Lecture 10
Evaluation of Relational Operators
(Select (σ) / Project (π) )

Chapter 14: Ramakrishnan & Gehrke

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl446
Lecture Outline

Evaluation of Relational Operators

• 14.1) Algorithms for Evaluating Selections (Επιλογές)
  – Selection with No Index, Unsorted Data
  – Selection using No Index, Sorted Data
  – Selection using Hash Index
  – Selection using B+Tree Index

• 14.2) Complex Selections (Σύνθετες Επιλογές με χρήση λογικών τελεστών)

• 14.3) Algorithms for Evaluating Projections (Προβολές)
  – Projections based on Sorting
  – Projections based on Hashing
Relational Operations
(Σχεσιακοί Τελεστές)

- We will consider how a DBMS implements:

Next Lecture
- 14.4) Join - Συνένωση (⊗) Allows us to combine two relations
- 14.5) Set-difference - Διαφορά (-): Tuples in Relation 1, but not in Relation 2.
- 14.5) Union - Ένωση (∪): Tuples in Relation 1 or in Relation 2.
- 14.6) Aggregation - Συνάθροιση (SUM, MIN, etc.) and GROUP BY

- Since each op returns a relation, operators can be composed!
- After we cover the operations, we will discuss how to optimize queries formed by composing them.
  - Relational Algebra operators are closed: a set is said to be closed under some operation if the operation on members of the set produces a member of the set.
Access Paths (Μονοπάτι Προσπέλασης)

- An **Access Path** (Μονοπάτια Προσπέλασης) is a way of retrieving tuples from a table (in response to a query):
  - It consists of a i) **File-Scan** or an ii) **Index** that matches the query selection

- **Tree index matching**
  - e.g., Tree index on \(<a, b, c>\) matches the selection \(a=5 \text{ AND } b=3\), and \(a=5 \text{ AND } b>6\), but not \(b=3\) (as this does not match the prefix of the search key)

- **Hash Index Matching**
  - e.g., Hash index on \(<a, b, c>\) matches \((a=5 \text{ AND } b=3 \text{ AND } c=5)\); but it does not match \(b=3\), or \((a=5 \text{ AND } b=3)\), or \((a>5 \text{ AND } b=3 \text{ AND } c=5)\).

- **Most Selective Access Path** (Μονοπάτι Προσπέλασης με Ψηλότερη Επιλεξιμότητα): The **Access Path** that retrieves the fewest irrelevant pages in order to answer a query Q.
  - Using the **most selective access path** minimizes the cost of data retrieval!
  - e.g., **Hash Index** on SID is a **more selective access path** compared to **File Scan** (as the latter returns ALL tuples).
Schema for Examples
(Σχήμα για Παραδείγματα)

• Assume the following Schema:

  Sailors (\textit{sid}: integer, \textit{sname}: string, \textit{rating}: integer, \textit{age}: real)
  Reserves (\textit{sid}: integer, \textit{bid}: integer, \textit{day}: dates, \textit{rname}: string)

• Also assume the following values:
  – \textbf{Sailors}: Each tuple is \textbf{50 bytes} long, \textbf{80 tuples} per page, \textbf{N=500 pages} of such records stored in the database.
  – \textbf{Reserves}: Each tuple is \textbf{40 bytes} long, \textbf{100 tuples} per page, \textbf{M=1000 pages} of such records stored in the database.
The Selection Operation I
(Ο Τελεστής Επιλογής I)

• Consider the selection query listed below.

```
SELECT *
FROM Reserves R
WHERE R.rname = 'Joe'
```

• **Selection with No Index, Unsorted Data:**
  - **Idea:** Scan \( R \), checking condition on each tuple on-the-fly and returning qualifying objects.
  - **Cost:** \( M \), where \( M \) is the \# of pages for Reserves

• **Can we improve the above approach?**
  - e.g., if data is **Sorted** or if **Hash index** on \( R.rname \) is available then this query could be answered more quickly!

• **We shall now only focus on simple** \( \sigma_{R.attr \ oper \ value(R)} \)
  queries and then extend the discussion to more complex boolean queries (e.g., \( \sigma_{R1.attr \ oper \ value \ AND \ R2.attr \ oper \ value(R)} \))
The Selection Operation II
(Ο Τελεστής Επιλογής II)

• **Selection using No Index, Sorted Data:**
  – **Idea:** Perform binary search over target relation; Identify First Key; and finally scan remaining tuples starting at first key.
  – **Search Cost:** $\log_2 M$
  – **Retrieval Cost:** \#matching_records / PageSize (i.e., \#matching_pages)
    - For the R relation the search cost is: $\log_2 1000 \sim 10$ I/Os

• In practice it is difficult to maintain a file sorted.
• It is more realistic to use a B+Tree using Alternative 1 (see next slide)
The Selection Operation III

(Τελεστής Επιλογής ΙΙΙ)

- **Selection using B+Tree Index:**
  - **Idea:** Use tree to **find the first index entry** that points to a qualifying tuple of R; Then **scan the leaf pages** to retrieve all entries in the key value that satisfy the selection condition.
  - **Search Cost:** \( \log_F M \), (typically 2-3 I/Os) \( F: \) branching factor
  - **Retrieval Cost:** i) Unclustered: \#matching_records (each record on separate page); Clustered: \#matching_records/PageSize (i.e., \#matching_pages)

- **Why is a B+Tree NOT always superior to Scanning?**
  - **Query:** SELECT * FROM Reserves R WHERE R.rname = ‘Joe’
  - **R** relation features 1000 pages.
  - **Assumption:** Selectivity (επιλεξιμότητα) of Query is 10% (i.e., 10%*1000 pages * 100 tuples/page = 10,000 tuples)
  - **Clustered Index Cost:** 3 I/Os + 100 I/Os (tuples on 100 consec. pages)
  - **Unclustered Index Cost:** 3 I/Os + 10,000 I/Os (each tuple on differ. Page)
  - It is cheaper to perform a linear scan that only costs 1000 I/Os!
The Selection Operation IV
(Ο Τελεστής Επιλογής IV)

**Selection using Hash Index:**
- **Idea:** Use hash index to find the index entry that points to a qualifying tuple of R; Retrieve all entries in which the key value satisfies the selection condition.
- **Search Cost:** Const (typically 1.2 I/Os, recall lin./extd. hashing)
- **Retrieval Cost:**
  - Unclustered: #matching records (each record on separate page);
  - Clustered: #matching_records/PageSize (i.e., #matching_pages)

**Example**
- **Query:** SELECT * FROM Reserves R WHERE R.rname = ‘Joe’
- **Assumption:** Selectivity (επιλεξιμότητα) of Query is 10% (i.e., 10%*1000 pages * 100 tuples/page = 10,000 tuples)
- **Clustered Index Cost:** 1.2 I/Os + 100 I/Os (tuples on 100 consec. pages)
- **Unclustered Index Cost:** 1.2 I/Os + 10,000 I/Os (each tuple on differ. Page)
- Again, it is cheaper to perform a linear scan that only costs 1000 I/Os.
Complex Selections  
(Σύνθετες Επιλογές)

- So far we considered selection conditions of the form $\sigma_{R.\text{attr oper value}}(R)$.
- In general, the selection condition can be a boolean expression (λογική έκφραση) using logical operators (AND, OR, etc).
  - i.e., $\sigma_{(R.\text{attr oper value}) \ § \ ... \ § (R.\text{attr oper value})}(R)$, where § is a binary logical operator such as $\land$ (conjunction/σύζευξη), $\lor$ (disjunction/διάζευξη), etc.

$$\text{(day<8/9/94 AND rname=’Paul’) OR bid=5 OR sid=3}$$

Selection conditions are first converted to Conjunctive Normal Form (CNF):
- A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses (όρων), where a clause is a disjunction of literals (λεκτικά στοιχεία).
- e.g., $(\text{day<8/9/94 OR bid=5 OR sid=3}) \text{ AND (rname=’Paul’ OR bid=5 OR sid=3})$
- Every propositional formula (προτασιακή εξίσωση) can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: double negative ($\neg(\neg A) \Leftrightarrow A$), De Morgan ($\neg(A \lor B) \Leftrightarrow \neg A \land \neg B$), distributive law/επιμεριστική ($A \land B) \lor C \Leftrightarrow (A \lor C) \land (B \lor C)$

<table>
<thead>
<tr>
<th>CNF</th>
<th>NOT CNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \land B.$</td>
<td>$(A \land B) \lor C$</td>
</tr>
<tr>
<td>$\neg A \land (B \lor C)$</td>
<td>$\neg(B \lor C)$</td>
</tr>
<tr>
<td>$(A \lor B) \land (\neg B \lor C \lor \neg D) \land (D \lor \neg E)$</td>
<td>$\neg(B \lor C)$</td>
</tr>
</tbody>
</table>
Complex Selections
(Σύνθετες Επιλογές)

Selection WITHOUT Disjunctions

Template: \( \sigma_A \land B \land \ldots \land Z (R) \)

First Approach

- Compute the most selective access path \( R' = \sigma_A \) (i.e., the one that returns the fewest irrelevant results compared to \( \sigma_B(R) \ldots \sigma_Z(R) \))
  - This could be a composite selection e.g., \( (\sigma_A \land B \land C(R)) \ldots \) depends on what access methods (indexes) are available.
- Then apply on-the-fly the rest conditions on \( R' \) (i.e., \( \sigma_B \land \ldots \land Z(R') \))

Example

- Consider \( day<8/9/94 \text{ AND } bid=5 \text{ AND } sid=3 \).
- A B+ tree index on \( day \) can be used; then, \( bid=5 \) and \( sid=3 \) must be checked for each retrieved tuple on-the-fly.
- Similarly, a hash index on \( <bid, sid> \) could be used; \( day<8/9/94 \) must then be checked on-the-fly.
Complex Selections
(Σύνθετες Επιλογές)

Selection WITHOUT Disjunctions

\[ \sigma_{A \land B \land \ldots \land Z}(R) \]

Second Approach

• Compute \( R_A = \sigma_A(R) \) and \( R_B = \sigma_B(R) \), and \( \ldots \) and \( R_z = \sigma_z(R) \) using independent access methods (if indexes are available)
• Intersect RID Sets: \( \text{sort}(R_A) \cap \text{sort}(R_B) \cap \ldots \cap \text{sort}(R_z) \)
  – Note: Intersecting sorted runs is cheaper than intersecting arbitrary runs.
• Each DBMSs uses different ways to achieve RID intersection.

Example

• Consider \( \text{day}<8/9/94 \ AND \ bid=5 \ AND \ sid=3 \).
• If we have a B+ tree index on \( \text{day} \) and a hash index on \( \text{sid} \), both using Alternative (2)
• Use both indexes (i.e., \( \text{day}<8/9/94 \ [B+tree] \ \text{and} \ \text{sid}=3 \ [Hash \ Index] \))
• Intersect results, retrieve records and then check \( \text{bid}=5 \).
Complex Selections
(Σύνθετες Επιλογές)

• Selection WITH Disjunctions

\[ \sigma_{(A_1 \lor \ldots \lor A_n) \land (B_1 \lor \ldots \lor B_n) \land \ldots \land (Z_1 \lor \ldots \lor Z_n)}(R) \]

Case A: \( \sigma_{(A_1 \lor A_2)}(R) \) e.g., day<8/9/94 OR bid=5

• If \( A_1 \) has an index and \( A_2 \) has an index, then
  – utilize both indexes and take the union of results.
• If \( A_1 \) has an index but \( A_2 \) needs a file scan, then
  – utilize only file scan & check both conditions on-the-fly

Case B: \( \sigma_{(A_1 \lor A_2) \land B_1}(R) \) e.g., (day<8/9/94 OR bid=5) AND sid=3

• If only \( B_1 \) has an index (For \( (A_1 \lor A_2) \) check case-A), then
  – utilize the index and check the \( A_1 \lor A_2 \) conditions on-the-fly
• Most DBMSs don’t handle this class of queries very efficiently!
The Projection Operation  
(Τελεστής Προβολής)

- Consider the projection query \( \pi_{\text{sid}, \text{bid}}(\text{Reserves}) \) listed below.

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} \quad \text{R.sid, R.bid} \\
\text{FROM} & \quad \text{Reserves R}
\end{align*}
\]

- recall that in relational algebra all rows have to be distinct as the query answer is a set.

- The projection operator is of the form

\[ \pi_{\text{attr}_1, \text{attr}_2, \ldots, \text{attr}_m}(R) \]

- The implementation requires the following
  - Remove unwanted columns (on-the-fly)
  - Eliminate any duplicate tuples produces.
    - This step is the difficult one!

- We will describe two techniques to cope with duplicate elimination: i) based on Sorting; and ii) based on Hashing.
Projection Based on Sorting 
(Προβολή μέσω Ταξινόμησης)

- First **approach**: essentially the External Merge Sort Alg.
- **Phase 1**: Create Sorted Runs (with selected attributes)
  - **Scan** R and **produce** a set of tuples that contain only the **desired attributes** i.e., only <R.sid, R.bid> (First Step of ExternalMergeSort)
    - **Read_Cost**: M I/Os & **Write_Cost**: T I/Os, where T is some fraction of M (i.e., depending on fields projected out) **Total**: M + T I/Os
    - **Example**: Assume that T=250 then **Total Cost**: 1000 + 250 = 1250 I/Os

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```
Projection Based on Sorting
(Προβολή μέσω Ταξινόμησης)

• **Phase 2: Sort** tuples using an External Sort Algorithm based on the projected keys (i.e., <R.sid, R.bid>)
  
  – **Cost**: $2T \times (\#\text{passes})$ I/Os where $\#\text{passes}: \left\lceil \log_{\frac{B}{B-1}} \frac{T}{B} \right\rceil$
    
    (i.e., cost of the External Merge Sort without first step)
  
  – **Example**: Using $B=20$ Buffer pages and $T=250$ I/Os
    
    $\left\lceil \log_{\frac{20}{19}} \frac{250}{20} \right\rceil = 2$
    
    **Total Cost**: $2 \times 250 \times 2 = 1000$ I/Os

• **Step 3: Scan the sorted result** (on-the-fly, consequently costs nothing), comparing adjacent tuples and discard duplicates
  
  – e.g., <1,2>, <1,2>, <2,2>, <2,3>,
  
  – **Total Cost**: $M + T + (2T \times \left\lceil \log_{\frac{B}{B-1}} \frac{T}{B} \right\rceil )$

  – Using Example: $1000 + 250 + 2 \times 250 \times 2 = 2250$ I/Os
Projection Based on Hashing
(Προβολή Μέσω Κατακερματισμού)

- **Second approach to remove duplicates during projections**

- **Phase 1: Partitioning (Φάση Διαμέρισης)**
  - Read R (page-at-a-time) using one input buffer
  - For each tuple, discard **unwanted fields**, apply hash function $h_1$ to choose one of $B-1$ output buffers. (**Cost:** $M$ I/Os)
  - Result is **$B-1$ partitions of tuples** (unwanted fields are eliminated).
    - 2 tuples from different partitions guaranteed to be distinct! (**Cost:** $T$ I/Os), where $T$ is some fraction of $M$ (i.e., depending on the fields that are projected out)

```
SELECT DISTINCT R.sid, R.bid
FROM Reserves R
```
**Phase 2: Duplicate Elimination (Φάση Απαλοιφής Διπλοτύπων):**

- For each partition, read in a page-at-a-time and build an in-memory hash table, using hash function $h_2(<> h_1)$ on all fields. *(Cost: $T$ I/Os)*
- If a new tuple hashes to the same value as some existing tuple, compare the two to check whether the new tuple is a duplicate. Discard duplicates as they are detected.

**Total cost (Projection using Hashing):** $M + 2T$ I/Os

- Assuming that $T=250$, the cost of the project query is $1000 + 2\times250 = 1500$ I/Os

SELECT DISTINCT R.sid, R.bid
FROM Reserves R
Use of Indexes for Projections  
(Χρήση Ευρετηρίων για Προβολή)

- So far we have NOT considered using Indexes for Projections.
- If an **existing index** contains all wanted attributes as its search key then we can apply an **index-only** scan.
  - e.g., Q="SELECT DISTINCT R.rname FROM R" and **Hash Index <R.rname>** is available.
  - We can use the index to identify the **R.rname** set (i.e., index scan). We must then use **sorting** or **hashing** to eliminate duplicates.
- If an **ordered (i.e., tree) index** contains all wanted attributes as **prefix** of search key, can do even better:
  - e.g., Q="SELECT DISTINCT R.rname FROM R" and **B+Tree Index <R.rname>** is available.
  - We can use the index to identify **R.rname** set (index scan). discard unwanted fields, compare adjacent tuples to check for duplicates.
  - We do not even need to apply **sorting** or **hashing for the duplicate elimination part!"
Projections: Sorting vs. Hashing
(Προβολή: Ταξινόμηση vs. Κατακερματισμό)

• **Sort-based** approach is the standard
  – Better Handling of skew (e.g., if the distribution of hash values is very non-uniform then Hash-based will suffer)
  – The result is **sorted** at the end of the execution!
  – DBMS anyway implement sorting algorithms thus projection can be implemented with little additional effort.

• What do commercial DBMSs implement?
  – **Sorting:** IBM DB2, Oracle and Sybase ASE, SQL Server
  – **Indexing:** Informix, SQL Server
Summary (Σύνοψη)

• A virtue (αρετή) of relational DBMSs: queries are composed of a few basic operators;
  – The implementation of these operators can be carefully tuned (and it is important to do this!).

• Many alternative implementation techniques for each operator;
  – No universally superior technique for most operators.

• Must consider available alternatives for each operation in a query and choose best one based on system statistics, etc.
  – This is part of the broader task of optimizing a query composed of several operators.