



**POLITECNICO
DI TORINO**

Recap Database Management System (01NVVOV)

Jacopo Nasi
Computer Engineer
Politecnico di Torino

I Period - 2017/2018

17 ottobre 2017

Indice

1	Database Management System	4
1.1	Introduction	4
1.2	Buffer Manager	5
1.3	Physical Access	7
1.4	Query Optimization	10

License

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

You are free:

- **to Share:** to copy, distribute and transmit the work
- **to Remix:** to adapt the work

Under the following conditions:

- **Attribution:** you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work)
- **Noncommercial:** you may not use this work for commercial purposes.
- **Share Alike:** if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

More information on the Creative Commons website (<http://creativecommons.org>).



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

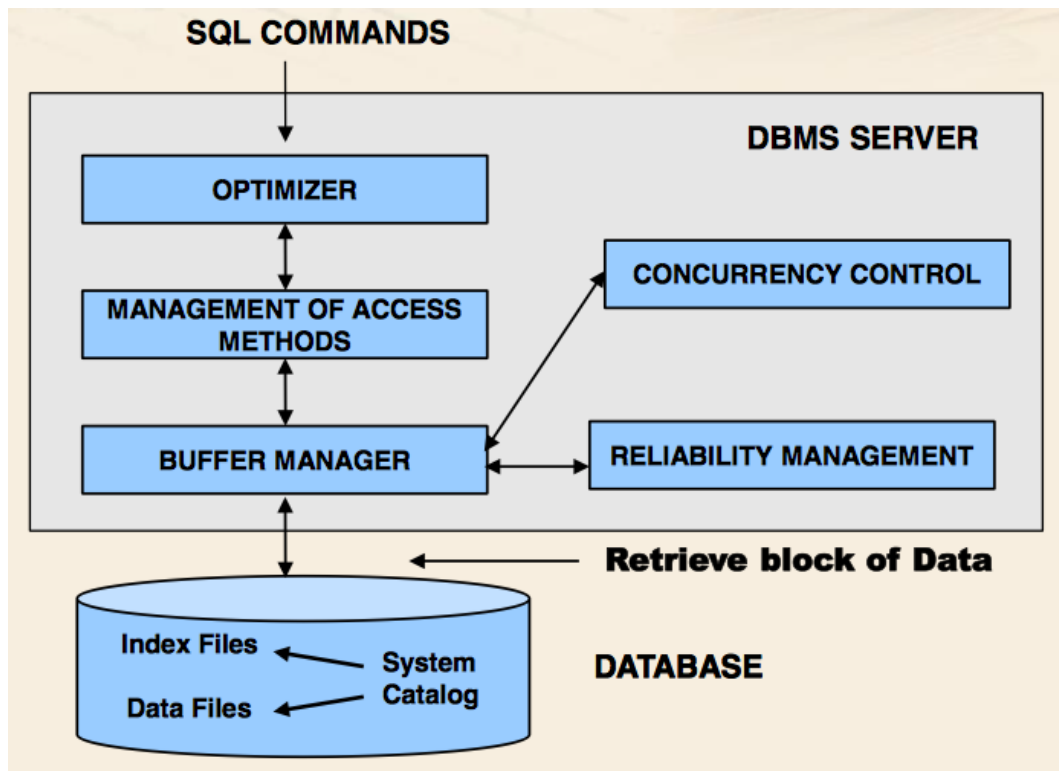


Figura 1: DBMS Architecture

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:

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity means that they cannot be divided in smaller units, is not possible 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 committed 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 transfer 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 the management strategies:

1. Data Locality: Data referenced recently 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 stores, for every page, the physical location of the page on the secondary memory (file identifier and block number) and two state variables, one that counts the number of transactions 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 if the page is not already in the buffer. There are two behaviours:

- Page already in buffer: Return the pointer to the data.
- Page not in buffer: It searches 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 decreases the Count.

Set Dirty Primitive it 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 transfer 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.

There 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 efficiency. 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, starting 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 is 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 by 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:

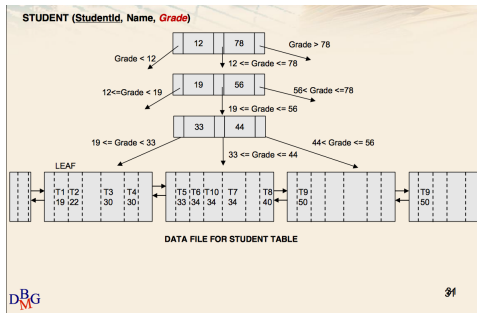


Figura 2: Clustered

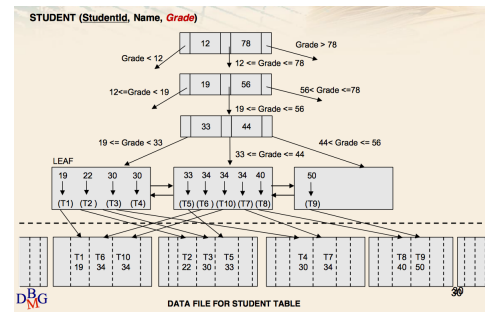


Figura 3: Unclustered

- **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]

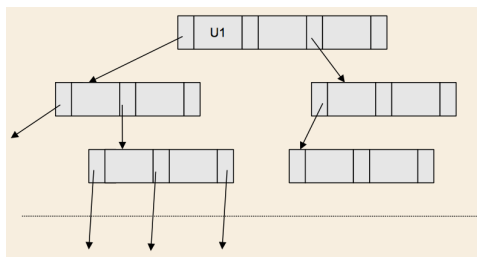


Figura 4: B-Tree

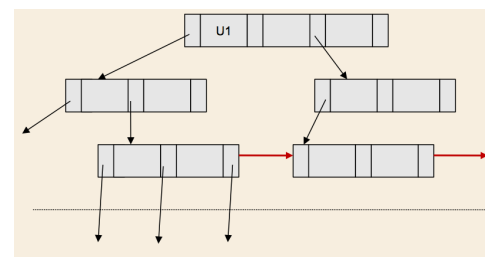


Figura 5: B^+ -Tree

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

- **Advantages:**
 - Very efficient 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 well-know structure is guarantees direct and efficient 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 completely fill the blocks to allow new data insertion.

- **Advantages:**

- Very efficient for queries with equality predicate on the key.
- No sorting of disk blocks is required.

- **Disadvantage:**

- Inefficient for range queries.
- Collision may occur.

The unclustered version 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 has j like attributes for the key field, 0 otherwise. the main characteristics are:

- **Advantages:**

- Very efficient 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 guarantee the data independence property, in fact, the form in which the SQL query is written does not affect the way in which it is

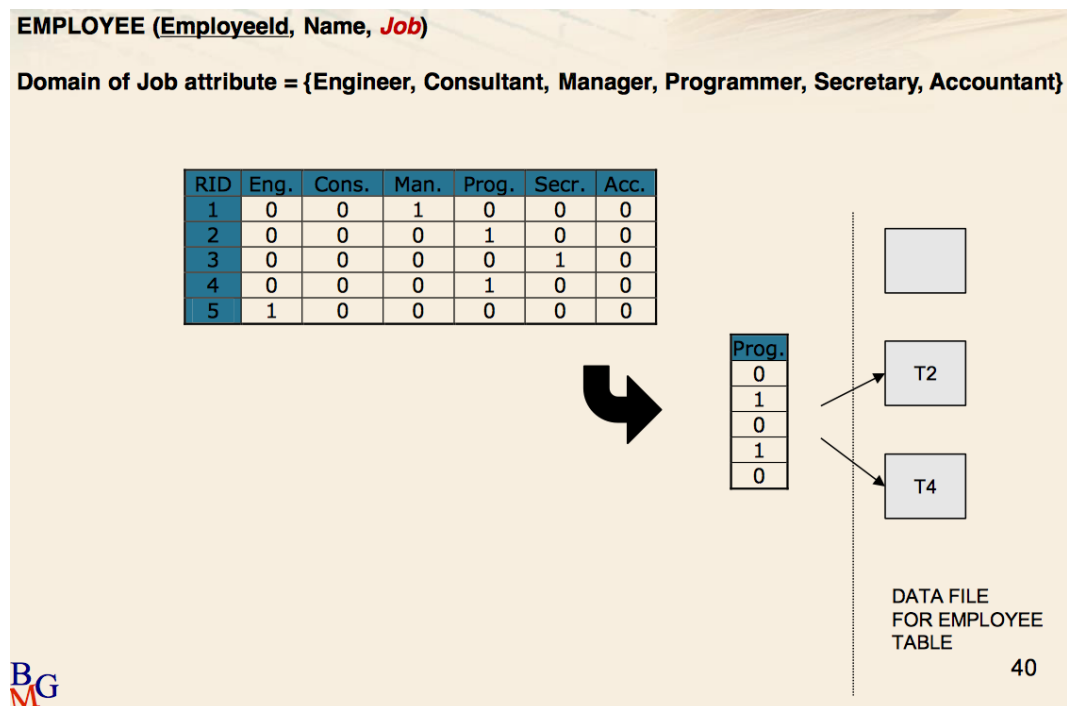


Figura 6: Bitmap Index

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, Syntactic and Semantic analysis : check the SQL for Lexical errors (e.g., misspelled keywords), Syntactical errors in the SQL grammar and for Semantic errors not existing object called in the query (require data dictionary). The output of this block is an internal representation 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 independent 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.

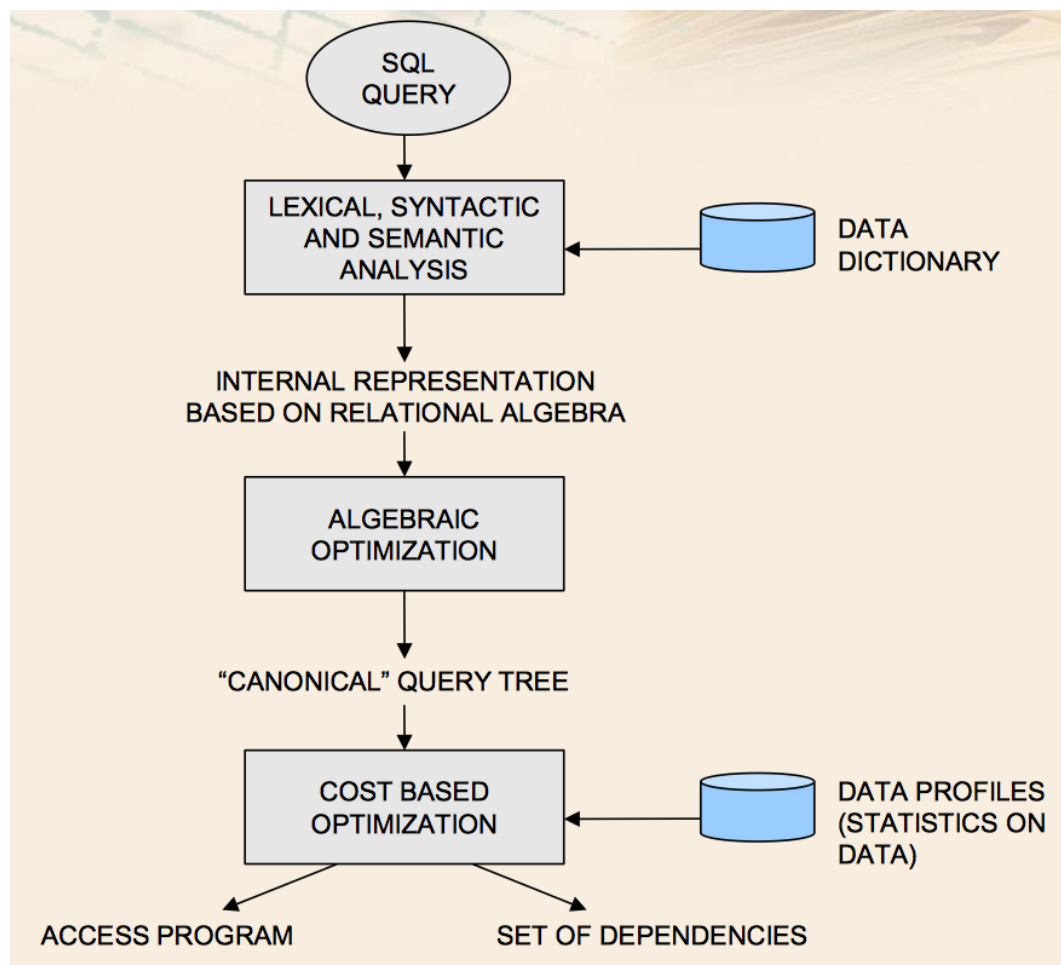


Figura 7: Optimization Execution Plan

The phase of **Algebraic Optimization** require a little more of analysis. The part is based on equivalence transformations, *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-known transformations:

1. **Atomization of selection:** Applying all attributes of selection one at the time or all together can provide different performance in case of indices.
2. **Cascading Projections:** It is possible to perform directly the final projection or 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 sub-set 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:** $\sigma_{F_1 \vee F_2}(E) = (\sigma_{F_1}(E)) \cup (\sigma_{F_2}(E))$
9. **Other:** $\sigma_{F_1 \wedge F_2}(E) = (\sigma_{F_1}(E)) \cap (\sigma_{F_2}(E))$
10. **Distributing join respect union:** Perform join of a table E with two merged table $(E_1 \cup E_2)$, is equivalent to join E with E1 and E2 separate and then 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 quantitative information on the characteristics of tables and columns:

- Cardinality of tuples.
- Size in bytes of tuples.
- Size in bytes of each attributes.
- Number of distinct 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 efficient 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 remaining 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 FORCE)
 - Efficient when the inner table is small and fits in memory or when the join attribute in the inner table is indexed.
 - Not cost symmetric. 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. Efficient for large and already-sorted tables.
 - Requires sorting both table (already sorted or through clustered index).
- **Hash Join:** It apply the same hash function to the join attribute of both table. Tuples to be joined will fall into 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.