CSCI317 Database Performance Tuning

Implementation of Relational Operations

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Implementations of Relational Operations

Outline

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is an elementary condition $a = v_a$ are the following

- Sequential scan of entire table r
- Access through an index on attribute a and then and access to the individual data blocks in a relational table r

Query processing plan of selection with an equality condition without an index

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER
 WHERE S_ACCTBAL = 100;
```

```
-----------------------------------------------------------------------------
| Id  | Operation         |    Name   | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           |    1 |    59 |   25    (0)| 00:00:01
|*  1 |  TABLE ACCESS FULL| SUPPLIER |     1 |    59 |   25    (0)| 00:00:01 |
------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
     1 - filter("S_ACCTBAL"=100)
```

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is an elementary condition $a = v_a$ are the following

- Sequential scan of entire table r
- Access through an index on attribute a and then and access to the individual data blocks in a relational table r

Query processing plan of selection with an equality condition with an index

```
EXPLAIN PLAN FOR
  SELECT S_NAME, S_ADDRESS
  FROM SUPPLIER
  WHERE S_SUPPKEY = 007;
```

```
---------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows | Bytes | Cost (%CPU)|  Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |    1 |    57 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID | SUPPLIER      |    1 |    57 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | SUPPLIER_PKEY|    1 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
   2 - access("S_SUPPKEY"=007)
```

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is an elementary condition $a \geq v_a$ are the following

- Sequential scan of entire table r
- Access through an index on attribute a, then horizontal scan of leaf level blocks in a index and access to the selected data blocks in a relational table r

Query processing plan of selection with an inequality condition without an index

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER
 WHERE S_SUPPKEY >= 7;
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 2995 |  157K |   25    (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| SUPPLIER | 2995 |  157K |   25    (0)| 00:00:01 |
-----------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("S_SUPPKEY">=7)
```

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is an elementary condition $a \geq v_a$ are the following

- Sequential scan of entire table r
- Access through an index on attribute a, then horizontal scan of leaf level blocks in a index and access to the selected data blocks in a relational table r

Query processing plan of selection with an inequality condition with an index

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER
 WHERE S_SUPPKEY >= 7777;
```

```
-----------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |     1 |    54 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED | SUPPLIER     |     1 |    54 |     3   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                  | SUPPLIER_PKEY|     1 |       |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
   2 - access("S_SUPPKEY">=7777)
```

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is a Boolean expression in Conjunctive Normal Form (an expression in CNF: ($t_{11} \vee t_{12} \vee \ldots) \wedge (t_{21} \vee t_{22} \vee \ldots) \wedge \ldots \wedge (t_{n1} \vee t_{n2} \vee \ldots)$ where each t_{ij} is an elementary condition like $a = v_a$) are the following

- Sequential scan of entire table r
- Access through the indexes on all attributes used in $(t_{i1} \vee t_{i2} \vee \ldots t_{im})$

Query processing plan of selection with CNF condition without an index

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER
 WHERE S_ACCTBAL = 0 AND S_PHONE = 1234567;
```

```
------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    1 |    71 |   25    (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| SUPPLIER |    1 |    71 |   25    (0)| 00:00:01 |
------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("S_ACCTBAL"=0 AND TO_NUMBER("S_PHONE")=1234567)
```

Selection

The possible implementations of selection operation $\sigma_{\varphi}(r)$ where φ is a Boolean expression in Conjunctive Normal Form (an expression in CNF: ($(t_{11} \vee t_{12} \vee \ldots) \wedge (t_{21} \vee t_{22} \vee \ldots) \wedge \ldots \wedge (t_{n1} \vee t_{n2} \vee \ldots)$ where each t_{ij} is an elementary condition like $a = v_a$) are the following

- Sequential scan of entire table r
- Access through the indexes on all attributes used in $(t_{i1} \vee t_{i2} \vee \ldots t_{im})$

Query processing plan of selection with CNF condition with an index

```sql
CREATE INDEX IDX1 ON SUPPLIER(S_ACCTBAL);
```

```sql
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER
 WHERE S_ACCTBAL = 0 AND S_PHONE = 1234567;
```

```sql
-----------------------------------------------------------------------------------------------
| Id  | Operation                            | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |          |    1 |    71 |    2    (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED | SUPPLIER |    1 |    71 |    2    (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                   | IDX1     |    1 |       |    1    (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
   1 - filter(TO_NUMBER("S_PHONE")=1234567)
   2 - access("S_ACCTBAL"=0)
```

Implementations of Relational Operations

Outline

Projection

The possible implementations of projection operation $\pi_{X}(R)$, where X is a nonempty subset of schema(r) are the following

- Sequential scan of entire table r and elimination of duplicates when required
- Full scan of a leaf level data blocks in an index on Y, where $X \subseteq Y$ and elimination of duplicates whenever it is necessary
- Elimination of duplicates can be performed by hashing or sorting

Query processing plan for projection without an index

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER;
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 3000 |   149K|   25    (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| SUPPLIER | 3000 |   149K|   25    (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

Projection

The possible implementations of projection operation $\pi_{X}(R)$, where X is a nonempty subset of schema(r) are the following

- Sequential scan of entire table r and elimination of duplicates when required
- Full scan of a leaf level data blocks in an index on Y, where $X \subseteq Y$ and elimination of duplicates whenever it is necessary
- Elimination of duplicates can be performed by hashing or sorting

Query processing plan for projection with an index

```
CREATE INDEX IDX3 ON SUPPLIER(S_NAME, S_ADDRESS);
```

```
EXPLAIN PLAN FOR
 SELECT S_NAME, S_ADDRESS
 FROM SUPPLIER;
```

```
---------------------------------------------------------------------------
| Id | Operation              | Name | Rows | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
| 0  | SELECT STATEMENT       |      | 3000 | 149K  | 9 (0)      | 00:00:01 |
| 1  |  INDEX FAST FULL SCAN| IDX3 | 3000 | 149K  | 9 (0)      | 00:00:01 |
---------------------------------------------------------------------------
```

Projection

The possible implementations of projection operation π_{X}(R), where X is a nonempty subset of schema(r) are the following

- Sequential scan of entire table r and elimination of duplicates when required
- Full scan of a leaf level data blocks in an index on Y, where X ⊆ Y and elimination of duplicates whenever it is necessary
- Elimination of duplicates can be performed by hashing or sorting

Query processing plan for projection with an index and elimination of duplicates

```
EXPLAIN PLAN FOR
 SELECT DISTINCT S_NATIONKEY
 FROM SUPPLIER;
```

```
-----------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |   25 |    50 |   25    (0)| 00:00:01 |
|   1 |  HASH UNIQUE       |           |   25 |    50 |   25    (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL| SUPPLIER  | 3000 |  6000 |   25    (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

Implementations of Relational Operations

Outline

- Selection
- Projection
- Join
- Union
- Difference
- Intersection
- Antijoin
- Cross join

Join

Nested loop implementation of join operation $r \bowtie_{\varphi} s$

Nested loop implementation of join operation

```
for each block br ∈ r do
  get(br);
  for each block bs ∈ s do
    get(bs);
    use a condition φ to join all rows in br with all rows in bs
```

cost = $b_r + (b_r * b_s) + b_{rs}$ where b_{rs} represents the total number of blocks written to store the results of join operation

Join

Nested loop implementation of join operation $r \bowtie_{\varphi} s$

Nested loop implementation of join operation

```
EXPLAIN PLAN FOR
 SELECT /*+ USE_NL(CUSTOMER ORDERS) */ *
 FROM ORDERS JOIN CUSTOMER
 ON ORDERS.O_TOTALPRICE = CUSTOMER.C_ACCTBAL
 WHERE O_TOTALPRICE < 5000;
```

```
------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 3807 |  996K | 1063K   (1)| 00:00:42 |
|   1 |  NESTED LOOPS       |          | 3807 |  996K | 1063K   (1)| 00:00:42 |
|*  2 |   TABLE ACCESS FULL| ORDERS    | 3786 |  403K | 1949    (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| CUSTOMER  |    1 |   159 | 280     (0)| 00:00:01 |
------------------------------------------------------------------------------
Predicate Information (identified by operation id):
  2 - filter("ORDERS"."O_TOTALPRICE"<5000)
  3 - filter("ORDERS"."O_TOTALPRICE"="CUSTOMER"."C_ACCTBAL" AND
             "CUSTOMER"."C_ACCTBAL"<5000 AND "CUSTOMER"."C_ACCTBAL">=0)
```

Join

Transient memory nested loop implementation of join operation

$r \bowtie_{\varphi} s$

```
for each block br ∈ r do                Transient memory nested loop implementation of join operation
  get(br ); /* Assume that a relational table r fits in transient memory */
for each block bs ∈ s do
  get(bs );
  use a condition φ to join all rows in br with all rows in bs
```

cost = $b_r + b_s + b_{rs}$ where b_{rs} represents the total number of data blocks written to store the results of join operation

Join

Index nested loop implementation of join operation $r \bowtie_{\varphi} s$

Index nested loop implementation of join operation

```
for each block br ∈ r do
  get(br ); /* Assume that relational table s is indexed on join attribute */
  for each row t ∈ br do
    traverse an index on s with a value of join attribute from t;
    access table s and get all rows is selected through index scan;
    use a condition φ to join a row t with all rows ∈ is;
```

cost = $b_r + height(i_s)*t_r + t_{rs} + b_{rs}$ where $height(i_s)$ is the height of an index i_s, t_r is the total number of rows in r, t_{rs} is the total number of rows in the result of join, and b_{rs} is the total number of data blocks written to store the results of join operation

Join

Index nested loop implementation of join operation $r \bowtie_{\varphi} s$

Index nested loop implementation of join operation

```
EXPLAIN PLAN FOR
 SELECT /*+ ORDERED USE_NL(CUSTOMER) */ *
 FROM ORDERS JOIN CUSTOMER
 ON ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY;
```

```
-----------------------------------------------------------------------------------------------
| Id  |                 Operation     | Name          | Rows  | Bytes | Cost  (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               | 450K  |  109M | 452K     (1)| 00:00:18 |
|   1 | NESTED LOOPS                  |               |       |       |             |          |
|   2 |  NESTED LOOPS                 |               | 450K  |  109M | 452K     (1)| 00:00:18 |
|*  3 |   TABLE ACCESS FULL           | ORDERS        | 450K  |   43M | 1950     (1)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN           | CUSTOMER_PKEY |     1 |       |    0     (0)| 00:00:01 |
|   5 |    TABLE ACCESS BY INDEX ROWID| CUSTOMER      |     1 |   153 |    1     (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
   3 - filter("ORDERS"."O_CUSTKEY">=0)
   4 - access("ORDERS"."O_CUSTKEY"="CUSTOMER"."C_CUSTKEY")
```

Join

Sort-merge implementation of join operation $r \bowtie_{\varphi} s$

Sort-merge implementation of join operation

```
sort table r over the attributes involved in join condition;
sort table s over the attributes involved in join condition;
merge sorted r and sorted s over a join condition;
```

cost = (split) $read(b_r) + write(b_r)$ + (sort and merge) $read(b_r)$ + (split) $read(b_s) + write(b_s)$ + (sort and merge) $read(b_s)$ + (write results) $write(b_{rs})$

Merge of the respective buckets is performed directly after in-memory sorting

Join

Sort-merge implementation of join operation $r \bowtie_{\varphi} s$

Sort-merge implementation of join operation

```
EXPLAIN PLAN FOR
 SELECT /*+ USE_MERGE(ORDERS,CUSTOMER) */ *
 FROM ORDERS JOIN CUSTOMER
      ON ORDERS.O_TOTALPRICE = CUSTOMER.C_ACCTBAL;
```

```
----------------------------------------------------------------------------------------
| Id  | Operation              | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |          | 41034 |   10M |       | 14724   (1)| 00:00:01 |
|   1 | MERGE JOIN             |          | 41034 |   10M |       | 14724   (1)| 00:00:01 |
|   2 |  SORT JOIN             |          | 40910 | 6352K |    14M| 1715    (1)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL    | CUSTOMER| 40910 | 6352K |       | 283     (1)| 00:00:01 |
|*  4 |  SORT JOIN             |          |  450K |   46M |   115M| 13009   (1)| 00:00:01 |
|   5 |   TABLE ACCESS FULL    | ORDERS   |  450K |   46M |       | 1950    (1)| 00:00:01 |
----------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
   3 - filter("CUSTOMER"."C_ACCTBAL">=0)
   4 - access("ORDERS"."O_TOTALPRICE"="CUSTOMER"."C_ACCTBAL")
       filter("ORDERS"."O_TOTALPRICE"="CUSTOMER"."C_ACCTBAL")
```

Join

Hash implementation of join operation r $\bowtie_{\varphi}$ s

```
Hash implementation of join operation
```

apply a hash function h to a join attribute in a table r to produce the hashed buckets bck_{r1}, ... , bck_{rn} from a table r;

apply a hash function h to a join attribute in a table s to produce the hashed buckets bck_{s1}, ... , bck_{sn} from a table s;

compute $bck_{r1} \bowtie bck_{s1}$, ... , $bck_{rn} \bowtie bck_{sn}$

cost = (read and hash) $b_r + b_s$ +

(write to hash buckets) $b_r + b_s$ +

(read the buckets to compute join) $b_r + b_s$ +

(write the results) b_{rs} =

$3 * (b_r + b_s) + b_{rs}$

Join

Hash implementation of join operation $r \bowtie_{\varphi} s$

Hash implementation of join operation

```
EXPLAIN PLAN FOR
 SELECT *
 FROM ORDERS JOIN CUSTOMER
 ON ORDERS.O_TOTALPRICE = CUSTOMER.C_ACCTBAL;
```

```
-------------------------------------------------------------------------------------
| Id  | Operation            | Name     | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |          | 41034 |   10M |       | 5140    (1)| 00:00:01 |
|*  1 |  HASH JOIN           |          | 41034 |   10M | 6832K | 5140    (1)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| CUSTOMER| 40910 | 6352K |       |  283    (1)| 00:00:01 |
|   3 |   TABLE ACCESS FULL| ORDERS   | 450K  |   46M |       | 1950    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
    1 - access("ORDERS"."O_TOTALPRICE"="CUSTOMER"."C_ACCTBAL")
    2 - filter("CUSTOMER"."C_ACCTBAL">=0)
```

Join

Hybrid hash implementation of join operation $r \bowtie_\varphi s$

Hybrid hash implementation of join operation

```
logically split table r into two partitions r1 and r2
for each partition p ∈ {r1, r2} do
  apply a hash function h to join attribute in a table r to produce the hashed buckets bck_p1 , ... , bck_pn
  of a partition p;
    for each block b_s ∈ s do
      apply a hash function h to join attribute in each row v ∈ b_s to find a number of bucket j
      compute v ⋈ bck_pj;
```

cost = (read and hash partitions) b_r +

(write partitions to hash buckets) b_r +

(read and hash (twice) table s to compute join) $2 * b_s$ +

(write the results) + $b_{rs} = 2 * (b_r + b_s) + b_{rs}$

Implementations of Relational Operations

Outline

- Selection
- Projection
- Join
- Union
- Difference
- Intersection
- Antijoin
- Cross join

Union

Hash implementation of union operation `r ∪ s`

```
Hash implementation of union operation
```

apply a hash function h to entire rows in a table r to produce the hashed buckets bck_{r1} , ... , bck_{rn} from a table r;

apply a hash function h to entire rows in a table s to produce the hashed buckets bck_{s1} , ... , bck_{sn} from a table s;

compute $bck_{r1} \cup bck_{s1}$, … , $bck_{rn} \cup bck_{sn}$

cost = (read and hash) $b_r + b_s$ +

(write to hash buckets) $b_r + b_s$ +

(read the buckets to eliminate duplicates) $b_r + b_s$ +

(write the results) + b_{rs} =

$3 * (b_r + b_s) + b_{rs}$

Union

Sort implementation of union operation `r ∪ s`

Hash implementation of union operation

```
EXPLAIN PLAN FOR
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE > 100
  UNION
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE < 10;
```

sql

```
----------------------------------------------------------------------------------
| Id | Operation            | Name   | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|  0 | SELECT STATEMENT     |        | 450K|  2636K|       | 5356   (37)| 00:00:01 |
|  1 |  SORT UNIQUE         |        | 450K|  2636K|  5304K| 5356   (37)| 00:00:01 |
|  2 |   UNION-ALL          |        |     |       |       |            |          |
|* 3 |    TABLE ACCESS FULL| ORDERS|  450K|  2636K|       | 1949    (1)| 00:00:01 |
|* 4 |    TABLE ACCESS FULL| ORDERS|     1 |     6 |       | 1949    (1)| 00:00:01 |
----------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("O_TOTALPRICE">100)
   4 - filter("O_TOTALPRICE"<10)
```

Implementations of Relational Operations

Outline

Difference

Hash implementation of difference operation `r - s`

Hash implementation of difference operation

```
apply a hash function h to entire rows in a table r to produce the hashed buckets
bck_r1 , ... , bck_rn from a table r;
```

```
apply a hash function h to entire rows in a table s to produce the hashed buckets
bck_s1 , … , bck_sn from a table s;
```

```
compute bck_r1 - bck_s1 , ... , bck_rn - bck_sn
```

cost = (read and hash) $b_r + b_s$ +

(write to hash buckets) $b_r + b_s$ +

(read the buckets to compute difference) $b_r + b_s$ +

(write the results) + b_{rs} =

$3 *(b_r + b_s) + b_{rs}$

Difference

Sort implementation of difference operation `r - s`

Hash implementation of difference operation

```
EXPLAIN PLAN FOR
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE < 100
  MINUS
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE > 10;
```

```
---------------------------------------------------------------------------------------
| Id | Operation             | Name    | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT      |         |    1 |  2636K|       | 5356   (64)| 00:00:01 |
|  1 |  MINUS                |         |      |       |       |            |          |
|  2 |   SORT UNIQUE         |         |    1 |     6 |       | 1949    (1)| 00:00:01 |
|* 3 |    TABLE ACCESS FULL| ORDERS |    1 |     6 |       | 1949    (1)| 00:00:01 |
|  4 |   SORT UNIQUE         |         |  450K|  2636K|  5304K| 3407    (1)| 00:00:01 |
|* 5 |    TABLE ACCESS FULL| ORDERS |  450K|  2636K|       | 1949    (1)| 00:00:01 |
---------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
3 - filter("O_TOTALPRICE"<100)
   5 - filter("O_TOTALPRICE">10)
```

Implementations of Relational Operations

Outline

Intersection

Hash implementation of intersection operation $r \cap s$

Hash implementation of intersection operation

```
apply a hash function h to entire rows in a table r to produce the hashed
buckets bck_r1, ... , bck_rn from a table r;
```

```
apply a hash function h to entire rows in a table s to produce the hashed
buckets bck_s1, ... , bck_sn from a table s;
```

```
compute bck_r1 ∩ bck_s1 , ... , bck_rn ∩ bck_sn
```

cost = (read and hash) $b_r + b_s$ +

(write to hash buckets) $b_r + b_s$ +

(read the buckets to compute intersection) $b_r + b_s$ +

(write the results) b_{rs} =

$3 * (b_r + b_s) + b_{rs}$

Intersection

Sort implementation of intersection operation `r ∩ s`

Hash implementation of intersection operation

```
EXPLAIN PLAN FOR
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE < 100
  INTERSECT
 SELECT O_TOTALPRICE FROM ORDERS WHERE O_TOTALPRICE > 10;
```

```
-------------------------------------------------------------------------------------
| Id | Operation            | Name   | Rows | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT     |        |    1 |  2636K|       | 5356   (64)| 00:00:01 |
|  1 |  INTERSECTION        |        |      |       |       |            | |
|  2 |   SORT UNIQUE        |        |    1 |     6 |       | 1949    (1)| 00:00:01 |
|* 3 |    TABLE ACCESS FULL| ORDERS |    1 |     6 |       | 1949    (1)| 00:00:01 |
|  4 |   SORT UNIQUE        |        | 450K|  2636K|  5304K| 3407    (1)| 00:00:01 |
|* 5 |    TABLE ACCESS FULL| ORDERS | 450K|  2636K|       | 1949    (1)| 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("O_TOTALPRICE"<100)
   5 - filter("O_TOTALPRICE">10)
```

Implementations of Relational Operations

Outline

- Selection
- Projection
- Join
- Union
- Difference
- Intersection
- Antijoin
- Cross join

Antijoin

Hash implementation of antijoin operation $r \sim_{\varphi} s$

```
Hash implementation of antijoin operation
apply a hash function h to a antijoin attribute in a table r to produce the hashed
buckets bck_r1, ... , bck_rn from a table r;
```

```
apply a hash function h to antijoin attribute in a table s to produce the hashed
buckets bck_s1, ... , bck_sn from a table s;
```

compute $bck_{r1} \sim_{\varphi} bck_{s1}$, ... , $bck_{rn} \sim_{\varphi} bck_{sn}$

cost = (read and hash) $b_r + b_s$ +

(write to hash buckets) $b_r + b_s$ +

(read the buckets to compute antijoin) $b_r + b_s$ +

(write the results) b_{rs} =

$3 * (b_r + b_s) + b_{rs}$

Antijoin

Hash implementation of antijoin operation $r \sim_{\varphi} t$

Hash implementation of antijoin operation

```
EXPLAIN PLAN FOR
 SELECT *
 FROM CUSTOMER
 WHERE C_CUSTKEY NOT IN (SELECT O_CUSTKEY
                         FROM ORDERS);
```

```
------------------------------------------------------------------------------------------
| Id | Operation            | Name     | Rows   | Bytes| TempSpc| Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT     |          | 14838 | 2376K|        | 2960    (1)| 00:00:01 |
|* 1 |  HASH JOIN RIGHT ANTI |         | 14838 | 2376K|   7472K| 2960    (1)| 00:00:01 |
|* 2 |   TABLE ACCESS FULL  | ORDERS   |   450K| 2197K|        | 1949    (1)| 00:00:01 |
|  3 |   TABLE ACCESS FULL  | CUSTOMER | 45000 | 6987K|        | 283     (1)| 00:00:01 |
------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("C_CUSTKEY"="O_CUSTKEY")
   2 - filter("O_CUSTKEY">=0)
```

Implementations of Relational Operations

Outline

Cross join

Nested loop implementation of cross join operation `r x s`

```
Nested loop implementation of Cross join operation
for each block b_r ∈ r do
  get(b_r);
  for each block b_s ∈ s do
    get(b_s);
    concatenate all rows in b_r with all rows in b_s
```

cost = $b_r + (b_r * b_s) + b_{rs}$ where b_{rs} represents the total number of blocks written to store the results of join operation

Cross join

Nested loop implementation of cross join operation `r x s`

Nested loop implementation of Cross join operation

```
EXPLAIN PLAN FOR
 SELECT *
 FROM CUSTOMER CROSS JOIN ORDERS;
```

```
--------------------------------------------------------------------------------
| Id | Operation              | Name     | Rows | Bytes| Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|  0 | SELECT STATEMENT       |          |   20G| 5054G| 87M     (1)| 00:57:05 |
|  1 |  MERGE JOIN CARTESIAN  |          |   20G| 5054G| 87M     (1)| 00:57:05 |
|  2 |   TABLE ACCESS FULL    | CUSTOMER| 45000| 6987K| 283     (1)| 00:00:01 |
|  3 |   BUFFER SORT          |          |  450K|   46M| 87M     (1)| 00:57:05 |
|  4 |    TABLE ACCESS FULL   | ORDERS   |  450K|   46M| 1948    (1)| 00:00:01 |
--------------------------------------------------------------------------------
```

References

R. Ramakrishnan and J. Gherke Database Management Systems, 3rd ed. Mc Graw-Hill, 2003, chapter 14, (UoW Library closed collection)

Oracle Database, SQL Tuning Guide 19c