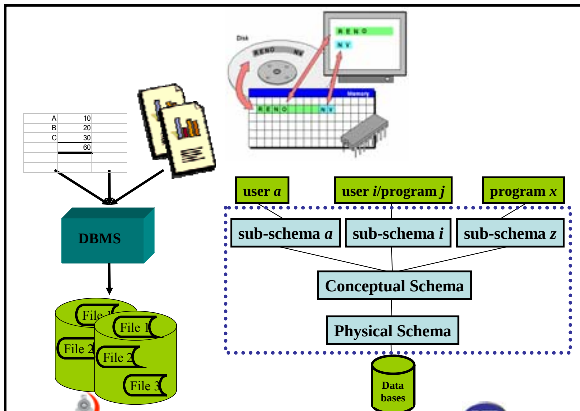
Data Storage and Querying

Dr G.N.Wikramanayake
Dr Jeevani Goonetillake
University of Colombo School of Computing

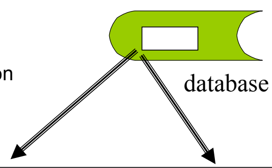
 BIT



 BIT

Physical View

- The DBMS must know
  - exact physical location
  - precise physical structure



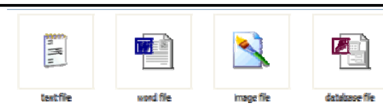
Employee record

A.B.C. De Silva	222, Galle Road, Colombo
Name (20 characters)	**Address (40 characters)**

650370690V	Senior Lecturer
NID (10 char)	**Designation (15 char)**

 BIT

File



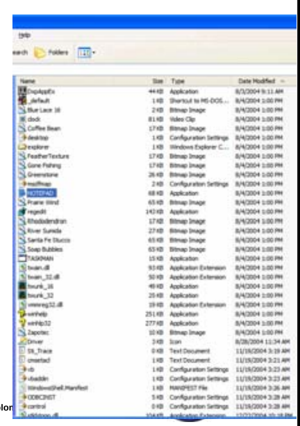
- A collection of data records with similar characteristics
- Student list, course list at the university
- price list, stock records at a supermarket

Item#	Description	Shelf#	Max Stock	Reorder	Balance
1152	Milk	D01	100	25	70
1167	Bread	D01	70	25	30
1175	Sugar	G02	300	50	120
1172	Rice	G04	50	20	35

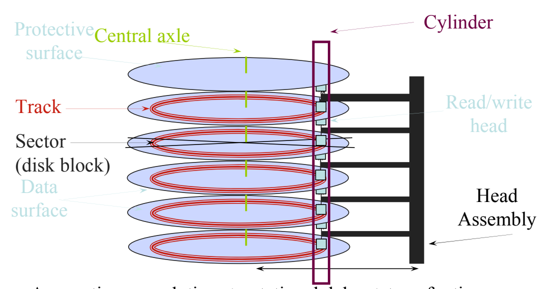
 BIT

Physical Design

- Provide good performance
  - Fast response time
  - Minimum disk accesses



Disk Assembly



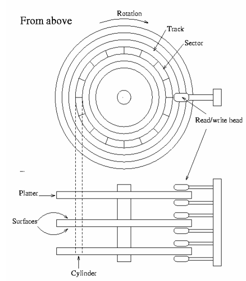
Access time = seek time + rotational delay + transfer time

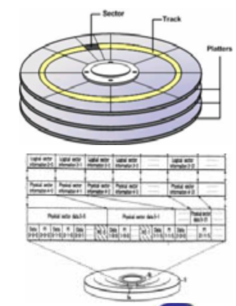
 BIT

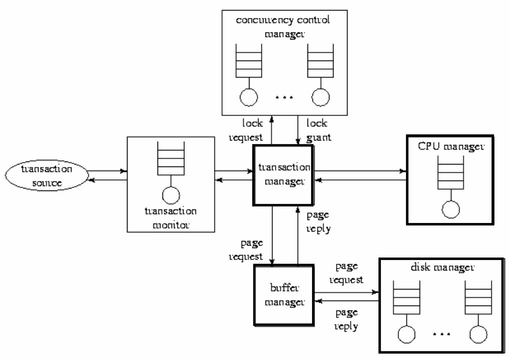
Disks

- 6 disks (platters); 12 surfaces; 2 outer protected surfaces; 10 inner data surfaces (coated with a magnetic substance to record data)
- Each surface 200-400 concentric tracks
- Read/write heads placed over specific track; with one active at a time
- Set of corresponding tracks is a cylinder, i.e. track I of all 10 surfaces

Disks







Data Block

- A data block (sector) is the smallest unit of data defined within the database.
- block size may be defined by the DB_BLOCK_SIZE

Definitions

- Seek time
  - Average time to move the read-write head to the correct cylinder
- Rotational delay
  - Average time for the sector to move under the read-write head
- Transfer time
  - Time to read a sector and transfer the data to memory

More Definitions

- Logical Record
  - The data about an entity (a row in a table)
- Physical Record
  - A sector, page or block on the storage medium
- Typically several logical records can be stored in one physical record.

File Organization Techniques

- Three techniques
  - Serial or Heap (unordered)
  - Sorted
    - Sequential (SAM)
    - Indexed Sequential (ISAM)
  - Hashed or Direct or Random

Serial

- Used for temporary files such as transaction files, dump files

Transaction#	***Item#***	Description	Quantity
101	1152	Milk	01
101	1167	Bread	02
102	1167	Bread	01
102	1175	Sugar	01
103	1172	Rice	01
103	1152	Milk	01

Heap File

Tony Lama was the first record added, Digital was the last.

ID	Company	Industry	Symbl.	Price	Earns.	Dividnd.
1767	Tony Lama	Apparel	TONY	45.00	1.50	0.25
1152	Lockheed	Aero	LCH	112.00	1.25	0.50
1175	Ford	Auto	F	88.00	1.70	0.20
1122	Exxon	Oil	XON	46.00	2.50	0.75
1231	Intel	Comp.	INTL	30.00	2.00	0.00
1323	GM	Auto	GM	158.00	2.10	0.30
1378	Texaco	Oil	TX	230.00	2.80	1.00
1245	Digital	Comp.	DEC	120.00	1.80	0.10

Heap File Characteristics

- Insertion
  - Fast: New records added at the end of the file
- Retrieval
  - Slow: A sequential search is required
- Update - Delete
  - Slow:
    - Sequential search to find the page
    - Make the update or mark for deletion
    - Re-write the page

Sequential

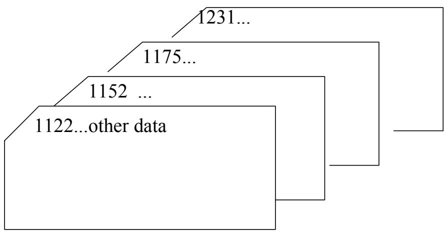
- Records are recoded in key sequence, but have no index
- Used for master files in a normal batch processing

Item#	Description	Price
1152	Milk	35.00
1167	Bread	30.00
1172	Rice	60.00
1175	Sugar	50.00

Sequential (Ordered) File

ID	Company	Industry	Symbl.	Price	Earns.	Dividnd.
1122	Exxon	Oil	XON	46.00	2.50	0.75
1152	Lockheed	Aero	LCH	112.00	1.25	0.50
1175	Ford	Auto	F	88.00	1.70	0.20
1231	Intel	Comp.	INTL	30.00	2.00	0.00
1245	Digital	Comp.	DEC	120.00	1.80	0.10
1323	GM	Auto	GM	158.00	2.10	0.30
1378	Texaco	Oil	TX	230.00	2.80	1.00
1480	Conoco	Oil	CON	150.00	2.00	0.50
1767	Tony Lama	Apparel	TONY	45.00	1.50	0.25

Sequential Access

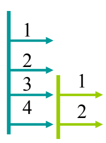


Sequential File Characteristics

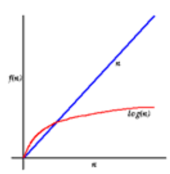
- Older media (cards, tapes)
- Records physically ordered by primary key
- Use when direct access to individual records is not required
- Accessing records
  - Sequential search until record is found
- Binary search can speed up access
  - Must know file size and how to determine mid-point,

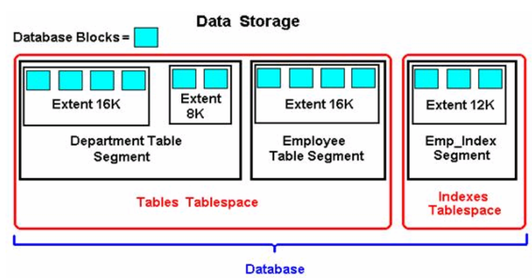
Search Sugar

- Sequence of file based on Description
- 4 sequential accesses to reach sugar
- 2 binary search accesses to reach sugar



Description	Price
Bread	30.00
Milk	35.00
Rice	60.00
Sugar	50.00
Water	20.00





Database Blocks are allocated to extents as specified in data storage clauses. Each object (table, index, cluster) is allocated a single segment which is comprised of one or more extents. Segments are stored to Tablespaces as determined by the DBA when the object is created. All of the Tablespaces taken together comprise the Database.

Inserting Records in SAM files

- Insertion
  - Slow:
    - Sequential search to find where the record goes
    - If sufficient space in that page, then rewrite
    - If insufficient space, move some records to next page
    - If no space there, keep bumping down until space is found
  - May use an “overflow” file to decrease time

Deletions and Updates to SAM

- Deletion
  - Slow:
    - Find the record
    - Either mark for deletion or free up the space
    - Rewrite
- Updates
  - Slow:
    - Find the record
    - Make the change
    - Rewrite

Binary Search to Find GM (1323)

	ID	Company	Industry	Symbl.	Price	Earns.	Dividnd.
	1122	Exxon	Oil	XON	46.00	2.50	0.75
	1152	Lockheed	Aero	LCH	112.00	1.25	0.50
	1175	Ford	Auto	F	88.00	1.70	0.20
	1231	Intel	Comp.	INTL	30.00	2.00	0.00
1.	1245	Digital	Comp.	DEC	120.00	1.80	0.10
3.	1323	GM	Auto	GM	158.00	2.10	0.30
2.	1378	Texaco	Oil	TX	230.00	2.80	1.00
	1480	Conoco	Oil	CON	150.00	2.00	0.50
	1767	Tony Lama	Apparel	TONY	45.00	1.50	0.25

- Takes 3 accesses as opposed to 6 for linear search.

Indexed Sequential

- Disk (usually)
- Records physically ordered by primary key
- Index gives physical location of each record
- Records accessed sequentially or directly via the index
- The index is stored in a file and read into memory when the file is opened.
- Indexes must be maintained

Indexed Sequential Access

- Given a value for the key
  - search the index for the record address
  - issue a read instruction for that address
  - Fast: Possibly just one disk access

Indexed Sequential Access: Fast

Find record with key 777-13-1212

Key	Cyl.	Trck	Sect.
279-66-7549	3	10	2
452-75-6301	3	10	3
789-12-3456	3	10	4

777-13-1212 < 789-12-3456
Search Cyl. 3, Trck 10, Sect. 4
sequentially.

222-66-7634 255-75-5531 279-66-7549
333-88-9876 382-32-0658 452-75-6301
701-43-5634 777-13-1212 789-12-3456

Inserting into ISAM files

- Not very efficient
  - Indexes must be updated
  - Must locate where the record should go
  - If there is space, insert the new record and rewrite
  - If no space, use an overflow area
  - Periodically merge overflow records into file

Deletion and Updates for ISA

- Fairly efficient
  - Find the record
  - Make the change or mark for deletion
  - Rewrite
  - Periodically remove records marked for deletion

Use ISAM files when:

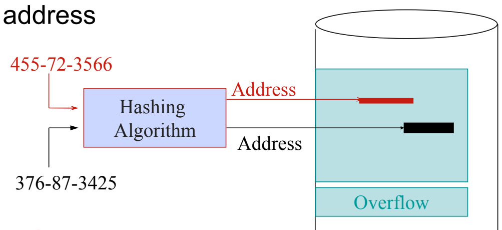
- Both sequential and direct access is needed.
- Say we have a retail application like Foley's.
- Customer balances are updated daily.
- Usually sequential access is more efficient for batch updates.
- But we may need direct access to answer customer questions about balances.

Random

- Randomly organized file contains records stored without regard to the sequence of their control fields.
- Records are stored in some convenient order establishing a direct link between the key of the record and the physical address of that record

Direct or Hashed Access

- A portion of disk space is reserved
- A "hashing" algorithm computes record address



Hashed Access Characteristics

- No indexes to search or maintain
- Very fast direct access
- Inefficient sequential access
- Use when direct access is needed, but sequential access is not
- Data cannot be sorted easily

Query Optimization

A query typically has many possible execution strategies and the process of choosing a suitable one for processing a query is known as query optimisation.

The job of the heuristic query optimiser is to transform this initial query tree into a final query tree that is efficient to execute.

The optimiser must include rules for equivalence among relational algebra expressions that can be applied to the initial tree, guided by the heuristic query optimisation rules to produce the final optimised query tree.

Execution strategy - The DBMS must then devise an execution strategy for retrieving the result of the query from the internal database files.

Query code generator - According to the chosen execution plan, the code generator generates the code to execute the plan.

Runtime database processor - The runtime database processor has the task of running the query code (compiled or interpreted) to produce the query result. If a runtime error occurs the runtime database processor generates an error message.

Translating SQL Queries into Relational Algebra

SELECT p.pno, d.dno, e.ename
FROM Project as p, Department as d, Employee as e
WHERE d.dno=p.dept and d.mgr=e.empno and p.location='Colombo';

T1 $\leftarrow$ Project $\infty_{dno=dept}$ Department
T2 $\leftarrow$ T1 $\infty_{mgr=empno}$ Employee
T3 $\leftarrow$ $\sigma_{location='Colombo'}$(T2)
Result $\leftarrow$ $\pi_{pno,\ dno,\ ename}$(T3)

JOIN Project and Department over dno=dept giving T1
JOIN T1 and Employee over mgr=empno giving T2
RESTRICT T2 where location='Colombo' giving T3
PROJECT T3 over pno, dno, ename giving Result

Queries in Relational Algebra

Equivalent Queries

(a) Result $\leftarrow$ $\pi_{pno,\ dno,\ ename}$ ($\sigma_{location='Colombo'}$ ((Project $\infty_{dno=dept}$ Department) $\infty_{mgr=empno}$ Employee))

(b) T1 $\leftarrow$ $\sigma_{location='Colombo'}$(Project)
T2 $\leftarrow$ T1 $\infty_{dno=dept}$ Department
T3 $\leftarrow$ T2 $\infty_{mgr=empno}$ Employee
Result $\leftarrow$ $\pi_{pno,\ dno,\ ename}$(T3)

Basic algorithms for executing query operations

- Each DBMS typically has a number of general database access algorithms that implement relational operations such as SELECT or JOIN or combinations of these operations.
- The query optimisation module will consider only execution strategies that can be implemented by the DBMS access algorithms (i.e. storage structures and access paths).

Sorting is one of the primary algorithms used in query processing.

For example, whenever an SQL query specifies an ORDER BY clause, the query result must be sorted.

Sorting is also a key component in sort-merge algorithms used for JOIN and other operations (such as UNION and INTERSECTION), and in duplicate elimination algorithms for the PROJECT operation (when an SQL query specifies the DISTINCT option in the SELECT clause).

There are many options for executing a SELECT operation.

A number of searching algorithms are possible for selecting records from a file.

Linear search (brute force), binary search, using a primary index or hash key, clustering index and using secondary index (B-tree) on an equality comparison are examples of searching.

Selecting Records, e.g.

- Primary index (records ordered on a key field)
  T1 $\leftarrow$ $\sigma_{empno='12345'}$(Employee) (single record)
  T1 $\leftarrow$ $\sigma_{dno>='5'}$(Department) (multiple records)
- Clustering Index (records ordered on a non key field)
  T1 $\leftarrow$ $\sigma_{dno='5'}$(Employee) (multiple records)
- B$^+$-Tree Index (secondary index on equality comparison)
  T1 $\leftarrow$ $\sigma_{salary>=30000 \text{ and } salary<=35000}$(Employee) (multiple records)

Using Heuristics in Query Optimisation

- There are two main techniques for query optimisation: heuristic rules and systematic estimating.
- Heuristic rules are used to order the operations in a query execution strategy. The rules typically reorder the operations in a query tree or determine an order for executing the operations specified by a query graph.
- Systematic estimating is used to cost the different execution strategies and to choose the execution plan with the lowest cost estimate.
- Both strategies are usually combined in a query optimiser.

Transformation rules for relational algebra operations

- There are many rules for transforming relational algebra operations into equivalent ones.
- These are in addition to those discussed under relational algebra.
- These rules are used in heuristic optimisation.
- Algorithms that utilise these rules are used to transform an initial query tree into an optimised tree that is more efficient to execute.
- Here we look at some examples that demonstrate such transformations.

Rule 1 (cascade of σ)

Break up any SELECT operations (σ) with conjunctive conditions (AND) into a cascade (sequence) of individual SELECT operations.

$$\sigma_{c1 \text{ AND } c2 \text{ AND } \ldots \text{ AND } cn} (R) \equiv \sigma_{c1} (\sigma_{c2} (\ldots(\sigma_{cn} (R))\ldots))$$

This permits a greater degree of freedom in moving SELECT operations down different branches of the tree.

Rule 2 (commutative of σ)

The SELECT operation is commutative

$$\sigma_{c1} (\sigma_{c2} (R)) \equiv \sigma_{c2} (\sigma_{c1} (R))$$

Rule 3 (commutative of σ with π)

If the SELECT condition c involves only attributes a1, a2, …, an in the PROJECTION list, the two operations can be commuted.

$$\pi_{a1,\, a2,\, \ldots,\, an} (\sigma_c (R)) \equiv \sigma_c (\pi_{a1,\, a2,\, \ldots,\, an} (R))$$

Rule 4 (commutative of σ with X or ∞)

If all the attributes in the selection condition c involve only the attributes of one of the relations being joined (say R) the two operations can be commuted as

$$\sigma_c (R \infty S) \equiv (\sigma_c (R)) \infty S$$

Alternatively if the selection condition c can be written as c1 and c2, where c involves only the attributes of S, the operations commute as

$$\sigma_c (R \infty S) \equiv (\sigma_{c1} (R)) \infty (\sigma_{c2} (S))$$

The same rules apply if the ∞ is replaced by a X operation.

Rule 5 (commuting σ with set operations)

The σ operation commutes with $\cup$, $\cap$ and $-$. If θ stands for any one of these 3 operations then

$$\sigma_c (R \theta S) \equiv (\sigma_c (R))\ \theta\ (\sigma_c (S))$$

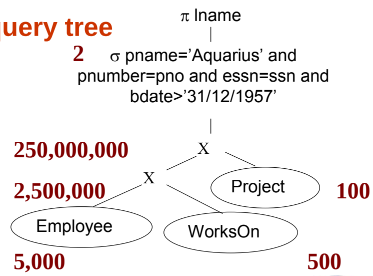
Using rules 2, 3, 4, and 5 concerning the commutative of SELECT with other operations, move each SELECT operation as far down the query tree as is permitted by the attributes involved in the select condition.

The objective is to reduce the number of tuples that appear in the Cartesian product.

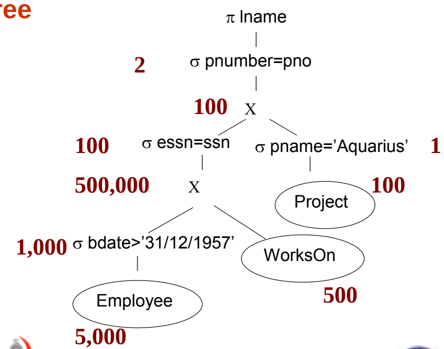
SELECT lname
FROM Employee, WorksOn, Project
WHERE pname='Aquarius' and pnumber=pno and essn=ssn and bdate>'31/12/1957';

(a) initial query tree

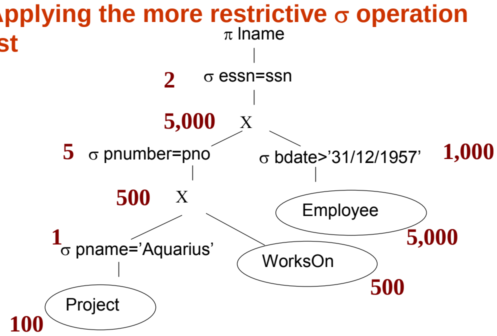
Depending on the capacity of the memory intermediate records may have to be written to disk. Thus increase total Read/Write operations.



(b) Moving σ operation down the query tree

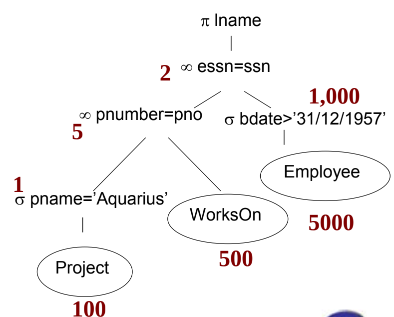


(c) Applying the more restrictive σ operation first

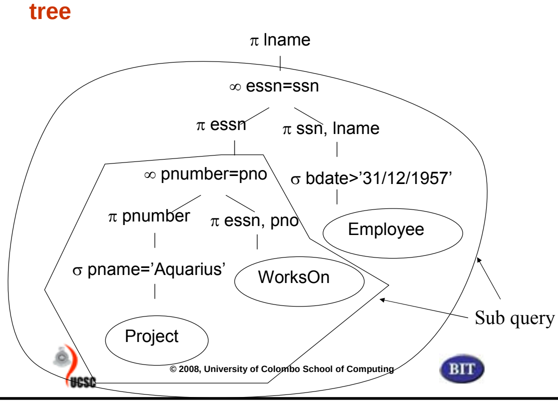


Use the information that pname is a unique attribute of Project relation.

(d) Replacing Cartesian product and Select with Join operations



(e) Moving Project operation (π) down the query tree



Using Selectivity and Cost Estimates in Query Optimisation

A query optimiser should not depend solely on heuristic rules, it should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate.

- Cost components
- Cost functions
- Examples

Cost Components for Query Execution

- Access cost to secondary storage
  - Cost of searching for, reading and writing data blocks that reside on secondary storage.
  - Cost of searching depends on access file structures such as ordering, hashing, indexes. Also based on how data are allocated on disk.
- Storage cost
  - Cost of storing any intermediate files that are generated by an execution strategy for the query

Cost Components …

- Computation cost
  - Cost of performing in-memory operations on the data buffers during query execution. E.g. searching, sorting, merging for join, performing computations on field values.
- Memory usage cost
  - Cost pertaining to the number of memory buffers needed during query execution.
- Communication cost
  - Cost of sending the query and its results from database server to client.

Catalog Information used in Cost Functions

- n_r - number of tuples in relation r
- s_r - size of tuple in relation r
- b_r - number of blocks containing tuples of r
- f_r - blocking factor – number of tuples of relation r that fit into one block
- x_a - number of levels of each multilevel index on attribute a .
- $d_{a,r}$ - number of distinct values in relation r for attribute a. $d_{a,r} = n_r = \pi_a(r)$ if a is unique
- $s_{a,r}$ - Selection cardinality of attribute a of relation r – average number records satisfying equality condition. If a is unique $d_{a,r} = n_r$, $s_{a,r} = 1$ else (if uniformly distributed), $s_{a,r} = n_r/d_{a,r}$

Cost Functions for SELECT

- Linear Search
  - Retrieve all file blocks; cost = b_r
  - For equal condition on average cost = $b_r/2$ if found else cost = b_r
- Binary Search
  - Search access cost = $\log_2 b_r + (s_{a,r}/ f_r) – 1$
  - If unique attribute average cost = $\log_2 b_r$
- Using a primary / secondary key index
  - Cost = $x_a + 1$
- Using a hash key
  - Cost ≈ 1