

Need for cost-based optimization

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select avg (dat...]

File Edit View Query Window Help

Database: tpcd

```
select avg (datepart (dd, l_shipdate - o_orderdate)),  
avg (datepart (dd, l_commitdate - o_orderdate))  
from orders join lineitem on o_orderkey = l_orderkey  
where l_shipdate between '9/15/1992' and '9/16/1992'
```

← Small range

SELECT ← Compute Scalar ← Stream Aggregate/A... ← Nested Loops/Inner... ← tpcd..LINEITEM.I_d... ← Clustered Index Se...

Index search, nested loops join

```
select avg (datepart (dd, l_shipdate - o_orderdate)),  
avg (datepart (dd, l_commitdate - o_orderdate))  
from orders join lineitem on o_orderkey = l_orderkey  
where l_shipdate between '9/15/1992' and '12/14/1992'
```

← Large range

SELECT ← Compute Scalar ← Stream Aggregate/A... ← Hash Match/Inner J... ← tpcd..LINEITEM.I_d... ← tpcd..ORDERS.O_PK

Large scans, hash join

Integrated hashing operation

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select l_partke...]

File Edit View Query Window Help

Database: tpcd

```
select l_partkey, count (*)  
from lineitem join part on l_partkey = p_partkey  
where l_shipdate between '9/15/1992' and '12/14/1992'  
group by l_partkey
```

SELECT

Join

Hash Match/Inner J...

tpcd..LINEITEM.L_P...

tpcd..PART.P_PK

Grouping

Integrated operation

A single hash operation does both grouping and join ... and saves time

Hash team root & member

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select l_partke...]

File Edit View Query Window Help

Database: tpcd

```
select l_partkey, count (*)  
from lineitem, part, partsupp  
where l_partkey = p_partkey and p_partkey = ps_partkey  
group by l_partkey
```

SELECT

Hash Match Root/Ag...

Hash Match Team/In...

Merge Join/Inner J...

tpcd..PART.P_PK, ...

tpcd..PARTSUPP.PS_...

tpcd..LINEITEM.L_P...

No overflow files or I/O costs for intermediate result due to team

Index intersection

SQL Server Query Analyzer - [Query - goetzg1.tpcd.sa - (untitled) - select * from ...]

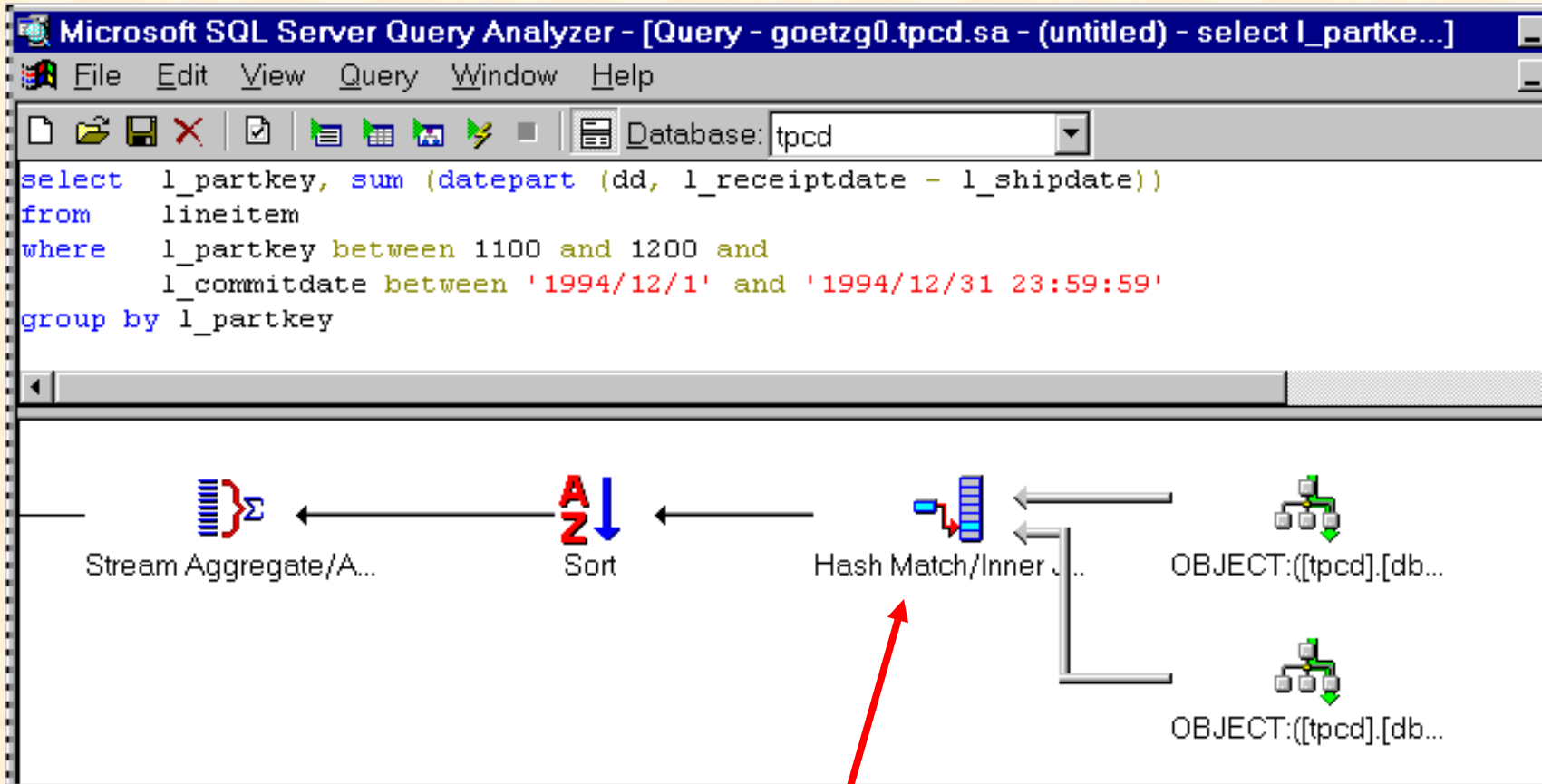
```
select *
from lineitem
where l_shipdate between '1994/1/1' and '1994/6/30 23:59:59' and
      l_suppkey between 500 and 600 and
      l_partkey between 2000 and 4000 and
      l_orderkey between 70000 and 100000
```

Estimated Execution Plan / Messages /

Query batch completed. Exec time: 0:00:00

One table, four predicates, four indexes exploited

Multiple indexes covering a query



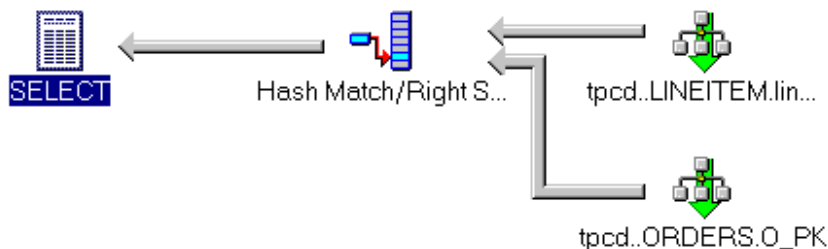
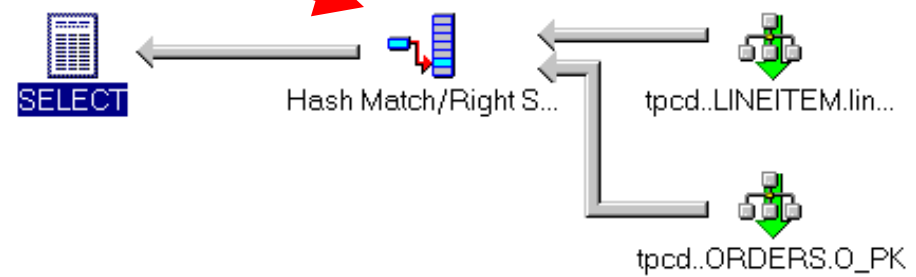
After joining two indexes of one table, all required columns are present – expensive record fetching is avoided

Nested query becomes semi-join

Exploit join algorithms designed for large inputs

```
SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - s
File Edit View Query Window Help
select *
from orders
where exists (
    select *
    from lineitem
    where l_shipmode = 'AIR' and
    l_orderkey = o_orderkey)
```

```
SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - s
File Edit View Query Window Help
select *
from orders
where o_orderkey in (
    select l_orderkey
    from lineitem
    where l_shipmode = 'AIR')
```



SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select p_name, ...]

File Edit View Query Window Help

Database: tpcd

```

select  p_name, p_partkey, count (*)
from    lineitem, part, partsupp
where   l_partkey = p_partkey and p_partkey = ps_partkey
group  by p_name, p_partkey

```

The diagram illustrates the execution plan for the query. It shows a sequence of operations from right to left:

- Three table scans: `tpcd..PART.P_PK`, `tpcd..LINEITEM.idx`, and `tpcd..PARTSUPP.PS_`.
- Two `Merge Join/Inner J...` operators connecting the scans.
- A `Stream Aggregate/A...` operator.
- A final `SELECT` operator at the leftmost end.

Multiple optimization techniques are needed to find this plan

- Join clause inferred between line item & part supply
- Group-by list reduced by functional dependencies
- Grouping (on alternative column) pushed down through join
- “Interesting orderings” between scans, joins, grouping

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select p_name, ...]

File Edit View Query Window Help

Database: tpcd

```

select  p_name, p_partkey, count (*)
from    lineitem, part, partsupp
where   l_partkey = p_partkey and p_partkey = ps_partkey
group  by p_name, p_partkey

```

Diagram illustrating a hash-based join plan:

- Input sources: `tpcd..PART,p_sizet...`, `tpcd..LINEITEM.L_P...`, and `tpcd..PARTSUPP.ps_...`.
- Intermediate operators: `Hash Match Team/In...` and `Hash Match Root/In...`.
- Final output: `SELECT`.

Multiple optimization techniques in a hash-based plan

Same as previous example, *plus*

- Integrated hash operation ...
- ... within a hash team
- Disk-order scans

Star joins: Cartesian products

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select count (*...)]

```
select count (*)
from fact, nation as supp_nation, nation as cust_nation
where f_supp_nationkey = supp_nation.n_nationkey and
      f_cust_nationkey = cust_nation.n_nationkey and
      supp_nation.n_name in ('FRANCE', 'GERMANY') and
      cust_nation.n_name in ('FRANCE', 'GERMANY')
```

SELECT

Stream Aggregate/A...

Nested Loops/Inner...

Nested Loops/Inner...

Filter

tpcd..NATION.N_PK ...

Filter

tpcd..NATION.N_PK ...

tpcd..fact.idx_f_n...

Cartesian product of two dimension tables prior to join with the fact table

Let the large fact table participate in fewer joins –
reduce star join cost without any query plan hints!

Star joins: semi-join reduction

SQL Server Query Analyzer - [Query - (local).tpcd.sa - (untitled) - select count (f...)]

File Edit View Query Window Help

Database: tpcd

```
select count (f_linestatus)
from fact, nation, part
where f_supp_nationkey = n_nationkey and n_name = 'argentina' and
f_partkey = p_partkey and p_size = 7
```

SELECT ← Stream Aggregate/A... ← Bookmark Lookup ← Hash Match/Inner J... ← Nested Loops/Inner... ← tpcd..PART_p_sizet...
Nested Loops/Inner... ← tpcd..NATION.N_PK...
tpcd..fact.idx_f_p...
tpcd..fact.idx_f_s...

First, **join** each dimension table with an index of the fact table;
then, **intersect** bookmark lists;
finally, **fetch** fact table rows

All star join technologies now also in SQL Server!