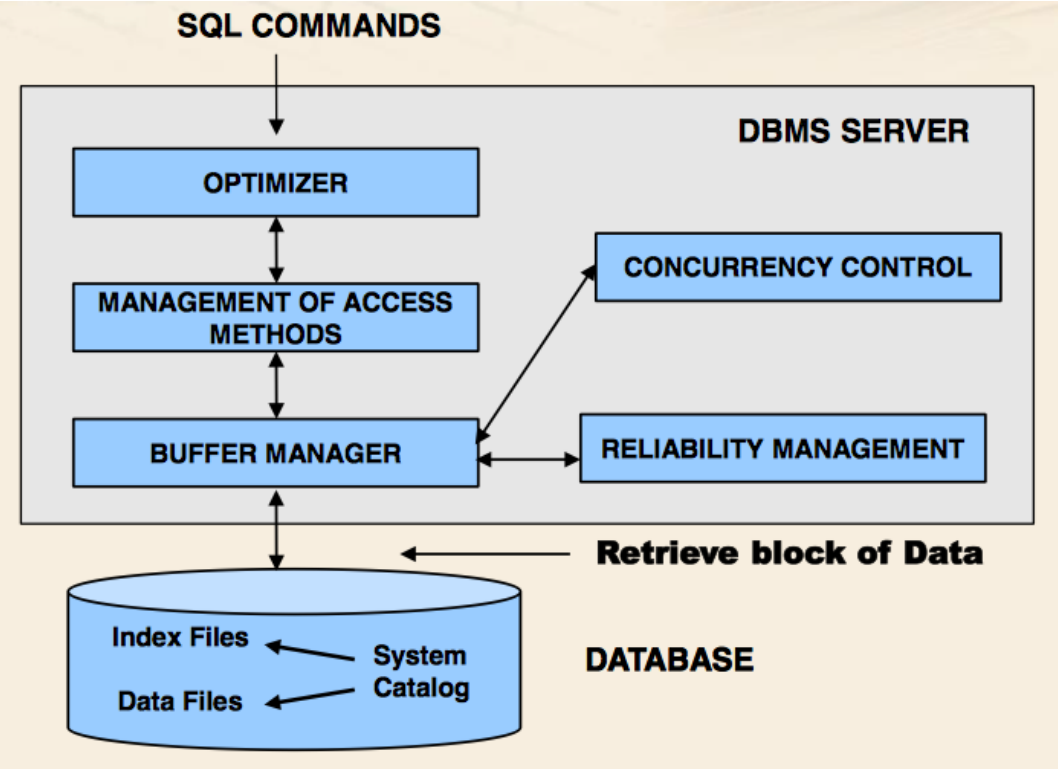


Figura 1: DBMS Architecture

always in the main memory (RAM) and, for this fact, is often stored in the secondary memory, like HDD. For this reason is necessary a system that define the operations to grab and manage the data from the secondary memory.
All the blocks has different behaviours. The **Optimizer** have multiple roles:

- Define an appropriate execution strategy for accessing data to answer queries.
- Receives in input the SQL instructions (DML).
- Check the lexical, syntactical and sematical correctness (not all the errors).
- Translate the query in an internal algebra rappresentation.
- Select the "right" strategy for accesing data.

- Guarantees the **data independence** property in the relation model.

The **Access Method Manager** is used for physical access to data and it implements the strategy selected by the optimizer. The **Buffer Manager** instead manage the page transfert from disk to main memory and vice versa and the main memory portion that is pre-allocated to the DBMS that is shared among many applications. The **Concurrency Control** coordinate the concurrent access to data (important for write operations) to guarantess the consistency of it. The **Realiability Manager** guarantees correctness of the database content duing the system crashes, the atomic execution of a transaction and it exploits auxiliary structures (log files) the correct the database in case of failure.
The **transaction** is an unit of work performed by an application, it's a sequence of one or more SQL RW operation charaterized by *correctness, reliability* and *isolation*. The START of a transaction is typically implicit and coincides with the first SQL instruction. The END instead can be of two differents types, it can be a COMMIT that it means the correct end of a transaction, or with ROLLBACK that it means error during the execution. In this second case the DBMS needs to go back to the state at the beginning of the transaction. The rollback can be of two type suicide, when is required by the transaction, and murder when is required by the system. The transaction have four important properties:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Atomicity means that they cannot be divided in smaller units, is not possibile to leave the system in a intermediate state of exec, guarantee by UNDO (undoes all the work perfomed, used for rollback) and REDO (redoes all work performed, used for commit the result in presence of failure). The consistency means that the transaction execution should not violate integrity constraints on a database, in case of it the system will perform solution to correct the violation. The system can be considered Isolated when the execution of a transaction is indipendent of the concurrent execution of other transaction, everything is enforced by the Concurrency Control block. The last properties means that, in presence of failures, the effect of a commited transaction IS NOT LOST, it guarantees the reliability of the DBMS and is enforced by the Reliability Manager block.

1.2 Buffer Manager

This block have a real important behaviour, it manages page transfer from disk to main memory and it's in charge of managing the DBMS buffer. The

operation of the pages transfert is the bottleneck of every system and this is why this block is really important. increasing the performance of this operation could really improve the speed of the entire system.
The buffer is:

- A large main memory block.
- Pre-allocated to the DBMS.
- Shared among executing transactions.

this part is organized in pages where the size depends on the size of the OS I/O block. There are two empirical law often used for hte management strategies:

1. Data Locality: Data referenced recntly is likely to be referenced again.
2. 20-80: The 20% of data is RW by 80% of transaction.

The buffer manager keeps additional snapshot information on the current content of the buffer, it shot, for every page, the physical location of the page on the secondary memory (file identifier and block number) and two state variables, one that count of the number of trasn using the page in that time (count), and the dirty bit that is set if the page has been modified.
It provides different access methods to load pages from disk and vice versa:

Fix Primitive used by transactions to require access to a disk page, after the page is loaded into the buffer a pointer is returned to the requesting transaction and the Count is incremented by 1. This procedure requires an I/O operation only id the page is not already in the buffer. There are two behaviour:

- Page already in buffer: Return the pointer to the data.
- Page not in buffer: It search a place for the page.
    1. Free pages
    2. Not free pages, Count=0; if the data is dirty it performs a synchronous write on the disk.

Unfix Primitive it tells the buffer manager that the transaction is no longer using the page and it decrease the Count.

Set Dirty Primitive is tells the buffer manager that the page has been modified by the running transaction and it sets the dirty bit to 1.

Force Primitive it requires a synchronous transfert of the page to the disk, when this operation is performed the transaction is suspended.

Flush Primitive is an autonomous transfert of the pages on the disks, is internal to the buffer manager and is runned when the CPU is not too much loaded. It transfer the page that are not valid (count=0) or not accessed since long time.
The are four writing strategies:

- **Steal**: The BM is allowed to select a locked page with Count=0 as victim. It writes on disk the dirty pages belonging to uncommitted trans. It can be undone.
- **No Steal**: The BM is not allow to steal.
- **Force**: All the pages are synchronous written on the disk during the commit operation.
- **No Force**: The pages are written asynchronously with the Flush Primitive.

The mostly used solution is **steal/no force** because of its efficency. The no force provides better I/O performance, steal may be mandatory for queries accessing a very large number of pages.

File System the BM is using services provided by the file system:

- Create/Delete of a file.
- Open/Close file.
- Read: It provides a direct access to a block in a file and it requires File Identifier, Block number and buffer page where to save data.
- Sequential Read: It provides seq. access to a fixed number of blocks in a file, it requires file identifier, strating block, number of blocks to be readed and the starting page for saving.
- Write and Sequential Write.
- Directory management.

1.3 Physical Access

Data may be stored in different format to provide efficient query execution. The **Access Method Manager** transform the decision taken by the optimizer into sequence of physical access to data. An access method is a software module specialized for single data structure that provide primitives for read and write. The AM can select the appropriate blocks of a file to be loaded in memory and it knows the organization of data into a page.
There are several solution for manage the data in relational system:

- Physical data storage
  - Sequential Structure
  - Hash Structure
- Indexing
  - Tree Structure
  - Unclustered Hash Index
  - Bitmap Index

In the sequential solution the tuples are stored in a given sequential order, in the case of the heap file are sorted in the insertion order, typically append at the end of the file.

- **PRO**: No wasted space, sequential read/write fast.
- **CONS**: Delete may cause wasted space.

this structure are frequently used jointly with unclustered indices to support search and sort operations.
In the ordered structures everything is sorted by the value of a given key, called sort key, it can contain one or more attributes.

- **PRO**: Sort, group by, search or join operations on the sort key really fast.
- **CONS**: Inserting new value preserving order.

the main problem of this solution is to keep the order of the data during new data insertion. There are two main solution, the first il leaving a percentage of free space in each block during the table creation; the second one create an overflow file containing tuples which do not fit into the correct block.
The ordered structure are typically used with B^+-Tree clustered (primary) indices where the index key is the sort key. Are used bt the DBMS too to storing intermediate operation results. This structure provide "direct" access to data based on a key (one or more attributes). This Tree have one root node with many intermediate nodes and each node has many children. The leaf nodes provide access to data in 2 different ways:

- **Clustered**: It store the data in the main memory. Used for primary key indexing. [figure 2]
- **Unclustered**: It store a pointer to the secondary memory of the data. Used for secondary indices. [figure 3]

There are two kind of B-Tree:

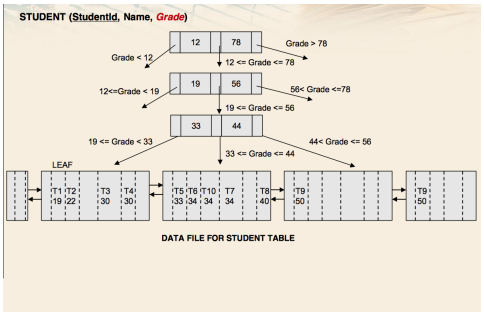


Figura 2: Clustered

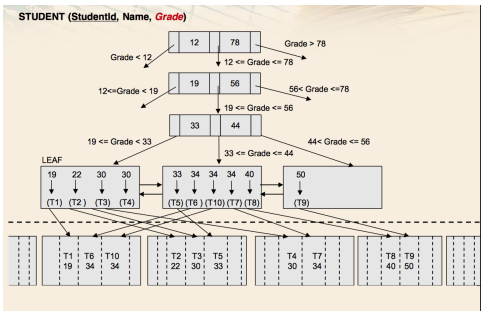


Figura 3: Unclustered

- **B-Tree**: Data pages are reached only through key values by visiting the tree. [figure 4]
- B^+**-Tree**: Provides link leaf allowing sequential access in the sort order. [figure 5]

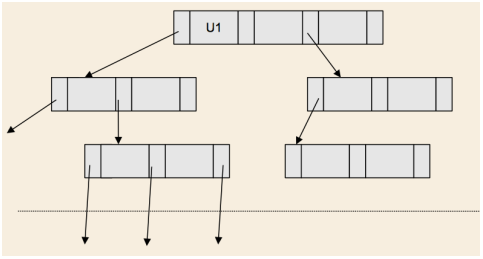


Figura 4: B-Tree

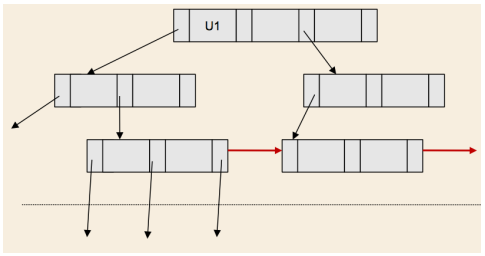


Figura 5: B^+-Tree

the B stands for **Balanced** where leaves are all at the same distance from the root and the search time is the same indipendently by the value.
This structure have some:

- **Advantages**:
  - Very efficent for range queries.
  - Appropriate for sequential scan in the order of the key field (always for clustered, not guarantee otherwise).
- **Disadvantage**:
  - Insertion may require a leaf or nodes split.
  - Deletions may require merging uncrowded nodes and re-balancing.

The **Hash** structure is another kind of weel-know structure is guarantees direct and efficent access to data based on the value of a key field (one or more attributes). Supposing to have B blocks in the hash structure the hash function is applied to the key value of a record and in return a values between

0 and b-1 which defines the position of the record, the idea is to not completly fill the blocks to allows new data insertion.

- **Advantages**:
  - Very efficent for queries with equality predicate on the key.
  - No sorting of disk blocks is required.
- **Disadvantage**:
  - Inefficent for range queries.
  - Collision may occur.

The unclustered versione is similar to the hash index, the main difference is that the actual data is stored in a separate structure and the position of tuples is not constrained to a block.

The **bitmap index** is another structure that provides direct and efficient access to data based on the value of a key field, it's based on a bit matrix. The bit matrix references data rows by means of RIDs (Rows IDentifiers), the actual data is stored in a separate structure and the tuples position is not constrained.
The bit matrix has:

- One column for each different value of the indexed attribute
- One row for each tuple.

the *(i, j)* position has a 1 if the tuple i as j like attributes for the key field, 0 otherwise. the main characteristics are:

- **Advantages**:
  - Very efficent for boolean expressions of predicates.
  - Appropriate for attributes with limited domain cardinality.
- **Disadvantage**:
  - Not used for continuous attributes.
  - Required space grows significantly with domain cardinality.

1.4 Query Optimization

The query optimizer is part of the Optimizer and its job is selecting an efficient strategy for query execution, this block is really important. Another important task is to guarantees the data independence property, in fact, the form in which the SQL query is written does not affect the way in which it is

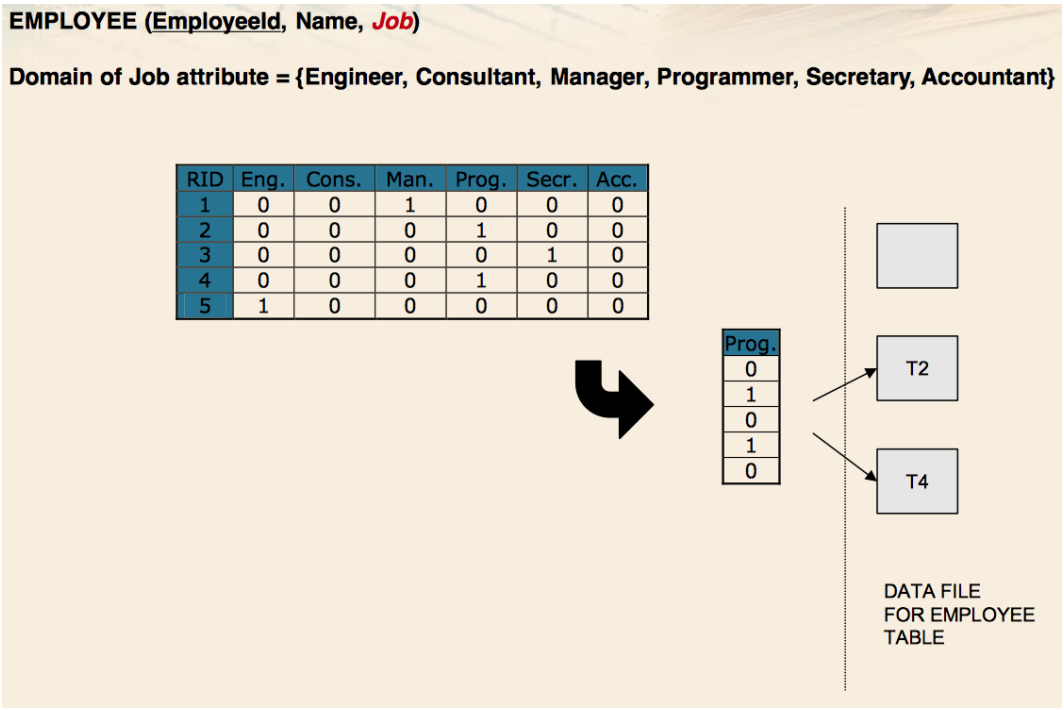


Figura 6: Bitmap Index

implemented and a physical reorganization of data does not require rewriting SQL queries.
The query optimizer generates a **Query Execution Plan** to use the best strategies to run the query, it evaluates many different alternative, it use data statistics, use best-know strategies and it adapts automatically on data changes. The plan has more phases as you can see in figure 7. The behaviour of each phase is:

Lexical, Synctactic and Semantic analysis : check the SQL for Lexical errors (e.g., misspelled keywords), Syntactical errors in the SQL grammar and for Semantic erros not existing object called in the query (require data dictionary). The output of this block is an internal rappresentation of extended relational algebra because it can represent the order in which operators are applied (procedural) and there are a lot of theorems and properties.

Algebraic Optimization : executing algebraic transformations is considered to be always beneficial, it should eliminate the difference among different formulations of the same query and is usually independet of the data distribution. The output of this phase is a "canonical" tree.

Cost Based Optimization : This phase select the best execution plan evaluating the execution cost, it use a selection of:

- Best access method for each table.
- Best algorithm for each relational operator among available alternatives.

the last step of this phase is the generation of the code implementing the best strategies, the output is the executable and all the dependencies used.
There are two types of execution modes:

- **Compile and Go**: Compilation and immediate execution, no storage of query plan and no need of dependencies.
- **Compile and Store**: The plan is stored in the DB together with its dependencies, it's executed on demand and it need to be recompiled in data structure changes.

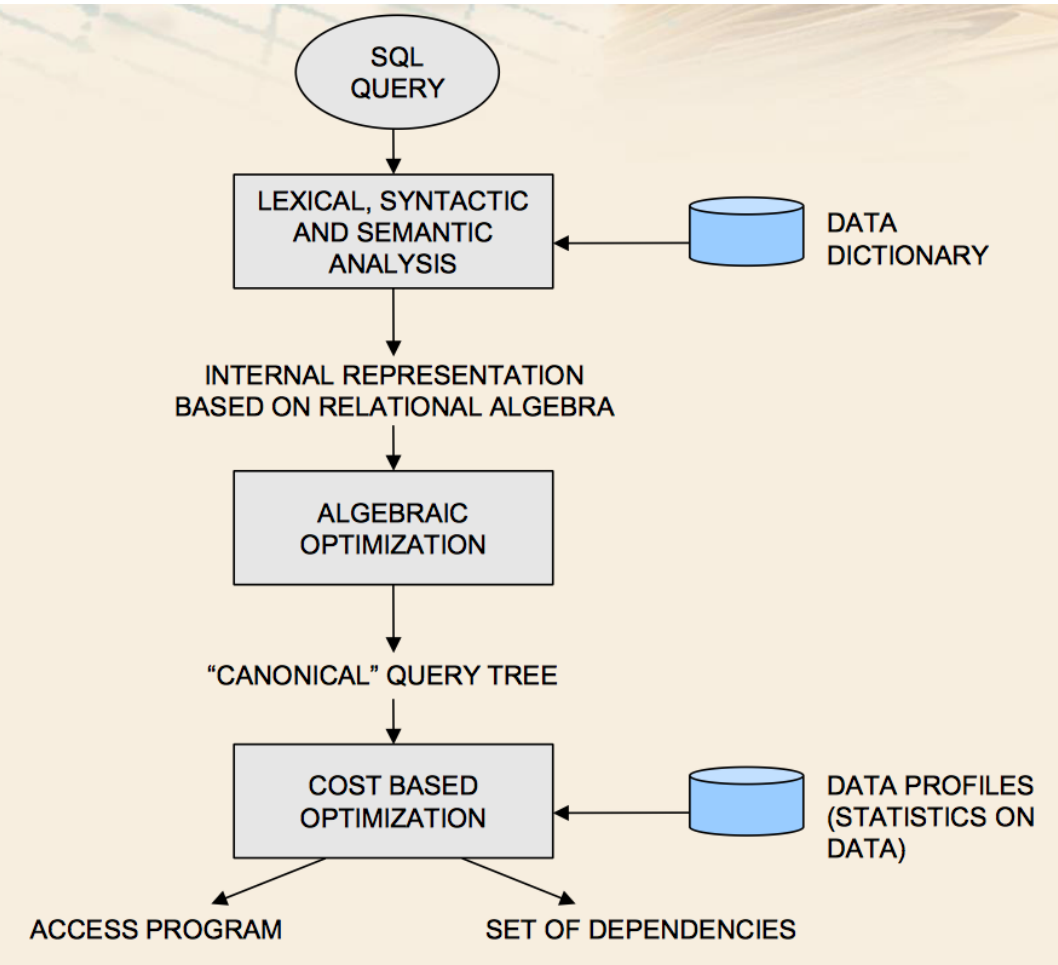


Figura 7: Optimization Execution Plan

The phase of **Algebraic Optimization** require a little more of analysis. The part is bbased on equivanlence trasformations, *two relational expressions are equivalent if they both produce the same query result for any arbitrary database instance.* The main objective of this part is to **reduce the size of the intermediate result**.
There are some well-know transformation:

1. **Atomization of selection**: Applying all attributes of selection one at a time or all together can provide different performance in case of indicies.

2. **Cascading Projections**: It possibile to perform directly the final projection o doing more projections with different sub-set and you can obtain the same result.

3. **Selection before join**: Anticipating the selection respect a join-operation can reduce the cardinality of the system reducing the number of operations for the join (is always used by the DBMS).

4. **Join derivation from Cartesian Product**: Perform a cartesian product and then a selection over data get the same result as the join operation but more slowly.

5. **Distributing selection respect union**: Is equivalent to select a subset and then merge it with another subset or merge the two set and then selecting it.

6. **Distributing selection respect difference**: The formulas explain better: $\sigma_F(E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2) = \sigma_F(E_1) - E_2$.

7. **Distributing projection respect union**: Projection of union of 2 table is equivalent to the union of 2 already projected table.

8. **Other**: $\sigma_{F1 \vee F2}(E) = (\sigma_{F1}(E)) \cup (\sigma_{F2}(E))$

9. **Other**: $\sigma_{F1 \wedge F2}(E) = (\sigma_{F1}(E)) \cap (\sigma_{F2}(E))$

10. **Distributing join respect union**: Perform join of a table E with two merged table (E1 $\cup$ E2), is equivalent to join E with E1 and E2 separate and the merge it.

The phase of **cost based optimization** is a little bit more complicated, it is based on:

- **Data Profiles**: Statistical information describing data distribution for tables and intermediate relational expression.

- **Approximate cost**: Evaluating cost by looking at CPU, HDD and main memory usage and time.

The profilings of table save quantitavive information on the characteristics of tables and columns:

- Cardinality of tuples.
- Size in bytes of tuples.
- Size in bytes of each attributes.
- Number of dinstinct values of each attribute.
- Min and max value of each attribute.

all this information are stored in the data dictionary that is periodically refreshed.

The access operators can perform different types of scans. The **sequential scan** execute sequential access to all tuples in a table (a.k.a Full Table Scan). The operation performed during a sequential scan:

- Projection.
- Selection (Simple predicate).
- Sorting based on attribute list (memory sort or sort on disk).
- Insert, Update or Delete.

the predicate evaluation is fundamental to provide an efficent access to the data. The index access it may be exploit with all kind of structures, in case of simple equality predicate all structure are appropriate. Instead, for range predicate, the only appropriate one is the B^+-Tree. For predicates with limited selectivity full scan is better (if available bitmap could be used). In case of conjunction fo predicates the most selective one is evaluated first through the index, then the other. A possible optimization could be computing the intersection of bitmaps coming from available indices and then a table read for remaing predicates. In the disjunction the index access can be used only if all predicates have and usable index, otherwise FTS.

The **join** operation can be a critical operation for a relational DBMS, the connection between tables is based on values instead of pointer. There are several algorithms that can be used for the join:

- **Nested Loop**: For each tuples of the outer table, the inner one is readed once. (BRUTE FROCE)
  - Efficent when the inner tavle is small and fits in memory or when the join attribute in the inner table is indexed.

  - Not cost symmertric. It depends on which table takes the role of inner.

- **Merge Scan**: It sort the two tables on the join attribute and it start a parallel scan.
  - Symmetric in terms of cost. Efficent for large and already-sorted tables.
  - Requires sorting both table (already sorted or trough clustered index).

- **Hash Join**: It applicate the same hash function to the join attrbute of both table. Tuples to be joined will fill the same bucket.
  - Very fast join.
  - Local sort and join is performed into each bucket.

- **Bitmapped Join Index**: It precompute the join. The position (i,j) of the matrix is 1 if the tuple with RID j of A joins with tuple with RID i in table B and 0 otherwise.
  - A data change need a recompute of table.
  - Used in OLAP queries.
  - It can exploit one or more bitmapped join indices (one for each pair of joined tables) and accessing the large central table is the last step.

The **Group By** is one of the most important function of SQL and is performed in 2 different ways: The first one is the sort based, it sort the data on the group by attributes and the compute aggregate functions on groups; the hash based one instead it perform and hash function over data, sort the bucket just created and than compute the aggregate function.

Execution plan selection is based on some data input, the data profiles (statistics over data) and the internal representation of the query tree, the output of this part is the "optimal" (it can't assure that it will be the best one) execution plan. This phase evaluate the cost of different alternatives for read tables and executing each relational operator exploting an approximate cost of execution.
The search is based on the following parameters:

- Scan type of data (full scan, index).
- Execution order among operators.
- Type of operators implementation (different join methods).

- Sorting time (when).

The approach work on a tree of alternatives where each nodes represent a decision on a variable and the leaf one complete query execution plan. Of course the system select the cheapest one. The general formula is $C_{Total} = C_{I/O} * n_{I/O} + C_{CPU} * n_{CPU}$ where $n_{I/O}$ is the number of I/O operations and n_{CPU} is the number of CPU operations.
The final plan is an approximation of the best solution. Th optimizer looks for a solution which is of the same order of magnitude of the "best" solution. In the **Compile and Go** execution mode the search is stopped when the time spend for the search is comparable to the time required to execute the current best plan.

1.5 Physical Design

The physical distribution of the data in the system is fundamental for providing good performance. Taking in account the logical schema of the DB, the features of the selected DBMS and thw workload this block provides a physical schema of the databse (table organization, indices) and all necessary set up parameters for storage and configurations.
The possible physical file organization are:

- Unordered (heap).
- Ordered (clustered).
- Hashing on hash-key.
- Clustering several relations.

The number of indicies is related to the structure type of the system. In case of clustered is possible to define only one index, instead, unclustered structures allow to define multiple different indices.
The workload distribution is different in case of a normal query or for an update. The first case involve:

- Accessed tables.
- Visualized Attributes.
- Attributes involve in selections and joins.
- Selectivity of selections.

the update case instead:

- Attributes and tables involved in selections.
- Selectivity of selections.

- Update type (Ins/Del/Up) and updated attributes.

The selection of the structure is important and it could be changed during the usage of the system for improvement (database tuning). Changes in the logical schema are allowed and they can or cannot preserve the BCNF (Boyce Code Normal Form). There isn't a general methodology the best solutions are trial and error, some general criteria and "common sense" heuristic.
Some general criteria could be:

- Primary Key is usually exploited for selection and joins, indexing it could be useful.
- Adding new indices for most common query predicates. Evaluate the actual plan and verify the improvement if available.
- Never index small table, the entire table requires few disk reads.
- Never index attributes with low cardinality (e.g. gender). This is not true for data warehouses.

from the heuristics point of view there are severals "common sense" ideas:

- For attributes involved in simple predicates of where clause equality:hash and range:B^+-Tree.
- Evaluated clustered improvement for slow queries.
- For where clauses involving many simple predicates use multi attributes index or appropriate key order.
- Maintenance cost.
- To improve joins use index on inner table in case of nested loop or B^+-Tree, for merge scan, on the join attribute.
- For group by hash index or B^+-Tree.
- Consider group by push down that anticipate the group respect to joins.

of course after all the changes a good choice could be update database statistics, for future improvements the database tuning could help. The last chance can be affecting optimizer decision, the main problem is the lost of data independence.

1.6 Concurrency Control

The workload of operational DBMS is measured in *transaction per second* (banking and flight reservation are on 10-1000 tps). This block provide concurrent access to data maximizing the throughput and minimizing response time. The elementary operations are of course **READ r(x)** and **WRITE w(x)**. The block that manage the concurrency is called scheduler is in charge of deciding if and when read/write request can be satisfied.
The most common anomalies are:

- **Lost Update**: It occur when a tr2 read a value that is already under operations by another tr. (figure 8)
- **Dirty Read**: When a tr2 read the value of x in an intermediate state which never become permanent. (figure 9)
- **Inconsistent Read**: When a tr1 read multiples times x with different value each time. (figure 10)
- **Ghost Update**: It occur when two transaction are working over multiple data at the same time performing read and write. (figure)

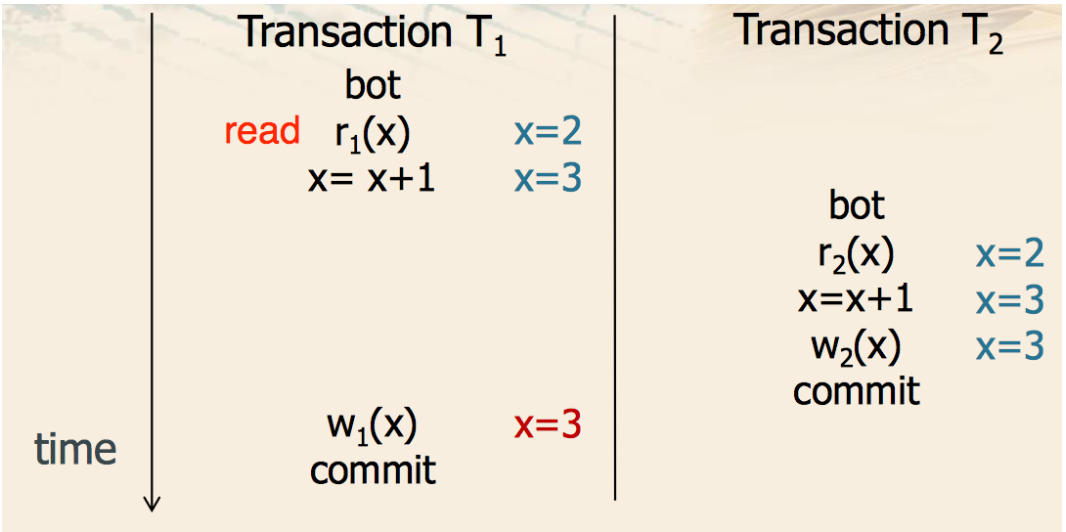


Figura 8: Lost Update Behaviour

Theory of Control a *transaction* is a sequence of R/W operations with the same TID (*Transaction Identifier*); the *schedule* is a sequence of read/write operations presented by concurrent transaction. The scheduler is in charge of accepts or reject the requests to avoid anomalies without knowing the outcome (commit/abort) of it.

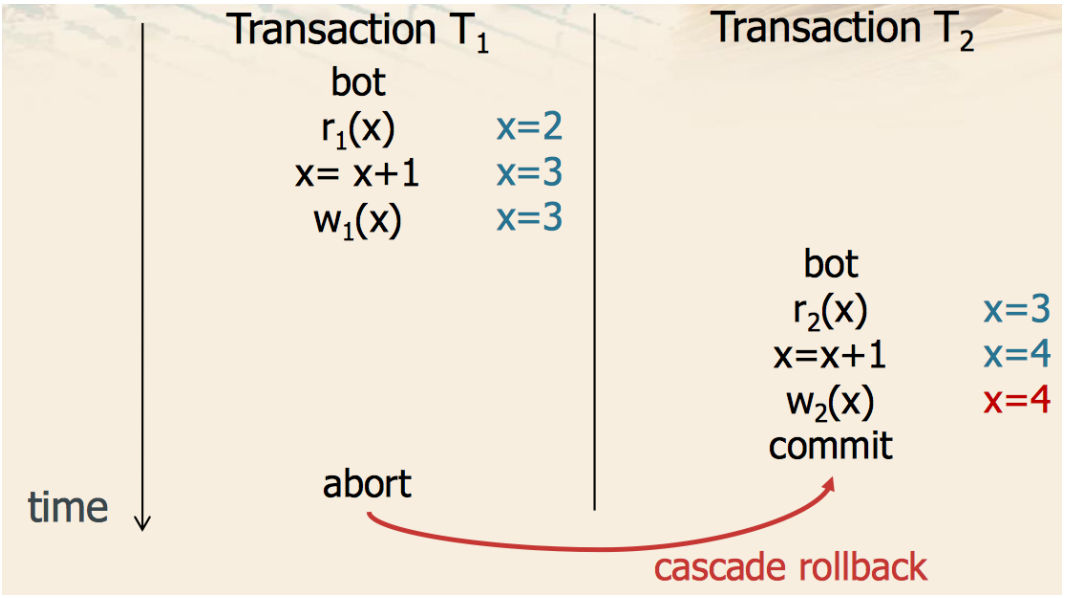


Figura 9: Dirty Read Behaviour

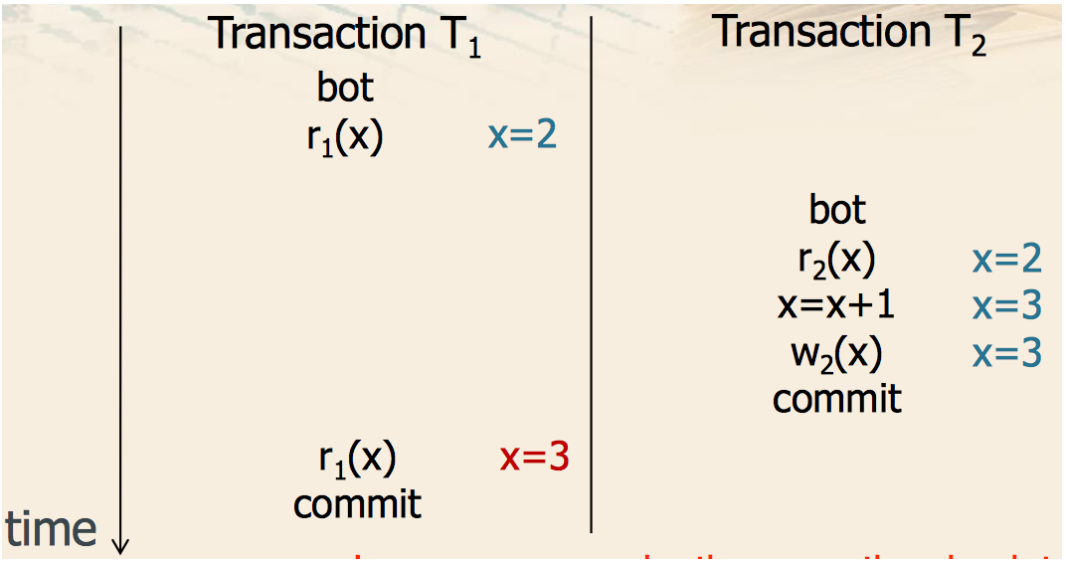


Figura 10: Inconsistent Read Behaviour

Commit projection is a simplifying hypothesis (the schedule only contains transaction performing commit), it avoid dirty read anomaly, it will be removed later.
In a **serial schedule**, the actions of each transaction appear in sequence, without interleaved actions. An arbitrary schedule S_i is correct when it yields the same results as an arbitrary serial schedule S_j of the same transactions. S_i is serializable, is equivalent to an arbitrary serial schedule of the same transaction. There are different equivalence classes between two schedules:

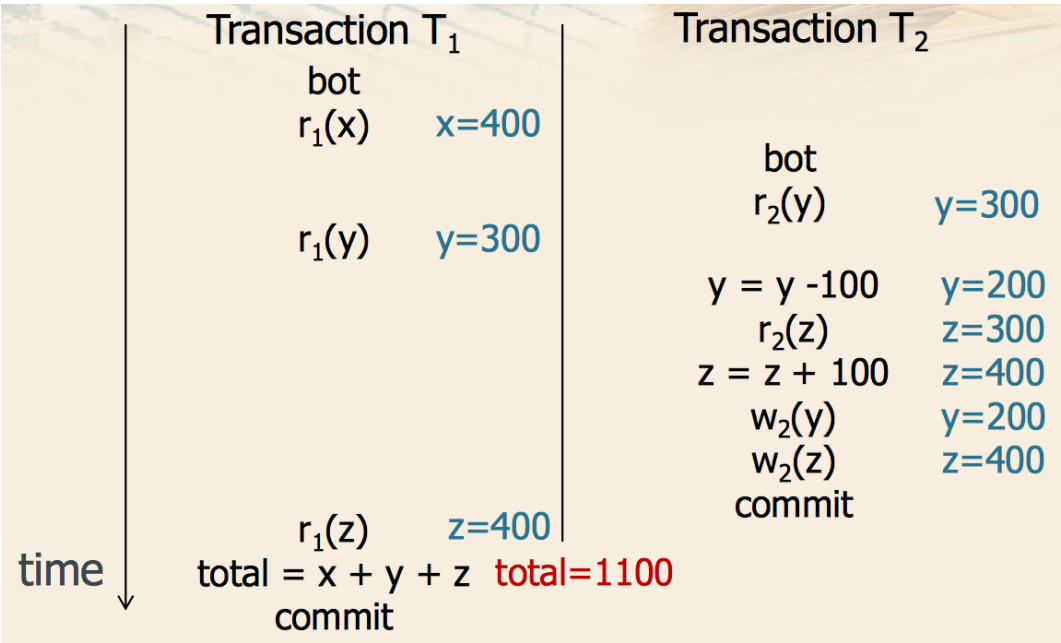


Figura 11: Ghost Update Behaviour

- View equivalence
- Conflict equivalence
- 2 phase locking
- Timestamp equivalence

each equivalence class find a set of acceptable schedules characterized by a different complexity.

View equivalence there some definitions to be introduced:

- **reads-from**: $r_i(x)$ reads-from $w_j(x)$ when:
  - $w_j(x)$ precedes $r_i(x)$ and $i \neq j$
  - There is no other $w_k(x)$ between them.
- **final write**: is a final write if it is the last write of x appearing in the schedule.

with this solution two schedules are view equivalent if they have the same reads-from set or the same final write set.
This techniques is easy to be understand using an example. Using the flow in figure 12. The corrisponding schedule is: $S = r_1(x)r_2(x)w_2(x)w_1(x)$ is this

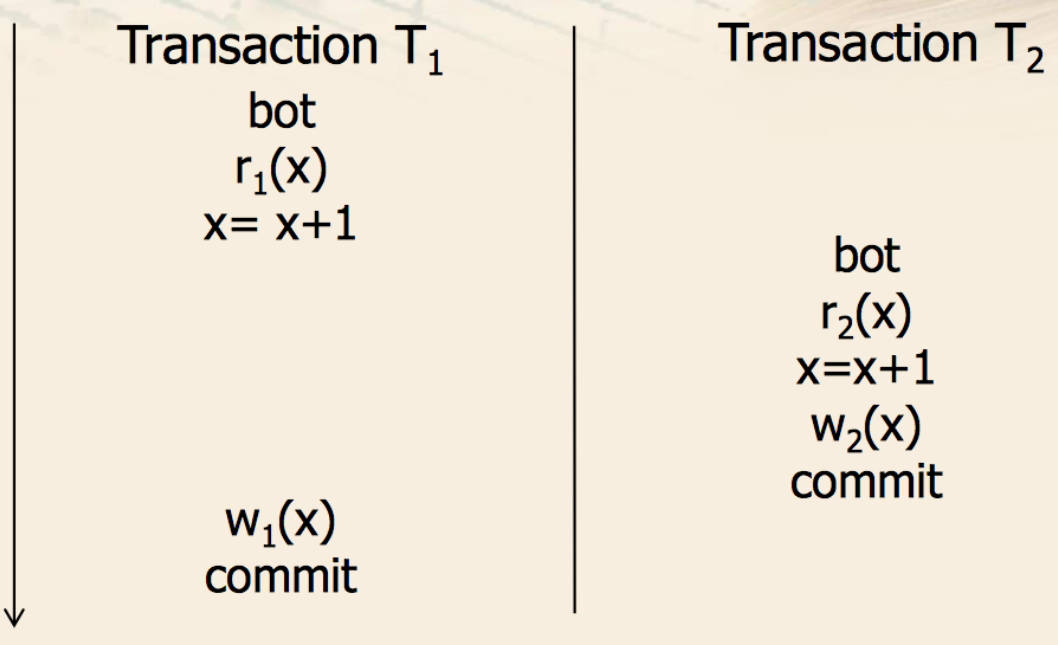


Figura 12: View Equivalence Example

schedule serializable?
There are only 2 possible serial schedules:

$$\begin{aligned} S_1 &= r_1(x)w_1(x)r_2(x)w_2(x) \\ S_2 &= r_2(x)w_2(x)r_1(x)w_1(x) \end{aligned} \tag{1}$$

In both cases S is not view equivalent to any serial schedule, not serializable, should be rejected.
The analization of this problem is linear if the schedule is given, in case of an arbitrary schedule it become NP-complete. For this problem is better to use a less accurate, but faster techniques.

Conflict equivalence Two actions are in conflict when both operate on the same object and at least one of them is a write. The conflict can be, RW, WR or WW. The conflict are equivalent if they have:

- Same conflict set.
- Each conflict pair is in the same order in both schedules.

A schedule is conflict serializable (**CSR**) if it is equivalent to an arbitrary serial schedule of the same transaction.
Dectecting the conflict serializability it is possible to exploit the conflict graph:

- Node: Each transaction.
- Edge i→j: if is there at least a conflict between T_i and T_j.

checking the cyclicity of a graph is linear in the size of the graph.

2 Phase Locking a lock is block on a resource which may prevent access to others. The operations are:

- **Lock**
  - Read Lock (R-Lock)
  - Write Lock (W-Lock)
- **Unlock**

Each operation is preceded by a request of R/W-Lock and is followed by a request of unlock. Of course the R-Lock is shared among different transaction, the write lock instead is exclusive, not compatible with any other lock.
The scheduler becomes a lock manager it receives transaction requests and grants locks based on locks already granted and so on...
When the lock request id **granted**:

- The resource is acquired by the requesting transaction.
- After the unlock, the resource, becomes again available.

When the lock is **not granted**:

- The requesting transaction is put in a waiting state.
- The wait is terminated when the resources is unlocked and becomes available again.

All the information for grant or not the lock are stored in the **lock table** with the **conflict table** (figure 13) used to manage lock conflicts.

The read locks are shared this is because the read not change the state of a data and multiple access can be exploited at the same time. A counter is used to count the number of R-Lock granted on a resources.
The lock manager, that coordinates the grant, exploit the lock table stored in main memory and, for each data object, use 2 bits to represent the 3 possible states (free, r-locked, w-locked) and a counter to count the number of waiting transaction. An example in figure 14.

There is also another version of the 2 phase locking the STRICT version. In this solution the unlock will be performed only at the end o the transaction and not when the resources is released; this difference guarantee to avoid the dirty read anomaly.
The procedure is really fast. If a transaction keep in wait over timeout the lock manager resume it and returns a NOT OK ERROR, the requesting transaction may perform a rollback o request again the same resources.

Request	Resource State		
	Free	R-Locked	W-Locked
R-Lock	Ok/R-Locked	Ok/R-Locked	No/W-Locked
W-Lock	Ok/W-Locked	No/R-Locked	No/W-Locked
Unlock	Error	Ok/It depends (free if no other R-Locked)	Ok/Free

Figura 13: Conflict Table

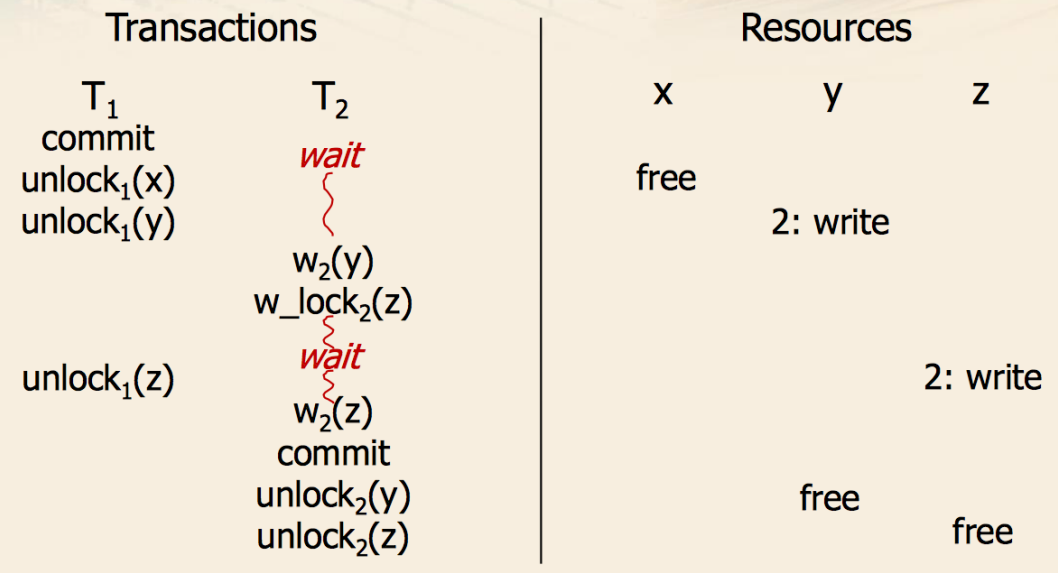


Figura 14: Conflict Table

Hierarchical Locking locks tables at different granularity levels:

- Table
- Group of Tuples (fragment)
- Single Tuple

- Single Field in a Tuple

this is an extension of the traditional locking. It allows a transaction to request a lock at the appropriate level of the hierarchy and it is characterized by a large set of locking primitives.
The **Locking Primitives** are:

- **Shared Lock (SL)**
- **eXclusive Lock (XL)**
- **Intention of Shared Lock (ISL)**: It shows the intention of shared locking on an object which is in a lower node in the hierarchy.
- **Intention of eXclusive Lock (IXL)**: Similar to ISL, but for exclusive lock.
- **Shared lock and Intetion of eXclusive Lock (SIXL)**: Shared lock of the current object and intention of exclusive lock for one or more objects in a descandant node.

The behavior is reported in figure 15.

	Resource State				
Request	**ISL**	**IXL**	**SL**	**SIXL**	**XL**
ISL	Ok	Ok	Ok	Ok	No
IXL	Ok	Ok	No	No	No
SL	Ok	No	Ok	No	No
SIXL	Ok	No	No	No	No
XL	No	No	No	No	No

Figura 15: Hierarchical Behavior

The selection of lock granularity depends on the application type:

- If it performs localized reads or updates of few objects (low lv - detailed ganularity).

- If it performs massive reads or updates (high lv - rough granularity).

the effect of lock granularity:

- If it is too coarse, reduces concurrency.
- If it is too fine, it forces significant overhead on the lock manager.

The predicate locking addresses the ghost update of type b (insert) anomaly, for the 2PL a read operation is not in conflict with the insert of new tuple, they can't be locked in advance. The PredLock allows locking all data satisfying a given predicate.

There are several isolation level:

- **SERIALIZABLE**:
  - Highest
  - PredLocking
- **REPEATABLE READ**:
  - Strict 2PL without PredLock
  - Read existing obj can be correctly repeated
  - No protection ghost update
- **READ COMMITTED**:
  - Not 2PL
  - The read lock is released as soon as the object is read
  - Reading intermediate states of a transaction is avoided (dirty reads)
- **READ UNCOMMITTED**:
  - Not 2PL
  - Data reads without acquiring the lock
  - Only allowed for read only transaction

Deadlocks can be frequent, the solution for solving it are implementing **timeout** the transaction waits for a given time, after the expiration of TO it receives a negative answer and it performs rollback. The time length could be LONG and SHORT (can be overloads the system). Other solution can be performed Pessimistic 2PL that acquire all locks before the transactions start (not always feasible) or the timestamp that put in wait mode only younger transaction. The deadlocks detection is performed using the wait graph, it could be expensive to build and mantain.

1.7 Reliability Management

It is responsible of the atomicity and durability ACID properties, it implements the following transactional commands:

- BEGIN transaction (B)
- COMMIT (C)
- ROLLBACK (A, for Abort)

it also provides recovery primitives WARM and COLD RESTART.
It manages the reliability of R/W requests by interacting with the buffer manager, it may generate new R/W requests for reliability purposes. It exploits the **log file** a persistent archive recording DBMS activity and is stored in a "stable" memory (not affected by failure, abstraction). It prepares data for performing recovery by means of the operations *checkpoint* and *dump*.

Log file is a sequential file written in stable memory that records transaction activities in chronological order. The record can be related to a transaction or system, they are saved interleaved between different transaction.
The transaction log are written in this way:

- BEGIN B(T)
- COMMIT C(T)
- ABORT A(T)
- INSERT I(T, O, AS)
- DELETE D(T, O, BS)
- UPDATE U(T, O, BS, AS)

where O represent the written object, AS is the After State (state of object O after modification) and BS is the Before State.

Checkpoint are periodically operation requested by the RM to the BM, it allows a faster recovery process. During the checkpoint, the DBMS writes data on disk for all completed transactions.
The flow of the checkpoint is:

1. All the TIDs of all active transaction are recorded
    - After the checkpoint start, no transaction can commit until the checkpoint ends.

2. The pages of concluded (C or A) transaction are synchronously written on disk.
    - By means of the froce primitive.
3. A the end of step 2, a chekcpoint record is synchronously written on the log.
    - Contains the set of active transactions.
    - It is written by means of the focre primitive.

Dump is a complete copy of the database, typically performed when the system is offline, stored in stabel memory, may be incremental. At the end a dump record is written in the log.

The log is designed to allow recovery in presence of failure, WAL or Commit precedence. The WAL (Write Ahead Log, SYNC) the BS of data in a log record is written in stable memory vefore database data is written on disk, during recovery it will allows UNDO operation. The COMMIT PRECEDENCE (ASYNC) solution write the AS in a stable memory before commit, this will allows the execution of redo operations.

Recovery Management there are two types of failures, the SYSTEM caused by software problem or power supply interruption that are causing losing in the main memoery content (buffer) but not on the disk. Or the MEDIA failure, caused by failure of devices managing secondary memory, this will lose the DB content on disk but not the log.
When a failure occurs the system is stopped, the type of recovery to be started depends on the failure type: SYS=WARM and MEDIA=COLD. When the recovery ends the system becomes again available to transactions.
The **WARM RESTART** is one of the solution for the recovery procedure:

- All the transaction completed before the checkpoint not need a recovery action.
- The transaction which committed, but for which some write on disk are not already performed REDO is needed.
- Active transaction at the time of failure (not committed) UNDO is needed.

the checkpoint is not needed to enable recovery, it only provides faster restart, because, without it, the entire log until the last dump needs to be read. This solution reads backwards the log to detecs action which should be undo or redo, then star to read forward the log and perform all the actions.
The **COLD RESTART** is the second solution for recovery, it is performed when a portion of the database on disk get a failure. The main steps are:

1. Access the last dump to restore the damaged portion of the DB on disk.
2. Starting from the last dump record, read the log forward and redo all actions on the database and transaction commit/rollback.
3. Perform a warm restart.

1.8 Triggers

The traditional DBMS is passive, query and updates are explicitly requested by users, the knowledge of processes operating on data is typically embedded into applications. The active DBMS instead have a Reactivity service provided that monitors specific database vents and triggers actions in response. Reactivity is provided by automatically executing rules, they can be:

- Event: Modification operation.
- Condition: Predicate on the DB state, cond==true: action==execute.
- Action: Sequence of SQL instructions or application procedure.

The rule engine is the component in charge of tracking events and executing rules when appropriate. The execution of the rules is interleaved with traditional transactions.
SQL provides instructions for defining triggers (CREATE TRIGGER) the syntax and semantics are covered in the SQL3 standard. The structure is divided in 3 main part:

- WHEN: the events takes place.
- IF: the condition is true.
- THEN: the action is executed.

there are also some execution modes:

- Immediate: before or after the triggering statement.
- Deferred: executed immediately before commit. (Not commercial)

and the granularity:

- Tuple (or row level): One separate exec of the trigger *for each tuple* affected by the triggering statement.
- Statement: One single trigger execution *for all tuples* affected by the triggering statement.

Oracle Triggers The base structure of a trigger is:

```
CREATE TRIGGER TriggerName
Mode Event {OR Event }
ON TargetTable
[[ REFERENCING ReferenceName]
FOR EACH ROW
[WHEN Predicate]]
PL/SQL Block
```

It can be divided in:

- Mode is BEFORE or AFTER
- Event ON TargetTable is:
  - INSERT
  - DELETE
  - UPDATE
- FOR EACH ROW specifies row level execution semantics.
  - OLD.ColName and NEW.ColName are used for accessing to the two types of data.
- WHEN is used only for row level execution.

The execution algorithm is:

1. Before statement triggers are executed.
2. For each tuple in *TargetTable* affected by the triggering statement.
   (a) Before row triggers are executed.
   (b) The triggering statement is executed + Integrity constraints are checked on Tuples.
   (c) After row triggers are executed.
3. Integrity constraints on tables are checked.
4. After statement triggers are executed.

the execution order for triggers with the same event, mode and granularity is not specified and it could be source of non determinism. If an error occurs the roll back of the triggers operation is performed. The triggers could also called in cascade, the maximum is defined by the user. The *mutating table* is the table modified by the statement triggering the trigger. The MT cannot be accessed in row level triggers, may only be accessed in statement triggers (limited access only on Oracle application).

1.9 Distributed Architectures

A possible architectural implementations is using a distributed system, the main advantages are related to performance improvement, increased availability and stronger reliability. Of course the classic client/server mechanicsm is much easier to be implemented and mantained. The distributed one are able to collaborate and are autonomous, the only problem is to guaranteeing the ACID property that requires more complex techniques.

Client/Server there are two main types of structure, the first is the **2-TIER**:

- n clients: With some application logic.
- DBMS Server: Provides access to data.

the **3-TIER** solution istead:

- n clients: Browser.
- Application server: Business logic and also web server.
- DBMS Server: Provides access to data.

Distributed system are accessed by many user at the time. Each user can be also access more than one DBMS server. Each server need to have a local autonomy, each manages its local data, concurrency control, recovery, etc... The localization instead could be the most important difference respect c/s system, this can perform a geographical distribution. Also the data availability, less probability off total block, but more in terms of local block. Least but not last, the scalability.

Design given a relation R, a data fragment is a subset of R in terms of

- Tuples:Horizontal = Not overlapped, union of table possible.
- Schema:Vertical = Overlapped on PK, join of table.
- Both:Mixed

The distributed system are based on data fragmentation over different servers. THe allocation schema describes how fragments are stored on different server nodes, it could be redundant or not redundatan if some fragments are replicated or not on different servers. When there are replication the data availability increase, but also the complexity, synchronization is needed.
The trasparency levels explains how data distribution are visible by the query programmer, the could be invisible, in this case programmer will call only one

table without knowing the fragment division. Another option is knowing the existence of fragments but not their allocation, in this case each fragment not to be used like a different table.

Classification the client is only responsible to request the execution of the query, the taks to redistributing the computation is demanded to the DBMS server. The transaction could be classificated:

- **Remote Request**:
  - Read only request
  - Single remote server
- **Remote Transaction**:
  - Any SQL command
  - Single remote server
- **Distributed Transaction**:
  - Any SQL command
  - Each SQL statement is addressed to one single server
  - Global atomicity is needed
- **Distributed Request**:
  - Each SQL command may refer to data on different servers
  - Distributed optimization is needed
  - Fragmentation transparency is in this class only

Technology using more system requires synchronization. To guarantee the ACID property some techniques need to be implemented:

- Atomicity: 2 phase commit.
- Consistency: Enforced only locally.
- Isolation: Strict 2PL and 2 Phase Commit.
- Durability: Extension of local procedures to manage atomicity in case of failure.

The Distributed Query Optimization have the tasks to split in different subqueries a single query execution request. After fast execution plan definition, the DBMS, start the different operations and coordinates everything fot a correct information exchange.

The **2-Phase Commit** protocol has the objective to coordinate the conclusion of a distributed transaction. The behaviour is similar to a wedding. There is one coordinator, the Transaction Manager (like priest) and several DBMS servers which take part to the transaction, Resource Manager (like the couple). There also 2 new logs, the TM add some information related to the protocol start/end, the RM add the ready log to synchronize the commit with the other system. The procedure is similar to the window-networks protocol, with packets and ack. Figure 16 schematize the flow.

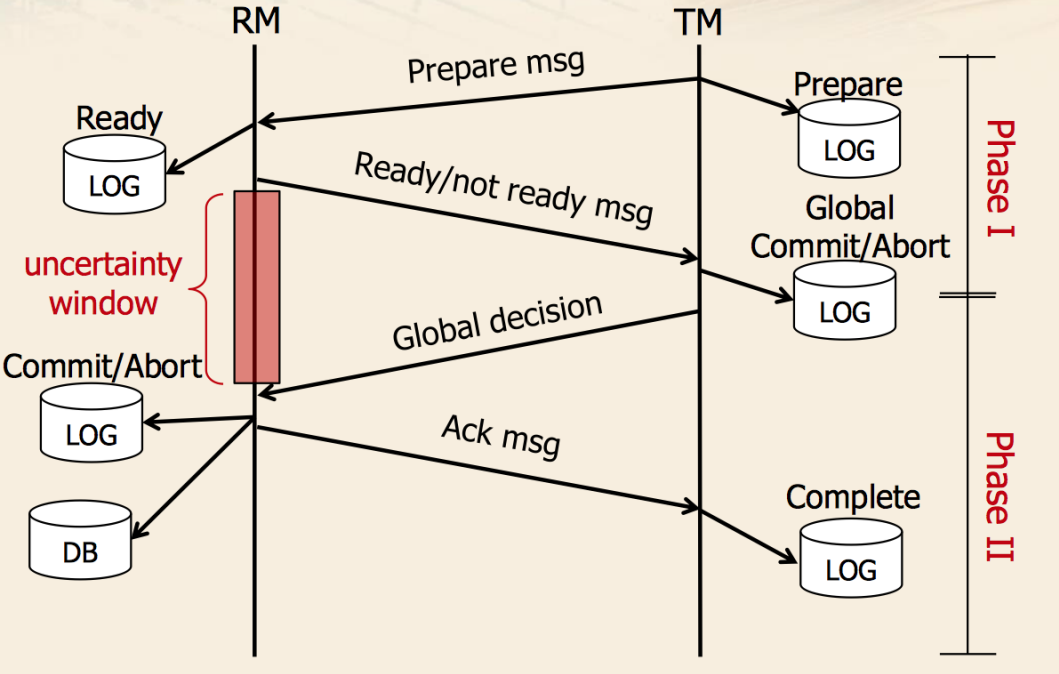


Figura 16: 2-Phase Commit

The 2 phases of the coordinator are:

- **I Phase**:
  - Prepare (Outgoing)
  - Ready (Incoming)
- **II Phase**:
  - Global Decision (Outgoing)

Also the recovery phase is a little bit modififed, the warm restart will read the READY log and asked to the TM how to procede to the restore (Remote Recovery Request). In case of the failure of the coordinator:

- Last Record=PREPARE: A global abort is written in log and send to all partecipants.
- Last Record=GLOBAL DECISION: Repeat the phase II.

2 Data Warehouses

2.1 Introduction

The database was delevoped for giving a service to the final users, like university, flight companies and other stuff. During the 80's they have understand that this system could be also used for made analisys over data. This analisys could be usefull for improving decision, forecast, cost reduction and other stuff. The goal of Business Intelligence is to provides support to strategic decision, trasforming company data inyo actionable information. This request requires of course an appropriate hardware and software infrastructure.
The data ware house is a database devoted to decision support, which is kept separated from company operational databases. The data which is:

- Devoted to specific subject
- Integrated and Inconsistent
- **Time Dependent**, non volatile

all data related to the timestamp are bigger than the other.
The data are kept separated for multiple questions:

- **Performance**: Complex queries reduce performance. Data may varies during operation, etc... The system could be developed for doing some operations but the warehousing use the system in a different way.
- **Data Management**: The data for the service could be different from the data needed for the analytics. (ex. Data addresses changing). There could be also inconsistency problem.

One of the rappresentation solution proposed is the (hyper)cube with three or more dimensions (figure 17). In this solution the cross of the three axes could be our important data, 3 like number of Milk sales in 2-3-2000 by the SupShop or the total amount in $ etc... The empty cells rappresent an inconsistency data, product not sales in that day.
The hypercube is a rappresentation based on the relational rappresentation, the STAR MODEL. This model have:

- Numerical measures: Value stored in the fact table. (ex. #sales)
- Dimensions: Describe the context of each measure in the fact table.

in figure 18 the Dimensions are *Shop, Date and Product* and the fact is the *Sale*.

An important analisys is related to the dimensions of the data warehouse, supposing: to store 2 years of data, for 300 shops for 3000 product sold every day in every shop, the fact table reach the dimensions of 660 milions of row

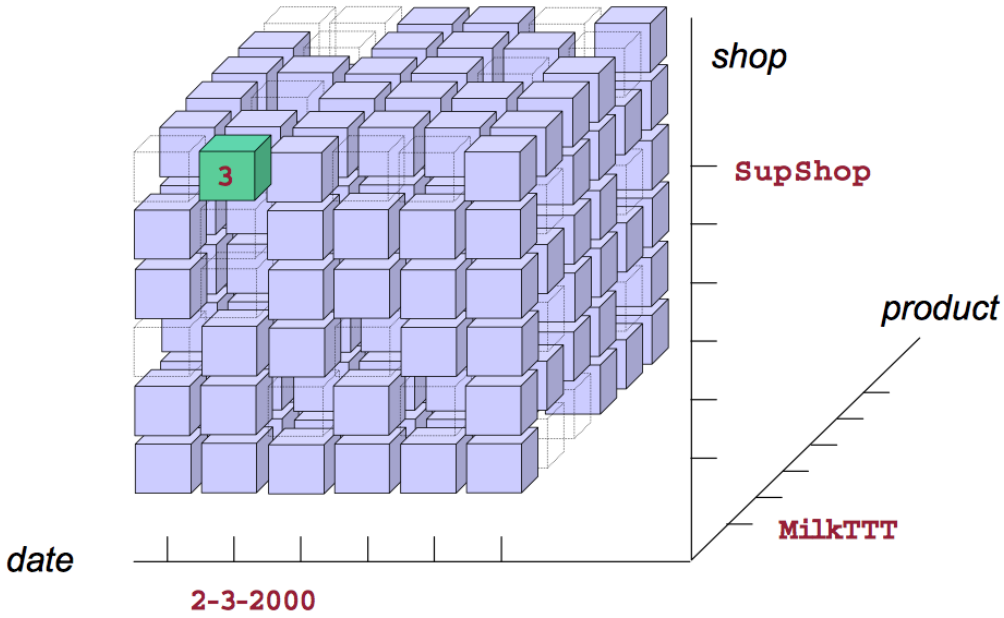


Figura 17: Hypercube rappresentation

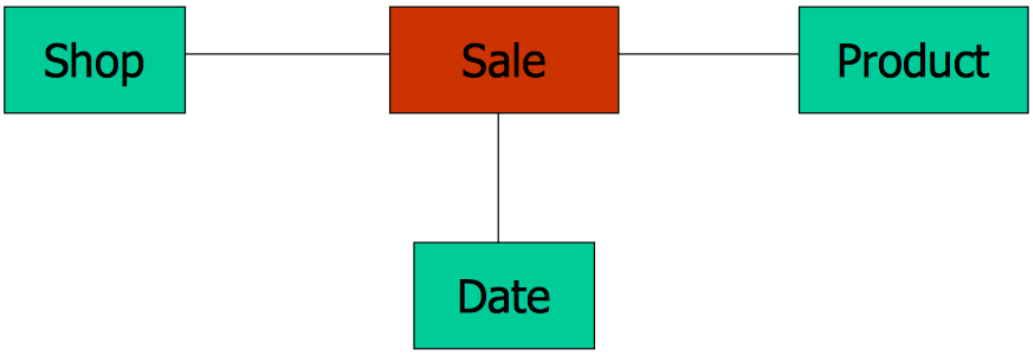


Figura 18: Star Model

(around 20GB of data).
The main operations over this kind of data is the computation with aggregated functions, with more complex operand (moving average, top ten, etc...) or applications of data mining techniques. This important evaluation are unusefull if the data is presentate bad.

Architecture the system is build on several block:

- Data Sources: External source, the DB.
- Data Warehouse: The big system.

- Data Marts: Smaller data warehouse related only to a single departement.
- OLAP servers: Multi dimensional data rappresentation.
- Metadata: Schemas information.

The data warehouse contains all the information on the company business and it requires a long time to be developed, the data mart are departimental information subset focused on a given subject, the implementation is faster than the warehouses, they requires a careful design. The data mart could be dependent (fed by the company warehouse) or independent (fe ddirectly by the sources). The most know solutions are:

- **ROLAP**: (Relational OLAP)
  - Extended relational DBMS (not sparse).
  - SQL extension for aggrgate computation.
  - Specialized access methods which implement efficient OLAP data access.
- **MOLAP**: (Multidimensional OLAP)
  - Data rappresented in multidimensional matrix (sparse data required compression).
- **HOLAP**: (Hybrid OLAP) use MOLAP for fast and user stats and ROLAP for high detailed data.

There are 3 different solution for the system architecture, 1,2 and 3 level.
The first it better to be avoided, is not a good choice to have DBMS and DW on the same side. The other two are, the 2 level rappresented in figure 19. These solution split the data and the DW, the only problem is data as soon as the data is added an "On the fly" data transformation is requires. Instead, in the three level solution (figure 20), the problem is avoided introducing a staging area used for managing and cleaning operation.

ETL Extraction, Transformation, Loading this process prepares data to be loaded into the data warehouse, is usually performed during the first load of the DW or during periodical DW refresh. The part are:

- **Data Extraction**: Data acquisition from sources.
- **Data Cleaning**: Techniques for improving data quality (correctness and consistency).
- **Data Transformation**: Data conversion from operational format to DW format.
- **Data Loading**: Update propagation to the data warehouse.

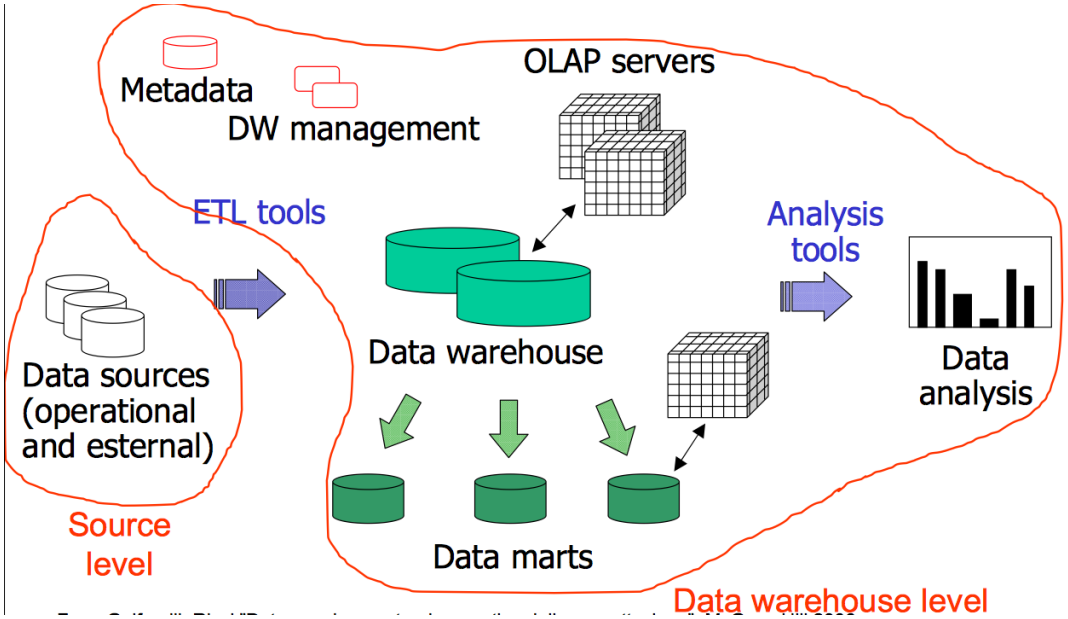


Figura 19: Two Level Architecture

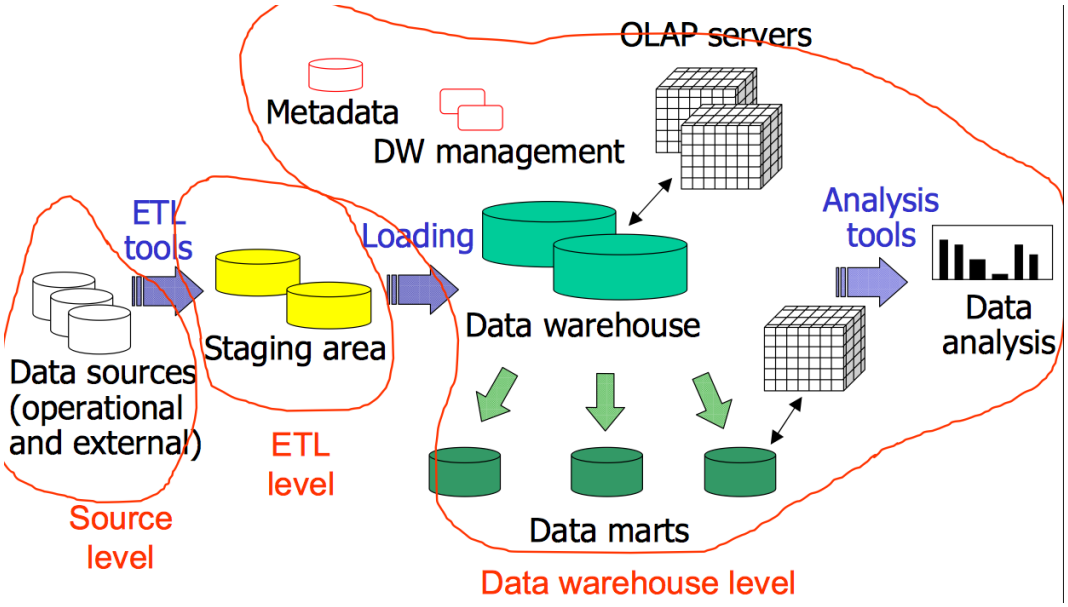


Figura 20: Three Level Architecture

Metadata these are data about data. There are different types:

- Data transformation and loading: Describe data sources and needed transformation operations.
- Data management: Describe the structure of the data in the DW, also for materialized view.

- Query management: Data on query structure and to monitor query execution (execution plan, memory and CPU usage).

2.2 Design

Using this type of structures involve some risks. Often the user think that data warehousing can solve the company's problems but is not like this, the behavior of this system is to provide that to solve in a better way the problem, not to solve problem. Also the source data could generate problems, incomplete or unreliable, also non itegrated or non optimized business processes. Good idea could be "politically correct" and developing a easy-to-use system.

Approach there are two types of development:

- **Top-Down**: Global and complete representation, complex and expensive.
- **Bottom-Up** incremental growth, separately focused on specific business areas.

in general the system follow a common flow, the Kimball lifecycle is a good rappresentation.
The data mart design:

- Operational source schemas
- Reconciled schema: Not easy do design.
- User Requirements
- Fact schema: non standard rappresentation, usefull for developing.
- Feeding: Define script for saving data.
- Physical Design: Good system for improving performance.

Requirement Analysis the first phase of designing is the analysis of the whole problem. We need to know:

- Collects: The data needed in the data mart and the constraints related to previous information system.
- Sources: Business user and operational system administrator.
- Select: Is good to strat from the most important sector of the company and feeded by few reliable sources.

The application requirements, what we need to keep in the data mart, are different:

- Description of relevant facts: Could be sales, phone call, investements, etc... With its usefull information (dimension). Granularity and time span.
- Workload: From already existent report generate new report. Other stuff produced in natural language.

There are also some structural requirements:

- Feeding periodicity: Different from realtime or weekly reports.
- Available space
- System architecture: 1, 2 or 3 level.
- Deployment: Start up and training.

Conceptual Design there isn't a standard model for this of kind of design, the dimensional fact model is proposed from Golfarelli e Rizzi in their book. This model define, for a given fact, dimensions, hierarchies and measures. The part are, figure 21:

- **Fact**: It evolvs in time modelling a relevant events (ex. sales).
- **Dimension**: Describes the coordinate of the fact (ex. sales date, shop, etc...).
- **Measure**: Describes a numerical property of the fact (ex. number of sold unit).

The hierarchy is the collection of all associated attributes of each dimensions. The attributes describe the dimension at different abstraction levels, this hierarchy represent a generalization relationship among subset of attributes in a dimension. Each edge represent a functional dependency 1:n. An example of hierarchy is: $Shop \rightarrow ShopCity \rightarrow Region \rightarrow Country$.
There some advanced features in these model:

- Non-additivity: Not aggregatable over sum.
- Optional edge
- Convergence: Two different hierarchy converge at the sime point at the end of its.
- Optional dimension: Not always existent dimension.

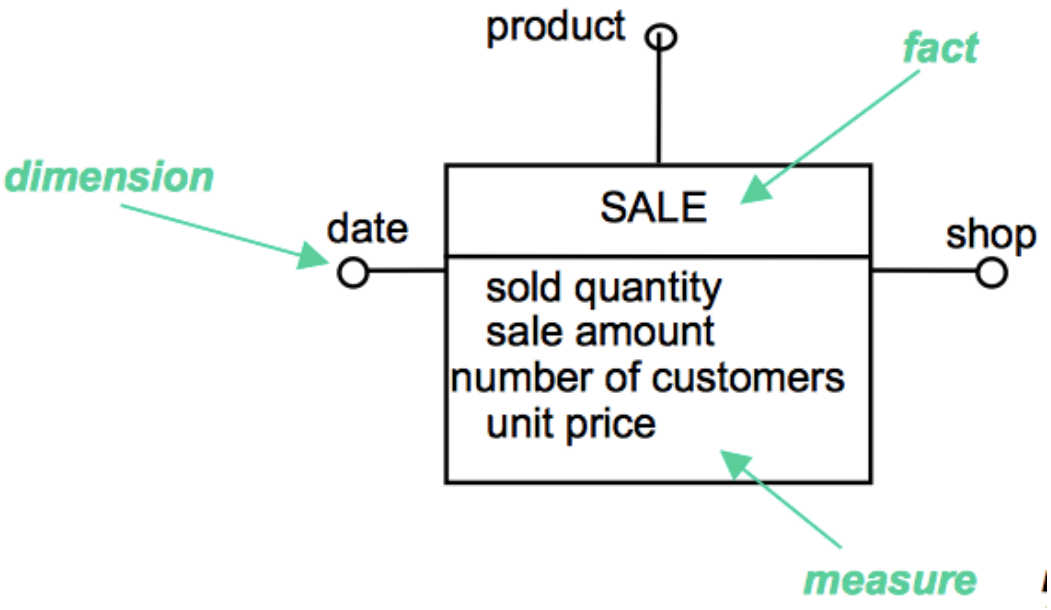


Figura 21: Conceptual Design

- Descriptive attribute: Not aggregatable for computation, only descriptive.

there other stuff likes:

- Multiple Edge: Relation n:n instead of 1:n.
- Shared Hierarchy: Different role from the same subset of data.

An interesting situation could be when the rappresentation of a fact without measure, is called factless fact schema. It can occur when you need to record the occurrence of an event, count the number of occurrence or representing events not occurred.

Aggregation computes measures with a coarser (less fine) granularity than those in the original fact schema. The reduction is obtained climbing the hierarchy. The standard operator are SUM, MIN, MAX, AVG, COUNT.
The measure can be additive, not additive (by SUM), not aggregable. Measure can be of 3 type:

- **Stream**: Evaluable at the end of a time period, aggragable by all operators (ex. sold quantity).
- **Level**: Evaluated at given time (snapshot), not additive along the time dimension (ex. Balance).
- **Unit**: Evaluated at a given time and expressed in relative terms, not additive (ex. Unit price product).

When is possible to compute aggregate from view of already aggregated value these operators are called **distributive**. Not all operators are distributive, AVG is **algebraic**, these types can compute higher level aggregations from more detailed data only when supplementary support measures are available. There is also another type called **Olistic** that can't compute aggregate from more detailed data.

Time Representation the data modification over time is explicitly represented by the event (ex. time of buying, timestamp measurement). Also dimension could be change, the number of professors, the number of stores, etc... The only difference is related to the speed of change, this is named *slowly changing dimension*.
The first (**TYPE I**) solution of this problem is to overwrite the data with the current value, this is used when the change is not important for the system like solving an error in a surname. This project the new situation over the past events. The second (**TYPE II**) solution is to directly correlate the events with the corresponding dimension value. This could be achived partitioning the data and creating a instance of the dimension. For example, *Purchases performed by married Mario Rossi* and *Purchases performed by unmarried Mario Rossi*.
The last solution (**TYPE III**) is similar to the type II because it create a new instance of the dimension, but it introduce a validity time stamp (start/end) for the dimension and a new attribute which allows to identify the root of the all dimension. The main utility of the last type is the projection to the future or to the past of and identity.

For example:
If we have a geo sub-division like nord-ovest and we compute the sels 2016 of this area, if we move one store to the area nord-center the computation of the sels of 2016 get a different results. Supposing to calculate the sels of the 2017, if we want to look at the results of the old nord-ovest configuration, but with the new data, the only solution is to use the TYPE III representation.

The third solution must be used only if the system really need this because could be really coplicated to be mantained.

Other the workload must be defined during the design phase, this can depends on user number, complexity and dimension. Probably a phase of tuning will be necessary for improving the system work.
The estimation of data volume is necessary due to a correct development of the system. Everything must be considered, from indicies, to materialized view, to time span, attribute length, etc.. This because this type of structure everything could become really big. Difficulties on volume estimation come from sparsity, because when we reducing granularity, with a high sparse cube, the reduction factor could be greater that expected.

Logical Design this phase start from the relational model (ROLAP):

- Conceptual fact schema
- Workload
- Data Volume
- System Constraints

Is a litle bit different from the traditional logical design because the data redundancy (lose of normalization) is acceptable due to a performance improve. Is a good idea to ad redundancy only in the dimension table and not in the fact table, because the number of records is really really bigger in the second one. The star schema (figure 22) is composed by:

- Dimensions:
  - One table for each dimension
  - Surrogate primary key (counter)
  - It contains all dimension attributes
  - Not explicit hierachies respect ROLAP schema
  - Totaly denormalized representation
- Facts
  - One fact table for each fact schema
  - Primary key composed by foreign keys of all dimensions
  - Measures are attributes of the fact table

Directly from the star schema we can derive the **Snowflake** schema that introduce a bit of normalization to reduce the size of the dimension table. This representation is created spliting in two or more table one dimension table, an example in figure derived directly from the previous star schema.

The main advantage are related to some space optimization, the disadvantage is that this solution need one or more further join. Normally the star schema is preferred. The only cases could be when one part of one dimension is shared above more dimension.

Multiple edges needs an appropriate implementation, there are two solution:

- **Bridge Table**: similar to classic relational schema add a table between the two tables linked by the edge to "merge" the cases. This solution allow to adds usefull attributes, like weight for computing specific calculations (ex. author income in a not equaly divison book).

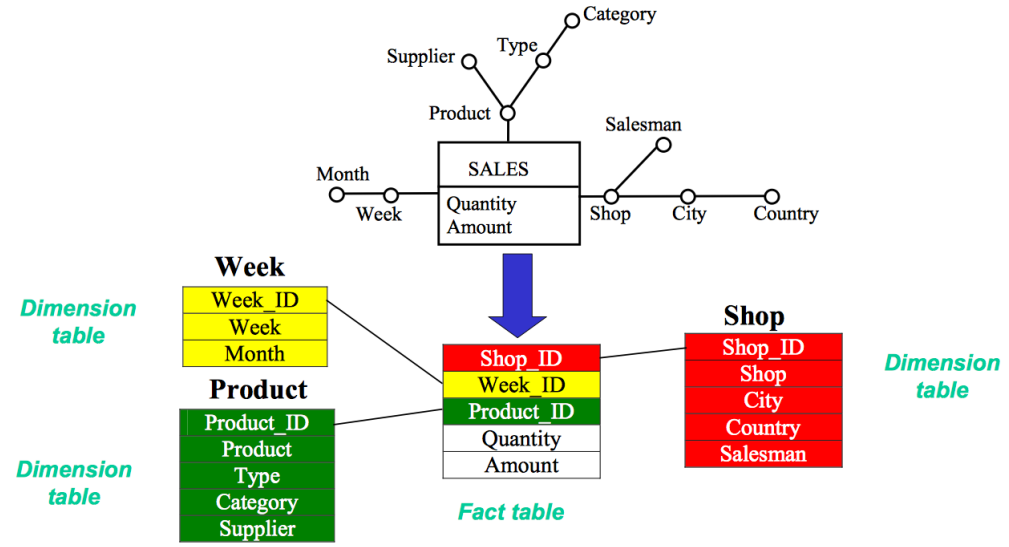


Figura 22: Star Schema

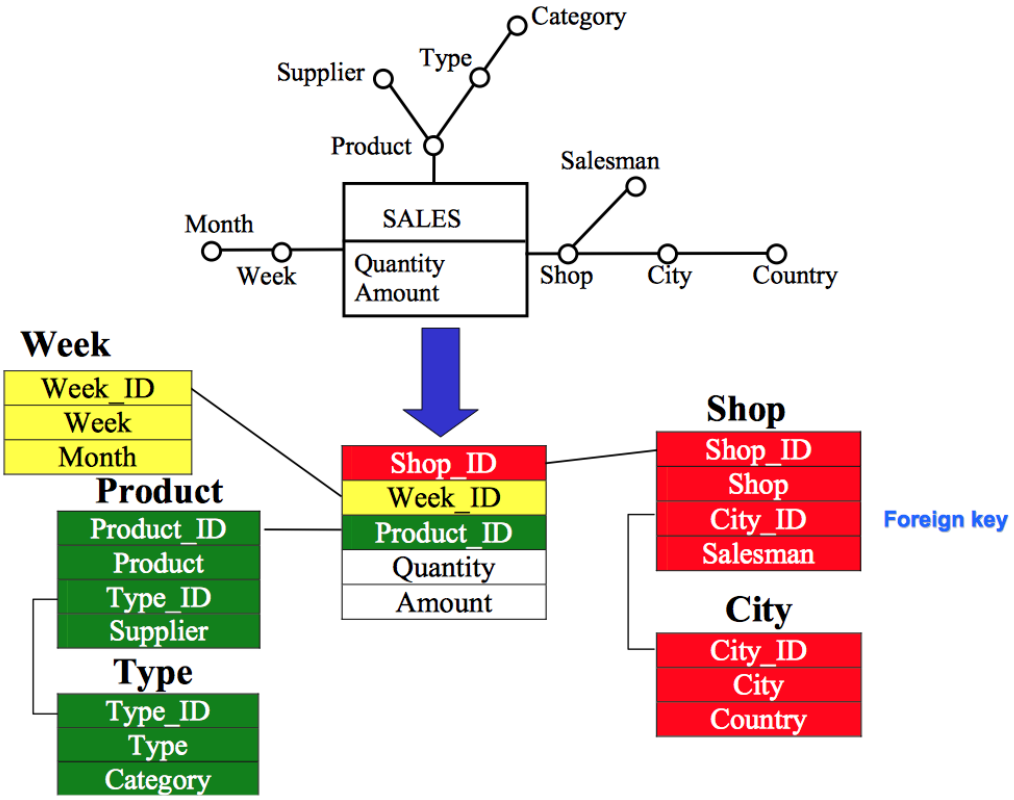


Figura 23: Snowflake Schema

- **Push Down**: Add 2 keys to the fact table. The weight in this case is wired in the fact table. The PD add problem related to increase of

redundancy, the only advantages is a minor number of join required. NOT GOOD SOLUTION.

In some cases could be necessary to create a dimension with only one attribute for a dimension, this is named **degenerate dimension**. There are two options to overcome the problem:

- Attribute directly integrate into the fact table (only attribute with very small size).
- Junk dimension: A single dimension containing several degenerate dimensions with no functional dependencies among them.

Materialized Views respect the view, the materialized ones, are table from all aspects are they aren't computed at the time the view is requested. The materialized implements an already aggregated result of a previous interrogation of the fact table. They are used for improve the performance during the computation of query less detailed. An example in figure 24. The views

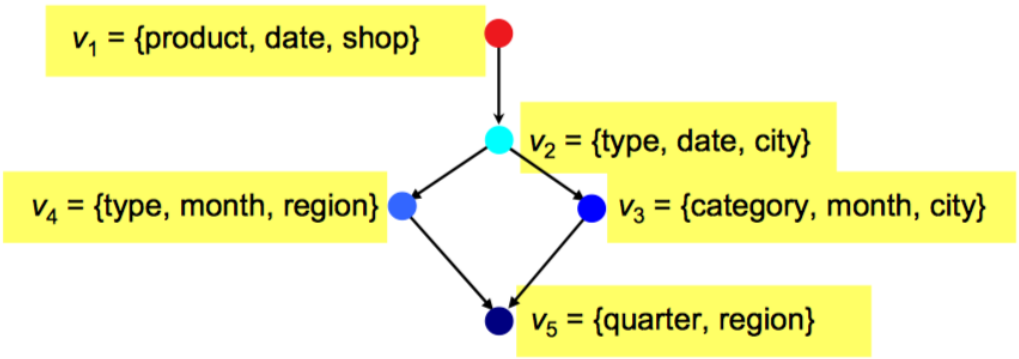


Figura 24: Example of materialized view

are SQL statement, generated starting from base tables or view with higher granularity. Considering that this views can takes up a lot of space is better to used their only when the are really usefull. A good advice is to develop views that are used from a lot of interrogation. Of course, generating a view that include a lot of aggreation attributes can be to big to be mantained, also the space occupied could be to much. In general the idea is to minimize this values:

- Disk space
- Update window
- Query cost

During the physical development of a data warehouse is important to manage the workload. The queries are computed with a lot of aggregation level and they will probably access to an huge part of each table involved. One of the main advantage of this system is the they are read-only and they don't require to be modififed. A periodical scheduled refresh is needed to keep the data valid, also indicies and other stuff.
Respect the OLTP this type of system use different index types like bitmap, join, bitmapped join index, etc...
An important phase of the development is the tuning where, looking at the real utilization behaviour the system if fitted to improve performance during real-life application. This phase is also used in case of structural changes. In more complex environment the parallelism become important, it can really increase the performance.
The index selection is performed on:

- Attributes frequently involved in selection predicates.
- High cardinality domain, with B-tree index.
- Low cardinality domain, with bitmap index.

Indexing only foreing keys in the fact table is rarely appropriate because the index could become big like the table reducing the real advantages of this stuff.

ETL the Extraction, Transformation and Loading phases are fundamental to prepare data to be loaded into the warehouse. Is performed at the first DW load and during periodical refresh, it's involve:

- Data extraction from OLTP or external sources.
- Data Cleaning.
- Data transformation.
- Data loading.

The process is eased by using the staging area. In general this phase is different if the system is during the first load or if is a periodical update/
The **extraction** acquire the data from the sources. If is the first time performa a *static* extraction grabbing snapshot of operational data. If is an update, it will use an *incremental* solution to extracting data since the last update only. In case of multiple sources is important to choose the best one. Of course is important to analyze how the data is collected:

- **Historical**: All modification, with a datastamp, are stored for a given time in the OLTP system (bank transaction, exam, etc...).
- **Partly Historical**: Not all the modification are stored, only a limited number of state.

- **Trasient**: Only the current data state is keeps on the OLTP.

Of course the real challenge is over trasient data.
The incremental extraction comparing 2 snapshot of the system it's a mess. One solution to this problem is to be **assisted by the application**, this means force the OLTP system to keep track of modification, on the selected table, in dedicated table. This means doubling the application load because every insert or update must be performed 2 times. This is the last solution, just used only on legacy system, because require a modification directly in the applications that perform the modification.
The best solution is using the **log**, this is written in any case for recovery pourpose, this means that using the log not require unnecessary data operations. Both operations are performed at the same time that the main transaction is done, are immediate.
Another possibile solution is using the **triggers**, there is the duplication like the application support, but this solution not require editing the single application, but editing only the DBMS.
The last solution is using the **timestamp**. The system need a new attribute for the *last modification time*. During the data extraction, first the system will find the first *"not yet added"* record, and then will start to scan the table from this point. This is the only deferred extraction and it means possible lose of intermediate state if the data is transient. A fast comparison over the various techniques in figure 25.

	Static	*Timestamps*	*Appilcation assisted*	*Trigger*	*Log*
Management of transient or semi-periodic data	No	Incomplete	Complete	Complete	Complete
Support to file-based systems	Yes	Yes	Yes	No	Rare
Implementation technique	Tools	Tools or internal developments	Internal developments	Tools	Tools
Costs of enterprise specific development	None	Medium	High	None	None
Use with legacy systems	Yes	Difficult	Difficult	Difficult	Yes
Changes to applications	None	Likely	Likely	None	None
DBMS-dependent procedures	Limited	Limited	Variabile	High	Limited
Impact on operational system performance	None	None	Medium	Medium	None
Complexity of extraction procedures	Low	Low	High	Medium	Low

Figura 25: Comparison of extraction techniques

Another important phase of the ETL is the **cleaning**. That phase have the task to clean all the data problem related to entry errors, different field formats and for evolving business pratices. This means remove duplicate, fill missing data, correct wrong use of field, remove impossibile or wrong data and

all the other inconsistency correlated. That problem can be solved using **data dictionary** used for entry or format errors. This of course can be performed only over data domains with limited cardinality. Another solution could be the **approximate fusion**, these techniques tries to find duplicates or similar, with a defined criterion, and perform operation to fix it when is possibile, otherwise the data must be purge again manually.

The **trasformation** required data conversion from the operational to the warehouse format. The phase is divide in two part:

- From operational to the reconciled data in the staging area.
  - Conversion
  - Matching
  - Data selection
- From reconcilied to the data warehouse.
  - Surrogate keys generation
  - Aggregation computation (view, fact table, indices, etc...)

the dimesion table loading is represented in figure 26 and the fact table loading in figure 27.

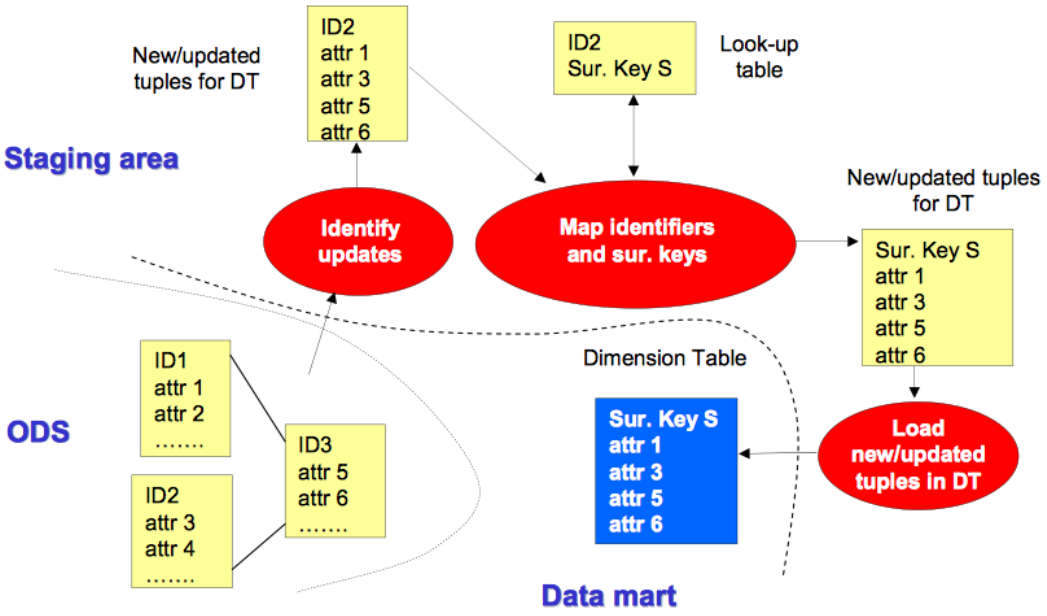


Figura 26: Dimension table loading

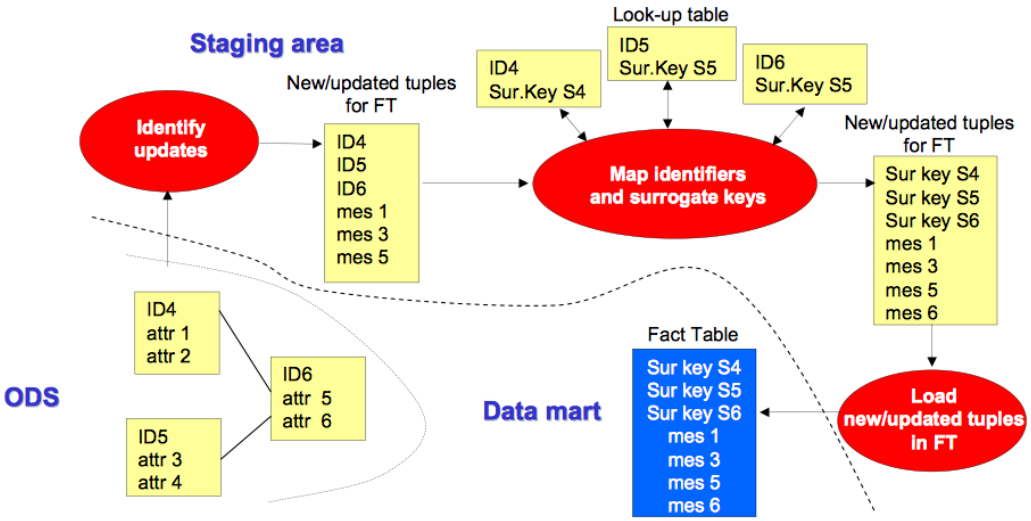


Figura 27: Fact table loading

2.3 Data Analisys

The analisys over data warehouse is slightly different from the normal query over relational DB. The OLAP analisys require to compute complex aggregate function for aggregation, for comparisons or data mining. The normal SQL isn't enough confortable to be used for this kind of computation. The datawarehouse could be query by various tools:

- Controlled Query Environments
- Query and Report generation tools
- Data mining

In the **Controlled query environment** have always a predefined structure with complex query, ad hoc analisys procedures and predefined reports. This environment requires ad hoc code development, stored procedures, predefined joins, aggregation and so on. Also some useful and flexible package for a better and easy-to-use report management are available. The system could be also fitted to use KPI of a specific economic area, like economical or finacial indicators.
When the CQE is not enough for the user a solution can be to develop an **Ad hoc query environment**. With this solution the user can define OLAP queries with a point and click techniques that generates SQL code, this means that the user must know a little bit the data structure of the warehouse. THe advantage is that, like the CQE, the user can exploits complex query with spreadsheet reports techniques.

OLAP Analisys this adds some useful operations to perform more complex query.
The rollup is a technique used for decreasing the data detail, this could be obtained by climbing up the hierarchy, from *group by store, month* to *group by city, month*, or dropping a whole dimension, from *group by product, city* to *group by product*. The drill down is the opposite of the rollup, it increase the data detail add a whole dimension or wakling dow the hierarchy. This operation could generate some problem related to spare matrix and data explosion, is problem can be solved using the slice and dice technique. This last operations selects a data subset by means of selection. The *slice* means selecting a slice not changing the granularity but adding only a equality predicate, the *dice* try to reduce the information of the slice over some reduced set. The result of this operation in figure 29, over the data in figure 28. The last

	Metrics	Dollar Sales									
	Customer Region	North-East	Mid-Atlantic	South-East	Central	South	North-West	South-West	England	France	Germa
Category	Year										
Electronics	1997	$ 138	$ 1.774	$ 384	$ 138	$ 2.346	$ 2.554	$ 2.184	$ 566	$ 199	$
	1998	$ 1.184	$ 4.529	$ 1.892	$ 7.232	$ 651	$ 9.488	$ 476	$ 2.683	$ 462	$ 7
Food	1997	$ 759	$ 682	$ 729	$ 262	$ 588	$ 469	$ 807	$ 156	$ 615	$ 1
	1998	$ 538	$ 925	$ 959	$ 677	$ 213	$ 1.503	$ 261	$ 165	$ 175	$ 1
Gifts	1997	$ 2.532	$ 1.355	$ 1.854	$ 1.413	$ 2.535	$ 2.132	$ 1.904	$ 908	$ 375	$ 1.0
	1998	$ 1.955	$ 2.785	$ 2.800	$ 2.695	$ 1.813	$ 2.844	$ 1.778	$ 1.158	$ 717	$ 6
Health & Beauty	1997	$ 624	$ 640	$ 1.317	$ 647	$ 588	$ 754	$ 654	$ 143	$ 292	$ 3
	1998	$ 611	$ 887	$ 566	$ 382	$ 499	$ 1.162	$ 1.044	$ 273	$ 72	
Household	1997	$ 5.354	$ 4.112	$ 5.410	$ 4.446	$ 3.058	$ 3.974	$ 2.654	$ 3.545	$ 2.875	$ 1.9
	1998	$ 5.787	$ 5.320	$ 5.416	$ 6.812	$ 4.334	$ 5.008	$ 7.588	$ 2.139	$ 3.649	$ 2.7
Kid's Korner	1997	$ 201	$ 398	$ 485	$ 186	$ 409	$ 323	$ 396	$ 105	$ 34	$
	1998	$ 247	$ 422	$ 441	$ 380	$ 221	$ 592	$ 290	$ 198	$ 19	$
Travel	1997	$ 624	$ 505	$ 564	$ 386	$ 300	$ 978	$ 416	$ 48	$ 38	
	1998	$ 608	$ 559	$ 1.096	$ 611	$ 464	$ 316	$ 573	$ 257	$ 198	$

Figura 28: Starting table

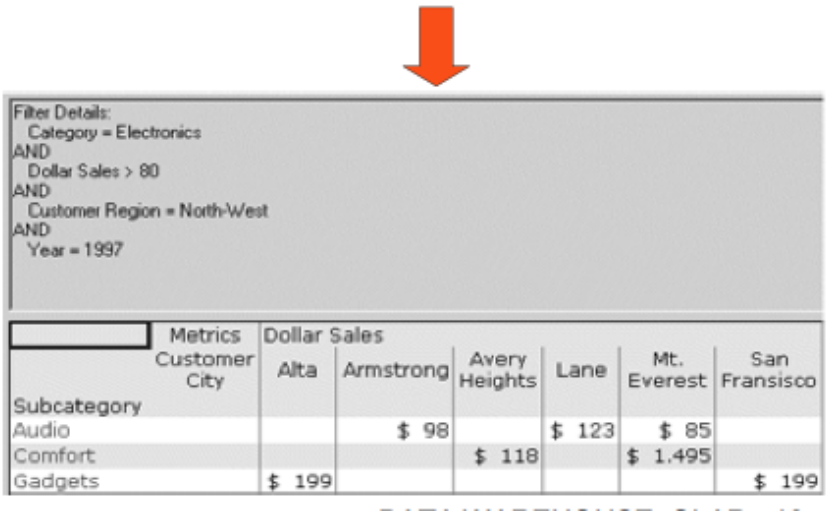


Figura 29: Table sliced and diced

operation is the pivot, this solution reorganize the multidimensional structure,

swapping the axises, without varying the detail level, this could be confortable to increase the readability of the same information.

Extension of SQL language this extension was introduced to support, in a better way, the new computation request of the data warehouse world. The were also standardized from the ANSI.
The computation window is a new clause introducted by the extended SQL. Is charaterized by:

- **Partitioning**: Rows are grouped, like *group by*, but without collapsing them.
- **Reordering**: Rows are ordered inside its group created by the partitioning.
- **Aggregation Windows**: It defines where perform the aggregation.

Example:
Show, for each city and month: (1) Sales amount and (2) Average on the current month and the two previous months, separately for each city. The query will be:

```
SELECT City, Month, Amount,
       AVG(Amount) OVER (PARTITION BY City
                         ORDER BY Month
                         ROWS 2 PRECEDING)
       AS MovingAvg
FROM Sales
```

Some consideration over this function: The sort order is required altought the computation of the window become inconsisten. When the window is not complete, like end of the month, the computation takes place on the available rows. Is possible to use several window at the same time. The aggregation window may be defined in two ways:

- Physical Level: It builds the group by counting rows (*current and 2 preceding* ***rows***).
- Logical Level: It builds the group by defining and interval on the sort key (*current and 2 preceding* ***months***).

both of this solution can implement a variable window using PRECEDING, BETWEEN and FOLLOWING. Is also posbbile to defined an UNBOUNDED window that change its size during that computation using only one fixed bound, useful for cumulative total. The difference is that the physical one not

look at missing rows and is possibile to order by more than one keys. Altought the logical can perform ordering only over numerical or data type where a distance can be defined and only by one key at the time. Another important characteristics is the possibility to compare detailed and total data without problem.
There is also a Ranking function that computes the rank of a value inside a partition, it could be of 2 types, **rank()** that computes the rank by leaving an empty slot after a tie, and **denserank()** that rank by leaving an empty slot after a tie. Of course this function requires an ordering altought would be meaningless, the partition is not necessary because it can compute also over the whole table. Is important to notice that the order by of the over is not the visualizing order, but is only the order for the ranking, if a ordered visualization is needed an external group by is compulsory.
The main important extension is related to the *group by* that allows to compute multiple aggretation without performing separte query, this increase the efficency of the whole system.
Using the rollup function over that computes multiple aggregates one at a time. The order in the expression is fundamental, it will compute the aggregate starting with all attribute and removing it one by one from right to left. The visualization of this computation have NULL value to represents the superaggregates, an example in figure 30. In this figure the first row with NULL

City	Month	Pkey	TotSales
Milano	7	145	110
Milano	7	150	10
Milano	...	...	...
Milano	7	**NULL**	8500
Milano	8	...	...
Milano	**NULL**	**NULL**	150000
Torino	...	...	150
Torino	...	**NULL**	2500
Torino	**NULL**	**NULL**	135000
...	...	...	...
NULL	**NULL**	**NULL**	25005000

Figura 30: Rollup with 3 values

present the total for the 7th month for Milano, the second one, with 2 NULL, is the total of all months for Milano, the last one, with 3 NULL, is the total

for all months, for all cities.
The **cube** function is used for computing all aggregates of all possible combination, in this case the order of the parameters is not important. This function is like performing more rollup at the time, is also well implemented respect software computation.

3 Data Mining

3.1 Introduction

During this year the companies the number of database and its size it increased a lot. All DB are potential source of useful information. Is know the our capacities of collecting data is really more the out capacity of analyzing it. The *data mining* is:

> *Non trivial extraction of* ***implicit***, ***previously unknow*** *and* ***potentially useful*** *information from available data.*

Of course, our pourpose, is to extract in an automatic way the information to represent it with abstracted models denotated pattern.
One example of this technique application is the profiling from e-commerce, search engines, social, georeference data, etc... These profling will be used for mirated advertisement, market basket analysis, brand reputation, sentiment analysis, etc...
Another important application of these techniques could be the biological data analysis for mapping DNA, demographic data, illnesses diffusion, etc...
The flow of the data analysis is named KDD, *Knowledge Discovert from Data*, a rappresentation in figure 31. The first step is the data selection. Starting

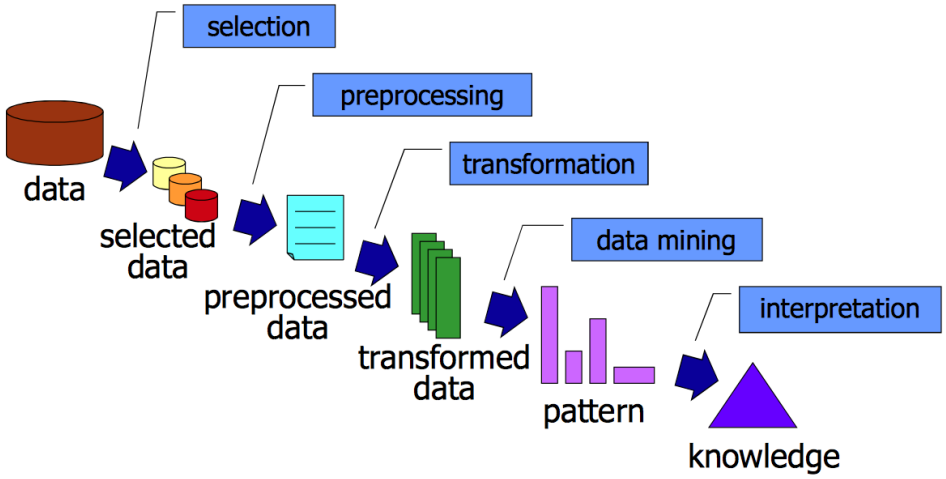


Figura 31: Knowledge Discovert from Data

from a big source, like a data warehouse, only a part of this repository will be used. The preprocessing phase is similar to the ETL phase, it perform data cleaning and data integration to generate data fitted for the mining. A good quality is needed, no good quality, no good quality pattern.
The data mining origins start from the statics, now are also used for machine learning, pattern recognition and database systems.

The are two types of analysis techniques, the **Descriptive method** that extract interpretable model describing data starting from pattern. Are useful for client segmentation for example. Another technique is the **Predictive methods** exploits some know variables to predict unknow or future values of other variables. An example could be the spam filter. This technique can't predict anomal events, not linke to the previuos history.
Some of the most important predictive methods are the **classification** this try to predict a class label by defining an interpretable model of a given phenomenon.
How this it works?
After a traning phase where the system understand from a traning set, where all data is labelled, the system define a model to parse unclassified data for correctly classifing it. Some examples are:

- Decision tree
- Bayesian classification
- Neural networks
- SVM

The selection of the methods to be used depends on accuracy, interpretablity, scalability, noise and outlier management. A more accurate process probably generate uninterpretables models and so on. There isn't a final technique, every problem need a fitted solution.
Another important predictive methods is the **clustering** that try to detect groups of similar data objects, computing distance, and identifying exceptions and outliers. The k-means is an example, like SOM or density-based. In these techniques the requirements are similar to the previous one, with more importance over the scalability and the noise managements. These results are really hard to be interpreted and this is why are often passed to a classification phase.
The **association rules** are newer and its behaviour is to extract frequent correlations or pattern from a transactional database. One of the most know result of this technique was the *Diapers* $\rightarrow$ *Beer* correlation. Using these methods allow solution for cross-selling, market basket analysis and shop layout design.
There are important, still open, issues related to:

- Scalibility to HUGE data volumes.
- Data dimensionality.
- Complex data structues, heterogeneous data formats.
- Data quality.

- Privacy preservation.
- Streaming data.

3.2 Data preprocessing

There are several data set types:

- Record
  - Tables
  - Document
  - Transaction
- Graph
  - WWW: Link from page to page
  - Molecular Structures
- Ordered
  - Spatial data
  - Temporal
  - Sequential
  - Genetic Sequence

The tabular data is a collection of record characterized by a fixed set of attributes. Presenting documents data is performed transforming it in a vector of terms. In this cases an important phase is the filtering of stock word (grammar article, word to singular, verbs to infinite, etc...).
There are different types of attribute:

- **Nominal**: ID Numbers, eye color, zip codes, etc... (D)
- **Ordinal**: Rankings, grades, height, etc... (DO)
- **Interval**: Calendar dates, temperatures, etc... (DOA)
- **Ratio**: temperature Kelvin, lenth, time, counts, etc... (DOAM)

the types depends on which of the following prepertis it possesses:

- **Distinctness**: $=, \neq$
- **Order**: $<, >$
- **Addition**: +, -

- **Multiplication**: *, /

another characterization is **discrete** or **continous** types.
The data quality is really important. Noise, outliers or missing values can generate a problems and must be managed for a correct data usage. There are a lot of reasons for missing values, from information not collected to attributes not applicable to all cases. In both case they must be handled by:

- Eliminate data objects
- Estimate missing values
- Ignore the missing value during analysis
- Replace with all possible values.

The preprocessing phase is divided in more step. The **aggregation** cobine two or more attributes into a single attribute. The purpose is to perform data reduction, change of scale and getting more "stable" data. The **data reduction** generates a reduced representation of the sataset. This rappresentation is smaller in volume, but it can provide similar analytical results. The techniques used to perform this tasks are: **Sampling** that is employed for data selection. Samples are easier to be used, because they are really little respect the entire sets and they become less expensive from the point of expenses and time consuming. Of course the idea is to preserve the quality of the entire set. There are several types of sampling:

- Simple Random Sampling: Equal probability.
- Without replacement: Each selected item is removed from the population (difference on probability).
- With replacement: Not removed from population, the item could be a duplicate.
- Stratified: Preserve the distribution of the starting schemes, performing random selection inside the single partition.

The **curse of Dimensionality** says that *when dimensionality increases, data becomes increasingly sparse in the space that it occupies.* On solution is to perform **dimensionality reduction**, it tries to reduce the amount of time and memory required by the algorithms, allow data to be more easilty visualized and may help to eliminate irrelevant features or reduce noise. The principal techniques used are:

- Principle Component Analysis: Merge data point on a vector, missing physical idea.
- Singulat Value Decomposition.

- Others: Supervised and non-linear methods.

Another way to reduce the dimensionality of data is to reduce the redundant features, for example: *purchase price of a product and the amount of sales tax paid*. Also removing irrelevant features is useful. How can i do this? There are several techniques:

- Brute-Force: Try all possible feature as input.
- Embedded: Feature selection occurs naturally as part of the algorithm.
- Filter: Selection before the run.
- Wrapper: Use the algo like a blackbox to find best subset of attributes.

Is also possible to create a new feature that can capture the important information in a data set much more efficiently than the original attributes. Thi can be achived with: Feature extraction, mapping to new space and feature construction.
The last part of data reduction is the **discretization** that split the domain of a continuous attribute in a set of intervals, reducing the cardinality of the attribute domain. Solutions:

- N intervals with the same width ($W = (V_max - V_min)/N$): Badly affect by outliers and sparse data. Incremental.
- N intervals with (approximately) the same cardinality: Fits better outliers and spare, but not incremental.
- Clustering: Fit well sparse data and outliers.

an example in figure 32.

In some cases an **attribute transformation** could be useful. Is a function that maps the entire set of values of a given attribute to a new set of replacement values such that each old value can be identifies with one of the new values. This normalization can be achieved with solution like: min-max normalization (rescaling), z-score and deciaml scaling.
A lot of operations are performed computing distances:

- **Similarity**: Numerical measure of how alike two data objects are, high value implies similar object, often falls in [0,1] range.
- **Dissimilarity**: Numerical measure of how different two data objects are, lower means different objects.
- **Proximity** refers to a similarity or dissimilarity.

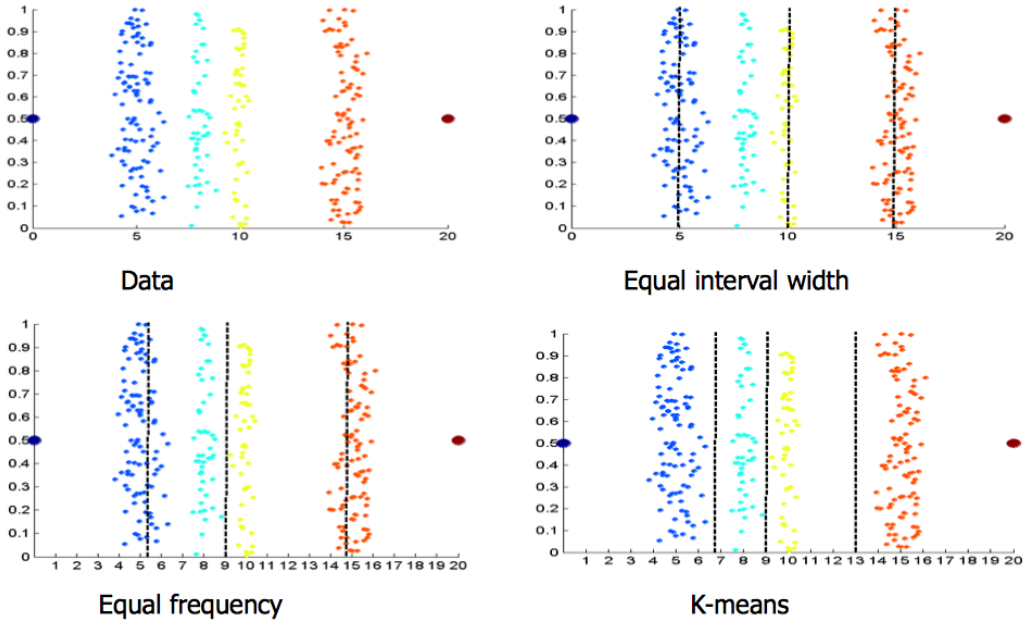


Figura 32: Discretization

In case of a simple attribute the solutions are reported in figure 33.

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal	$d = \frac{\lvert p-q \rvert}{n-1}$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - \frac{\lvert p-q \rvert}{n-1}$
Interval or Ratio	$d = \lvert p - q \rvert$	$s = -d$, $s = \frac{1}{1+d}$ or $s = 1 - \frac{d-min\_d}{max\_d-min\_d}$

Figura 33: Similarity and Dissimilarity for simple attributes

When the similarty or dissimilarty must be computed over different attributes a more complex instruments is necessary, some examples are:

- **Euclidean Distance**: $dist = \sqrt{\sum_{k=1}^{n}(p_k - q_k)^2}$ It requires standardization if scales differ.
- **Minkowski Distance**: $dist = (\sum_{k=1}^{n} |p_k - q_k|^r)^{\frac{1}{r}}$ this is the general case, the r=2 is the euclidean, r=inf is the supremum.

Is some cases could be also useful computing distances between binary vectors, one solution is using the simple matching that is: SMC = Number of

Matches / Number of attributes $(M_11 + M_00)/(M_01 + M_10 + M_11 + M_00)$. The Jaccard coefficents instead calculate similarities with: J = number of 11 matches / number of not-both-zero attributes values $(M_11)/(M_01 + M_10 + M_11)$. The final computation distance over 2 point with more attributes can be evaluated with the following formula:

$$similarity(p, q) = \frac{\sum_{k=1}^{n} \omega_k \delta_k s_k}{\sum_{k=1}^{n} \delta_k} \tag{2}$$

3.3 Association Rules

The objective of the association rules is to extract frequent correlation or patterns from a transactional database. An example could be: Diapers /*Rightarrow* Beer that it means that, for example, 30% of transaction containing diapers also contains beer. **It is not a cause-effect correlation!**.
A collection of a transaction is:

- Set of items.
- Without order correlations between them.

The association rule extraction is an exploratory technique that can be applied to any data type. It can also be applied over textual data document, etc...

Definitions An association rule is compesed by multiple parts:

- **Itemset**: Set including one or more items.
- **k-itemset**: Itemset that contains k items.
- **Support count** (#): Frequency of occurrence of an itemset.
- **Support**: Fraction of transaction that contains an itemset.
- **Frequent Itemset**: Itemset whose support is greater than or equal to a *minsup* threshold.

We need also to define some rule quality metrics. Given the association rule A $\Rightarrow$ B (A, B are itemsets) the:

- **Support**: Fraction of transaction containing both A and B: $(\#\{A, B\}/|T|)$
  - $|T|$ is the cardinality of the transactional DB.
  - A priori probability of itemset AB.
  - Rule frequency in the DB.
- **Confidence**: Frequency of B in transaction containing A: $(sup(A, B)/sup(A))$ [Conditional probability]

An example with data from table below:

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diapers, Milk
4	Beer, Bread, Diapers, Milk
5	Coke, Diapers, Milk

From the itemset {Milk, Diapers} the following rules may be derived:

- Rule: Milk $\Rightarrow$ Diapers
  - Support: $\#\{Milk, Diapers\}/\#trans = 3/5 = 60\%$
  - Confidence: $\#\{Milk, Diapers\}/\#\{Milk\} = 3/4 = 75\%$
- Rule: Diapers $\Rightarrow$ Milk
  - Support: 60%
  - Confidence: $\#\{Milk, Diapers\}/\#\{Diapers\} = 3/3 = 100\%$

Given a set ot transaction T, association rule mining is the extraction of the rules satisfying the constraints of the minimum support and confidence. That rules is not easy to be obtained.
During the execution of these extraction we like to have a **complete** (**all** rules satisfying both constraints) and **correct** (**only** the rules satisfying both constraints). This is true for the association rules, but not for all techniques that we will study during this course.

Extraction The first easy solution is the **brute-force** approch, it:

- Enumerate all possible permutations.
- Compute support and confidence for each rule.
- Prune all the rules that do not satisfy the threshold.

Of course this method is computationally unfeasible. The fisrt variation from the brutal method is the following. Given an itemset, the extraction process may be split:

1. Generate frequent (passing the threshold) itemsets.
2. Generate rules from each frequent itemset.

The two step above have a lot of problem anyway, the first one can be performed with different techniques like Apriori, FP-growth, etc.. Best it still remain computionally expensive. The second one is more standard and it can be improved too much and is the generation of all possible binary partitioning of each frequent itemset. The BF approch have a huge complexity $\tilde{}O(|T|2^d w)$ where d is the number of items and w the transaction length.
How to improving the efficiency?

- Reduce the number of candidates.
- Reduce the number of transaction (e.g. Remove all the transaction with less then x items searching for y items).
- Reduce the number of comparisons.

Apriori Principle [Agr94] is the first studied algorithm, it say that: "*If an itemset is frequent, then all of its subsets must also be frequent*".
Of course the support of an itemset can never exceed the support of any of its subsets. This is guarantee by the antimonote property of the support measure: $if A \subseteq B \rightarrow sup(A) \geq sub(B)$. This allow a candidates reductions.
It's a level-based approch, at each iteration extracts itemsets of a given length k. For each level, two main steps are executed:

1. Candidate Generation:
   - Join Step: Generate candidates of length k+1 by joining frequent itemsets of length k.
   - Prune Step: Apply Apriori principle: Prune length k+1 candidate itemsets that contain at least one k- itemset that is not frequent.
2. Frequent iteset generation:
   - Scan DB to count support for k+1 candidates.
   - Prune candidates below minsup.

The generation of candidates start ffrom L_k, that is the set of candidate of length k in a lexicographical order. For each candidate of length k:

- Self-join with each candidate sharing same prefix l_{k-1} prefix.
- Prune candidates by applying Aprirori principle.

An example to clarifying:
$L_3 = \{abc, abd, acd, ace, bcd\}$

- Self-join:
  - **abcd** from *abc* and *abd*.

  - **acde** from *acd* and *ace*.
  - **bcd** will be prune because there are not frequent items in the next step.

- Prune with Apriori:
  - **acde** is removed because *ade* is not in L3.
  - $C_4 = \{abcd\}$

Some criticalities can be found:

- Scan transaction DB to count support of each itemset:
  - Total number of candidatesmay be large.
  - One transaction may contain many candidates.

The best efficient data structure is the hash-tree. The leaf will contains list of items, the interior node are hash table.
The candidate generation can have real problem with the generation of the 2-itemset because is a cartesian product, also extracting long frequent itemsets will requires generating all frequent subsets. The number of scans is n + 1 when longest frequent pattern length is n.
Other factors are:

- Minimum support threshold: Lower support threshold increases number of frequent itemsets.
- Dimensionality (number of items): More space to store support count. If the number of frequent increase, both computation ans I/O costs may also increase.
- Size of DB: Apriori multiple passes, run time increase with the number of transactions.
- Average transaction width: Width increase in dense data sets. May increase max length of frequent itemsets and trasversals of hash tree.

Some improving algorithm were proposed:

- Hash-based itemset counting [Yu95]
- Transaction reduction [Yu95]
- Partitioning [Sav96]
- Sampling [Toi96]
- Dynamic Itemset Counting [Motw98] (Sergey Brin)

The second algorith studied is the **FP-growth** [Han00] that start from the ideas that Apriori makes too many scans of the DB. Is base to exploits a main memory compressed rappresentation of the database, the FP-tree:

- High compression for dense data distrivution.
- Complete representation for frquent pattern mining.

Frequent pattern mining by means of FP-growth:

- Recursive visit of FP-tree.
- Divide-and-counquer approach.

This solution require only two databse scans. The construction of the FP-tree is based on three steps:

1. Count item support and prune items belowe minsup threshold.
2. Build Header Table by sorting items in decreasing support order.
3. Create FP-tree:
   - For each transaction t in DB:
     - Order transaction t items in decreasing support order (like HT).
     - Insert transaction t in the FP-tree:
       * Using existing path for common prefix.
       * Create new branch when path becomes different.

At the end, also some pointer directly to the point and between all point is created, are called item pointer.
The algorithm of the FP-growth is based on 2 simple steps:

1. Scan header table from lowest suppot item up.
2. For each item i in header table extract frequent itemsets including item i and items preceding it in Header Table:
   (a) Build **Conditional Pattern Base** for item i (i-CPB): Select prefix-paths of items i from FP-Tree.
   (b) Recursive invocation of FP-growth on i-CPB.

The example is too large to be write here, looks at 3-DMassrules from the slides course.
Some other solution are implement like the vertical data layout, etc...

One of the main problem of these systems is the waste of space of the rappresentation. This is why a compact repersentation is needed. An itemset

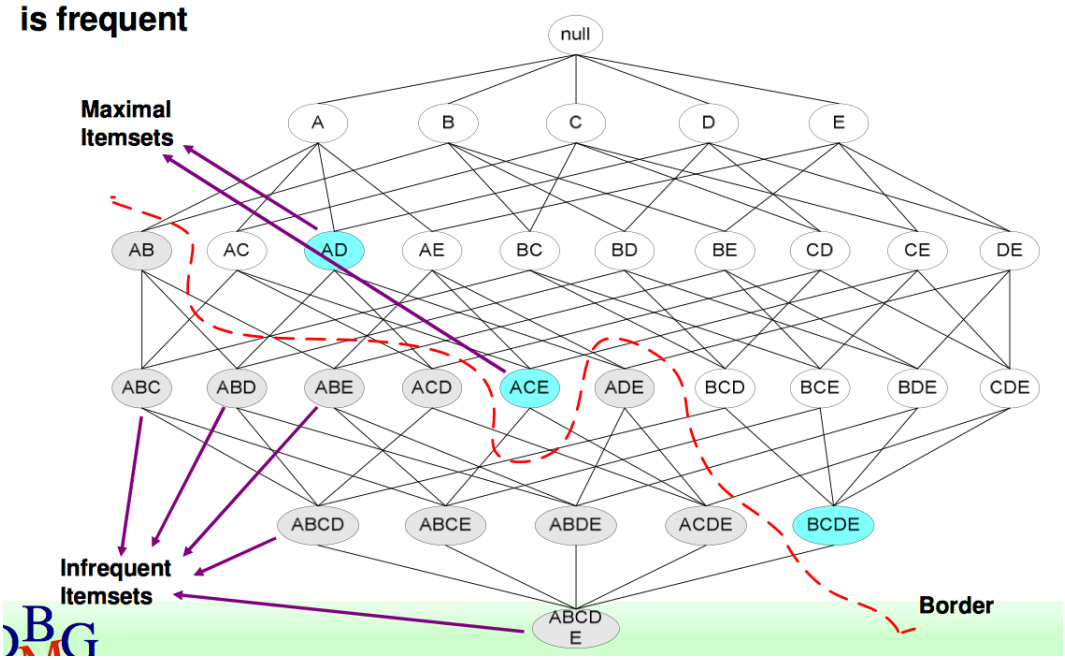


Figura 34: Maximal Frequent Itemset

is **Maximal Frequent** if none of its supersets is frequent, of course it can be computed only with a border line defined by the support, althought it loose is sense. In figure 34, there are some highlighted elements AD, ACE and BCDE, they are MFI because there no other items longer then them with a sufficient frequency, infrequent, to be kept. They are useful because they can collapse a lot of informations. Some algorithms for generating it was developed.
Another important label for the itemsets is the **Closed Itemset** that occurs when if none of its immediate supersets has the same support as the itemset. In the case of table below:

itemset	sup
{A}	4
{B}	*5*
{C}	3
{D}	4
{A,B}	*4*
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	*4*
{C,D}	3

The itemset {A} is not a CI because there is another itemset, longer, that include A with the same support of it and it's {A,B}, both for itemset {D}

because it as the same support of {D,B} but is shorter. Is short word they are the longest itemset for a given length. The main difference respect the MFI is that they reduce the number of the itemsets but without losing information, you can generate all itemsets from them. Of course a Maximal is always a closed itemset.

One of the most hard choice is the selection of an appropriate *minsup* threshold:

- Too **high** *minsup*: Itemset including rare but interesting items may be lost.
- Too **low** *minsup*: Computionally very expensive, very large number of frequent itemsets.

A large number of pattern may be extracted, a solution is to rank patterns by their interestingness. For objective measures it useful to rank patterns based on statistics computed from data. There are also subjective measures that rank patterns according to user interpreatation.

Confidence measure is the confidence always a reliable measure? No, this example will explain why:
In high school of 5000 students:

- 3750 Eat Cereals
- 3000 Play basket
- 2000 Eat cereals and play basket

One of the extractable rule is: Play Basket ⇒ Eat Cereals (Sup = 40%, Conf = 66.7%) and is misleading because eat cereals has sup 75% (that is greater than 66.7%). The problem is caused by the high frequency of rule head ("eat cereals"). Data used are:

	Basket	**Not Basket**	**Total**
Cereals	2000	1750	3750
Not Cereals	1000	250	1250
TOTAL	3000	2000	5000

The solution to this problem is using another measurement called **Correlation** or **Lift**. The formula is:

$$r : A \Rightarrow B$$
$$Correlation = \frac{P(A,B)}{P(A)P(B)} = \frac{conf(r)}{sup(B)} \tag{3}$$

The result will be:

- Statistical Independence: C = 1
- Positive Correlation: C ¿ 1
- Negative Correlation: C ¡ 1

Using the previous data the correlation become:

- Play Basket ⇒ Eat Cereals: C = 0.89 (Negative Correlations)
- Play Basket ⇒ Not(Eat Cereals): C = 1.34

3.4 Classification

The classification is the process used to predict the class label. The process start from a training data, already classified by definition, the phase will generated a model. When unclassified data is passed through the model can be classified. A schema of classification process in figure 35. There are several

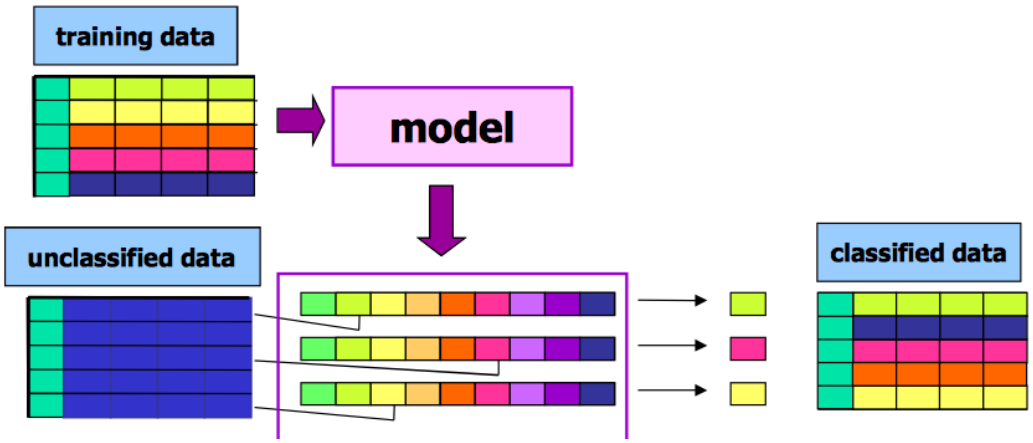


Figura 35: Classification process

different approaches to the classification problem and they can be classified by its characteristics like accuracy, interpretability, scalability and noise/outliers management. Given a:

- Collection of class labels
- Collection of data objects labelled with a class label

the goal is to find a descriptive profile of each class, which will allow the assignment of unlabeled objects to the appropriate class. After the training process is important to check the quality of the model, this is perfomed using a test set that is a collection of labeled data objects used to validate the calssification model, most of the time is directly direved from the training set.

Decision Trees are the simplest method of classifing. The main structure is a tree where will flow some data to be checked by all different nodes to providing a classification. The process will end when, a block of data, reach a leaf (termination node, blue) where there is a classification class tag. This An example in figure 36. Each internal nodes is a splitting attribute, the number

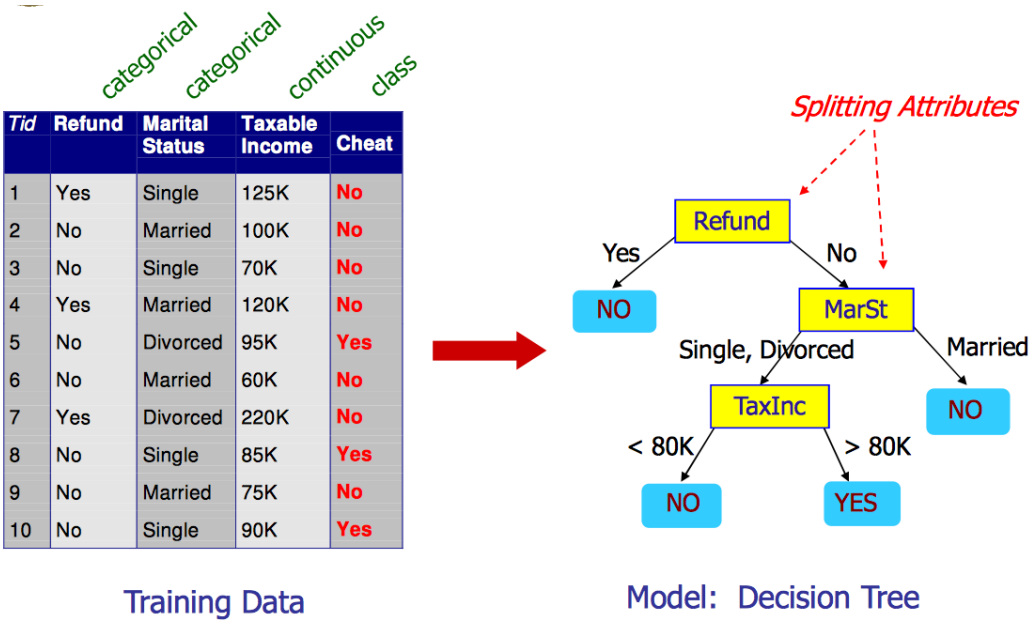


Figura 36: From data to decision tree

of exiting arch is correlated to the number of different value of the attributes, it could be binary or more.
The first algorith to build a decision trees is the Hunt's, nowadays the ID3 or C5.0 are more used. The flow is:

- If D_t contains records that belong to the same class y_t: Then is a leaf node labeled as y_t
- If D_t contains records that belong to more than one class:
  1. Select the "best" attribute A on which to split D_t and label node t as A.
  2. Slipt D_t into smaller subsets and recursively apply the procedure to each subset.
- If D_t is an empty set: The t is a leaf node labeled as the default (majority) class, y_d.

All this flow is achieved with a greedy strategy, an exshaustive search will be superflous. The best attributes for the split is selected locally at each step (not a global optimum). The main problem are related to the kind of test

condition, the selection of the best and the stopping condition.
The structure of test condition depends on attribute type that could be: nominal, ordinal and continuous and it depends on number of outgoing edges, 2 or multy-way split. The splitting can be of 2 different types, multy-way that it use as many partitions as distinct values, o binary that divide all the values in 2 subset or that have only teo distinct possible values. In case of continuos attributes there are different techniques:

- Discretization: The subset are created using average, percentiles, clustering. They can be Static, defined at once the beginning, or dynamic, discretized during the tree iduction.
- Binary: Considering all possible splits and find the best cut (computationally intensive).

The selection of the best attribute is really important, our goal is to reach the end of the process, a leaf, as fast as possible, this is achieved by having a subset with only value of a single class. Using an attribute that not divide our data in "good" partitions, become unuseful, non-homogeneous and high degree of impurity (*"Own car?"*). Defining good partition that collect a lot of data of a single class is important, more homogeneous, low degree of impurity (*"Car type?"*). Also dividing is too much way is a waste of resources, not all attribute are profits for our tasks (*"Student ID?"*). An example in figure 37.

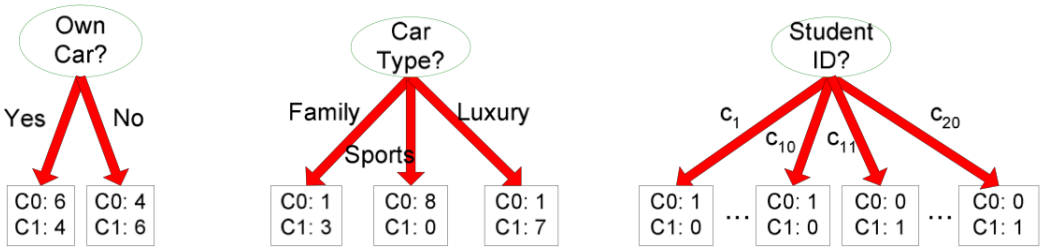


Figura 37: Attribute selection

How evaluate the impurity? The **gain** is the must used, it evaluate the density of the different classes before the splitting (M0), it apply two or more attribute and that it recompute the density after that applies (MA, MB), at the end it will compare the difference between the M0, MA and M0, MB and it will kept the purest one.
Another weel know index of purity is the **GINI**, that is, for a given node t:

$$GINI(t) = 1 - \sum_j [p(j|t)]^2 \tag{4}$$

The value is:

- Maximum (1-1/#ofClass): When the records are equally distributed among all classes, higher impurity degree.
- Minimum (0.0): When all records belong to one class, implying a lower impurity degree.

The idea is to divide the set in a weight way, a little but "more pure" class will have a greater weight, a big but "less pure" class will have a littler weight. All is achieved by using the formula:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i) \tag{5}$$

where n_i is the number of records at child i and n is the number of records at node p. In case of continuos attribute the system will start from the present value of that attribute and than it will try all the possibile splitting value to evaluate the best (GINI Index) solution, an example in figure .

Cheat	No		No		No		Yes		Yes		Yes		No		No		No		No	
Taxable Income																				
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		***0.300***		0.343		0.375		0.400		0.420	

Figura 38: GINI index for continous attribute

Another well-know solution is the INFO (or Entropy impurity measure). The main difference is how this two index label the value. The entropy value is more "pessimistic" than the GINI one, because, for the same set, will give a worst quality of purity index for a class with a not-high impurity. In figure is easy to understand.

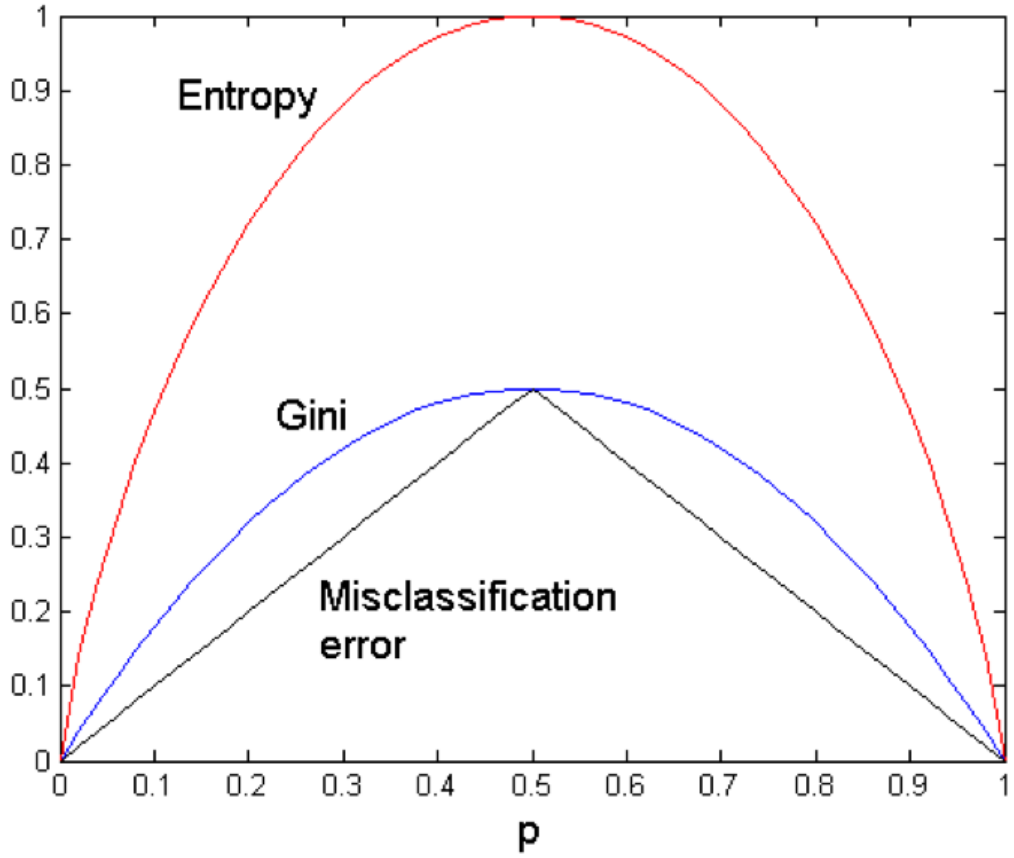


Figura 39: GINI and INFO comparisons

Stopping Criteria the termination, for the Tree Induction, occurs in 3 different cases:

- Stop exapanding a node when all the records belong to the same class.
- Stop exapanding a node when all the records have similar attribute values.
- Early termination.

Pro & Cons the decision tree based calssification have the advantages of:

- Inexpensive to construct.
- Extremely fast at classifying unknow records.
- Easy to interpret for small-sized trees.
- Accuracy is comparable to other classification techniques for many simple data sets.

Instead, the main disadvantage is:

- Accuracy mat be affected by missing data.

There are also some well-know issues related to this classification:

- Under-over fitting.
- Missing values.
- Cost of classification.

The first issues occurs when the generated tree is too specific (high number of nodes) for the training set, it means that the error rate will increase during is application. The under fitting is the problem of having few nodes that implies a not very precise model. In figure 40 the underfitting border is when the blue curve become more stable (around 80 nodes).

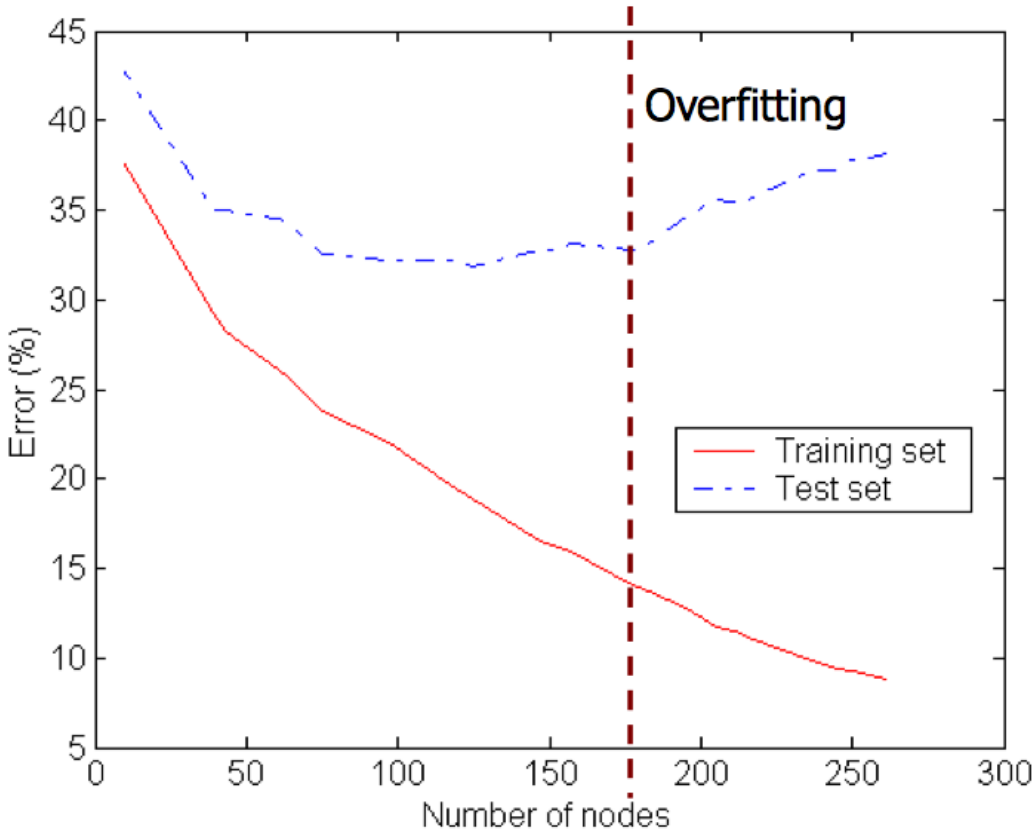


Figura 40: Under and over fitting problem

This problem is solved using the **pruning**. The PRE-pruning:

- Stop the algorithm before it becomes a fully-grown tree.

- Typcal stopping conditions are:
  - If all instances belong to the same class.
  - If all the attribute values are the same.
- Other more restrictive conditions:
  - User defined threshold for the number of instances.
  - If class distribution of instances are independent of the available features.
  - If the exapanding will not improve the impurity measure.

The POST-pruning solution:

- Grow decision tree to its entirety.
- Trim the nodes of the decision tree in a bottom-up fashion.
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined from majority class of isntances in the sub-tree.

This is why the test set MUST NOT be the same of the training.
The handling of missing attribute values is important. They affect the tree decision construction in three ways:

- How impurity measures are computed.
- How to distribute instance with missing value to child nodes.
- How a test instance with missing value is classified.

Search Strategy Finding an optimal decision tree is NP-hard. The algorithm presented so far uses a greedy, top-down, recursive partitioning strategy to induce a reasonable solution. There are other strategies like bottom-up and bi-directional. The best situation where apply the decision tree is when the data is easy to be split in different class using boundaries parallels to the axes (41). When the problem isn't solvable in this way is better to use different approaches (42).

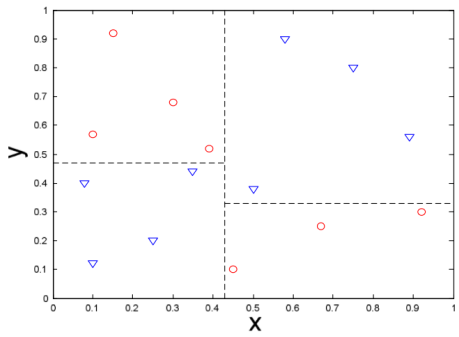


Figura 41: Good use

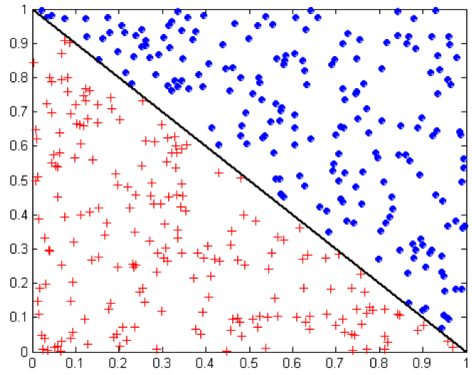


Figura 42: NOT Good use

Rule Classification Another possible technique is to classify records by using a collection of *"if...then..."* rules. The result of applying this methods could be of three type:

- Row matched by one rule.
- Row matched by more rules.
- Row not matched.

Rules can have two characteristics:

- Mutually exclusive: Rules are independent of each other. Every record is covered by at most one rule.
- Exhaustive Coverage: If it accounts every possible combination of attribute values. Each record is covered by at least one rule.

Is possible to translate a Decision Tree to a Rule-based classification. The rules just created are ME and EC by definition because you can follow only one path at the time of the tree. Simplifying the rules could cause the lose of the ME, in these case a record may trigger more than one rule, the possible solutions are:

- Ordered rule set.
- Unordered rule set - Use voting schemes.

When the lose is over the exhaustivity, having a record that not trigger any rules, the solution is using a default class (majority).
The advantages of this kind of classification are:

- As highly expressive as decision tree.
- Easy to interpet.

- Easy to generate.
- Can classify new instances rapidly.
- Performance comparable to decision tree.

Associative Classification This model is defined by means of association rules: (Condition) $\rightarrow$ y, where the body is an itemset. This scheme not look at one item at the time but to both (item, value) togheter, the difference is in the model generation:

- Rule Selection & Sorting: Based on support, confidence and correlation thresholds.
- Rule Pruning: Database coverage, the training set is covered by selecting topmost rules according to previous sort.

Some PRO of this approach are:

- Interpretable model.
- Higher accuracy that DT (correlation among attributes is considered).
- Efficient classification.
- Enaffected by missing data.
- Good scalability in the training set size.

The WEAK point are:

- Rule generation may be slow.
- Reduced scalability in the number of attributes.

Neural Networks They are inspired by the structure of the human brain, with elaboration units (neurons) and connection network (synapses). The structure is simplyfied in figure 43.

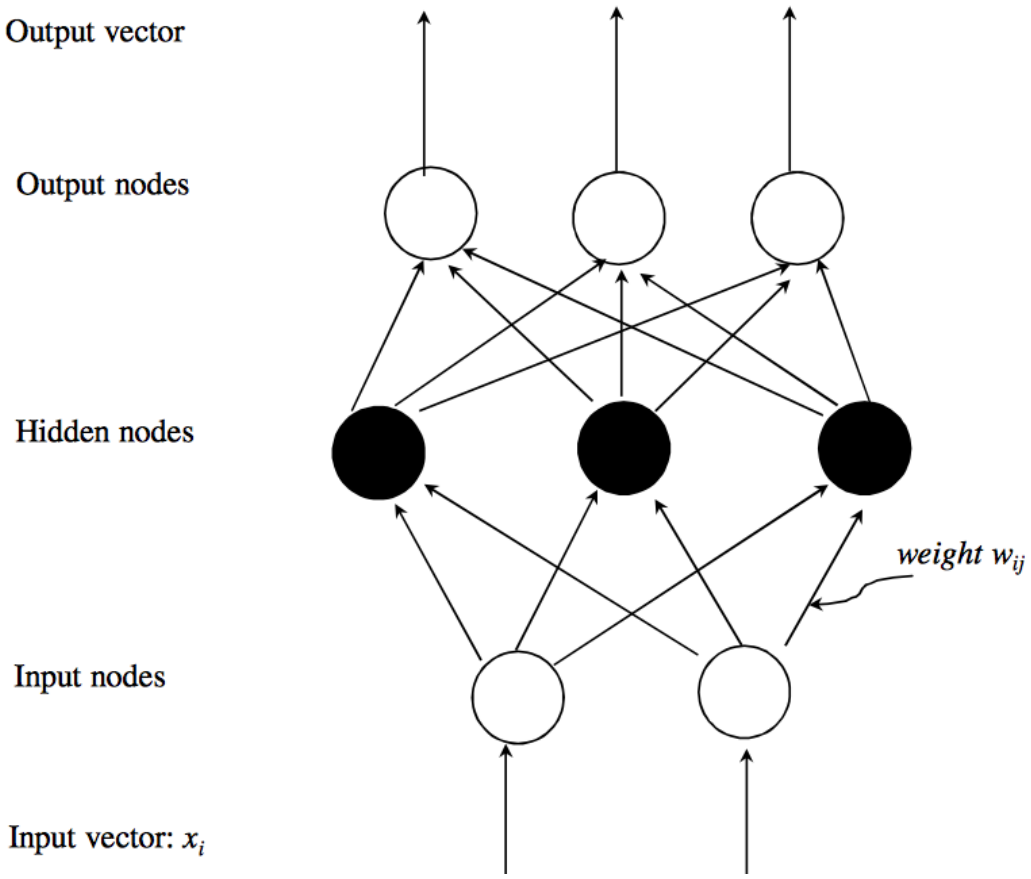


Figura 43: Structure of a neural network

The general structure of a neurons use an input and a weight vectors to generate a weighted sum (scalar product). After the addition of an offset it compute a function and then it provides the output values.
The model of the networks is based on all the weight and all the offset value of each node.
The training is based on an iterative approach. The base algorithm is:

1. Assign random values to weights and offsets.
2. Process instances in the training set one at a time:
   (a) For each neuron, compute the result when applying weights, offset and activation function for the instance.
   (b) Forward propagation until the output is computed.
   (c) Compare the computed output with the expected output, and evaluate error.
   (d) Backpropagation of the error, by updating weights and offset for each neuron.

The process ends when:

- The % of accuracy above a given threshold.
- The % of parameter variation (error) below a given thereshold.
- The maximum number of epochs (number of times training sets parsed) is reached.

These system has some strong points:

- High Accuracy.
- Robust to noise and outliers.
- Supports both discrete and continous output.
- Efficient during classification.

The main weak points insted:

- Long training time: Weakly scalable in training data size. Complex configuration.
- Not interpretable model: Application domain knowledge cannot be exploited in the model.

Bayesian Classification Is really different from the previous methods explained. Is based on the Bayes theorem:

$$\begin{gathered} P(C,X) = P(C|X)P(X) = P(X|C)P(C) \\ P(C|X) = \frac{P(X|C)P(C)}{P(X)} \end{gathered} \tag{6}$$

The approch is:

- Let the class attribute and all data attributes be random variables (C = Any Class, $X =< x_1, ..., x_k >$).
- Classification:
  - Compute $P(C|X)$ for all classes (Probability that record X belongs to C).
  - Assign X to the class with the ***maximal*** $P(C|X)$.

The computeation is achieved using the Bayes theorem, where P(C) is A Priori probability, without valculating the fraction (we not need the probability but only the highest value). The problem is how to calculate the $P(X|C)$, i.e $P(x_1, ..., x_k|C)$ and it was solved using a Naïve hypothesis: All attributes are

statistical indipendent, of course this is not always true and it can affect the model quality.
For example, strating from the data below:

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

after the computation of all possibilities below:

outlook	
P(sunny|p) = 2/9	P(sunny|n) = 3/5
P(overcast|p) = 4/9	P(overcast|n) = 0
P(rain|p) = 3/9	P(rain|n) = 2/5
temperature	
P(hot|p) = 2/9	P(hot|n) = 2/5
P(mild|p) = 4/9	P(mild|n) = 2/5
P(cool|p) = 3/9	P(cool|n) = 1/5
humidity	
P(high|p) = 3/9	P(high|n) = 4/5
P(normal|p) = 6/9	P(normal|n) = 2/5
windy	
P(true|p) = 3/9	P(true|n) = 3/5
P(false|p) = 6/9	P(false|n) = 2/5

P(p) = 9/14
P(n) = 5/14

When a new data ($X =< rain, hot, high, false >$) to be labeled is insert the flow will compute this value:

- For class p: $P(X|p)P(p) = 3/9 * 2/9 * 3/9 * 6/9 * 9/14 = 0.010582$
- For class n: $P(X|n)P(n) = 2/5 * 2/5 * 4/5 * 2/5 * 5/14 = \mathbf{0.018286}$

3.5 Clustering Fundamentals

What is Cluster Analysis? Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) objects in other groups.

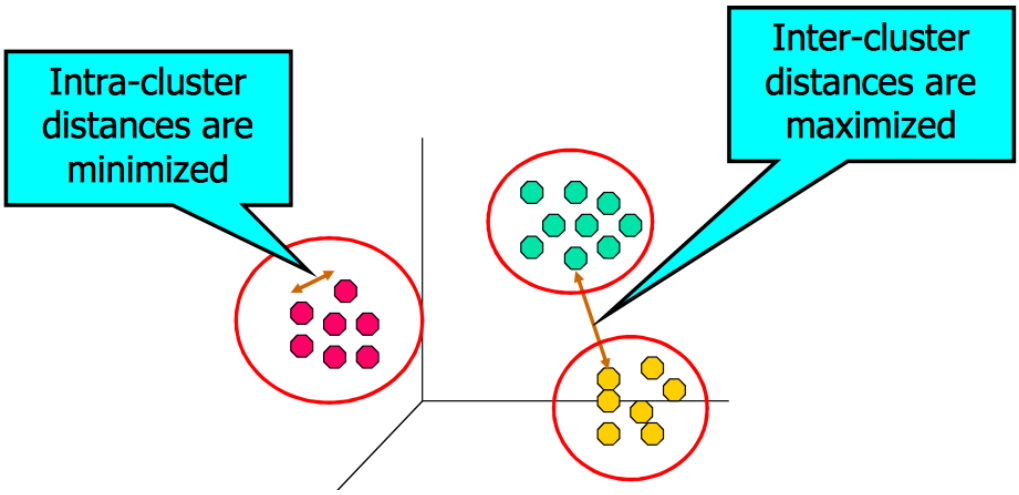


Figura 44: The clustering ideas

This is a data-mining technique not-supervisioned, it means that, respect the classification, it not require the class tag. The main applications of this analysis are:

- **Understanding**: Group related documents for browsing, group of genes or proteins with similar functionality, etc...
- **Summarization**: Reduce the size of large data sets.

The problem of clustering is related to its definition that can be ambiguous, the best cluster division is always related to the request of the problem, there isn't a always-best solution.

Type of cluster A clustering is a set of clusters, there are two main type of cluster:

- **Partitional**: A division data objects into non-overlapping subsets (cluster) such that data object is in exactly one subset.
- **Hierarchical**: A set of nested cluster organized as hierarchical tree.

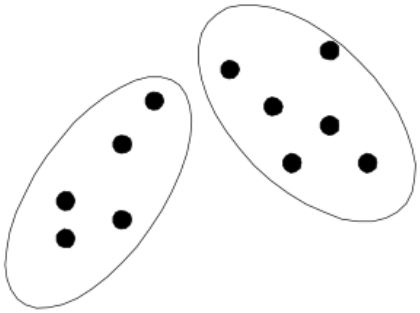


Figura 45: Partitional Cluster

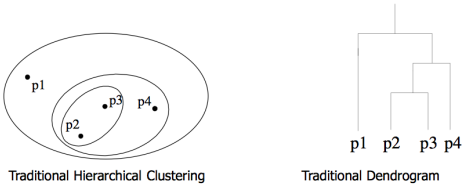


Figura 46: Hierarchical Cluster

Other distrinctions between sets of clusters are:

- Exclusive vs Non-exclusive: In eXclusive, point not belong to multiple clusters.
- Fuzzy vs non-fuzzy: In fuzzy, a point belongs to every cluster with some weight (0¡w¡1).
- Partial vs Complete: In some cases, we only want cluster some of the data.
- Heterogeneous vs homogeneous: Cluster of widely sizes, shapes and densisties.

The types can be also distrincted by its shapes:
In a **Well Separated** cluster the point are distribuited such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster (figure 47). **Center-based** means that an objects in a cluster is more similar to its center than to the center of any other cluster. The center can be found with the average of all point in the cluster (*centroid*, it may not be a record), or with the most "representative" point (*medoid*) of a cluster (figure 48). Another type is the **Contiguity-based** that collect

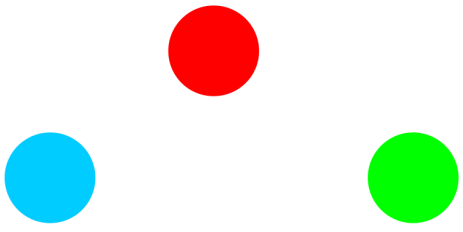


Figura 47: Well Separated

Figura 48: Center-based

closer point to one or more other points in the cluster than to any point not in the cluster, they can have strange shapes (figure 49). In the **Density-based**

the clusters are identified by high-density region of point, which are separated by low-densisty regions. The **Share Property** or conceptual clusters shares some common property or represent a particular concept (figure 50).

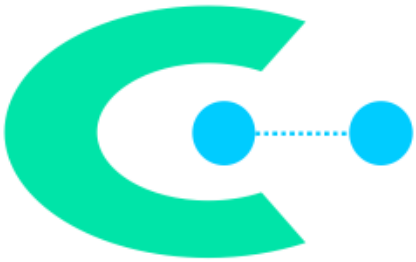


Figura 49: Contiguity

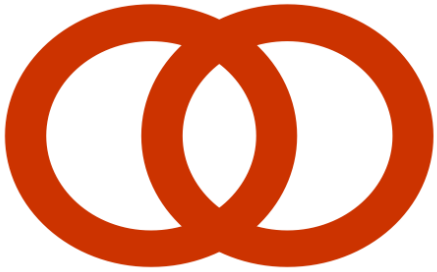


Figura 50: Share Property

Algorithms One of the most important is the **K-menas**. Used for partitional clusters, it associate each cluster with a centroid and each point is assigned to the cluster with the closest centroid. The number of cluster must be choose by the user. After the centroid choice is necessary to recompute the centroid. The algorithm is really simple.
The initial centroids are often chosen randomly, it means that the clusters produced vary from one run to another. The distance can be evaluated in different ways, Euclidean, cosine, similarity, etc... The k-means will converge in the first few iterations (figure 51). The complexity is O(n*K*I*d), where n = #ofPoints, K = #ofClusters, I = #ofIterations and d = #ofAttributes.

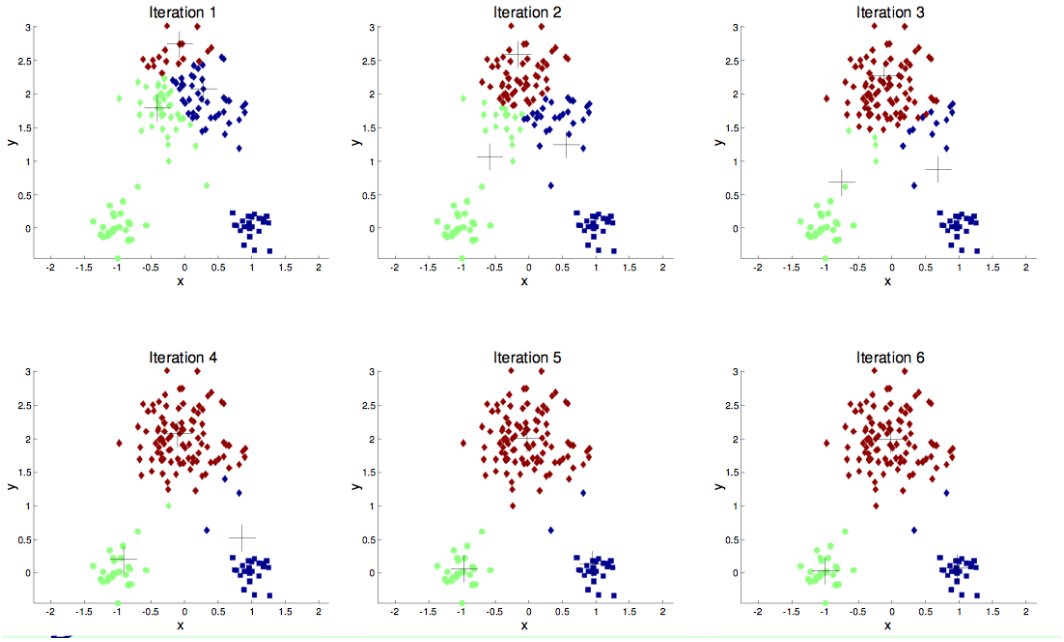


Figura 51: Convergence of K-means

In situations of 10-15 dimensions is impossible to evaluated the "goodness" of a solutions, this can be achieved using **SSE** or Sum of Squared Error. For each point, the error is the distance to the nearest cluster. The formula is:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2(m_i, x) \tag{7}$$

x is a data point and m_i is the representative point for cluster its cluster C_i. Given more cluster, we can choose the one with the smallest error. One way to reduce SSE is to increase K.
In some cases the basic k-means algorithm can yeild empty clusters, is important to handling it. Some solutions are:

- Choose the point that contributes most to SSE.
- Choose a point from the cluster with the highest SSE.
- If there are several empty clusters, the above can be repeated several times.

Is possible to increase the quality of the system with some pre-processing operation like:

- Normalization (Always need).
- Eliminate Outliers.

There also some post-processing operations:

- Eliminate small cluster that may represent outliers.
- Split 'loose' clusters (high SSE).
- Merge clusters that are 'close' and that have relatively low SSE.
- Can use these step during the clustering process.

The **bisecting K-means** is an alternative to the k-means, it applies the normal k-means by splitting a cluster, the worst one (high SSE), in 2 cluster. These step are iterated many times until the number of cluster is K. Normally this solution give better results than the normal k-means.
Both k-means solution has problem when clusters are of differing in size (52), densities (53) or non-globular shapes (54), also outliers can cause problems.
One solution is to use an high number of K and then merge it in a post-processing phase.

The **hierarchical Clustering** algorithms produces a set of nested clusters organized as a hierarchical tree. They can be visualized as a dendrogram (a tree like diagram that records the sequences of merges or splits). They can be created bottom-up or top-down. They don't have to assume any particular number of clusters. Two main types:

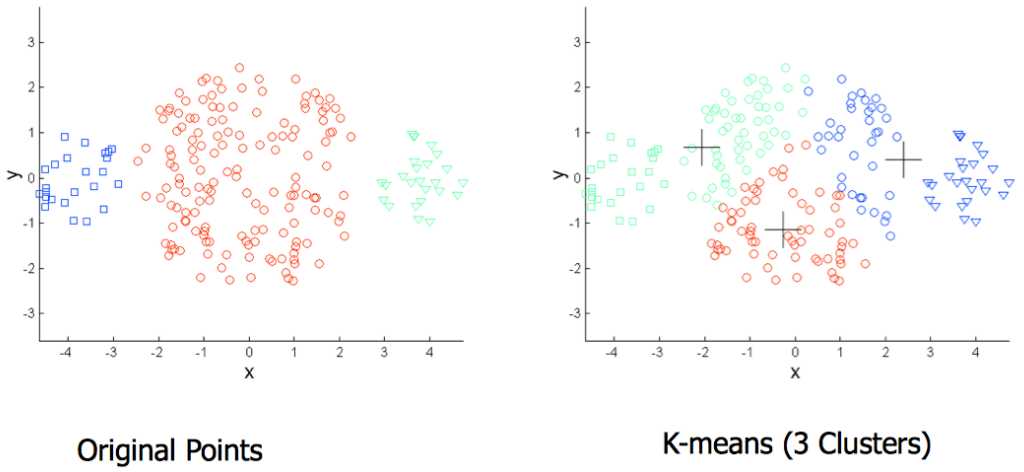


Figura 52: Size problem

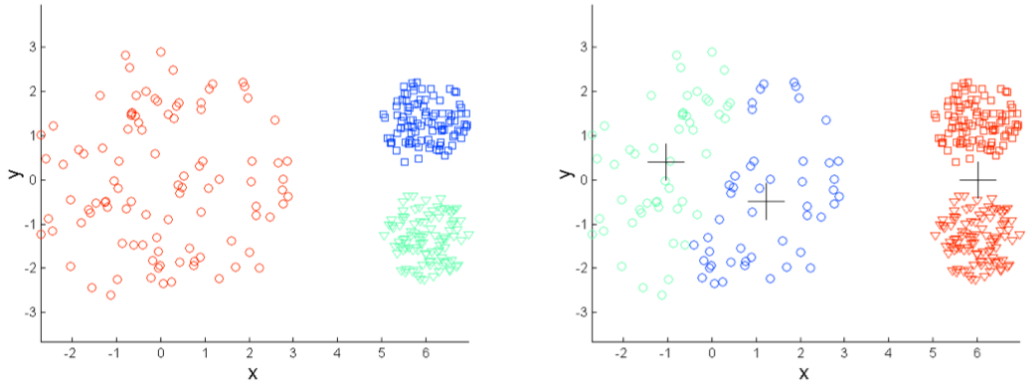


Figura 53: Density problem

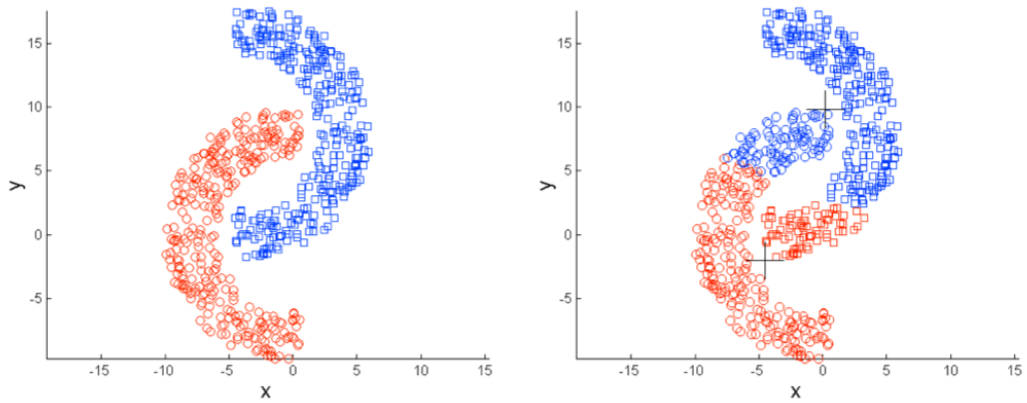


Figura 54: Shape problem

- Agglomerative: Points as individual clusters. Each step, merge the closest pair of clusters until only one cluster left.
- Divisive: Start with one, all-inclusive cluster. Each step, split a cluster until each cluster contains a point.

Since in the intermediate steps we will have only clusters, is necessary to have measurement for distance between points, already studied in classification section, and between clusters. All the computation is made with a proximity matrix.
There are 5 main measure, the first 3 are computing all couples combination between two cluster and than select the MIN, MAX or Average distance between them. Another possibilities is to compute the distance between centroids. All these possibilities are done to fill the proximity matrix.
The advantage of the **MIN** is that it can handle non-elliptical shapes, the limitations is that is sensitive to noise and outliers. Instead, the **MAX** is less susceptible to noise and outliers, but it tends to break large clusters (like K-means).
The most expensive step is the computation of the matrix, the space complexity is $O(N^2)$, where N is the number of points, the time complexity is $O(N^3)$.

Another algorithm is the **DBSCAN** that is density-based, it's good to check outliers and noise. The density is computed like the number of points eithin a specified radius ϵ. The points are classified, given ϵ and $minPts$, with three label:

- **Core**: If it has more than a $minPts$ within ϵ.
- **Border**: Fewer than $minPts$ in ϵ but near to a core point.
- **Noise**: Any point not a Border or a Core ones (added to the cluster 0).

An example in figure 55. The implementation is not too easy, the pseudo code is the following:

$current\_cluster\_label \leftarrow 1$
for all core points **do**
  if the core point has no cluster label **then**
    $current\_cluster\_label \leftarrow current\_cluster\_label + 1$
    Label the current core point with cluster label $current\_cluster\_label$
  end if
  for all points in the Eps-neighborhood, except i^{th} the point itself **do**
    if the point does not have a cluster label **then**
      Label the point with cluster label $current\_cluster\_label$
    end if
  end for
end for

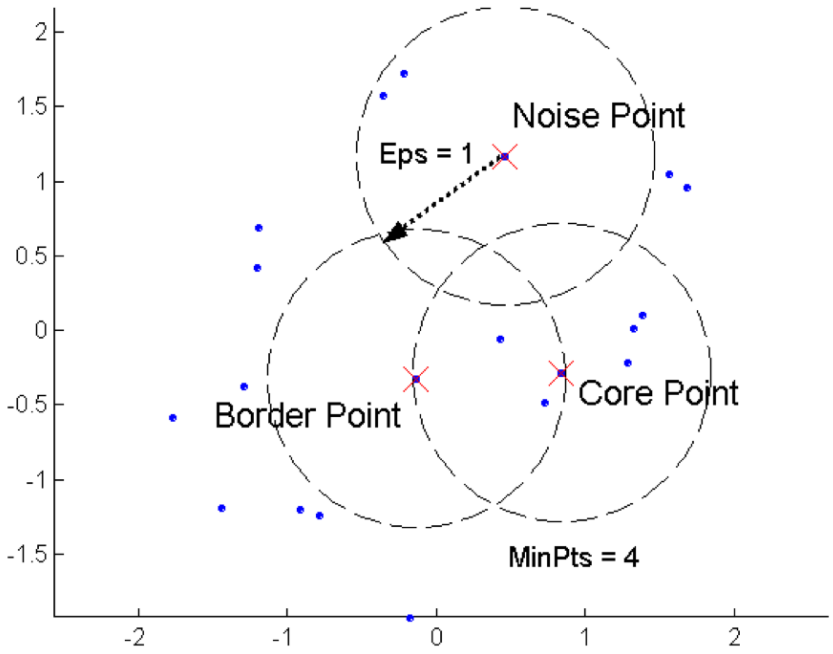


Figura 55: DBSCAN example

The strenght of the DBSCAN is that is really resistant to noise and it can hadle clusters of different shapes and sizes. It not works well in case of varying densisties and high-dimensional data. The first problem can be solved performing two iterations of the algorithm, in the first one you'll find the high-dense clusters, in the second one the less-dense.
How determing ϵ and minPts? The ideas is to plot a graph with sorted distance of every point to its k^{th} nearest neighbor. This because for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance, instead the noise points are farther. Normally the value of k is choose at the knee of the curve (around 10 in figure 56), in case of multiple knees it means that there are different density zones.

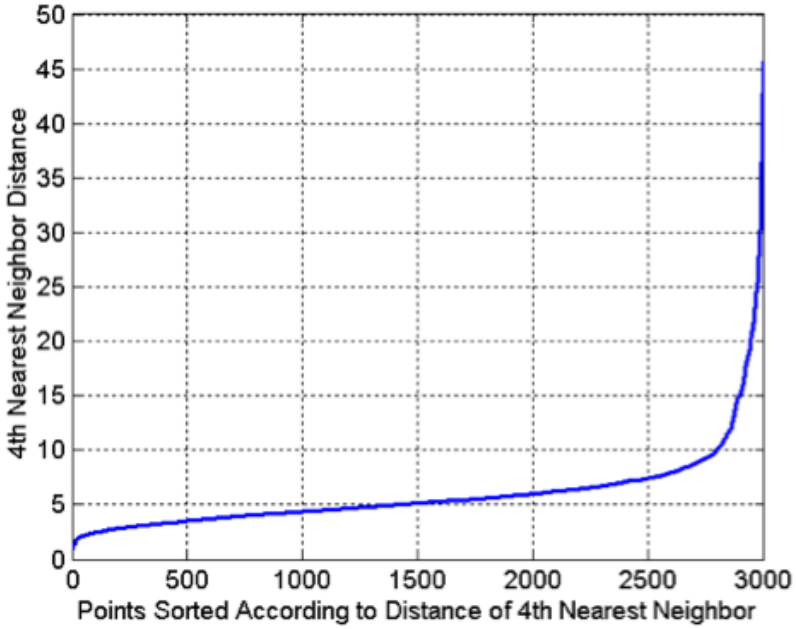


Figura 56: Example of graph for determining ϵ and *minPts*

Cluster Validity this is the most difficult tasks. Some numerical measure can be exploited to evaluate the "goodness" of the solution. They are classified in two main classes:

- External Index: Used to measure the extent to which cluster labels match externally supplied class labels, like a training sets (entropy, purity).
- Internal Index: Used to measure the goodness of a clustering structure without respect to external information (SSE, cohesion (how near), separation (how far), rand-index).

"The validation of clustering structures is the most difficult and frustrating part of cluster analysis.
Without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage."
— **Jain and Dubes**, Algorithms for Clustering Data

4 Beyond Relational Databases

4.1 Introduction

The first NoSQL databases was developed around 1998. The main difference are three:

- No joins: Complex and unuseful in some cases.
- Schema-less: No tables, implicit schema.
- Horizontal scalability: Adding machines not require to stop.

In this kind of systems the table are substituted with **specialized storage** solutions like document-based, key-value pairs, graph databases and columnar storage. Removing the scemas constraints allows better option for managing dynamic changing documents, semi-structured or un-structured data. The advantages of an Horizontal scalability is that sufficient to add a server to your pool in order to reduce the load. In the classic system the only solution is to increase the power of the system. The main drawbacks is that there isn't a standardized language for the managing, each system has is own language. The old complex joins are now solved with non standard interfaces. The SQL system are better for managing flat and structured data, the NoSQL are better for complex (e.g. Hierarchical) data like JSON and XML. Some example of NoSQL software are **MongoDB**, BigTable, Redis, Cassandra, HBase and CouchDB. In some cases NoSQL software developer, like the Mongo ones, have introduct some "more SQL" feature to its build to better fit some problem.

4.2 Structure

These new kind of system have also introduced new types of data structures, the most important are in figure 57.

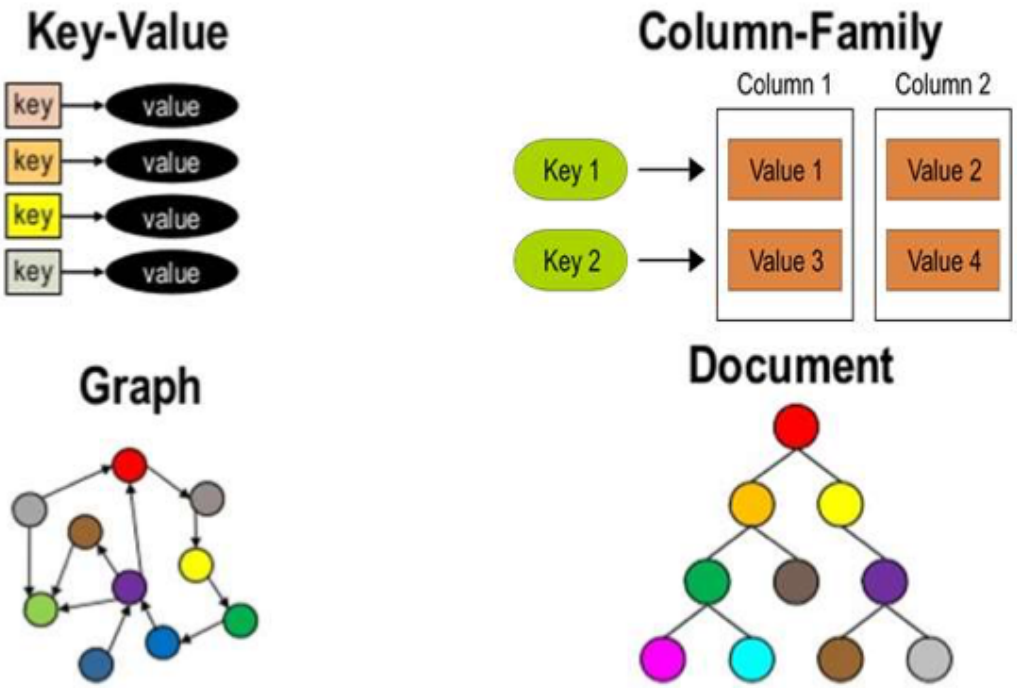


Figura 57: Types of NoSQL databases

The first type is the **key-value** is:

- **Simplest** data stores.
- Macth keys with value.
- No Structure.
- **Great performance**.
- Easy scalable.
- Very fast.
- Examples: Redis, Riak, Memcached (used to solve FB problem).

The second one is the **column-oriented** databases that are:

- Store data in columnar format.
- A column is a (complex) attribute.
- Key-value pairs stored and retrived on key in a parallel system (similar to indices).
- Rows can be constructed from cloumn values.

- Column stores can produce row output (**tables**).
- Completely transparent to application.
- Example: Cassandra, HBase, Hypertable, AMZ DynamoDB.

Another structure is the **graph**, the main feature are:

- Based on graph theory.
- Made up by **Vertex** and **Edges**.
- Used to store information about networks.
- Good fit for several real world application.
- Examples: Ne04J, Infinite Graph, OrientDB.

The last structure is the **document** ones, where document means complex, that is:

- Database stores and retives documents.
- Keys are mapped to documents.
- Documents are self-describing (**attribute=value**).
- Has hierarchical-tree nested data structures (e.g. Maps, lists, datetime, ...)
- **Heterogeneous** nature of documents.
- Examples: **MongoDB**, CouchDB, RavenDB.

4.3 NoSQL example: CouchDB

The CouchDB is good to be studied because is kept is soul entirely on the NoSQL side respect other software, is a Pure NoSQL database.
CouchDB is a document-oriented database can be queried and indexed in **MapReduce** fashion, it offers incremental **replication** with bi-directional conflict detection and resolution. Is written in Erlang, a robust functional programming language ideal for building concurrent **distributed systems**. Erlang allows for a flexible design that is easily scalable and readily extensible. Is also RESTful JSON API that allows access from any environment that permit HTTP request, it don't need a client.

MapReduce is a scalable distributed programming model to process Big Data. The first analisys over these model come from Google in 2004, they have try to change the point of view trying to passe the code to the node ehere the data is stored and not get all the data to the central code node. It consist in two function:

- **Map**:
  - Process each record (document) indipendently.
  - Return a list of key-value pairs.
- **Reduce** (optional):
  - Reduce the list of key-value returned by the map to a single value (it can be a complex value such as a map).

The map function is called one document by one and it get back a key, that index in a sorted way the result, and the value itself. Performing a function like this:

```
Function(doc) {
  emit(doc.exam, doc.mark);
}
```

The result take the value of the DB on the left and transform it to the table on the right of the figure 58.

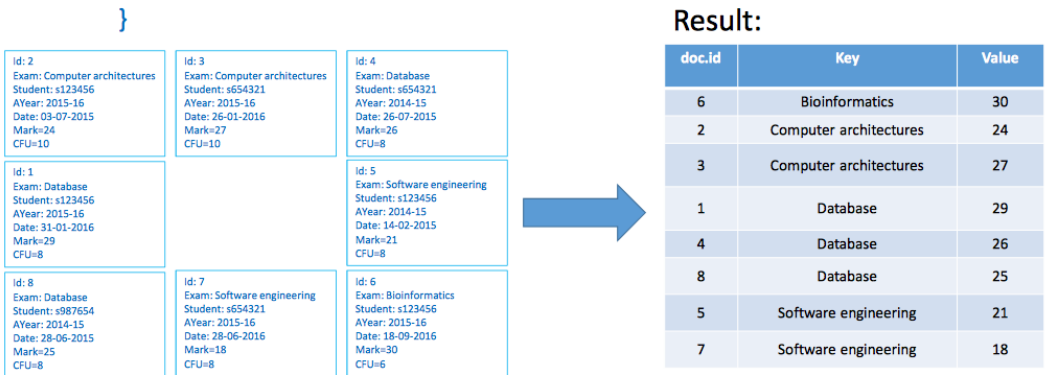


Figura 58: Map function example

The key can be more complex adding, for example the accademy year. Also the value can be complex adding the CFU for computing later an weighted average and so on.
The reduce function perform operations over the value returned by the map function. The most simple SQL-equivalent operations performed by means of reducers are "group by" aggregations, but with more more powerful functions to be implemented. It also possible to perform reduce over already reduced sets. For example, using this function:

```
Function(key, value) {
  S = sum(values);
  N = len(values);
  AVG = S/N;

  return AVG;
}
```

The output start from the left table and perform operations to return a structure like the right one of the figure 59.

Map

doc.id	Key	Value
6	Bioinformatics	30
2	Computer architectures	24
3	Computer architectures	27
1	Database	29
4	Database	26
8	Database	25
5	Software engineering	21
7	Software engineering	18

Reduce

Key	Value
Bioinformatics	30
Computer architectures	25.5
Database	26.67
Software engineering	19.5

Figura 59: Reduce function example

as you can see all the exams are grouped and there is no difference between the same exam created with the exam name like keys. If this operations is computed over a starting table with (ExamName, AccYear) like keys the result will be with exams grouped by name and accamedic year; the result is similar to the *GROUP BY Exam, AYear*. One of the most important feature of this system is the parallelization capabilities, in figure 60 an example of how the result will be computed with more nodes with data.

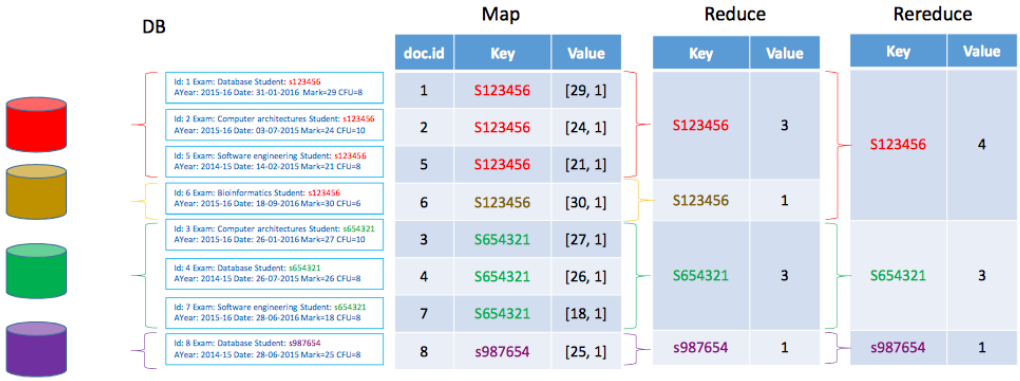


Figura 60: Parallel workflow example

4.4 Replication

The replication try to keep the same data, portions of it, in different places, like local and/or remote servers, clusters, data centers. The main goals are:

- Redundancy helps surviving failures (availability).
- Better performance.

To achieve this feature the are two possible solutions.

Master-Slave this first solution implement a *master* server that takes all the writes, updates and insert and some *slaves* servers that take all the reads (they can't write). This means only a read scalability and the master become a single point of failure. In the case of CouchDB also the Master-Master replications can be implemented.

Synchronous replication is a different solution, the master will waits the commit of all slaves, before commiting. This is similar to the 2-phase commit in RDB. It could be performance killer! A treade-oof could be wait only for a subset (majority) of slaves before commit.

Asynchronous replication have a master that commit all operations without waiting the slaves. Each slaves independently fetchs update from master, which may fail...

- If no slave has replicated, the you've lost the data commited to the master.
- If some slaves have replicated and some haven't, then you have to reconcile.

This is a faster but unreliable solution.

4.5 Distributed databases

Different autonomous machines, working together to manage the same dataset. There are 3 typical problems in distributed databases:

- **Consistency**: All distributed DB provide the same data to the application.
- **Availability**: Database failures (e.g. Master Node) do not prevent survivors from continuing to operate.
- **Partition** tolerance: The system continues to operate despite arbitrary message loss, when connectivity failures causes network partitions.

CAP Theorem also know as Brewer's theorem, states that it is impossible for a distributed system to simultaneously provide all three of the previous guarantees. It becomes a theorem in the 2002. A representation in figure In

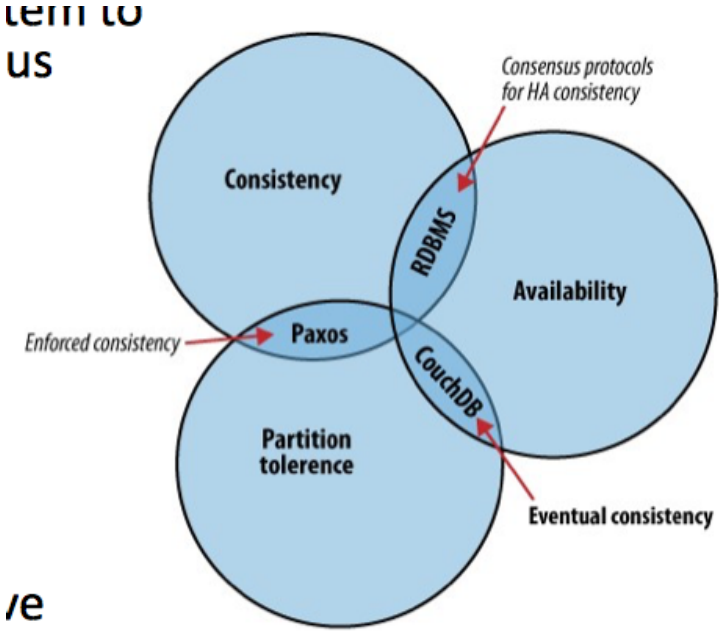


Figura 61: Cap theorem chart

general is not to difficult to be understand. Allowing at least one node to update state will cause the nodes to become inconsistent, thus forfeiting C. If the choice is to preserve consistency, one side of the partition must act as if it is unavailable, thus forfeiting A. Only when no network partition exist, is it possible to preserve both consistency and availability, thereby forfeiting P.
The general delief is that for wide-area system, deesigners cannot forfeit P and therefore have a difficult choice between C and A. This is because is impossible to have a distributed systems without allowing network partition in case of

failures, therefore is not still a distributed system.
The **CP** solutions guarantee the consistency of the data, it similar to the 2-phase locking where an agreement is needed to commit the operations. This is means that everything is been blocked since all response have comed back. For example, guaranteeing C allows to show the correct availability for a product to all the Amazon's costumers. It can be really expensive because is better to manage situation where more people have bought only one item, rather than not sell it. In this case **AP** (a.k.a best effort) become a good solution, this sacrifice the consistency but it can guarantee a good service, even though the system as a shole has benn broken up into incommunicable regions (partitions). This situations not means that the global consistency is always broken, it only means that is **can't be guarantee all the time**.

"Each node in a system should be able to make decisions purely based on local state. If you need to do something under high load with failures occurring and you need to reach agreement, you're lost. If you're concerned about scalability, any algorithm that forces you to run agreement will eventually become your bottleneck. Take that as a given."

— **Werner Vogels**, Amazon CTO and Vice President

In general the the "2 of 3" view is misleading on several fronts. First, because partitions are rare, there is a little reason to forfeit C or A when the system is not partitioned. Second, the choice between C and A can occur many times within the same system at very fine granularity; not only can subsystem make different choices, but the choice can change according to the operation or even the specific data or user involved. Finally, all three properties are more continuous that binary. Availability is obviously continous from 0 to 100 percent, but there are also many levels of consistency, and even partitions have nuances.

BASE this kind of systems prefer to use a different properties respect the ACID one. The BASE consists in:

- **Basically Available**: The system provides availability, in terms of the CAP theorem.
- **Soft state**: Indicates that the state of the system may change over time, even without input, because of the eventual consistency model.
- **Eventual consustency**: Indicates that the system will become consistenct over time, given that the system doesn't receive input during that time.

4.6 Conflict resolution

Supposing that there are two customers, A and B. A books a hotel room, the last available. B does the same, on a different node of the system, which was not consistent. The hotel room document is affected by two conflicting updates. The applications should solve the conflict with custom logic (it's a business decision). The DB can:

- Detect the conflict.
- Provide a local solution, e.g., latest version is saved as the winning version.

In CouchDB is guarantee that each instance that sees the same conflict comes up with **same winning** and losing revisions. It does so by running a deterministic algorithm to pick the winner:

- The revision with the longest revision history list becomes the winning revision.
- If they are the same, the _rev cvalues are compared is ASCII sort order, and the highest wins.

4.7 HTTP RESTful API

This are an important characteristics of this systems. It's really easy to get a document, you only need to use your browser and write its url, e.g., *http://localhost:5984/test/some_doc_id*. And so on. All operations have a proper procedure to be used, but this is beyond the pourpose of this course.

4.8 Conclusions

If you're building an app today, then there might be a need for using two or more databases at the same time, this is because there isn't a solutions that fits everything you need, some times using a more structured ways is preferred, some times is not usable. In general the choice depends on your needs, and it's probably that not just one DB can solve your problem.

5 Big Data

5.1 Introduction

Big Data is about Seach, lot of bytes, storage and analytics. Big data not makes magic and it could give bad result if not performed well. Where get all this data? Of course from the web, mobile, social, sensors from healt to scientific, log files, IoT, and so on...
One of the most common definition of BD is: "*Data whose scale, diversity and complexity require new architectures, techniques, algorithms and analytics to manage it and extract value and hidden knowledge from it*".

BigData Vs There are some know Vs regarding BD, they are:

- Volume: Scale of data
- Variety: Different forms of data.
- Velocity: Analysis of streaming data.

Other 2 important think were added during times:

- Veracity: Uncertainty of data.
- Value: Exploit information provided by data.

The volume of our data growth really really faster, we speaking now of thousant of terabytes normally. They increase in exponential way, the forecast say that in 2020 will reach decines of petabytes. The variety of this kind of data is impressive, from numerical to image, audio, text, video and so on... A single application may generate many different formats. The data could be fast as you can't store it, like the stream, you look at it only in a short time window, for reacting of for logging it in a different way, and just it has generated, because storing it would be crazy and unuseful. Speaking about data quality become really important, since this data is logged automatically is important to detect fault, e.g crazy temperature sensor, or other stuff that can ruin all your data. More data you have, more important become the quality. The last important V is the value, in some cases the analysis of big data can be translated to business advantage and this is really interesting.

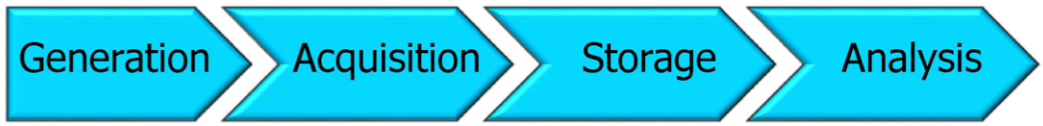
Challenges There are a lot of new challenges related to the BD usage:

- Technology & Infrastructure: A lot of new architecture, programming paradigms and techniques are needed. *Transfer the processing power to the data* (Hadoop ecosystem).
- Data management & Analysis: New emphasys on "data" $\rightarrow$ Data Science.

5.2 Data Science

is: "*Extracting meaning from very large quantities of data*" is an abused word nowadays because the big is not really defined and every kind of analysis over data seems data science. DS collects a lot of different stuff, statistics, machine learning, visualization, pattern recognition, etc... The data science process is based on the KDD already mentioned in 3.1 paragraph.

Value chain the process to the data science is chematized in figure below:



The step are:

- **Generation**:
  - Passive Recording: Typically structured data, like bank transaction, shopping records, government, etc...
  - Active Generation: Semistructured or unstructured data, like User generated content...
  - Automatic Production: Location-aware, context-dependent, highly mobile data, sensor-based internet-enabled devices, etc...
- **Acquisition**:
  - Collection: Pull or push.
  - Transmission: transfert with high capacity link.
  - Processing: Integration, cleaning and redundancy elimination.
- **Storage**:
  - Storage Infrastructure: HDD, SSD, DAS, NAS, SAN, etc...
  - Data Management: File System (HDFS), key-value stores (memchached), column-oriented, document-based, etc...
  - Programming Models: Map reduce, stream processing and graph processing.
- **Analysis**:
  - Objectives: Descriptive, predictive or prescriptive analytics.
  - Methods: Satistical analysis, data mining, network, clustering, etc...

Difference traditional systems use database and datawarehouse with well-defined strucuture small enough to be fitted to the machines. In big data the dataset is not suitable for database and it's why new solution like HDFS has been developed, may need near real-time analysis that is a different approch from data warehousing. There are also a lot of different programming paradigms.
Also the processing phase is changed, the traditional computation is processor bound and required a data trasfert from disks to CPUs to be processed this could be a problem because during these years the storage capacity has grown a lot, but the speed not the same...
The solutions is to "***transfert the processing power to the data***" it means that the various CPUs will compute only the subset of data that they have near in order to avoid data transfert. The problem is that this solution is not problem free, it must manage, hardware failure, network data transfert, data loss, etc... Everything is managed by Hadoop in figure 5.2. The principles

ideas are:

- Distribute data across nodes automatically.
- Processing executed on local data.
- No need of data transfert to start.
- Auto replication.
- Developer need only to focus over the logic.