**Joins Test Case**

Sample Data:

create table emp(deptid int,empid int);

create table dept (deptid int, salary int);

insert into emp(deptid,empid)

select n,random()\*1000

from generate\_series(1,50000) n;

insert into dept(deptid,salary)

select n,random()\*1000

from generate\_series(1,20000) n;

create index idx\_dept1 on emp(deptid);

create index idx\_dept2 on dept(deptid);

Nested Loop:

* nested loop: Joins two tables by fetching the result from one table and querying the other table for each row from the first.
  + The least performant form of join.
  + Fast to produce first record.
  + Negative performance possible if the second child is slow.
  + Only join capable of executing CROSS JOIN.
  + Only join capable of inequality join conditions.

Statement:

explain analyze select \* from emp e, dept d where e.deptid < d.deptid;

Hash Joins:

* hash joins: The hash join loads the candidate records from one side of the join into a hash table which is then probed for each record from the other side of the join.
  + Can only be used for equality join conditions.
  + The most performant for joining a large table against a small table.
  + Only for hashable data types.
  + Slow start due to hashing the smaller table.
  + Performance is negatively impacted if table stats out of date and incorrect.

Statement:

explain analyze select \* from emp e, dept d where e.deptid = d.deptid;

* merge join: The (sort) merge join combines two sorted lists like a zipper. Both sides of the join must be presorted.
  + Can only be used for equality join conditions.
  + Generally, the most performant for large data sets.
  + Requires ordered inputs - which can require slow sorts or index scans.
  + Slow to start up, as all index tuples are read and sorted.

Statement: (Create Index)

explain analyze select \* from emp e, dept d where e.deptid = d.deptid;

PostgreSQL join strategies

|  |  |  |  |
| --- | --- | --- | --- |
|  | Nested Loop Join | Hash Join | Merge Join |
| Algorithm | For each outer relation row, scan the inner relation | Build a hash from the inner relation, scan the outer relation, probe the hash | Sort both relations and merge rows |
| Indexes that help | Index on the join keys of the inner relation | None | Indexes on the join keys of both relations |
| Good strategy if | the outer table is small | the hash table fits into work\_mem | both tables are large |