

---

Advanced Database Technology  
Anna Östlin Pagh and Rasmus Pagh  
IT University of Copenhagen  
Spring 2005

## **Implementing relational operations**

February 28, 2005

Based on G UW Chapter 15 (except 15.9)

---

# — Lectures 6 & 7: Impl. of relational operations.

We consider the question: Given a query in e.g. SQL, how can the result be efficiently computed in the database?

Stages in query processing:

## 1. Query compilation (Chapter 16)

- Parse (SQL) query to a **query expression tree**. (÷ curriculum)
- Select a **logical query plan**, expressing the query in relational algebra. (Next week)
- Select a **physical query plan**, i.e., particular algorithms and an ordering for the relational operations. (Next week)

## 2. Query execution – **this lecture** (Chapter 15)

- Several possible algorithms for relational algebra operations.
- The best algorithm depends on the particular relations involved, and on the internal memory available.

# — This lecture

---

- Relational operations
- One-pass algorithms
- Algorithms based on sorting
- Algorithms based on hashing
- Index based algorithms
- Buffer management

# — Relational operations —

---

- Set/bag operations: Union, Intersection, Difference
- Projection and Selection
- Join
- Duplicate elimination
- Aggregation and Grouping

## — Set/bag operations —

---

Operations on two sets/bags  $R$  and  $S$ , where  $R$  and  $S$  must have the same set of attributes. We have the three set operations:

- Union,  $R \cup S$ ,
- Intersection,  $R \cap S$ , and
- Difference,  $R \setminus S$ .

# Projection

A **projection** of relation  $R$  on attributes  $A_1, \dots, A_n$  is denoted by

$$\pi_{A_1, \dots, A_n}(R)$$

and is the relation  $R$  restricted to columns for attributes  $A_1, \dots, A_n$ .

<i>title</i>	<i>year</i>	<i>length</i>	<i>filmType</i>	<i>studioName</i>	<i>starName</i>
Star Wars	1977	124	color	Fox	Carrie Fisher
Star Wars	1977	124	color	Fox	Mark Hamill
Star Wars	1977	124	color	Fox	Harrison Ford
Mighty Ducks	1991	104	color	Disney	Emilio Estevez

$$\pi_{title, length, studioName}(Movies) =$$

<i>title</i>	<i>length</i>	<i>studioName</i>
Star Wars	124	Fox
Mighty Ducks	104	Disney

# Selection

A **selection** of tuples satisfying condition  $C$  from relation  $R$  is denoted by

$$\sigma_C(R)$$

and is the relation  $R$  restricted to tuples for which condition  $C$  is satisfied.

$C$  can be any boolean expression, i.e. it may involve multiple attributes, constants, AND, OR, and NOT.

**Example:**

Movies		
<i>title</i>	<i>year</i>	<i>filmType</i>
Star Wars	1977	color
Mighty Ducks	1991	color
Wayne's World	1992	color

$$\sigma_{year > 1981}(Movies) =$$

<i>title</i>	<i>year</i>	<i>filmType</i>
Mighty Ducks	1991	color
Wayne's World	1992	color

## — Join (natural)

---

### Natural-Join:

Let  $R$  and  $S$  be two relations with attributes  $R_1, \dots, R_n$  and  $S_1, \dots, S_m$  respectively. The join of relations  $R$  and  $S$ , denoted

$$R \bowtie S$$

has attributes  $\{R_1, \dots, R_n\} \cup \{S_1, \dots, S_m\}$ .

If  $r \in R$  and  $s \in S$  agree on attributes  $\{R_1, \dots, R_n\} \cap \{S_1, \dots, S_m\}$  then the joint tuple for  $r$  and  $s$  is in  $R \bowtie S$ .

## — Duplicate elimination

---

**Duplicate elimination** is an operator that converts a bag to a set.

$\delta(R)$  denotes the relation with one copy of every tuple that appears in  $R$ .

# — Aggregation and Grouping —

---

**Aggregation operators** summarize in some way the values in a column of a relation. There are several aggregation operators, e.g., Sum, Avg, Min, and Max.

**Grouping** of a relation divides a relation in smaller parts depending on one or more attributes.

**Grouping operator:** Denoted  $\gamma_L(R)$ , where  $L$  is a list of elements. An element is an attribute or aggregation operators applied to an attribute. The relation is grouped depending on the attributes in  $L$ . The result of  $\gamma_L(R)$  is one tuple per group consisting of the grouping attributes and the aggregations.

## — One-pass algorithms —

---

Most often we need to at least read the relation(s) involved in the operation.

**One-pass algorithm:** When it is enough to read the relation(s) once.

In many cases two or more passes are needed, but it depends on the operation, the size of the relation(s), and the size of available main memory.

## — Problem Session —

---

- For the relational operations, which can be implemented using only one pass (regardless of the size of the relation(s))?
- For the binary relational operations, which can be implemented using only one pass, if one of the relations fits in memory?

Describe the algorithms and any constraints on the relations and the results.

- Set/bag operations: Union, Intersection, Difference
- Projection and Selection
- Join
- Duplicate elimination
- Grouping and Aggregation

## — Notation and things to note —

---

### Notation:

We use a somewhat different notation here.

$M$ : Number of blocks in main memory.

$B(R)$ : Number of blocks necessary for storing relation  $R$ .

### Note:

When analyzing an algorithm we do NOT count the number of I/O's to write the output to disk.

## Nested-loop join

---

Algorithm for join,  $R \bowtie S$ :

For each pair of tuples  $r \in R$  and  $s \in S$ , check if they join. If they do, write the joined tuple to output.

**“One-and-a half”-pass:**

$R$  is scanned once,  $S$  is scanned  $|R|$  times.

**Improvement:**

Block-based nested-loop join:

For  $M - 1$  blocks of  $R$  check all pairs with  $S$  (one block at the time).

$R$  is scanned once,  $S$  is scanned  $\lceil B(R)/(M - 1) \rceil$  times.

## — Algorithms based on sorting —

---

(We do as the book does, we start to look at algorithms where two passes are enough.)

General form of a two-pass algorithm based on sorting:

### Pass 1:

Repeat

- Read  $M$  blocks of  $R$  into memory.
- Sort the  $M$  blocks
- Write the sorted sublist back to memory.

### Pass 2:

Merge the sorted sublists in some way. How depends on which operation it is.

## — Duplicate elimination —

---

Algorithm based on sorting for duplicate elimination  $\delta(R)$ :

### Pass 1:

Sort the sublists as described before.

### Pass 2:

- Load one block of each sublist into memory.
- Write one copy of the minimum non-processed element to output.
- Load a new block of a sublist into memory when a block is processed.

Continue until all sublists are empty.

### Total time:

Pass 1:  $B(R)$  I/O's to read,  $B(R)$  I/O's to write sorted sublists to disk.

Pass 2:  $B(R)$  I/O's to read blocks (No I/O's to write output).

Total:  $3B(R)$  I/O's.

**Space:**  $M \geq \sqrt{B(R)}$ .

## — Grouping and aggregation —

---

Algorithm based on sorting for grouping and aggregation  $\gamma_L(R)$ :

### Pass 1:

Sort the sublists, but with the grouping attributes as sorting keys.

### Pass 2:

- Load one block of each sublist into memory.
- Repeat until all tuples are processed:
  - For the least value, process all tuples with this key by computing the aggregate.
  - Write the resulting tuple (grouping attributes and aggregates) to output

**Total time:**  $3B(R)$  I/O's.

**Space:**  $M \geq \sqrt{B(R)}$ .

# Union

---

Algorithm based on sorting for union  $R \cup S$ :

**Pass 1:**

Sort both  $R$  and  $S$  separately in sublists (as before).

**Pass 2:**

Merge the two relations by comparing the least unprocessed tuple in  $R$  and  $S$ .

If both  $R$  and  $S$  include the same tuple then only one is written to output.

**Total time:**  $3(B(R) + B(S))$  I/O's.

**Space:**  $M \geq \sqrt{B(R) + B(S)}$  (one buffer per sublist is still needed).

# Join

---

**Problem:** There may be many tuples to join. E.g., if all tuples are to be joined we have to use  $\frac{B(R)B(S)}{M-1}$  I/O's.

## Sort-based, two-pass, join algorithm:

Assume there are at most  $M/2$  blocks of tuples with equal join-values.

**Pass 1:** Sort both  $R$  and  $S$  separately in sublists.

**Pass 2:** Use  $M/2$  blocks for the merging.

Repeat

- For the least unprocessed value, copy the tuples to the remaining  $M/2$  blocks
- Write the joined tuples to output.

**Total time:**  $3(B(R) + B(S))$  I/O's. **Space:**  $M \cdot M/2 \geq B(R) + B(S)$ .

## — Algorithms based on hashing —

(Two-pass algorithms to start with.)

General form of a two-pass algorithm based on hashing:

### Pass 1:

Hash all tuples into (about)  $M$  buckets.

Time:  $2B(R)$  I/O's (read+write).

### Pass 2:

For each bucket, perform the operation by just looking at one bucket at the time.

Time:  $B(R)$  I/O's.

**Note 1:** All tuples with the same hash key hash to the same bucket.

**Note 2:** All tuples that hash to a bucket have to fit in main memory at the same time and there can be at most  $M$  buckets. Therefore:  $B(R) \leq M^2$ .

## — Duplicate elimination —

---

Algorithm based on hashing for duplicate elimination  $\delta(R)$ :

**Pass 1:**

Hash all tuples in  $R$  to  $M - 1$  buckets, using the whole tuple as key.

**Pass 2:**

Load each bucket (one at the time) into main memory, and for each value, write one copy of the tuple to output.

The algorithm works since all equal tuples hash to the same bucket.

## — Problem session —

---

Consider hash-based, two-pass algorithms.

Describe an algorithm for:

- Grouping and aggregation,  $\gamma_L(R)$ .
- Difference, both for sets and bags,  $R \setminus_S S$  and  $R \setminus_B S$ .
- Join,  $R \bowtie S$ .

What is the I/O-complexity of the algorithm?

What limitations are there w.r.t. the size of  $R$  and  $S$  and main memory ( $M$ )?

## — Index based algorithms —

---

If there is an index it can be used when implementing relational operations.

**Clustering indexes:** All tuples with the same index key appear together (in as few blocks as possible).

**Example:**

**Index based selection:**

$\sigma_C(R)$ , where the condition  $C$  is of the form  $a_i = x$ .

Easy with an index on attribute  $a_i$ , and very efficient if it is a clustering index.

B-trees support efficient selection for range conditions, like  $7 \leq a_i \leq 47$ .

## — Join using an index —

---

Assume we have a sorted clustering index on the join-value, e.g., a B-tree.

Join using an index can be implemented similarly to the sorting based join algorithm, but now we have the sorted order to start with. Therefore pass 1 is not necessary.

**Algorithm:** Just go through the two sorted lists and join the tuples.

Works in one pass as long as there are at most (about)  $M$  blocks of tuples with equal join-values.

## — Buffer management —

---

We have assumed that we can decide which blocks are stored in main memory.

Buffer management deals with the question:

**Which block shall be written back to disk when we need a free block in main memory?**

Note the problem with so-called pinned blocks: Blocks that can not be written back to disk, e.g., due to swizzling.

# — Buffer management strategies —

Also called **caching algorithms**.

Often used to avoid complex decisions on what to store in memory.

- Least-Recently-Used (LRU)
- First-In-First-Out (FIFO)
- The Clock Algorithm
- System control

## — Problem session —

---

So far we have only looked at two-pass algorithms based on sorting and on hashing.

Generalize the two-pass methods to work for any size of the relations, i.e. when  $B(R) > M^2$