Module 12

Query Processing and Optimization

Objectives

- Learn about issues affecting how queries should be processed

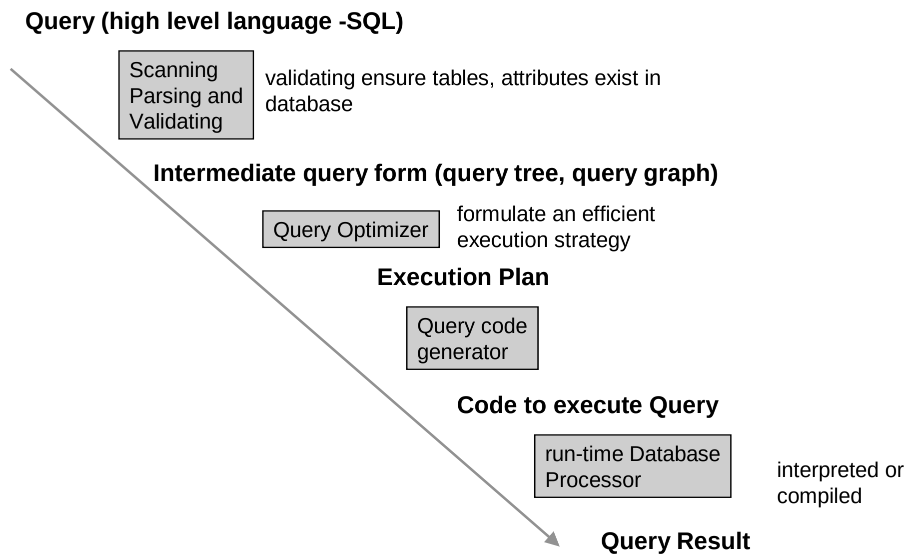
95.305

Topics

References

- **Elmasri & Navathe Chapter 16**

Query Processing Schematic



Optimizing Declarative Queries

- Procedural languages are optimized by the query programmer since the instruct how the database is to be navigated
- Declarative languages indicate what data is required, and not how to navigate the database (e.g. SQL)
- For declarative languages an optimization strategy can greatly improve the run-time efficiency

Optimization

- The optimizer has several implementations for operations like SELECT, JOIN to pick from
- Two main optimization techniques: Heuristics and Cost estimation
- Heuristics typically reorder queries in the execution plan (query tree)
- Estimation is based on computing a probable cost of various strategies
- The two techniques are usually combined by the query optimizer

Query Operations algorithms

- The optimizer has several algorithms for implementing SELECT, JOIN, etc.
- Choice of algorithm might depend on:

  -size of tables
  -knowledge of data in tables
  -whether an index for the appropriate attribute exists
  -whether a hash function exists for the query attributes
  ...

Company Relational Database Schema

EMPLOYEE

fname	minit	lname	<u>ssn</u>	bdate	address	sex	salary	superssn	dno

DEPARTMENT

dname	<u>dnumber</u>	mgrssn	mgrstartdate

DEPT_LOCATIONS

<u>dnumber</u>	<u>dlocation</u>

PROJECT

pname	<u>pnumber</u>	plocation	dnum

WORKS_ON

<u>essn</u>	pno	hours

DEPENDENT

<u>essn</u>	<u>dpendent_name</u>	sex	bdate	relationship

Note:
Attributes refering to the same thing can have different name
(pnumber vs. pno)

Attributes refering to different things can have the same name
(name of employee or
name of department)

Company Relations

EMPLOYEE

FNAME	INIT	LNAME	<u>SSN</u>	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	B	Smith	123456789	9-Jan-55	731 Fondern	M	30000	333445555	5
Franklin	T	Wong	333445555	8-Dec-45	638 Voss	M	40000	888665555	5
Alicia	J	Zelaya	999887777	19-Jul-58	3321 Castle	F	25000	987987987	4
Jennifer	S	Wallace	987654321	20-Jun-31	291 Berry	F	43000	888665555	4
Ramesh	K	Narayan	666884444	15-Sep-52	975 Fire Oak	M	38000	333445555	5
Joyce	A	English	453453453	31-Jul-62	5631 Rice	F	25000	333445555	5
Ahmad	V	Jabber	987987987	29-Mar-59	980 Dallas	M	25000	987654321	4
James	E	Borg	888665555	10-Nov-27	450 Stone	M	55000	NULL	1

DEPARTMENT

DNAME	<u>DNUMBER</u>	MGRSSN	MGRSTARTDATE
Research	5	333445555	22-May-78
Administration	4	987654321	1-Jan-85
Headquarters	1	888665555	19-Jun-71

...Company Relations

DEPT_LOCATIONS

DNUMBER	DLOCATION
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.50
123456789	2	7.50
666884444	3	40.00
453453453	1	20.00
453453453	2	20.00
333445555	2	10.00
333445555	3	10.00
333445555	10	10.00
333445555	20	10.00
999887777	30	30.00
999887777	10	10.00
987987987	10	35.00
987987987	30	5.00
987654321	30	20.00
987654321	20	15.00
888665555	20	NULL

...Company Relations

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Staffort	4
Reorganization	20	Houston	1
NewBenefits	30	Stafford	4

DEPENDENT

ESSN	D_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	5-Apr-76	DAUGHTER
333445555	Theodore	M	25-Oct-73	SON
333445555	Joy	F	3-May-48	SPOUSE
987654321	Abner	M	29-Feb-32	SPOUSE
123456789	Michael	M	1-Jan-78	SON
123456789	Alice	F	31-Jan-78	DAUGHTER
123456789	Elizabeth	F	5-May-57	SPOUSE

Implementations for SELECT

- **We will consider several select operations on the company database**

σ (ssn=123456789) **(**EMPLOYEE**)**

σ (DNUMBER > 5) **(**DEPARTMENT**)**

σ (DNO = 5) **(**EMPLOYEE**)**

σ (DNO = 5 AND SALARY > 30000 AND SEX = F) **(**EMPLOYEE**)**

σ (ESSN=123456789 AND PNO=10) **(**WORKS_ON**)**

Implementation of SELECT search

- **Brute force (Linear Search) Retrieve every record from the file and test whether its attribute values satisfy the select condition**
- **Works for any SELECT query**
- **Makes no assumptions about file structure**
- **Obviously quite slow: O(n) for n records**

...Implementation of SELECT search

σ (ssn=123456789) (EMPLOYEE)

- Binary Search
- If the selection condition involves equality on the ordering key binary search can be used
- O(log(n)) for n records

...Implementation of SELECT search

σ (ssn=123456789) (EMPLOYEE)

- Using primary key or hash function
- Works is selection condition is equality test on search key
- usually faster than O(log(n)) for n records
- Could be as fast as O(1)

...Implementation of SELECT search

σ (DNUMBER > 5) **(**DEPARTMENT**)**

- **Use primary index to retrieve multiple records**
- **Works if selection condition is an inequality on search key**
- **Use the index to find the corresponding equality condition (DNUMBER = 5) then chain through subsequent records**
- **Can only be used for range queries (e.g. 30000 <= SALARY <= 35000)**

...Implementation of SELECT search

σ (DNO = 5) **(**EMPLOYEE**)**

- **Use a clustering index to retrieve multiple records**
- **Works for selection conditions involving equality on a nonkey attribute with a clustering index**

...Implementation of SELECT search

σ (DNO = 5) **(**EMPLOYEE**)**

- **Use a secondary (B+ tree) index**
- **Works for retrieving a single record if indexing field has unique value (is key) or to retrieve multiple records if indexing field is not a key**
- **Can only be used for range queries (e.g. 30000 <= SALARY <= 35000)**

...Implementation of SELECT search

σ (DNO = 5 AND SALARY > 30000 AND SEX = F) **(**EMPLOYEE**)**

- **Conjunctive Selection**
- **If the condition is of the form X AND Y, use one for the previous methods to find all records satisfying X, and check each for Y**

...Implementation of SELECT search

σ (ESSN=123456789 AND PNO=10) **(**WORKS_ON**)**

- **Conjunctive Selection using Composite Index**
- **e.g. if an index exists on the composite key (ESSN, PNO), use the index directly**
- **Works if two or more attributes are used in an equality condition, and a composite index exists on those attributes**

...Implementation of SELECT search

σ (ESSN=123456789 AND PNO=10) **(**WORKS_ON**)**

- **Conjunctive Selection by Pointer Intersection**
- **Works if selection condition is a conjunction of equality conditions, and an index, with record pointers, exists for each attribute in the condition**
- **Strategy: index on each attribute value to find pointers to those records and take intersection of pointers**
- **Other conditions can also be tested within the intersection result.**
- **(Pointers must be to records, not blocks)**

Implementations for SELECT

σ (ssn=123456789) **(**EMPLOYEE**)**

σ (DNUMBER > 5) **(**DEPARTMENT**)**

σ (DNO = 5) **(**EMPLOYEE**)**

- **For these we can check whether and index, or hash, exists for the selection attribute, otherwise use brute force**
- **Optimization is needed most when a conjunction is involved**

Using Selectivity

- **To optimize a query it might be helpful to know how many records would satisfy a condition such as X='Ottawa'**
- **If** X **is a key a selectivity estimate would be**
  1/ |r(R)|
- **For a non-key attribute value** X **with** i **distinct values a selectivity estimate is**
  |r(R)|/i / |r(R)| = 1/i
- **So database might keep track of the number of distinct values of an attribute (statistics about the data)**

Disjunctive Conditions (OR)

σ (DNO = 5 OR SALARY > 30000 OR SEX = F) (EMPLOYEE)

- **Harder to deal with**
- **Little optimization can be done because the condition requires union of records**
- **If <u>any one</u> attribute does not have an index, we are forced to use brute force approach**
- **Only when every OR condition attribute has an index can we optimize**

Implementing the JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

DEPARTMENT ⊗ (MGRSSN = SSN) EMPLOYEE

- **JOIN is one of the most time-consuming operations in query processing**
- **Most joins are equi-joins (natural joins) so we only examine these**

R1 ⊗ (X=Y) R2

...Implementing the JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

DEPARTMENT ⊗ (MGRSSN = SSN) EMPLOYEE

- **Brute Force (Nested Loop)**
- **For each t in EMPLOYEE {**

```
For each s in DEPARTMENT {
  if (t[DNO] = s[DNUMBER])
       result := result union (t union s)
  }
}
```


...Implementing the JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

- **Using index on join attribute**
- **e.g. if index exists on** DNO **in** EMPLOYEE
- **Retrieve each record** t **from** DEPARTMENT **and use index to test for** EMPLOYEE **records which match** t[DNUMBER]

...Implementing the JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

- **Using Sort-merge JOIN**
- **If** EMPLOYEE **is physically sorted by** DNO **and** DEPARTMENT **sorted by** DNUMBER
- **Each file is scanned and matched against the other**
- **This method is very efficient since each file is only scanned once**
- **O(n) for n records in largest file**

...Implementing the JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

- **Using Hash JOIN**
- **If** EMPLOYEE **and** DEPARTMENT **are both hashed to the <u>same</u> file using the <u>same</u> hash function**
- **Requires a single pass through each file to hash into the same buckets**
- **Records from each file with equal join attributes will hash to the same bucket**
- **O(n) for n records in largest file**

Cost of JOIN operation

- Although we collect records with equal join attributes, it's actually disk block reads that count
- Strategy matters -consider brute force nested loop with

DEPARTMENT	EMPLOYEE
rD = 50 records	rE = 5000 records
bD = 10 disk blocks	bE = 2000 disk blocks

nb = 6 blocks buffer in main memory

Cost of JOIN operation

EMPLOYEE $\otimes$ (DNO = DNUMBER) DEPARTMENT

- For each t in EMPLOYEE {
  For each s in DEPARTMENT {
  if (t[DNO] = s[DNUMBER])
  result := result union (t union s)
  }
  }
- Strategy: read nb - 1 blocks from outer loop, then read through the entire inner loop file one block at a time
- Repeat for next nb-1 outer file blocks, etc.

Cost of JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

- **For each t in EMPLOYEE {**
  For each s in DEPARTMENT {
  if (t[DNO] = s[DNUMBER])
  result := result union (t union s)
  }
  }
- **Number blocks read for EMPLOYEE = bE = 2000**
- **Number of times nb-1 blocks are read**
  = bE/(nb-1) = 2000/5 = 400
- **Total number of blocks read for inner file**
  =bD * bE/(nb-1) = 10 * 400 = 4000
- **Total number block access**
  =bE + bE/(nb-1) * bD = 2000 + 2000/5*10 = 6000

Cost of JOIN operation

EMPLOYEE ⊗ (DNO = DNUMBER) DEPARTMENT

- **For each s in DEPARTMENT {**
  For each t in EMPLOYEE {
  if (t[DNO] = s[DNUMBER])
  result := result union (t union s)
  }
  }
- **Number blocks read for DEPARTMENT = bD = 10**
- **Number of times nb-1 blocks are read**
  = bD/(nb-1) = 10/5 = 2
- **Total number of blocks read for inner file**
  =bE * bD/(nb-1) = 2000 * 2 = 4000
- **Total number block access**
  =bD + bD/(nb-1) * bE = 10 + 10/5*2000 = 4010

- **Implementation Rule:** **Use the smaller file in the outer loop of a nested-loop JOIN**

...Implementing the JOIN operation

DEPARTMENT ⊗ (MGRSSN = SSN) EMPLOYEE

- **Assume secondary indices exist on** SSN **in** EMPLOYEE **and** MGRSSN **in** DEPARTMENT
- **index level** xSSN = 4, **index level** xMGRSSN = 2
- **OPTION 1 Retrieve each record** e **from** EMPLOYEE **and use index to test** DEPARTMENT **for records whose** MGRSSN **match** e[SSN]
- **OPTION 2 Retrieve each record** d **from** DEPARMENT **and use index to test** EMPLOYEE **for records whose** SSN **match** d[MGRSSN]

...Implementing the JOIN operation

DEPARTMENT ⊗ (MGRSSN = SSN) EMPLOYEE

- **OPTION 1 Retrieve each record** e **from** EMPLOYEE **and use index to test** DEPARTMENT **for records whose** MGRSSN **match** e[SSN]

- **total block accesses is approx.**
  = bE + (rE * (xMGRSSN + 1))

  = 2000 + (5000 * (2 + 1) = 17,000 block accesses

  Note: most EMPLOYEE records (4050 out of 5000) won't match join condition

...Implementing the JOIN operation

DEPARTMENT ⊗ (MGRSSN = SSN) EMPLOYEE

- **OPTION 2 Retrieve each record** d **from** DEPARMENT **and use index to test** EMPLOYEE **for records whose** SSN **match** d[MGRSSN]

- **total block accesses is approx.**
  = bD + (rD * (xSSN + 1))

  = 10 + (50 * (4 + 1) = 260 block accesses

  Note: every DEPARTMENT record will match some employee record

- **Implementation Rule:** For join using index structure

- Use either the smaller file, or the file that has a match for every record (high selection factor) in the outer loop

- If can be worthwhile for the database to create an index just to do the join

Optimization based on Heuristics

- **Heuristic: Apply** SELECT **and** PROJECT **operations before applying** JOIN
- **Both SELECT and PROJECT reduce the size of the files the JOIN must work with, and hence the result**

Example Query

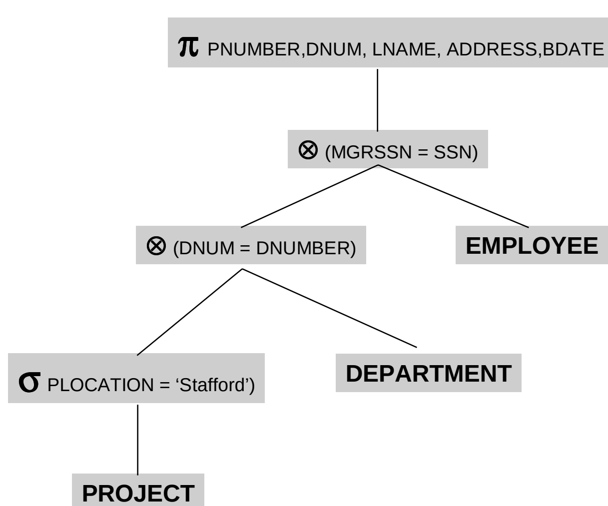
- **For every project in ‘Stafford’ retrieve the project number, controlling department number and name, address and birthdate of the department manager**

$\pi_{\text{PNUMBER,DNUM, LNAME, ADDRESS,BDATE}}$ (((

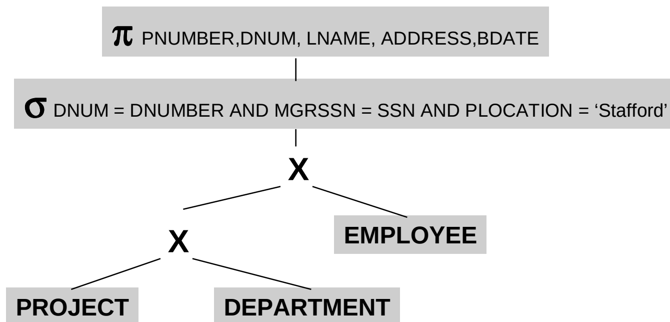
$\sigma_{\text{PLOCATION = 'Stafford')}}$ **(**PROJECT**))** $\otimes_{\text{(DNUM = DNUMBER(}}$
(DEPARTMENT) $\otimes_{\text{(MGRSSN = SSN(}}$ (EMPLOYEE))

SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER **AND**
MGRSSN=SSN **AND**
PLOCATION = ‘Stafford’

Query Tree (From relational Algebra)



Equivalent Canonical Query Tree

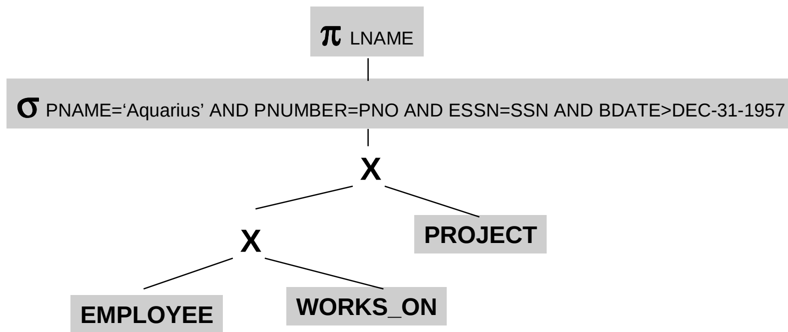


- **Canonical Tree is easy to generate but is very inefficient**
- **For PROJECT(100 tuples by 100 bytes)**
  DEPARTMENT(50 tuples by 50 bytes)
  EMPLOYEE(5000 tuples by 150 bytes)
- **Cartesian product would be: 10,000,000 by 300**

Example Query

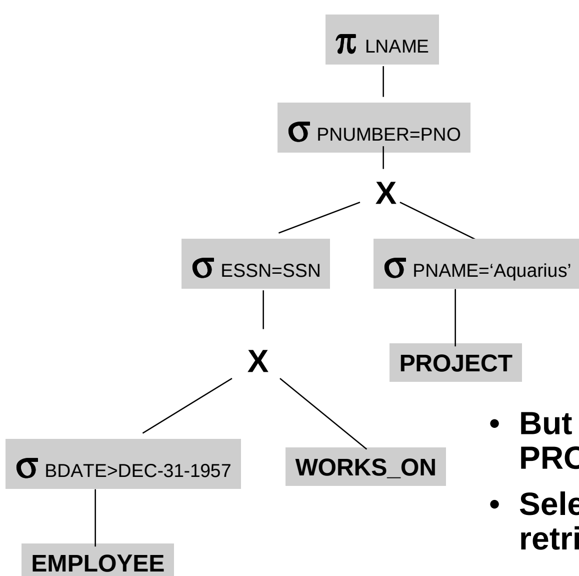
- **Find the last names of employees born after 1957 who work on project "Aquarius"**
- **SELECT** LNAME
  FROM EMPLOYEE, WORKS_ON, PROJECT
  WHERE PNAME = 'Aquarius' **AND**
  PNUMBER = PNO **AND**
  ESSN = SSN **AND**
  BDATE > 'DEC-31-1957'

Equivalent Canonical Query Tree



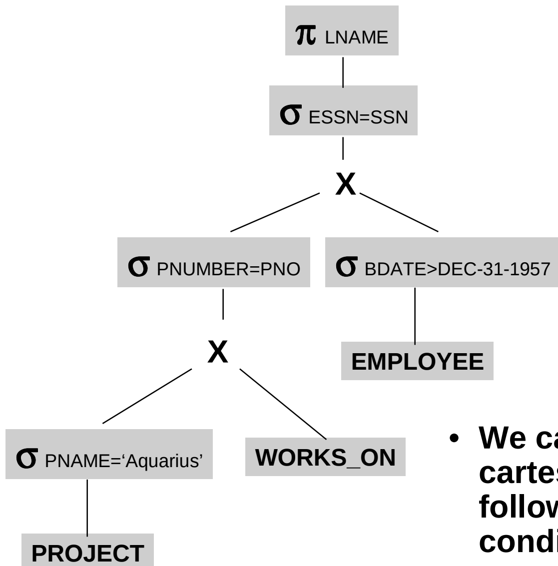
- But we need only one project record -for ‘Aquarius’
- We need only those employees with BDATE>DEC-31-1957
- We can produce an equivalent, more efficient, tree

Equivalent Query Tree



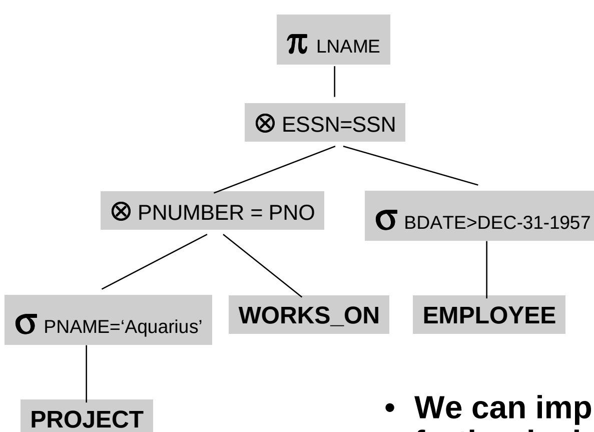
- But PNUMBER is key in PROJECT
- Select on PROJECT will retrieve a single record

Equivalent Query Tree



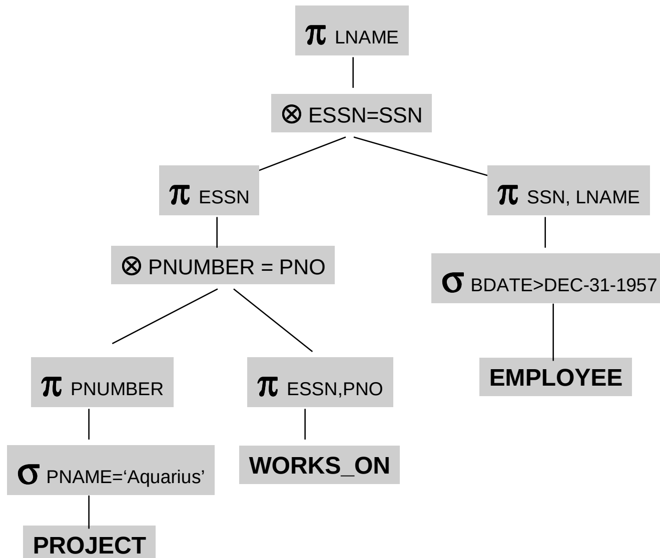
- **We can replace and cartesian product followed by a join condition with a JOIN**

Equivalent Query Tree



- **We can improve things further by keeping only the attributes we need**
- **i.e. do projects early**

Equivalent Final Query Tree



Optimization based on Cost Estimates

- **Optimizer can also compare various execution strategies by computing an estimated cost and picking the best**
- **Optimizer should not spend more time estimating costs than it would take to execute the actual query**

Query Execution Costs

- Access Cost: cost of searching for, reading, and writing disk blocks (Main cost)
- Storage Cost: cost of storing any temporary files generated by the execution strategy
- Computation Cost: cost of performing in-memory operations on data
- Communication Cost: Cost of shipping the query and its results from the database to client

Cost meta-data

- To be able to estimate cost of queries we need to keep statistics about the actual data (meta-data)
- meta-data includes
  r -> number of tuples in table
  b -> number of blocks in table
  bfr -> blocking factor (records per block)

  x -> number of levels of index
  bl1 -> number of first level index blocks
  d -> number of distinct data values for index attr.
  s = r/d -> average number of index tuples with same value (selection cardinality)

Cost Estimates for SELECT Strategies

- Linear Search: Cost ~ b/2
- Binary Search: Cost ~ log2(b) + s/bfr - 1
- Using Primary Key or Hash Key
  Cost ~ x + 1, (Cost ~1 for Hashing)
- Using Ordering Index for multiple records
  Cost ~ x + b/2
- Using a clustering index Cost ~ x + s/bfr
- Using a Secondary B+ tree index
  Cost ~ x + s (for equality test)
  Cost ~ x + bl1/2 + r/2 (for range query)
- Conjunctive Selection
  Same is Linear, Binary or B+ tree
- Conjunctive with composite index
  Same as Primary Index or B+ tree