ICOM 6005 – Database Management Systems Design

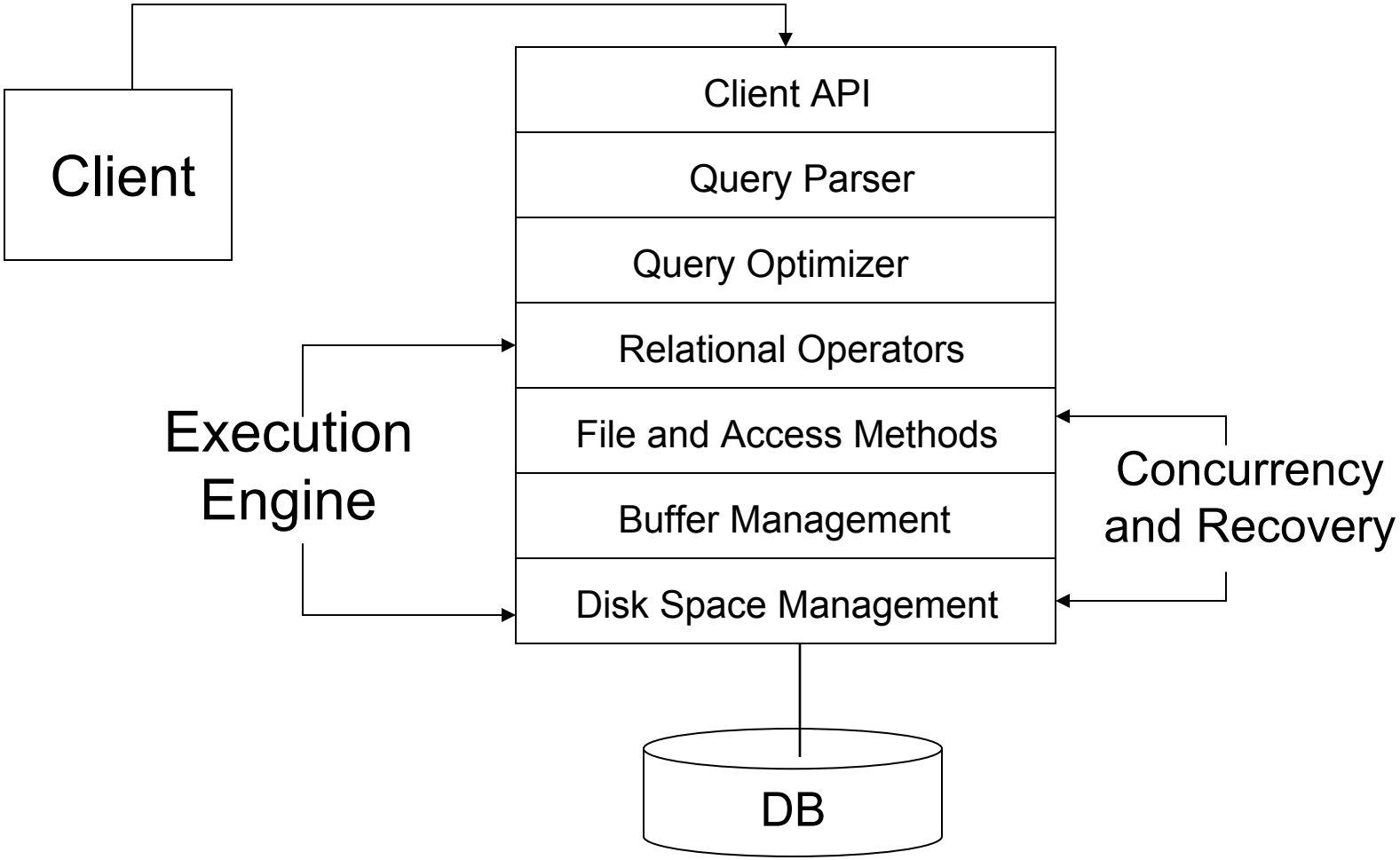
Dr. Manuel Rodríguez-Martínez

Electrical and Computer Engineering Department

Query Evaluation Techniques

- Read :
  - Chapter 12, sec 12.1-12.3
  - Chapter 13
- Purpose:
  - Study different algorithms to execute (evaluate) SQL relational operators
    - Selection
    - Projection
    - Joins
    - Aggregates
    - Etc.

Relational DBMS Architecture



Processing a query

- Parser –
  - transforms query into a tree expression (parse tree)
  - Looks into catalog for metadata about tables, attributes, operators, etc.
- Optimizer
  - transforms query expression tree into a query plan tree
    - Tree with metadata about relations, attributes, operators, and algorithms to run each operator
  - Searches several alternative query plan trees to find the cheapest
    - Based on I/O cost, CPU cost (and network cost)
- Execution Engine
  - Takes the plan from optimizer, interprets it and runs it.

Issues in selecting a query plan

- Need to understand cost of different plan
  - Plan – algorithm to run a given relational operator
  - Example- Selection: “Get all students with gpa = 4.00”
    - Plan 1 – scan heap file to find records with gap 4.00
    - Plan 2 – Use an index on gpa to find records with 4.00
    - Plan 3 – Sort table students on gap attribute, then scan sorted records for those with gpa = 4.00
- Need to have statistics about:
  - Relation size, attribute size
  - Distribution of attribute values
    - Uniform vs skewed
  - Disk speed
  - Memory size
  - Etc.

System Catalog

- The System catalog (Catalog for short)
  - Collection of tables with data about the database data (metadata) and system configuration
    - Often called data dictionary
- Closure property
  - Catalog is a collection of tables
  - Can be queried via SQL!!!
    - Easy way to find out information about the system
- Every DBMS vendor has its own way to organize the catalog
  - Also have quick mechanism to read the information

What information is stored?

- Table
  - Table name
  - File that stores and file type (heap file, clustered B+ tree, …)
  - Attributes names and types
  - Indices defined on the table
  - Integrity constrains
  - Statistics
- Index
  - Index name and type of structure (B+ tree, external hash, …)
  - Search key
  - Statistics
- Views
  - View name and definition

What are the statistics?

- Relational Cardinality
  - Number of tuples in table R : *Ntuples(R)*
- Relation Size
  - Number of pages used to store R : *NPages(R)*
- Index Cardinality
  - Number of distinct key values in index I : *NKeys(I)*
- Index size
  - Number of pages for index I: *NPages(I)*
    - B+-tree – number of leaf pages
- Index Height
  - Number of non-leaf levels: *IHeight(I)*
- Index Range
  - Min and max values in the index: *ILow(I)* and *IHigh(I)*

Some issues about stats …

- DBMS must periodically update statistics
- Often done once or twice per week
  - More fine tuning can be done to increase accuracy
- Histograms can also be used
  - Give a better distribution of values
    - Relation attributed and Indices search keys
- Tradeoff
  - Computing stats is expensive
  - Accurate stats yield very good query evaluation plans
- Often stats are a bit inaccurate, plus assumptions are simplified
  - Ex. : attribute values are distributed independently

Sample Catalog

- Tables:
  - *Sailors(sid:integer, sname:string, rating:integer, age:real);*
  - *Reservations(sid:integer, bid:integer, day:dates,rname:string);*

- Catalog table for attributes:
  - *Attribut_Cat(attr_name:string, rel_name:string, type:string, position:integer)*

Catalog Instance: Attribute_Cat

Attr_name	Rel_name	Type	Position
Attr_name	Attribute_Cat	string	1
Rel_name	Attribute_Cat	string	2
Type	Attribute_Cat	string	3
Position	Attribute_Cat	Integer	4
Sid	Sailors	Integer	1
Sname	Sailors	string	2
Rating	Sailors	integer	3
Age	Sailors	real	4
Sid	Reserves	integer	1
Bid	Reserves	Integer	2
Day	Reserves	Dates	3
rname	Reserves	string	4

Access paths

- An **access path** is a mechanism (algorithm) to retrieve tuples from a table(s).
  - Also called **access method**
- Three main schemes used for access paths
  - File Scan - iterate over all tuples in a table
  - Indexing - Use an index to extract records
    - Good for selection
  - Partitioning – sort or hash the data before the tuples are examined
    - Good for aggregation, projections and joins

Single-Table access paths

- Consider SQL query:

  Select sid, slogin, sname

  From Students

  Where gpa == 4.00 and age < 25;

- Need to read tuples and evaluate where clause!
- Accessing a table mostly done via two kinds of access paths
  - File Scan
    - Read each page on data file (e.g. heap file) and get every tuple, then evaluate predicate
  - Index Scan
    - Use B+-tree or Hashing to find tuples that match a condition (or part of it) on where clause

SQL and relational algebra

- Typically query parser will transform SQL query into parse tree, and then into query plan tree
- Query plan tree is a tree that represents a relational algebra expression!
- Every node in the tree implements a relational operator + the scan operator.
- The scan operator is used to fetch tuples from disk
  - First access path that gets evaluated!
- The tuples processed by a given node are then passed to its parent on the plan tree.

Conjuctive Normal form

- Where clause is assumed to be a conjunction:
  - Attr1 op value1 AND Attr2 op value2 … and AttrK op value K
- Each term Attr1 op value1 is called a predicate or conjunct.
- Each conjunct is a filter for the tuples.
  - Those tuples that do not pass the conjunct are excluded from the result
- Disjunctive normal form is also used but is less amicable to optimization
- Conjuncts are prime candidates for evaluation using an index.

Index Matching

- Given a query Q, with a predicate (conjunct) p, we say that an index I matches the predicate p if and only if
  - the index can be used to retrieve just the tuples that satisfy p.
- The index might match all the conjuncts in the selection or just a few (or even just one)
- Primary conjuncts – conjuncts that are matched by the index
- If the index matches the whole where clause, index is enough to implement the selection
- Otherwise, use index to fetch tuples, evaluate the other predicates iteratively
  - Each predicate will filter out unwanted tuples

Rules for matching

- Hash index: one or more conjunct in the form *attribute* = *value* in a selection have attributes that match the index search key.
  - Need to include all attributes in search key
- Tree Index (B+-tree or ISAM): one or more terms of the form *attribute op value* in a selection have attributes that match a prefix of the search key.
  - *op* can be any of : >, <, =, <>, >=, <=
- Note on prefixes:
  - If the search key for a B+tree is: <a, b, c> the following are prefixes of this search key:
    - <a>, <a,b>, <a,b,c>
  - The following are not prefixes:
    - <a, c>, <b, c>

Examples on matching

- Assume tables:
  - *Sailors(sid:integer, sname:string, rating:integer, age:real);*
  - *Reservations(sid:integer, bid:integer, day:dates,rname:string);*
- Hash Index on Reservations & key <rname, bid, sid>
  - Matches: rname = “Bob” and bid = 5 and sid = 10
  - Not matching : rname = “Bob”; rname = Bob and bid = 5
    - Matching a prefix tells me nothing, since hash key depends on the while set of attributes!
- Same scenario, but with B+-tree
  - Matches:
    - rname = “Bob” and bid = 5 and sid = 10;
    - rname = “Bob” and bid = 5; rname = “Bob”
  - Not matching: bid = 5 and sid = 10

More examples

- Index on Reserves with search key <bid, sid>
- Hash Index or B-tree match the condition
  - rname = “Tom” and bid = 10 and sid = 4
- Why?
  - Can be rewritten as bid = 10 and sid = 4 and rname = “Tom”
  - First two conjuncts bid = 10 and sid = 4 match search key
  - Conjuct rname = “Tom” must be evaluate afterward
    - Iteratively
- Matching deals with whether or not index can be used to make a first pass on the table.

Selectivity of Access paths

- Each access paths a **selectivity,** which is the number of pages to be retrieved by it
  - Both data pages and index pages (if any)
- It is helpful to define the notion of selectivity factors for conjuncts
  - Given a predicate p, the selectivity factor p is the fraction of tuples in a relation R that satisfy p.
  - Denoted as
- Selectivity for conjunct p applied to a relation R can then be computed as:

$$selectivity \approx \left\lceil \frac{(SFp * NTuples(R))}{\#tuples\ per\ page} \right\rceil$$

Several useful SF formulas

- Equality:
  - *Attr = value:* SF = 1/# of distinct values for attr in table R
  - If we have an index I on attribute *Attr* , then SF = 1/ *NKeys(I)*
  - If the information is not available use: 1/10
- Greate than:
  - *Attr > value*: SF = *(IHigh(I) – value)* / (*IHigh(I) - ILow(I)*)
  - Otherwise, use 1/3
- *Attr* between *value1* and *value2*
  - *SF = (value2 – value1)/(IHigh(I) – ILow(I))*
  - *Otherwise, use 1/4*

Selection and SF

- Selections can have where clause with CNF, or DNF.
- CNF case:
  - If where clause is of the form p1 and p2 and … and pn, then

$$SF = SF_{p1} \wedge SF_{p2} \wedge \ldots \wedge SF_{pn}$$

  - Assume that all predicates are independent