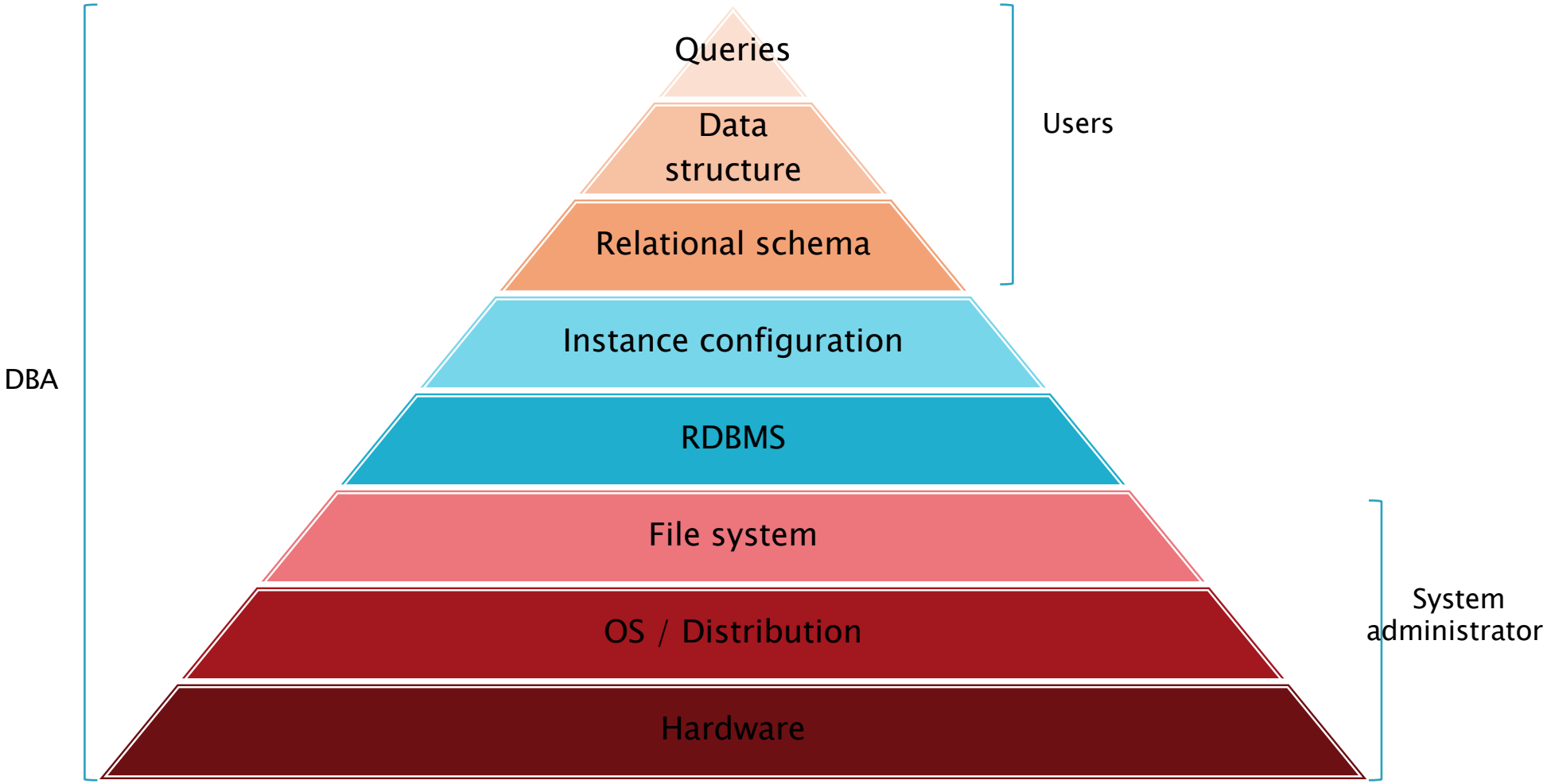
TUNING

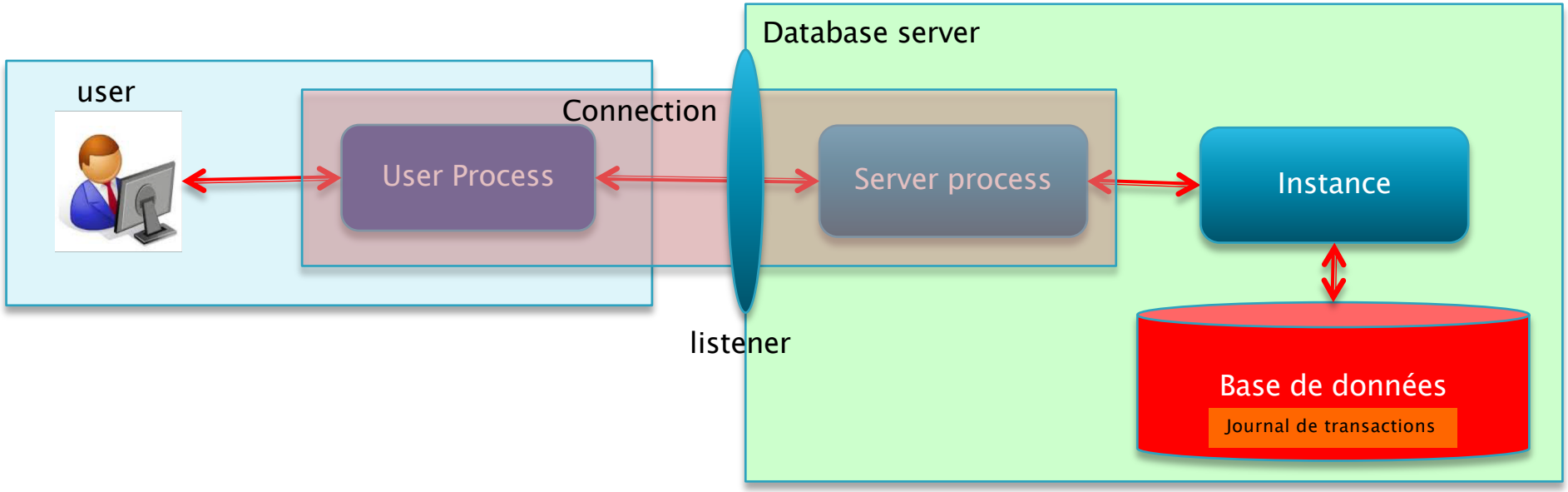
CCIN2P3

- In this section, we'll cover the following points :
  - Understand what happens in a RDBMS when you run a query
  - Describe how the rdbms compiles your SQL
  - Introducing the optimizer
  - Influencing optimizer choices

Introduction

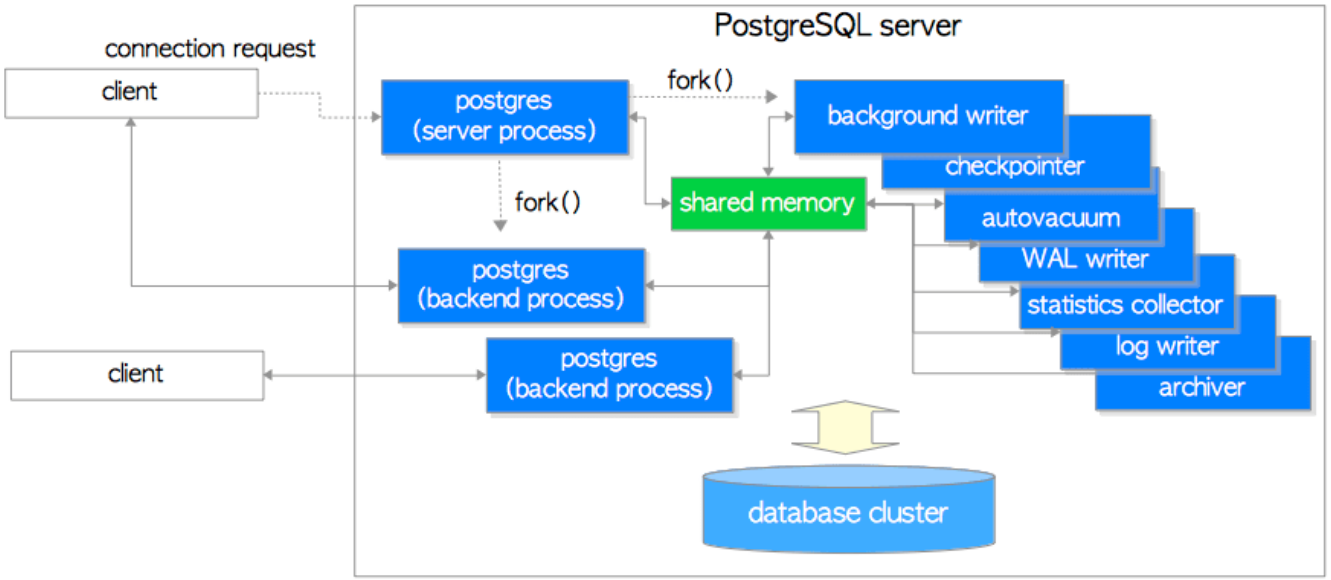


Internal mechanism

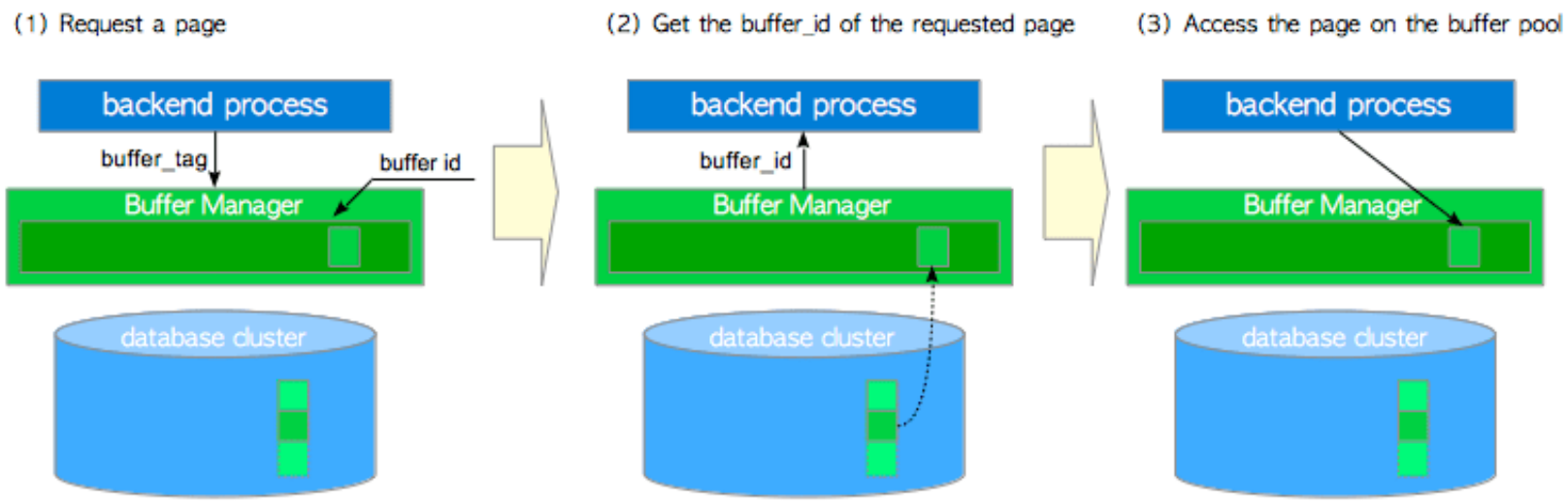


Internal mechanism

- The instance consists of memory structures and background processes. Its existence is transient, in your RAM and on your CPU.
- The memory plays an important role in performance like human-beings.
- To provide best performances, RDBMS split the memory into different pools for different use.



Internal mechanism



- (1) When reading a table or index page, a backend process sends a request.
- (2) The buffer manager returns the buffer_ID of the slot that stores the requested page. If the requested page is not stored in the buffer pool, the buffer manager loads the page from persistent storage to one of the buffer pool slots and then returns the buffer_ID's slot.
- (3) The backend process accesses the buffer_ID's slot (to read the desired page).
- When all buffer pool slots are occupied in the buffer manager, many dbms uses a Least Recently Used algorithm to free up space for new pages.
- The bigger your data cache is, faster the data access is.

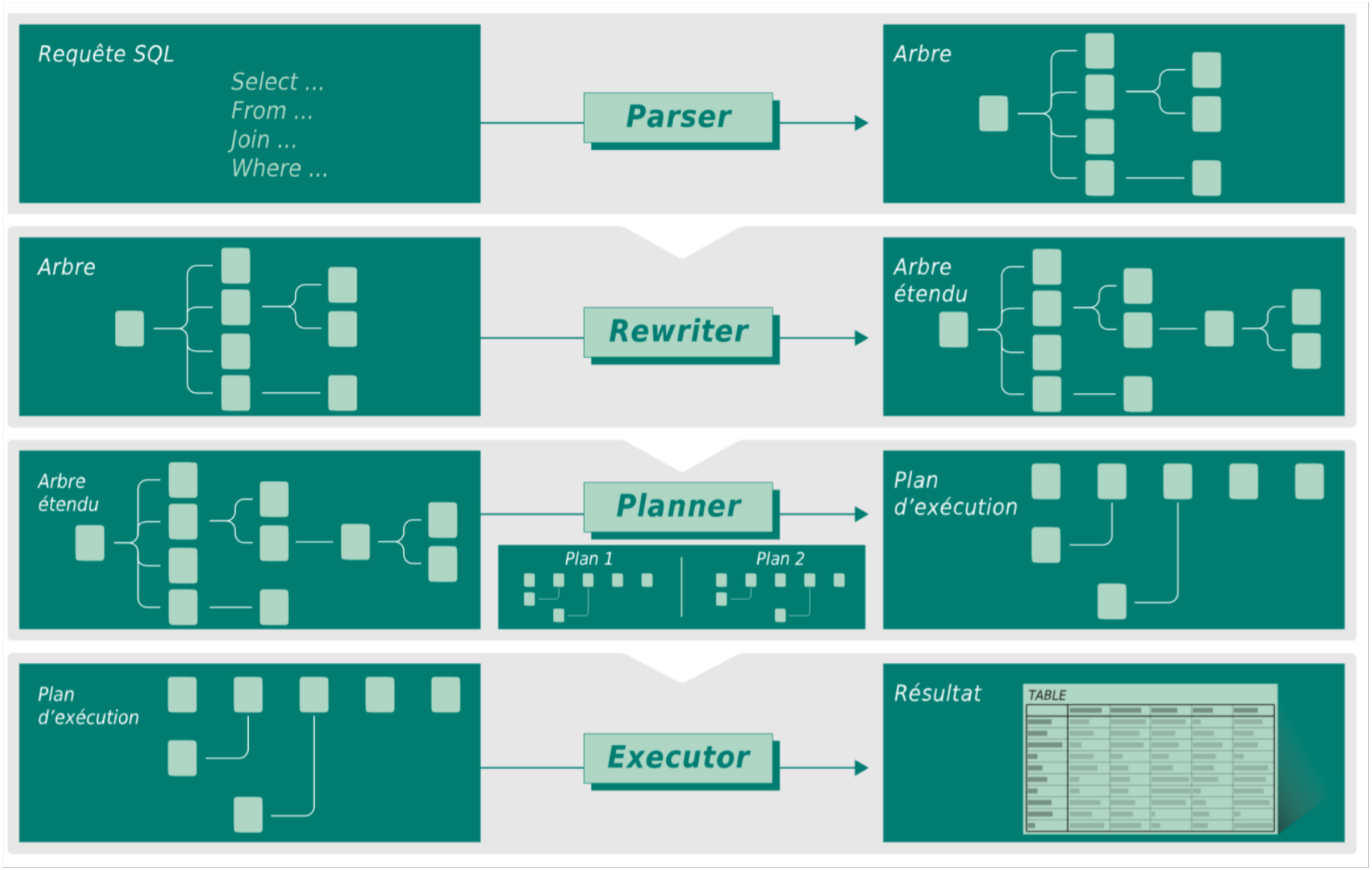
Internal mechanism

- Note that RDBMS use a buffer for storing SQL execution plan or PL/SQL programs.

- All queries and PL/SQL programs are kept in memory in order to be ready for immediate execution.

- Running a statement is not only a compilation operation, RDBMS need to process it through many phases:
  - Parser
  - Rewriter
  - Planner
  - Executor

Internal mechanism



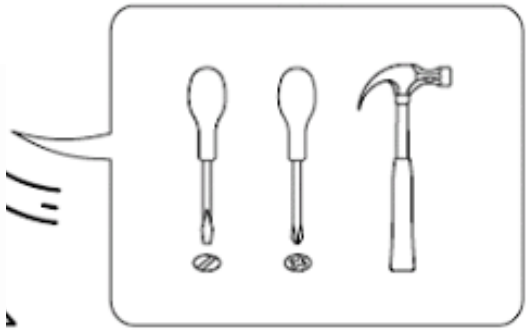
Internal mechanism

- Parser
  - Checks the statement for syntactic and semantic validity
  - Determines whether the process issuing the statement has privileges to run it
- Rewriter
  - Rewrite if required the SQL statement
  - For example, replace the view definition and apply rules.
- For some queries, SQL processing may be longer than the query execution.
- Keeping execution plan allows to save time

Internal mechanism

- Planner/Optimizer
  - It generates a set of execution plan and computes its cost for all of them.
  - Select the « cheapest » execution plan
- Executor
  - It runs the execution plan and request locks.
- The executor step is very CPU intensive. In Oracle, there is a mechanism of bind variables which allows to share the same execution plan to similar statements have the potential of being shared.
- Note that DDL (CREATE, DROP …) statements are only parsed and executed.
- Usually, running a SQL statement spend much time in planner and executor phases.

Role of the Optimizer



Role of the Optimizer

- The query optimizer determines the most efficient execution plan and is the most important step in the processing of any SQL statement.
- The execution plan is a set of operations that are performed in sequence to execute the statement.
- The optimizer considers many factors related to the objects referenced and the conditions specified in the query.
  - Statistics gathered for the system (I/O, CPU, and so on) as well as schema objects (number of rows, index, and so on)
  - WHERE clause qualifiers
  - Hints supplied by the developer
- The optimizer estimates execution plans from three types of measures:
  - Selectivity of a predicate
  - Cardinality
  - Cost

Role of the Optimizer : Selectivity

- Selectivity of a predicat
  - A predicat is the condition after the WHERE clause.
    - last_name = 'Smith' AND salary > 2000
  - It indicates how many rows from a table will satisfy the predicat.
    - Selectivity = # rows satisfying the predicat / Total # of rows
  - Selectivity lies in a value range from 0.0 to 1.0
  - **Selectivity = number of rows returned / total number of rows**

- Example :
  - A table basics contains 180 rows
  - What is the selectivity of the following statement :
    - Select * from emp;
    - Selectivity = 180/180 = 1
  - What is the selectivity of the following statement :
    - Select * from employee where id_dpt =‘COMPUTING’; // 10 employees
    - Selectivity = 10/180 = 0,05

- Cardinality
  - Cardinality is the number of rows returned or accessed by an operation
  - **Cardinality = Selectivity * Total Number Of Rows**
  - For an equality predicate

- Example : from the previous one
  - A table basics contains 180 rows
  - What is the cardinality of the following statement :
    - Select * from emp;
    - Selectivity = 180/180 = 1
    - Cardinality = 1 * 180 = 180 // result will return 180 rows
  - What is the selectivity of the following statement :
    - Select * from employee where id_dpt ='COMPUTING'; // 10 employees
    - Selectivity = 10/180 = 0,05
    - Cardinality = 0,05 * 180 = 10

Role of the Optimizer : Cost

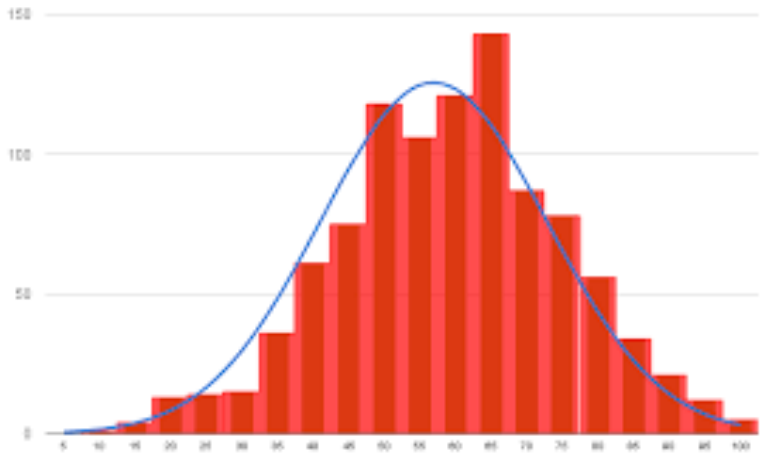
- Cost
  - Represents the number of units of resource that are used.
  - The query optimizer uses disk I/O, CPU usage, and memory usage as units of work.
  - The cost of a query plan is the number of work units that are expected to be incurred when the query is executed and its result is produced.

- Work units :
  - In Postgres, they are session variables :
    - seq_page_cost estimates the cost of a disk page fetch from a sequentiel access. (default 1)
    - random_page_cost estimates the cost of a non-sequentially-fetched disk page. (default 4)
    - cpu_operator_cost estimates the cost of processing each operator or function executed during a query
    - …
  - In Oracle, it is automatically set up from hardware performances.

- The optimizer estimates cardinality depending on statistics.

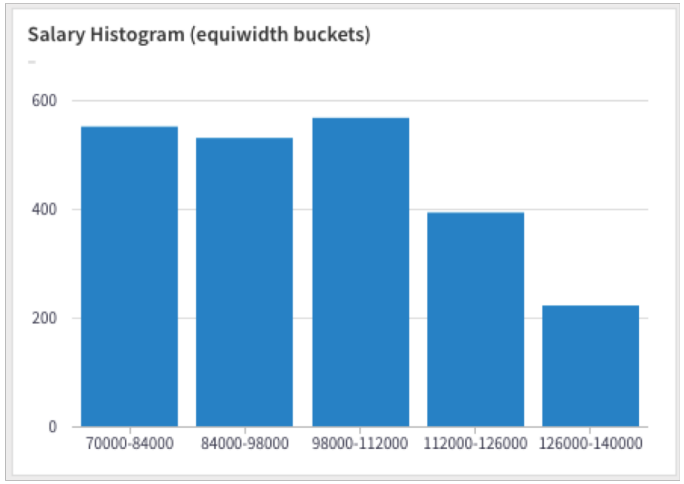
- Distinct values
  - For a single equality predicate, the optimizer assumes a uniform distribution and calculates the cardinality for the query by dividing the total number of rows in the table by the number of distinct values.

- The distribution of values in the column
  - The most common values in the column (< 250)
  - The frequencies of the most common values
    - number of occurrences of each divided by total number of rows.



- Histogram
  - A histogram is a special type of column statistic that sorts values into *buckets*

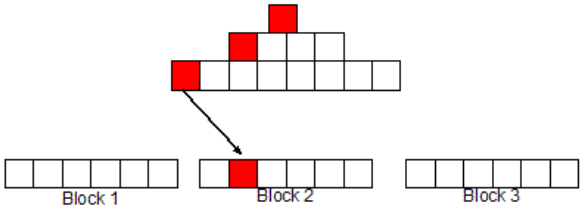
- With equal bucket widths
  - Well adapted for uniform data
  - **Bucket width** = [max(col value) - min (col value)]/ number of bucket)]
  - Create a histogram for a column salary :
    - five buckets
    - Range of value : $70,000 - $140,000
    - Bucket has a width of $14,000
    - Count the number row per bucket



Clustering Factor

- The clustering factor is a measure between the physical order and the logical order for column values.
- Range [0;1]

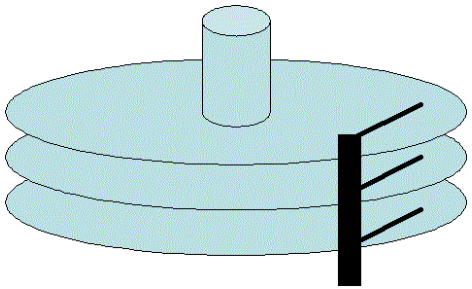
Recommended : Sequential scan



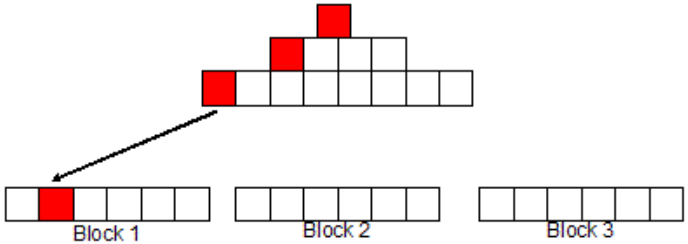
correlation = 0
=> Sequential scan

Un-Clustered table rows

Clustering_factor ~= num_rows



Super-large disks can be problematic



Correlation = 1
=> Indexed scan

Select order_nbr, item_name from ordor natural join item;

Clustered table rows

Clustering_factor ~= blocks

Statics

- Data may constantly change, statistics must be regularly gathered so that they are/keep relevant.
- Collecting statistics is not free. The system needs CPU and memory for computing statistics.
- More the sampling size is high and more your statitics should be accurate.
- Some RDBMS provide some tools for extending statistics :
  - For example, all statistics on postgres are on a single column but it is possible to extend the default statistics with the command CREATE STATISTICS

Access path

- An access path is the way used by a query to retrieve rows from the database.
- The access path takes into account the data structures that are used in the database.
- The optimizer then generates a set of possible execution plans using available access paths and estimates the cost of each plan, using the statistics for the index, columns, and tables accessible to the statement.
- The access path determines the way your data will be physically scanned.

Scan method

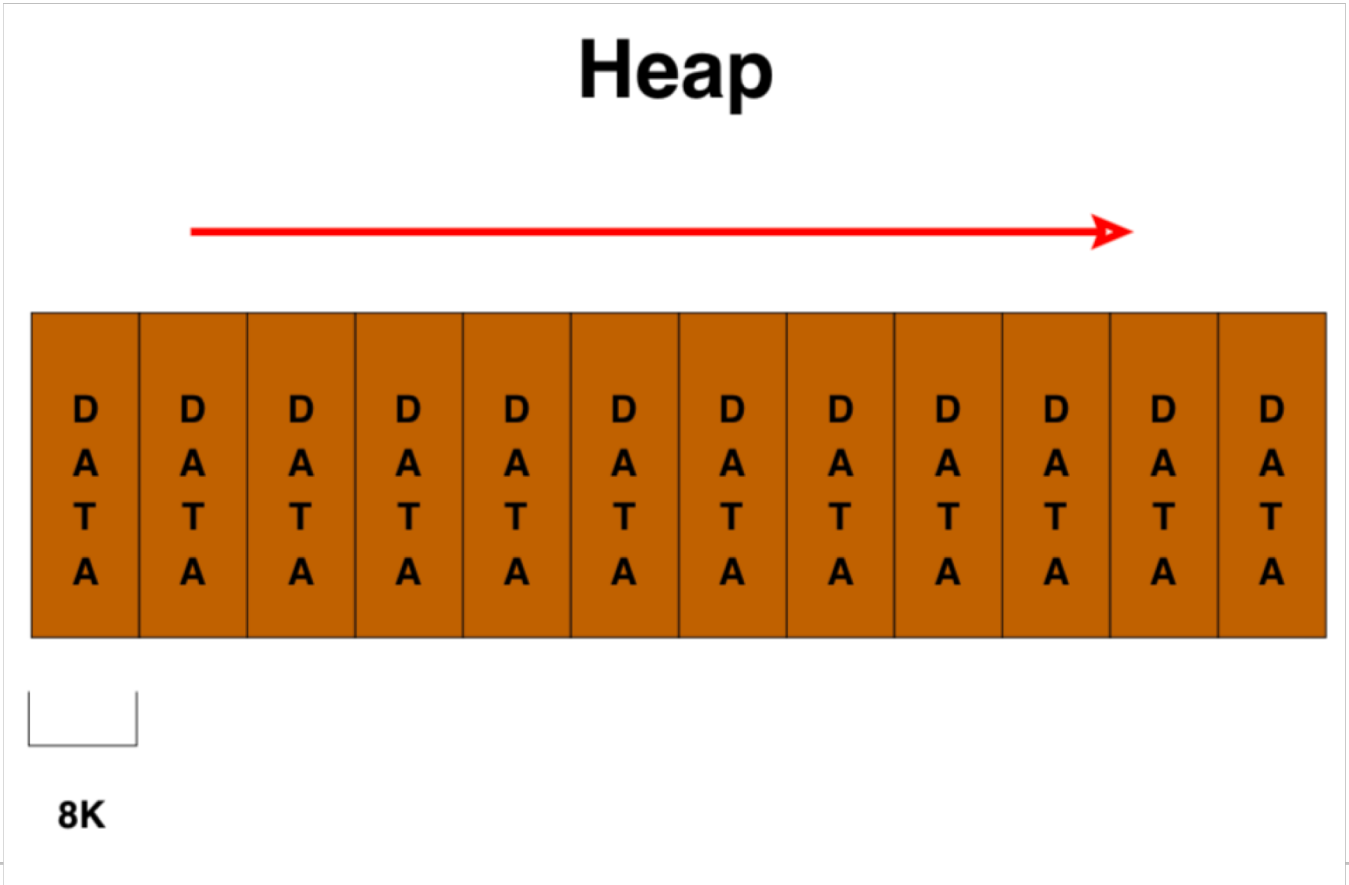
- Depending on RDBMS, the list of scan algorithm may be different.

- The most common methods are :
  - Sequential scan
  - Btree index scan
  - Index only scan
  - Bitmap index scan

- For joins
  - Nested Loop
  - Hash join
  - Merge join

Sequential scan

- This type of scan reads all rows from a table and filters out those that do not meet the selection criteria.
  - First method considered by planner
  - Usually expensive
  - Read data block sequentially
  - Usage : table without index / high selectivity (1) / small table (few blocks)

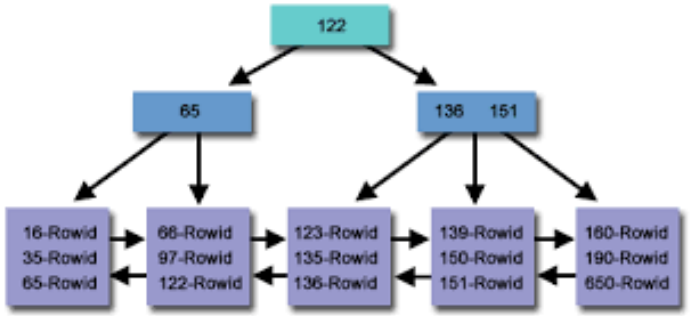


Btree index scan

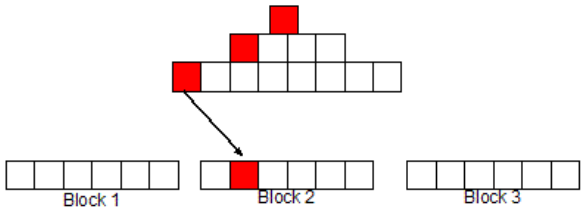
- index scan

- This type of scan reads all rows from an index and filters out those that do not meet the selection criteria.
  - Read index block and data block randomly (expensive operation)
  - Depending on the clustering factor close to 1
  - Usage : low selectivity (>0.1)

- Index only scan is the same thing as index scan except data



Index structure



Select order_nbr, item_name from ordor natural join item;

Un-Clustered table rows

Clustering_factor ~= num_rows

Bitmap index scan

- Bitmap indexes are appropriate for highly repetitive data => high selectivity (1)
- Recommanded for read only
- Very efficient for operator AND / OR
  - Select … where REGION = ’EAST’ OR REGION = ’WEST’

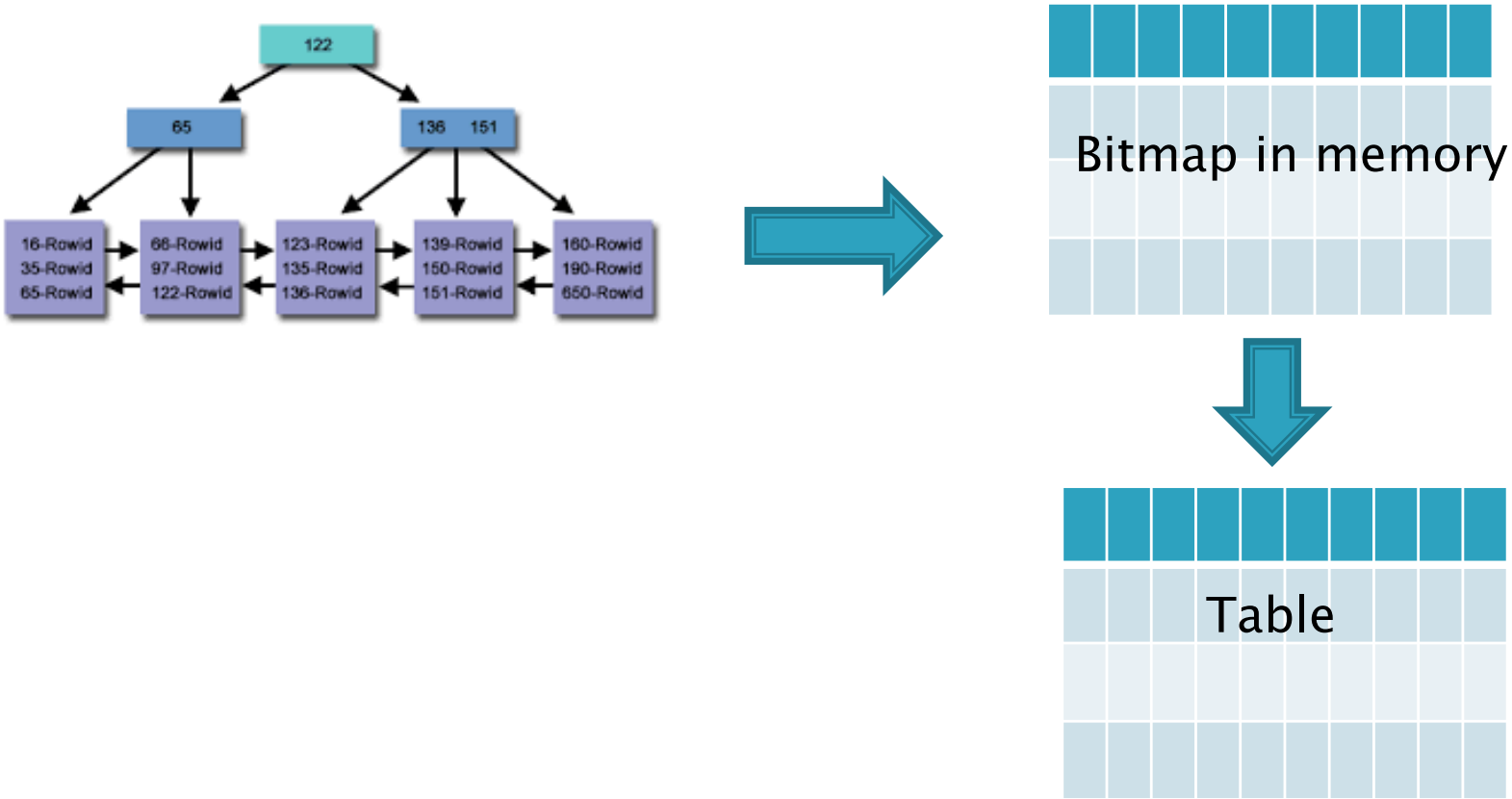
Region	1	2	3	4	5
North	0	1	0	0	0
South	0	0	1	0	0
East	1	0	0	1	0
West	0	0	0	0	1

Employee
East
East

0 0 1 0 OR 0 0 1 1 = 0 0 1 0

Bitmap Heap Scan

- A bitmap scan fetches all the tuple-pointers from the index in one go, sorts them using an in-memory "bitmap" data structure, and then visits the table tuples in physical tuple-location order.

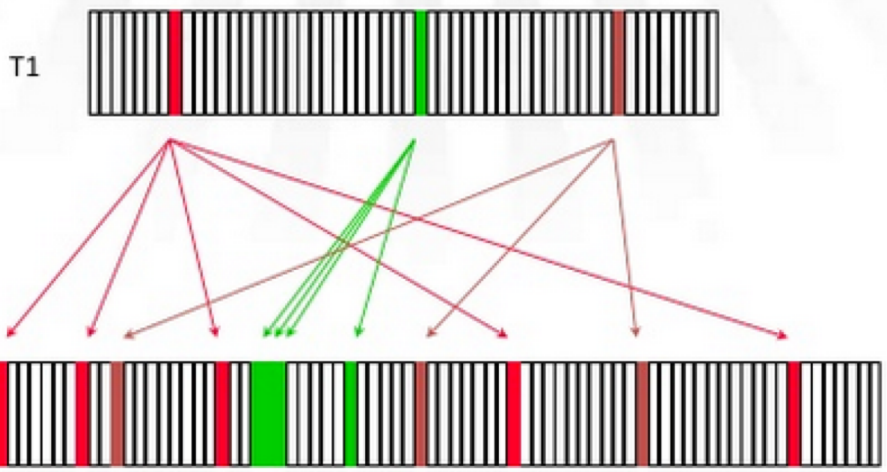


Join method

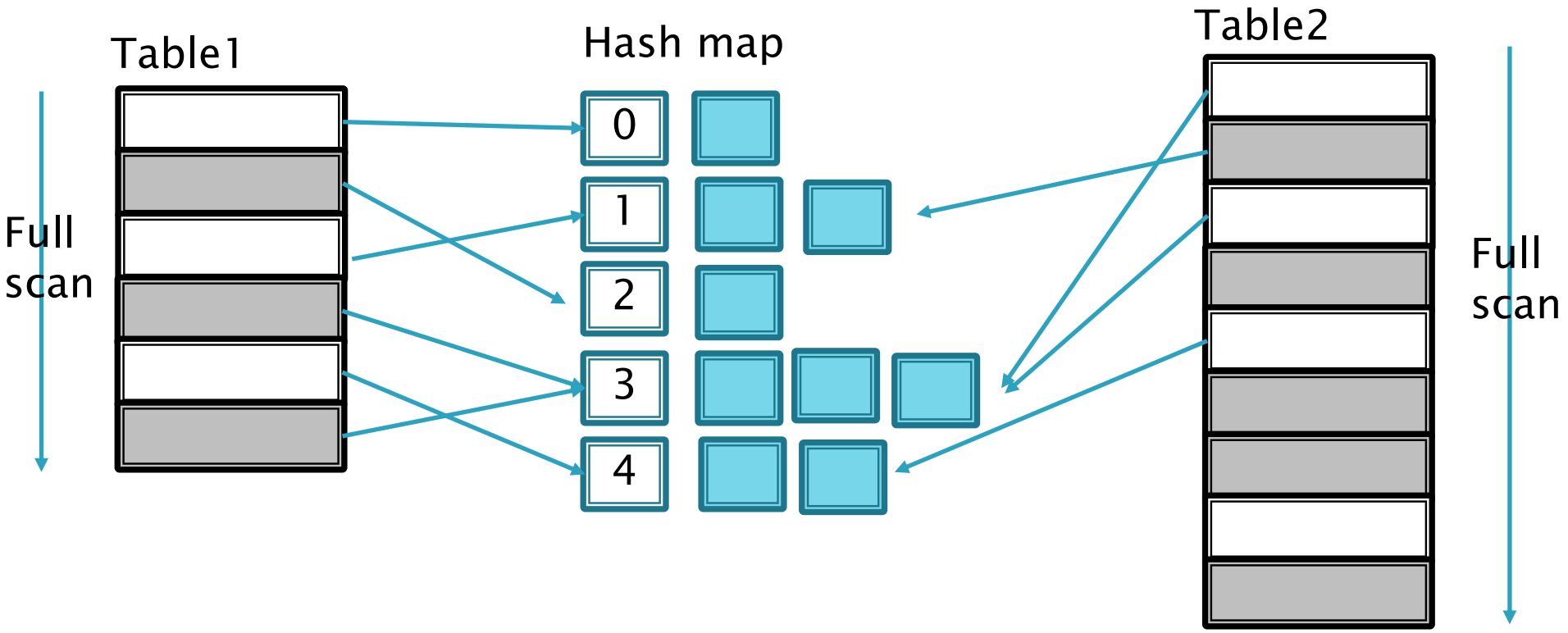
- A join is a binary operation that joins only two row sources.
- A multi-way join consists of joining two tables together and then joining the resultant row source to another table or row source until all the joins are complete.
- For each join, the optimizer determines (in sequence) the
  - Order in which to join the tables
  - Best join method to apply for each join
  - Access path for each row source
- The join order is affected by the following built-in rules:
  - A single-row predicate forces its row source to be placed first in the join order.
  - For outer joins, the table with the outer-joined table must come after the other table in the join order for processing the join.
- The costs of various join methods are compared, for each join order.
  - Nested loop join
  - Hash join
  - Merge join
- Note that some methods can not be used with some operations.
  - For example, a hash join can not be used for a non-equijoin.

- Nested Loop
  - Usage : for Small Tables / both tables are selectively indexed
  - Indexes on join columns

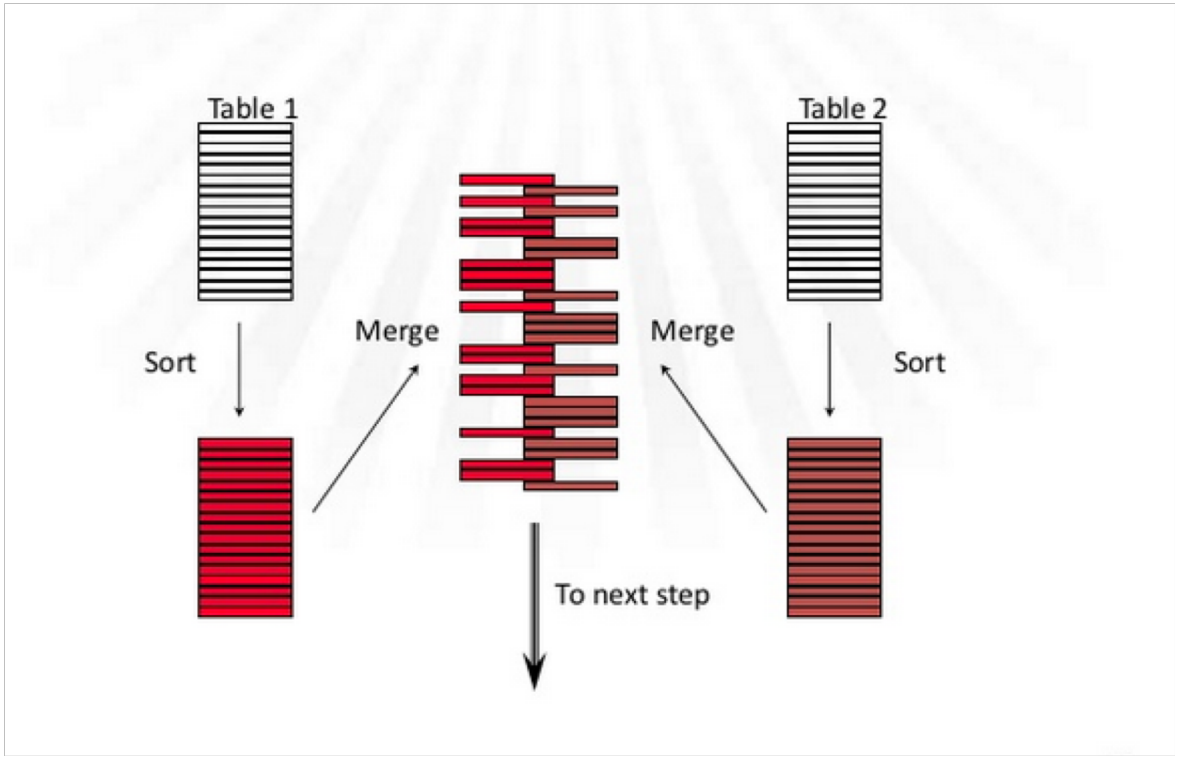
```
for r1 in (select rows from table_1 where colx = {value})
loop
      for r2 in (select rows from table_2 that match current row from table_1)
      loop
            output values from current row of table_1 and current row of table_2
       end loop
end loop
```



- Hash joins
  - Hash joins are performed only for equijoins
  - Perform a full scan of the smaller table
  - Build a hash table (in memory)
  - Perform a full table scan of the other table : may be splitted into partition
  - Collision possible : double check
  - Usage : Large tables / small and large table



- Merge join
  - Rows from each row source are sorted on the join predicate columns.
  - Recommended for Non-equijoin



Hints

- RDBMS provide some features to influence the behavior of the planner.

- Oracle implements hints. In the query, you specify for example the index to use.
  - SELECT /*+ INDEX(idx_test) */ * FROM test;

- Postgres uses system variables to enable or disable some scan method.
  - enable_bitmapscan
  - enable_hashjoin
  - enable_indexscan
  - enable_seqscan …

EXPLAIN

- EXPLAIN is a command displaying the execution plan that the planner will generate for the supplied statement.
- Watch out, the statement analyzed with explain is actually executed. If you want to see the execution plan for an insert or a delete, it is recommended to include the command into a transaction.
  - BEGIN
    - EXPLAIN INSERT INTO COPY_EMP select * from EMP ;
  - ROLLBACK
- There are different options with EXPLAIN :
  - ANALYZE : Carry out the command and show actual run times and other statistics
  - COSTS : Display additional information about estimated startup and total cost
  - TIMING : Include actual startup time and time spent in each node in the output
  - https://www.postgresql.org/docs/11/sql-explain.html

```
select * from test where i = 54;
```

Cost estimate

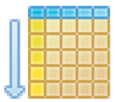
Width row

actual row

```
Index Scan using test_i_idx on test  (cost=0.43..43.53 rows=1034 width=41) (actual time=2.389..4.673 rows=1000 loops=1)
  Index Cond: (i = 54)
Planning Time: 3.270 ms
Execution Time: 4.776 ms
```

Row estimate

actual row

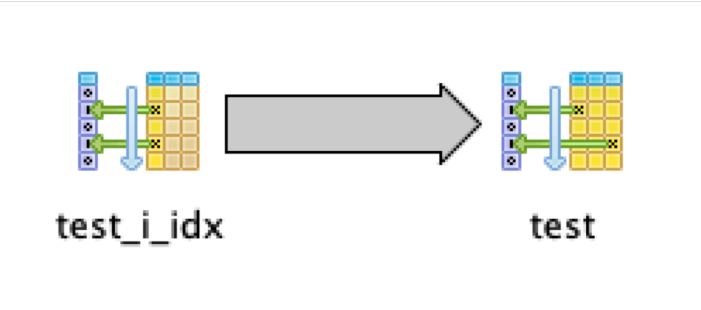


test_i_idx

EXPLAIN

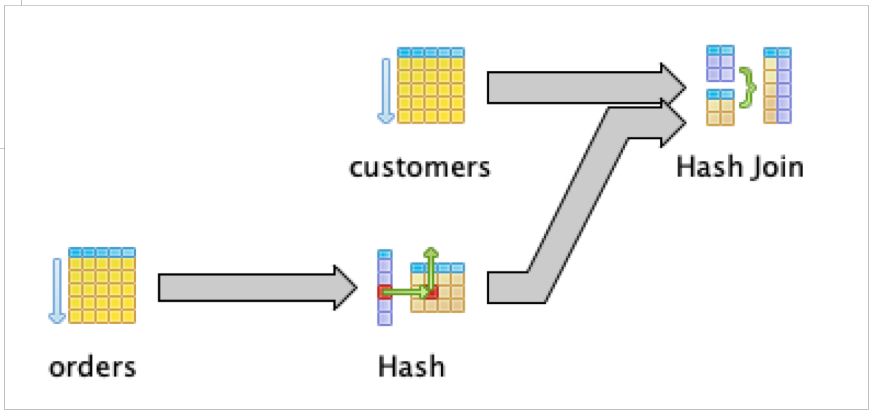
SELECT * FROM test WHERE i < 6000;

	QUERY PLAN text
1	Bitmap Heap Scan on test (cost=113900.19..315198.20 rows=6084613 width=41)
2	Recheck Cond: (i < 6000)
3	-> Bitmap Index Scan on test_i_idx (cost=0.00..112379.03 rows=6084613 width=0)
4	Index Cond: (i < 6000)



EXPLAIN

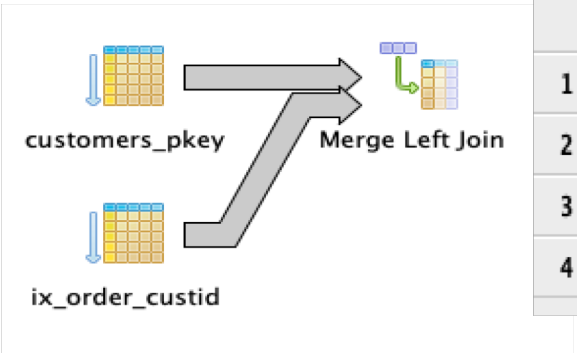
```
EXPLAIN (ANALYZE, TIMING ON, COSTS ON)
select C.customerid,firstname,lastname,orderdate,tax,orderid
from customers C INNER join orders O on C.customerid=O.customerid
```




```
Hash Join  (cost=1849.00..7282.00 rows=60000 width=36) (actual time=43.679..182.245 rows=59999 loops=1)
  Hash Cond: (c.customerid = o.customerid)
  ->  Seq Scan on customers c  (cost=0.00..3333.00 rows=100000 width=22) (actual time=0.005..53.931 rows=100000 loops=1)
  ->  Hash  (cost=1099.00..1099.00 rows=60000 width=18) (actual time=43.485..43.486 rows=59999 loops=1)
        Buckets: 65536  Batches: 1  Memory Usage: 3559kB
        ->  Seq Scan on orders o  (cost=0.00..1099.00 rows=60000 width=18) (actual time=0.468..21.042 rows=59999 loops=1)
Planning Time: 11.592 ms
Execution Time: 191.388 ms
```

EXPLAIN

```
EXPLAIN (ANALYZE, TIMING ON, COSTS ON)
select C.customerid,firstname,lastname,orderdate,tax,orderid
from customers C left join orders O on C.customerid=O.customerid order by C.customerid;
```



	QUERY PLAN text
1	Merge Left Join (cost=0.58..9500.58 rows=100000 width=36)
2	Merge Cond: (c.customerid = o.customerid)
3	-> Index Scan using customers_pkey on customers c (cost=0.29..4940.29 rows=100000 width=22)
4	-> Index Scan using ix_order_custid on orders o (cost=0.29..3560.28 rows=60000 width=18)

https://blog.2ndquadrant.com/hinting_at_postgresql/

SELECT questions FROM audience;