Foundations of Database Systems

Execution Plans in PostgreSQL

Werner Nutt

Execution Plans in PostgreSQL

Call EXPLAIN in psql

```
wdb=> EXPLAIN
wdb-> SELECT d.loc
wdb-> FROM   emp e, dept d, salgrade s
wdb-> WHERE  e.deptno = d.deptno AND
wdb->        e.sal BETWEEN s.losal AND s.hisal AND
wdb->        s.grade = 2;
```

EXPLAIN can also be called from pgAdmin

EXPLAIN Returns a Query Plan

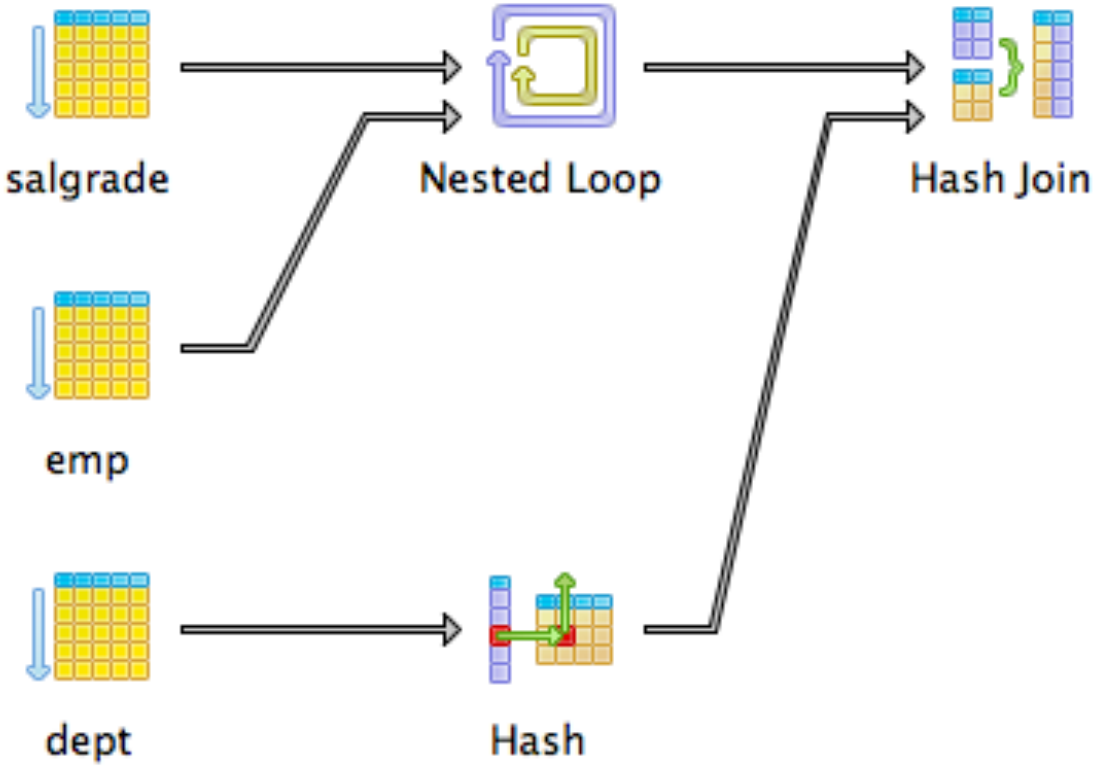
```
Hash Join  (cost=1.09..3.53 rows=2 width=7)
    Hash Cond: (e.deptno = d.deptno)
    ->  Nested Loop  (cost=0.00..2.41 rows=2 width=4)
          Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
          ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
              Filter: (grade = 2)
          ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=8)
    ->  Hash  (cost=1.04..1.04 rows=4 width=11)
          ->  Seq Scan on dept d  (cost=0.00..1.04 rows=4 width=11)
```

Principles

- Each operator, except the top one, is marked by **->**
- The tree structure is indicated by indentation
- For each step the plan contains the operator applied and estimates of
  - the time needed to produce the first tuple (= start-up time), measured in disk page fetches
  - the time needed to produce all tuples
  - the number of result tuples
  - the size of result tuples

There is a graphical representation of plans in pgAdmin

Graphical Plan



Explain Analyze

Runs the query and confronts the expected values in the plan with the real ones

```
Hash Join  (cost=1.09..3.53 rows=2 width=7)
                                    (actual time=0.074..0.107 rows=3 loops=1)
  Hash Cond: (e.deptno = d.deptno)
  ->  Nested Loop  (cost=0.00..2.41 rows=2 width=4)
                                    (actual time=0.037..0.063 rows=3 loops=1)
        Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
        ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
                                     (actual time=0.016..0.018 rows=1 loops=1)
              Filter: (grade = 2)
        ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=8)
                                    (actual time=0.003..0.014 rows=14 loops=1)
  ->  Hash  (cost=1.04..1.04 rows=4 width=11)
                                    (actual time=0.018..0.018 rows=4 loops=1)
        ->  Seq Scan on dept d  (cost=0.00..1.04 rows=4 width=11)
                                    (actual time=0.003..0.007 rows=4 loops=1)
```

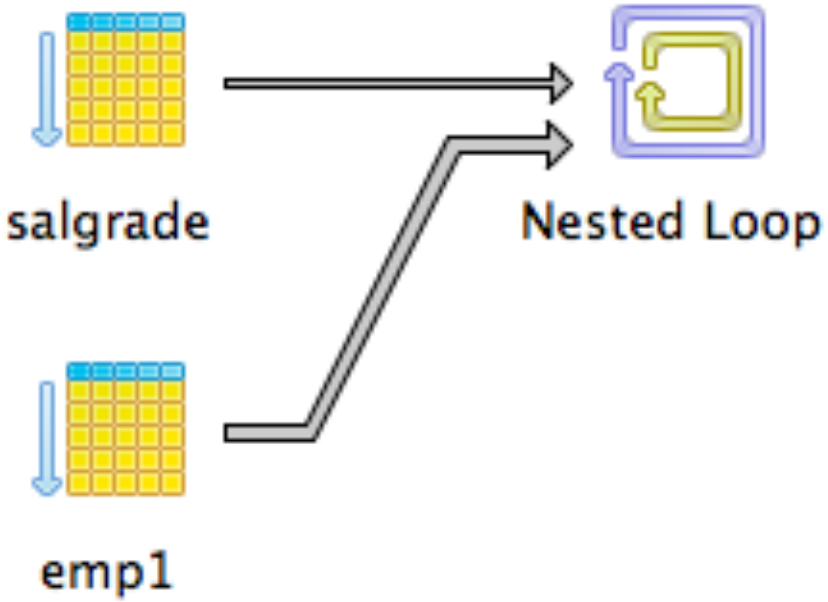
Operators: Query 1

```
EXPLAIN SELECT e.ename
FROM    emp1 e, salgrade s
WHERE   e.sal BETWEEN s.losal AND s.hisal AND
        s.grade = 2;
```

Operators: Plan 1

```
Nested Loop  (cost=0.00..35.06 rows=111 width=10)
  Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
  ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
        Filter: (grade = 2)
  ->  Seq Scan on emp1 e  (cost=0.00..19.00 rows=1000 width=14)
```

Operators: Graphical Plan 1

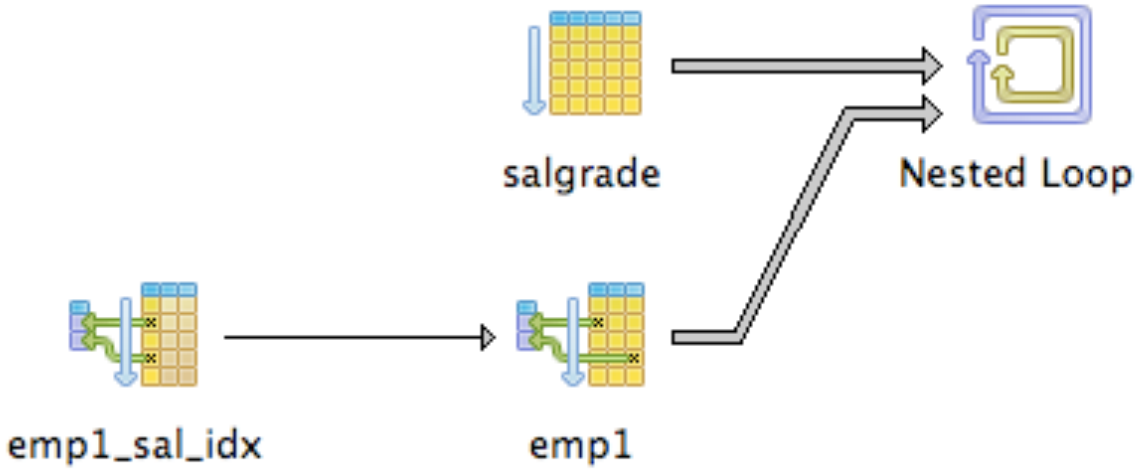


Operators: Query 2

```
CREATE INDEX emp1_sal_idx on emp1(sal);
```

```
EXPLAIN SELECT e.ename
FROM    emp1 e, salgrade s
WHERE   e.sal BETWEEN s.losal AND s.hisal AND
        s.grade = 2;
```

Operators: Graphical Plan 2



Operators: Plan 2

```
Nested Loop  (cost=5.39..18.23 rows=111 width=10)
  ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
        Filter: (grade = 2)
  ->  Bitmap Heap Scan on emp1 e  (cost=5.39..15.50 rows=111 width=14)
        Recheck Cond: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
        ->  Bitmap Index Scan on emp1_sal_idx  (cost=0.00..5.36 rows=111 width=0)
              Index Cond: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
```

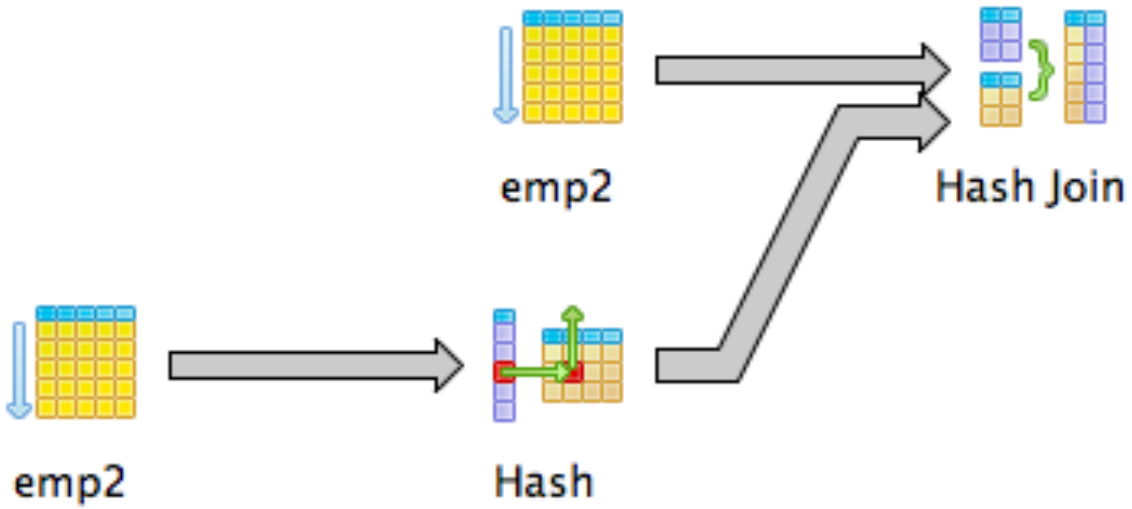
Hash Join vs. Sort Merge Join

emp2 has 10,000 tuples, **emp3** has 100,000 tuples

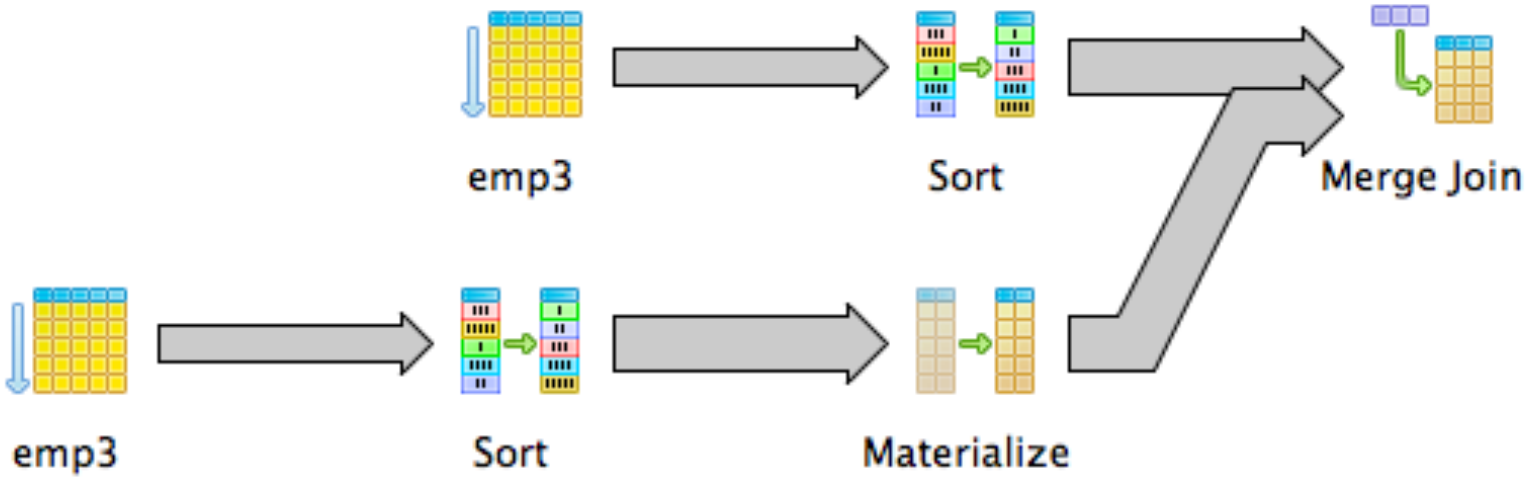
```
SELECT e1.ename, e2.ename
FROM   emp2 e1, emp2 e2
WHERE  e1.mgr = e2.mgr;
```

```
SELECT e1.ename, e2.ename
FROM   emp3 e1, emp3 e2
WHERE  e1.mgr = e2.mgr;
```

Joining `emp2`: Graphical Plan



Joining `emp3`: Graphical Plan



Hash Join vs. Sort Merge Join

```
Hash Join  (cost=312.00..1778.32 rows=26682 width=18)
  Hash Cond: (e1.mgr = e2.mgr)
  ->  Seq Scan on emp2 e1  (cost=0.00..187.00 rows=10000 width=13)
  ->  Hash  (cost=187.00..187.00 rows=10000 width=13)
        ->  Seq Scan on emp2 e2  (cost=0.00..187.00 rows=10000 width=13)
```

```
Merge Join  (cost=23754.82..29777.58 rows=368208 width=18)
  Merge Cond: (e1.mgr = e2.mgr)
  ->  Sort  (cost=11877.32..12127.32 rows=100000 width=13)
        Sort Key: e1.mgr
        ->  Seq Scan on emp3 e1  (cost=0.00..1861.00 rows=100000 width=13)
  ->  Materialize  (cost=11877.32..13127.32 rows=100000 width=13)
        ->  Sort  (cost=11877.32..12127.32 rows=100000 width=13)
              Sort Key: e2.mgr
              ->  Seq Scan on emp3 e2  (cost=0.00..1861.00 rows=100000 width=13)
```

Speeding Up Selections

```
SELECT COUNT (ename)
FROM   emp3 e
WHERE  e.sal = 4000;
```

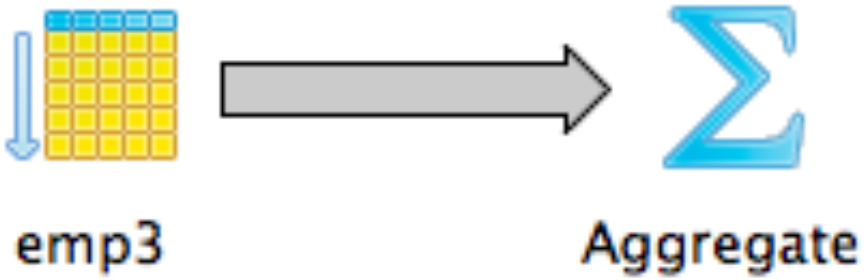
Example

- **emp3** has 100,000 tuples
- salaries are between 5 and 4000
- there are 3959 distinct values of **sal**
- there are 27 tuples with **sal=4000**
- relation **emp3** has size 7Mb

Selection without Index

```
Aggregate  (cost=2111.07..2111.07 rows=1 width=9)
                              (actual time=85.711..85.712 rows=1 loops=1)
  ->  Seq Scan on emp3 e  (cost=0.00..2111.00 rows=25 width=9)
                              (actual time=0.781..85.574 rows=27 loops=1)
        Filter: (sal = 4000)
Total runtime: 85.776 ms
```

Selection Without Index: Graphical Plan

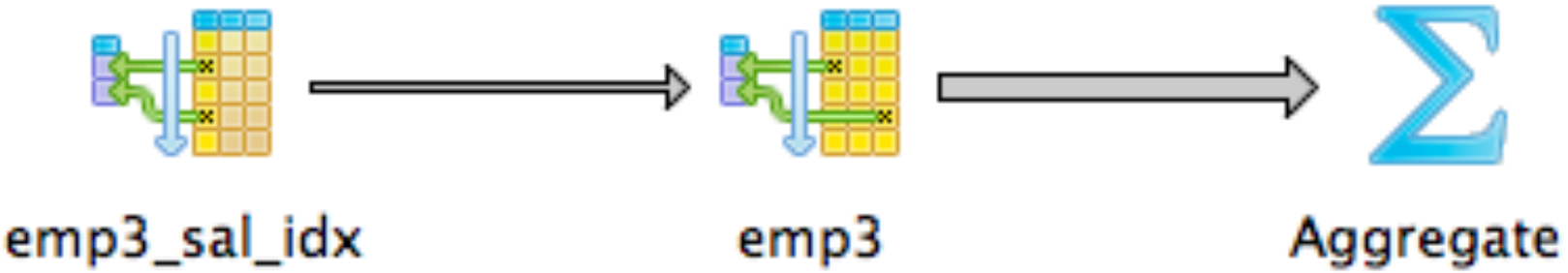


Selection with Index

```
CREATE INDEX emp3_sal_idx ON emp3(sal);
```

Aggregate (cost=92.05..92.06 rows=1 width=9)
(actual time=0.553..0.553 rows=1 loops=1)
-> Bitmap Heap Scan on emp3 e (cost=4.45..91.99 rows=25 width=9)
(actual time=0.206..0.489 rows=27 loops=1)
Recheck Cond: (sal = 4000)
-> Bitmap Index Scan on emp3_sal_idx (cost=0.00..4.45 rows=25 width=0)
(actual time=0.181..0.181 rows=27 loops=1)
Index Cond: (sal = 4000)
Total runtime: 0.632 ms

Selection with Index: Graphical Plan



Selection and Join on Primary Keys

```
EXPLAIN ANALYZE
SELECT *
FROM   emp3 NATURAL JOIN dept3
WHERE  empno = 54321;
```

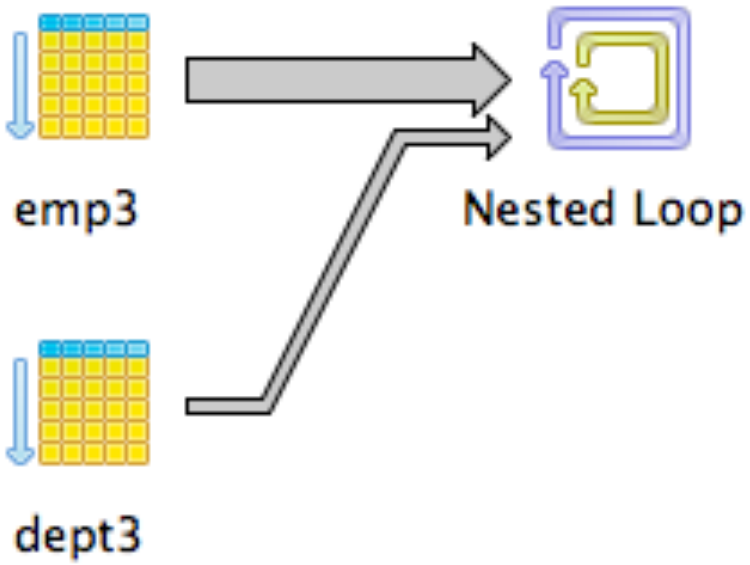
Statistics:

- **emp3** has 100,000 tuples
- **dept3** has 1,000 tuples

Without Indexes on the Primary Keys

```
Nested Loop  (cost=0.00..2140.50 rows=1 width=60)
                                (actual time=25.113..47.055 rows=1 loops=1)
  Join Filter: (emp3.deptno = dept3.deptno)
  ->  Seq Scan on emp3  (cost=0.00..2111.00 rows=1 width=43)
                                (actual time=23.250..45.006 rows=1 loops=1)
        Filter: (empno = 54321)
  ->  Seq Scan on dept3  (cost=0.00..17.00 rows=1000 width=21)
                                (actual time=0.085..1.157 rows=1000 loops=1)
Total runtime: 47.128 ms
```

Without Indexes: Graphical Plan



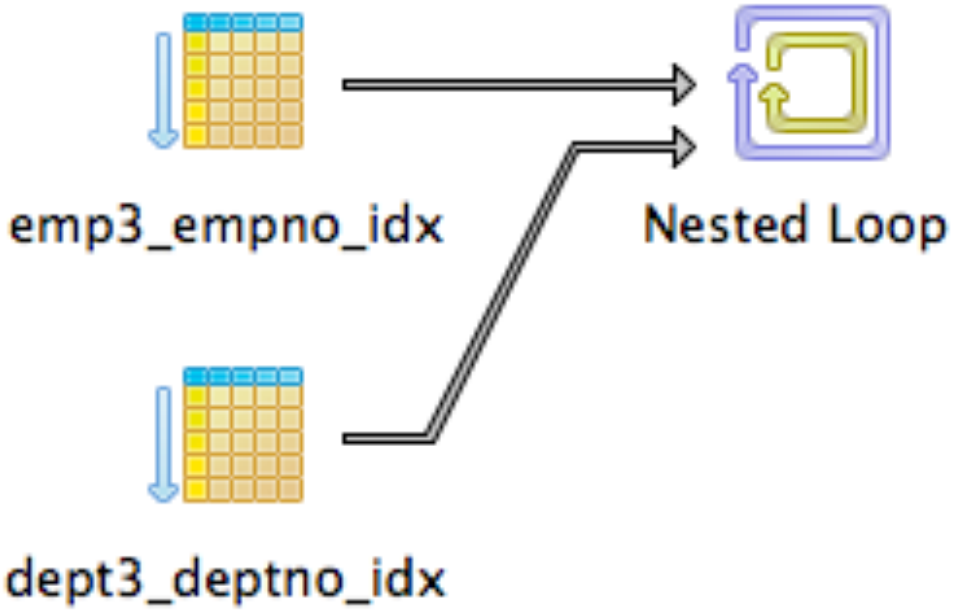
With Indexes on the Primary Keys

```
CREATE INDEX emp3_empno_idx ON emp3(empno);

CREATE INDEX dept3_deptno_idx ON dept3(deptno);
```

Nested Loop (cost=0.00..16.56 rows=1 width=60)
(actual time=0.218..0.226 rows=1 loops=1)
-> Index Scan using emp3_empno_idx on emp3 (cost=0.00..8.28 rows=1 width=43)
(actual time=0.094..0.095 rows=1 loops=1)
Index Cond: (empno = 54321)
-> Index Scan using dept3_deptno_idx on dept3 (cost=0.00..8.27 rows=1 width=21)
(actual time=0.109..0.113 rows=1 loops=1)
Index Cond: (dept3.deptno = emp3.deptno)
Total runtime: 0.271 ms

With Indexes: Graphical Plan



Controlling the Join Order (1)

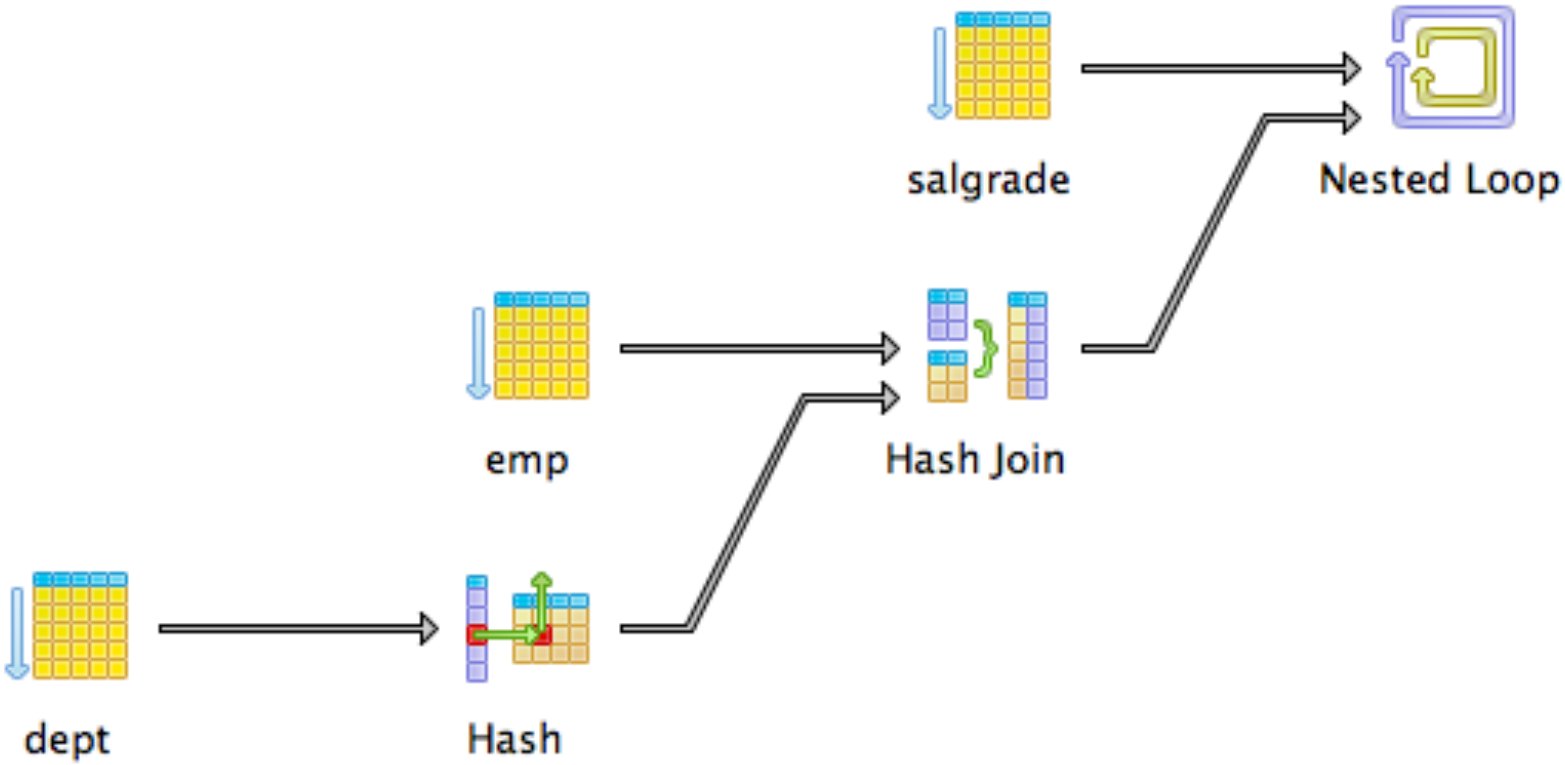
```
SELECT e.ename
FROM   dept d natural join emp e, salgrade s
WHERE  e.sal BETWEEN s.losal AND s.hisal AND
       s.grade = 2 AND
       d.loc = 'Boston';
```

Suppose we want that **salgrade** and **emp** are joined first

The Standard Plan

```
Nested Loop  (cost=1.06..3.42 rows=1 width=6)
  Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
  ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
        Filter: (grade = 2)
  ->  Hash Join  (cost=1.06..2.29 rows=4 width=10)
        Hash Cond: (e.deptno = d.deptno)
        ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=14)
        ->  Hash  (cost=1.05..1.05 rows=1 width=4)
              ->  Seq Scan on dept d  (cost=0.00..1.05 rows=1 width=4)
                    Filter: ((loc)::text = 'Boston'::text)
```

The Standard Plan (2)



Controlling the Join Order (3)

By default, joins in the **`FROM`** clause are translated into

- tables in the **`FROM`** clause
- conditions in the **`WHERE`** clause

In a session, we can set the parameter **`join_collapse_limit`**, which limits the number of relations in the **`FROM`** clause

```
SET join_collapse_limit = 1;
```

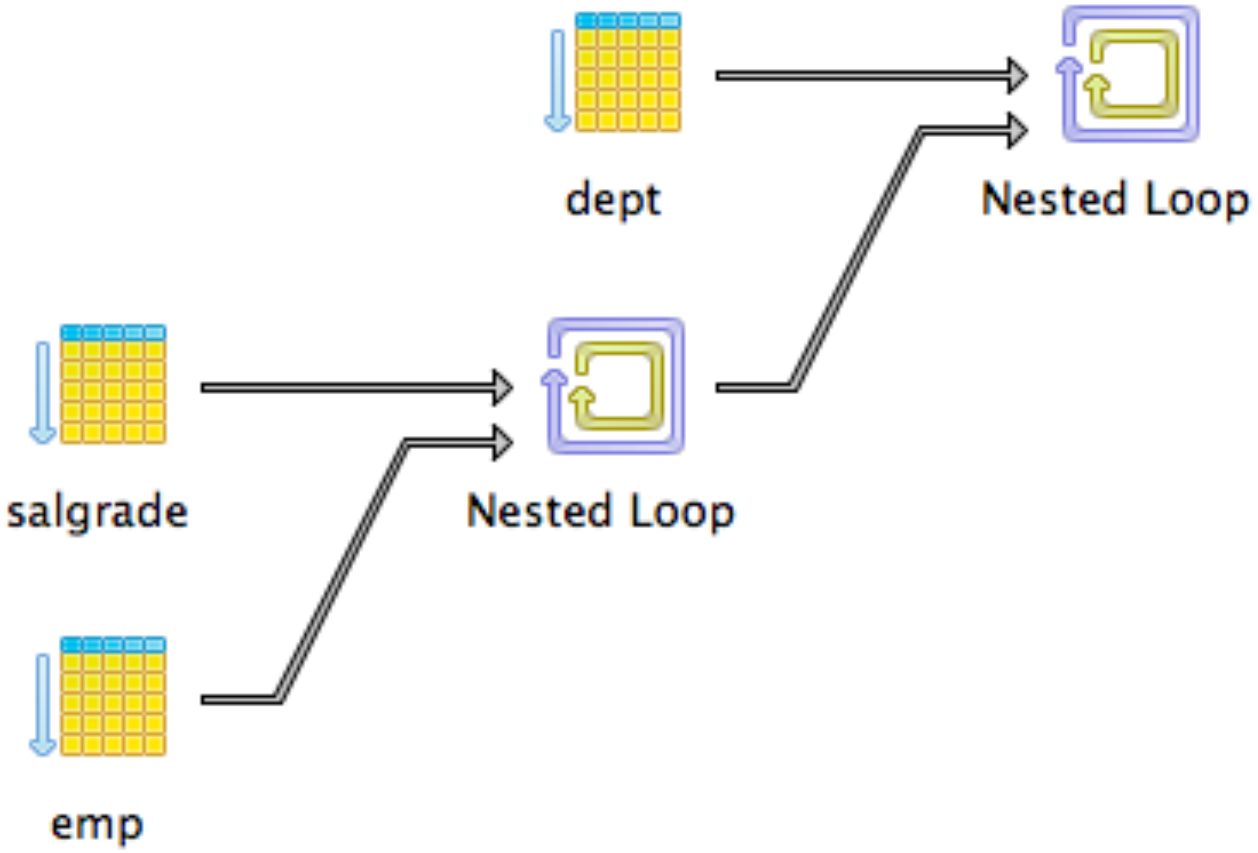
The Modified Query

```
SELECT e.ename
FROM   (emp e JOIN salgrade s
              ON e.sal BETWEEN s.losal AND s.hisal)
                    NATURAL JOIN dept d
WHERE   s.grade = 2 AND
        d.loc = 'Boston';
```

Controlling the Join Order (4)

```
Nested Loop  (cost=0.00..3.49 rows=1 width=6)
  Join Filter: (e.deptno = d.deptno)
  ->  Seq Scan on dept d  (cost=0.00..1.05 rows=1 width=4)
        Filter: ((loc)::text = 'Boston'::text)
  ->  Nested Loop  (cost=0.00..2.41 rows=2 width=10)
        Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
        ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8)
              Filter: (grade = 2)
        ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=14)
```

Modified Query: Graphical Plan



Which One Was Faster?

```
Nested Loop  (cost=1.06..3.42 rows=1 width=6) (actual time=0.350..0.350 rows=0 loops=1)
  Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
  ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8) (actual time=0.164..0.165 rows=1 loops=1)
        Filter: (grade = 2)
  ->  Hash Join  (cost=1.06..2.29 rows=4 width=10) (actual time=0.173..0.173 rows=0 loops=1)
        Hash Cond: (e.deptno = d.deptno)
        ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=14) (actual time=0.077..0.084 rows=14 loops=1)
        ->  Hash  (cost=1.05..1.05 rows=1 width=4) (actual time=0.057..0.057 rows=1 loops=1)
              ->  Seq Scan on dept d  (cost=0.00..1.05 rows=1 width=4) (actual time=0.045..0.047 rows=1 loops=1)
                    Filter: ((loc)::text = 'Boston'::text)
Total runtime: 0.429 ms
```

```
Nested Loop  (cost=0.00..3.49 rows=1 width=6) (actual time=0.078..0.078 rows=0 loops=1)
  Join Filter: (e.deptno = d.deptno)
  ->  Seq Scan on dept d  (cost=0.00..1.05 rows=1 width=4) (actual time=0.018..0.019 rows=1 loops=1)
        Filter: ((loc)::text = 'Boston'::text)
  ->  Nested Loop  (cost=0.00..2.41 rows=2 width=10) (actual time=0.021..0.049 rows=3 loops=1)
        Join Filter: ((e.sal >= s.losal) AND (e.sal <= s.hisal))
        ->  Seq Scan on salgrade s  (cost=0.00..1.06 rows=1 width=8) (actual time=0.006..0.008 rows=1 loops=1)
              Filter: (grade = 2)
        ->  Seq Scan on emp e  (cost=0.00..1.14 rows=14 width=14) (actual time=0.003..0.013 rows=14 loops=1)
Total runtime: 0.140 ms
```

References

For the preparation of these slides I have used the PostgreSQL Manual at

http://www.postgresql.org/docs/manuals/