University of Windsor

07 Indexing and Query optimization - Part I

Kalyani Selvarajah
School of Computer Science
University of Windsor

Advanced Database Topics
COMP 8157 01/02/03
Fall 2023

Today's Agenda

Organizing the data on the disk

Introduction to Indexing

Query Execution Plan (Part II)

https://raima.com/raima-database-manager/

Introductory Questions

Why index is important ?

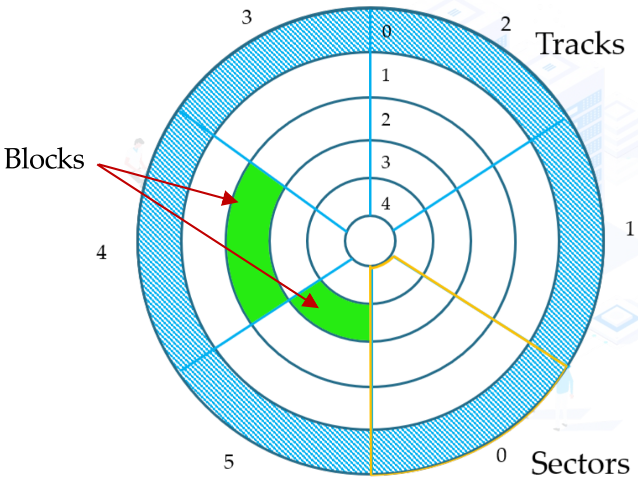
What are the downsides of indexes ?

What is query optimization ?

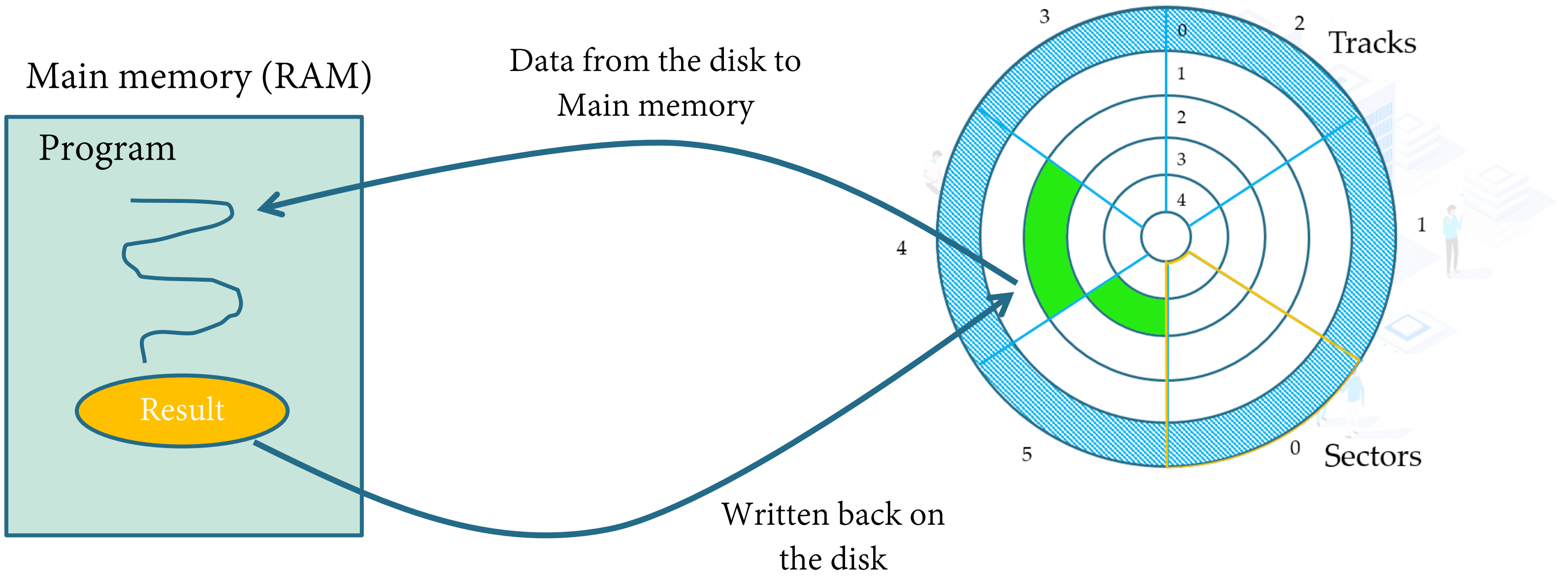
Disk Structure

$$Block\ Address\ = \langle Track\ Num, Sector\ Num\rangle$$

Let's assume a block size is 512 bytes

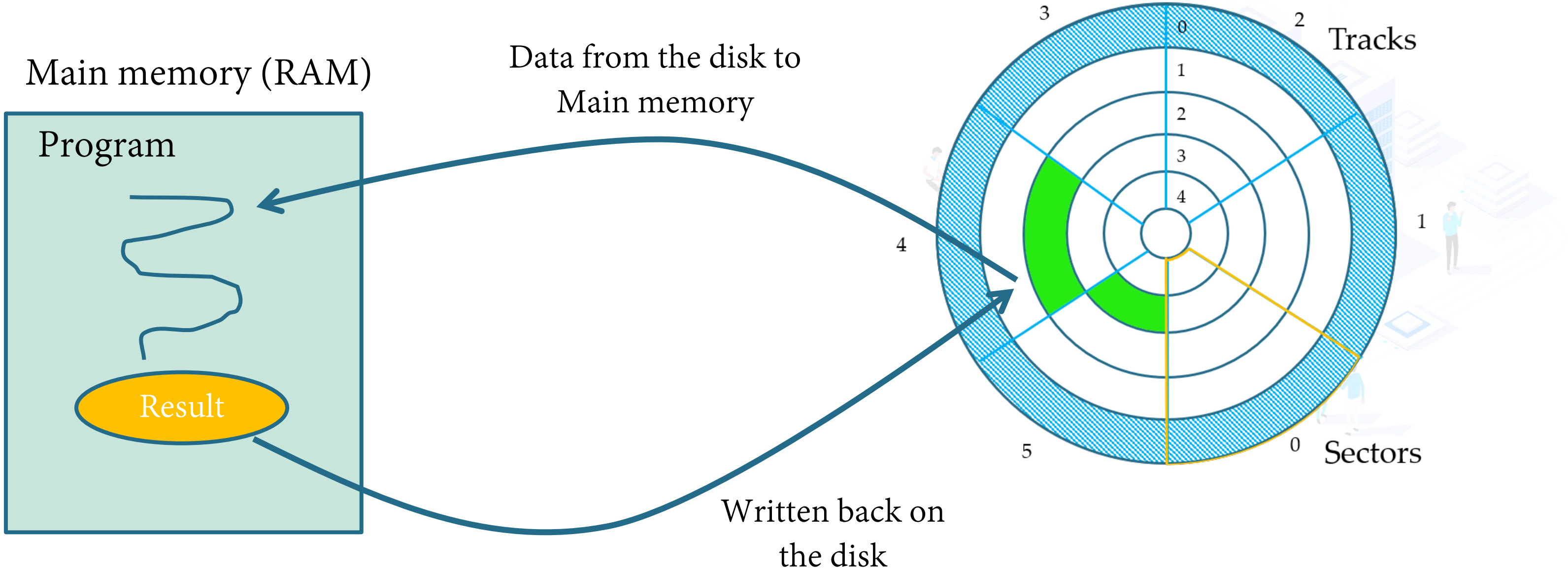


Disk Structure



So, the data cannot be directly processed upon the disk it has to be brought into the main memory and then access .

Disk Structure



Organizing the data inside the main memory that is directly used by the program is **Data Structures**.

Organizing the data on the disk efficiently so that it can be easily utilized that is **DBMS.**

How is data stored on Disk?

Employees

Fields	**Size**
Eid	10
Name	50
Depart	30
Gender	8
Salary	30
Total size	**128 bytes**

In each block how many rows can be stored?

Number of Records/ Block = 512/128

= 4

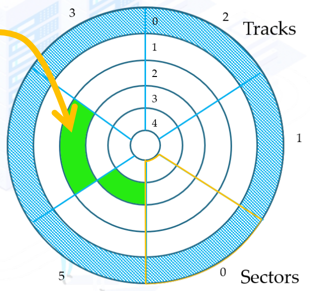
128 bytes

Employees

Eid	**Name**	**Depart**
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager
⋮	⋮	⋮

100 records

What is the size of a record?



Block Size = 512 bytes

How is data stored on Disk?

Employees

Fields	**Size**
Eid	10
Name	50
Depart	30
Gender	8
Salary	30
Total size	**128 bytes**

SELECT Eid=7
FROM Employees

How many blocks are required for 100 records?

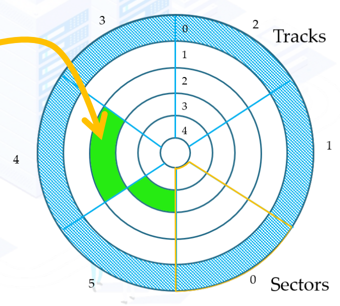
Number of Records/ Block = 512/128
= 4

100 Records can be stored in = 100/4
= **25 blocks**

Employees

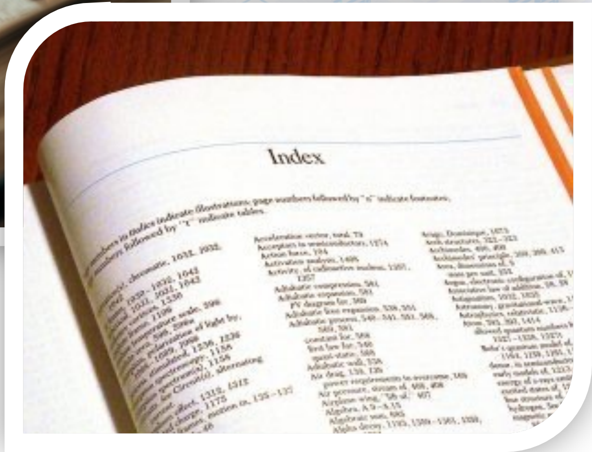
Eid	**Name**	**Depart**	....
1	Jenkins	Manager	
2	Williams	Sales Rep	
3	Smith	Sales Rep	
4	Crosby	Manager	
5	Albright	Secretary	
6	Sawyer	Sales Rep	
7	Thomas	Secretary	
8	Albright	Worker	
9	Crawford	Manager	
⋮	⋮	⋮	

100 records



Block Size = 512 bytes

What is Indexing?



It makes our search simpler and quicker.

- ✓ How do we create the indexes ?
- ✓ How these indexes help to access the data ?

What is Indexing?

Where do we store the index?

<K(i), P(i)>

Index

Key	Pointer
1	
2	
3	
4	
5	

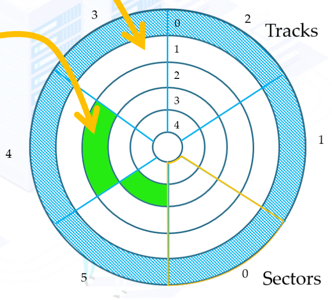
100 records

Dense Index

Employees

Eid	Name	Depart	....
1	Jenkins	Manager	
2	Williams	Sales Rep	
3	Smith	Sales Rep	
4	Crosby	Manager	
5	Albright	Secretary	
6	Sawyer	Sales Rep	
7	Thomas	Secretary	
8	Albright	Worker	
9	Crawford	Manager	

100 records



Block Size = 512 bytes

What is Indexing?

<K(i), P(i)>

Index

Fields	Size
Eid	10
Pointer	6
Total size	**16 bytes**

Index

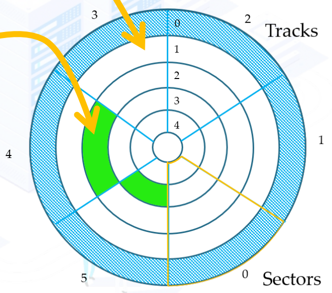
Key	Pointer
1	
2	
3	
4	
5	

100 records

Employees

Eid	Name	Depart
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager

100 records



Block Size = 512 bytes

Number of Records/ Block = 512/16
= 32

100 Records can be stored in = 100/32
≅ 4 blocks

What is Indexing?

<K(i), P(i)>

```
SELECT Eid=7
FROM Employees
```

Total number of blocks required = 4 +1

= **5 blocks**

100 Records can be stored in = 100/32

≅ **4 blocks**

Index

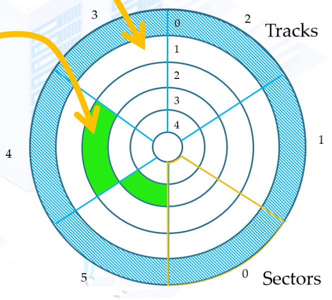
Key	Pointer
1	
2	
3	
4	
5	

100 records

Employees

Eid	Name	Depart
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager

100 records



Block Size = 512 bytes

What is indexing?

<K(i), P(i)>

Index

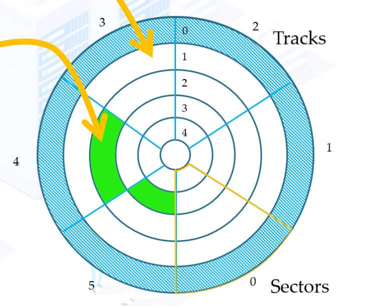
Key	Pointer
1	
2	
3	
4	
5	

1000 records

Employees

Eid	Name	Depart	....
1	Jenkins	Manager	
2	Williams	Sales Rep	
3	Smith	Sales Rep	
4	Crosby	Manager	
5	Albright	Secretary	
6	Sawyer	Sales Rep	
7	Thomas	Secretary	
8	Albright	Worker	
9	Crawford	Manager	

1000 records



Block Size = 512 bytes

What is multi level indexing?

<K(i), P(i)>

Index

Key	Pointer
1	
33	

Index

Key	Pointer
1	
2	
3	
4	
5	

1000 records

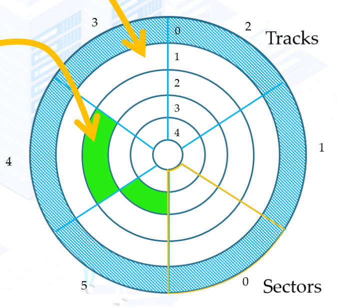
We require 40 blocks.

Employees

Eid	Name	Depart
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager

1000 records

We require 250 blocks



Block Size = 512 bytes

Can we have an index above an index?

What is multi level indexing?

<K(i), P(i)>

Index

Key	Pointer
1	
33	
65	

Sparse Index

Index

Key	Pointer
1	
2	
3	
4	
5	

32 entries

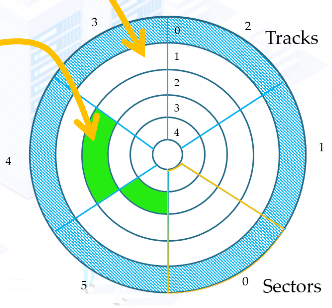
32 entries

40 blocks

Employees

Eid	Name	Depart
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager

1000 records



Block Size = 512 bytes

We require 250 blocks

Number of Records/ Block = 512/16

= 32

40 Records can be stored in = 40/32 = 1.25

≅ 2 blocks

What is multi level indexing?

<K(i), P(i)>

Index

Key	Pointer
1	
33	
65	

Sparse Index

Index

Key	Pointer
1	
2	
3	
4	
5	

32 entries

32 entries

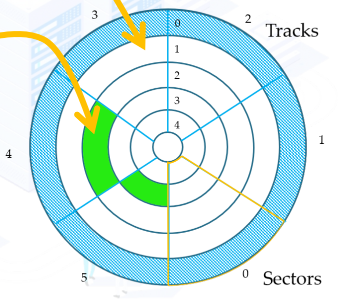
40 blocks

Employees

Eid	Name	Depart	....
1	Jenkins	Manager	
2	Williams	Sales Rep	
3	Smith	Sales Rep	
4	Crosby	Manager	
5	Albright	Secretary	
6	Sawyer	Sales Rep	
7	Thomas	Secretary	
8	Albright	Worker	
9	Crawford	Manager	
⋮	⋮	⋮	

1000 records

We require 250 blocks



Block Size = 512 bytes

SELECT Eid=7
FROM Employees

Number of Records/ Block = 512/16
= 32

40 Records can be stored in = 40/32 = 1.25
≅ **2 blocks**

What is multi level indexing?

<K(i), P(i)>

Index

Key	Pointer
1	
33	
65	

Index

Key	Pointer
1	
2	
3	
4	
5	

32 entries

32 entries

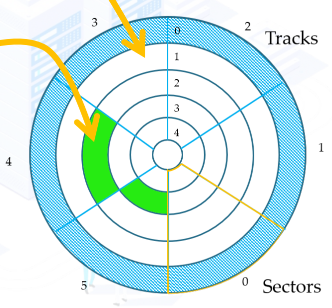
40 blocks

Employees

Eid	Name	Depart
1	Jenkins	Manager
2	Williams	Sales Rep
3	Smith	Sales Rep
4	Crosby	Manager
5	Albright	Secretary
6	Sawyer	Sales Rep
7	Thomas	Secretary
8	Albright	Worker
9	Crawford	Manager

1000 records

We require 250 blocks



Block Size = 512 bytes

Total number of blocks required =
2 +1 + 1
= **4 blocks**

Number of Records/ Block = 512/16
= 32

40 Records can be stored in = 40/32 = 1.25
≅ **2 blocks**

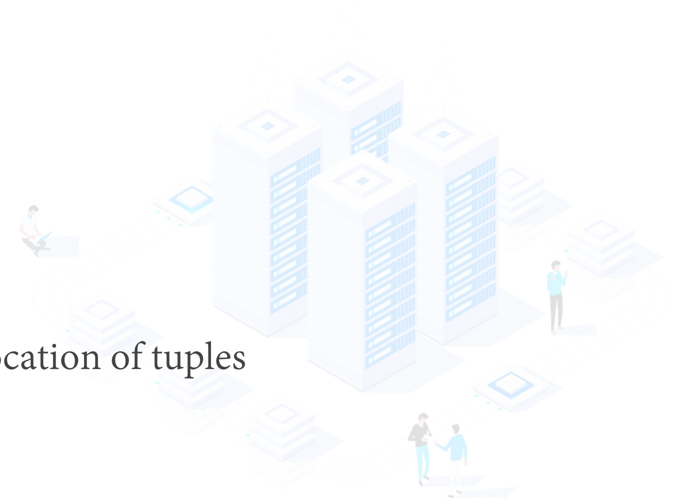
Indexes

Primary mechanism to get improved performance on a database

Persistent data structure, stored in database

Many interesting implementation issues

- focus on user/application perspective



Utility

- Index = difference between full table scans and immediate location of tuples
  - Orders of magnitude performance difference
- Underlying data structures
  - Balanced trees (B trees, B+ trees)
  - Hash tables

Indexes

Single-Key Index:

```
Select sName
From Student
Where sID = 18942
```

Many DBMS's build indexes automatically on **PRIMARY KEY** (and sometimes **UNIQUE**) attributes

Multiple Single-Key Indexes:

```
Select sID
From Student
Where sName = 'Mary' And GPA > 3.9
```

```
Select sName, uName
From Student, Apply
Where Student.sID = Apply.sID
```

Question:

Select *

From Student, Apply, University

Where Student.sID = Apply.sID and Apply.uName = University.uName And Student.GPA > 1.5 And University.uName = ‘UWindsor’

Suppose we are allowed to create two indexes, and assume all indexes are tree-based. Which two indexes do you think would be most useful for speeding up query execution?

1. Student.sID, University.uName
2. Student.sID, Student.GPA
3. Apply.uName, University.uName
4. Apply.sID, Student.GPA

Question:

Consider the following query:

Select * From Apply, University

Where Apply.uName = University.uName And Apply.major = 'CS' and University.enrollment < 5000

Which of the following indexes **could NOT** be useful in speeding up query execution ?

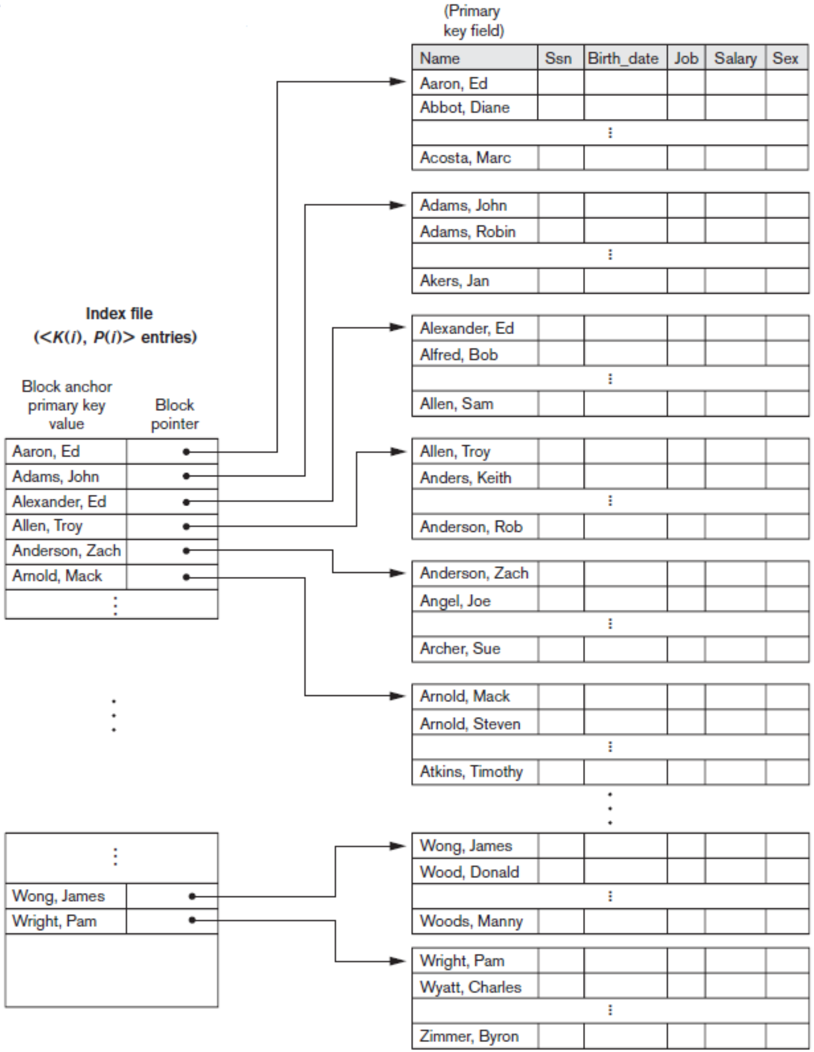
1. Tree-based index on Apply.uName
2. Hash-based index on Apply.major
3. Hash-based index on University.enrollment
4. Hash-based index on University.uName

Type of Indexing

1. Single-level Indexes:
    1. Primary index
    2. Clustering index
    3. Secondary index or non-clustering Index
2. Multilevel Indexes

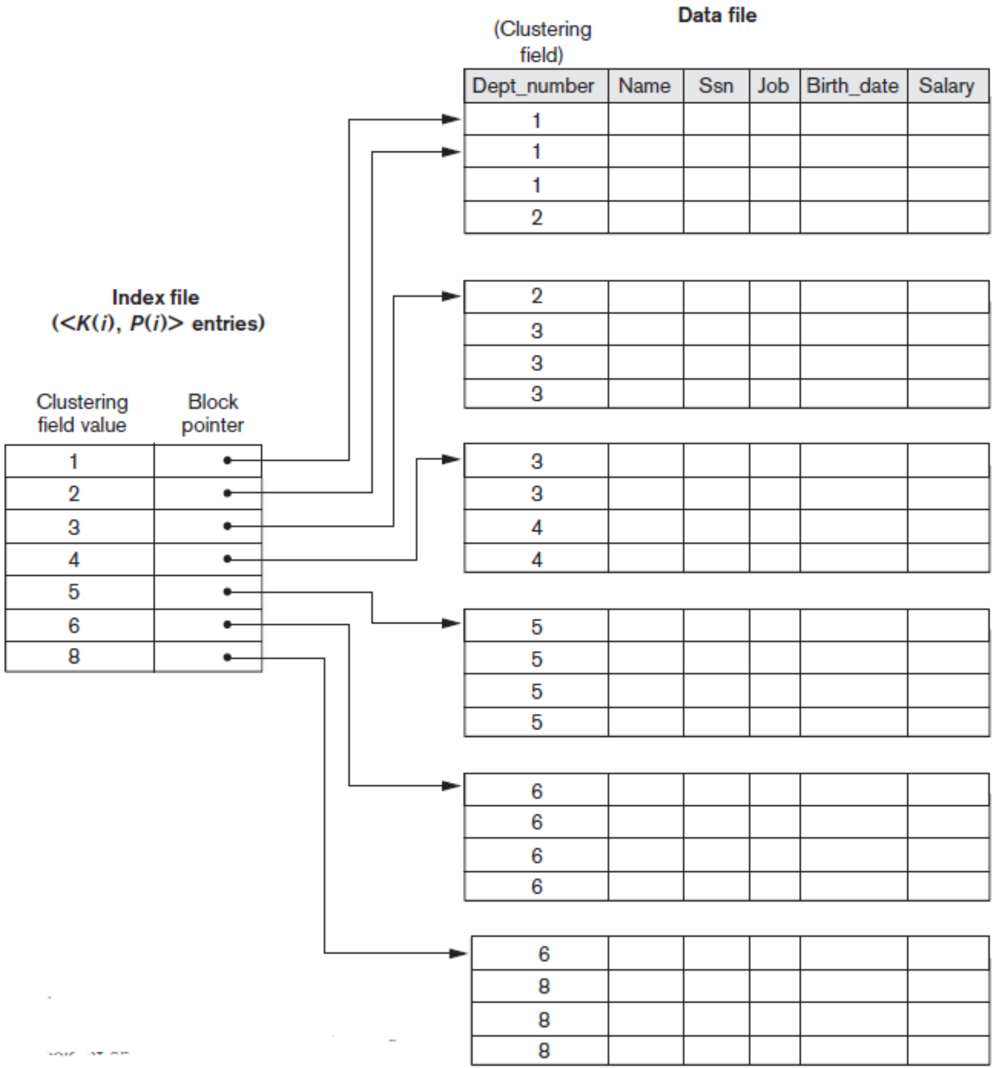
Primary Index

- A primary index is an ordered file whose records are of fixed length with two fields, and it acts like an access structure to efficiently search for and access the data records in a data file.
- The first field is of the same data type as the **ordering key field** (stored in some sorted order)—called the primary key—of the data file, and the second field is a pointer to a disk block (a block address).
- $<K(1) = (\text{Aaron, Ed}), P(1) = \text{address of block 1}>$
  $<K(2) = (\text{Adams, John}), P(2) = \text{address of block 2}>$
  $<K(3) = (\text{Alexander, Ed}), P(3) = \text{address of block 3}>$



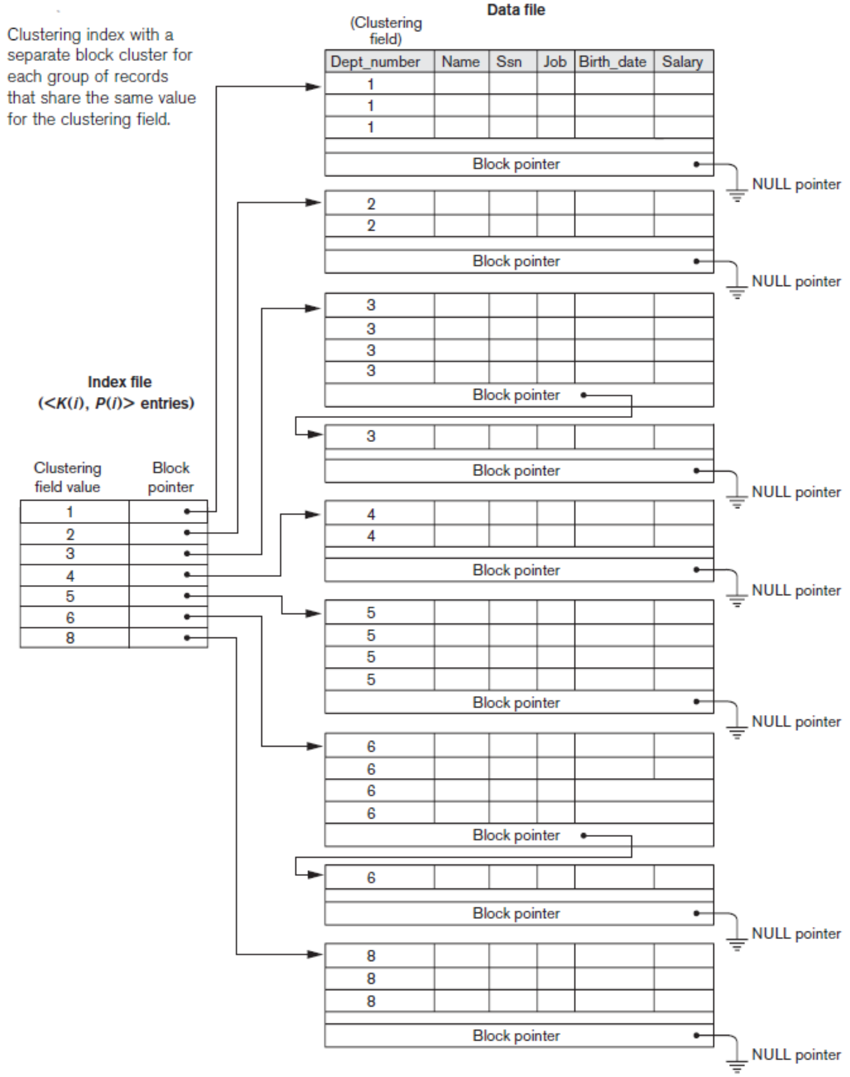
Clustering Indexes

- If file records are physically ordered on a non-key field, which does not have a distinct value for each record—that field is called the **clustering field.**
- A clustering index is also an ordered file with two fields; the first field is of the same type as the clustering field of the data file, and the second field is a disk block pointer.
- Notice that record insertion and deletion still cause problems because the data records are physically ordered.



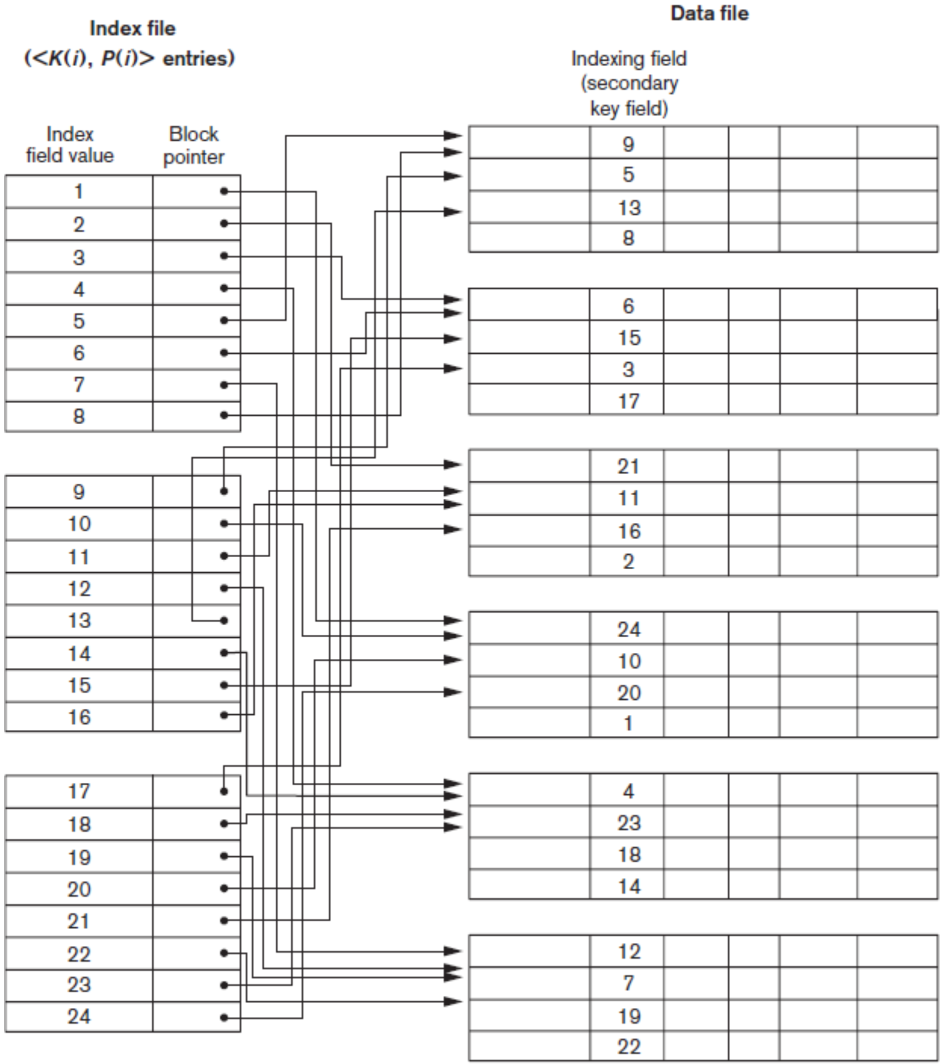
Clustering Indexes

- To alleviate the problem of insertion, it is common to reserve a whole block (or a cluster of contiguous blocks) for each value of the clustering field; all records with that value are placed in the block (or block cluster).
- This makes insertion and deletion relatively straightforward.

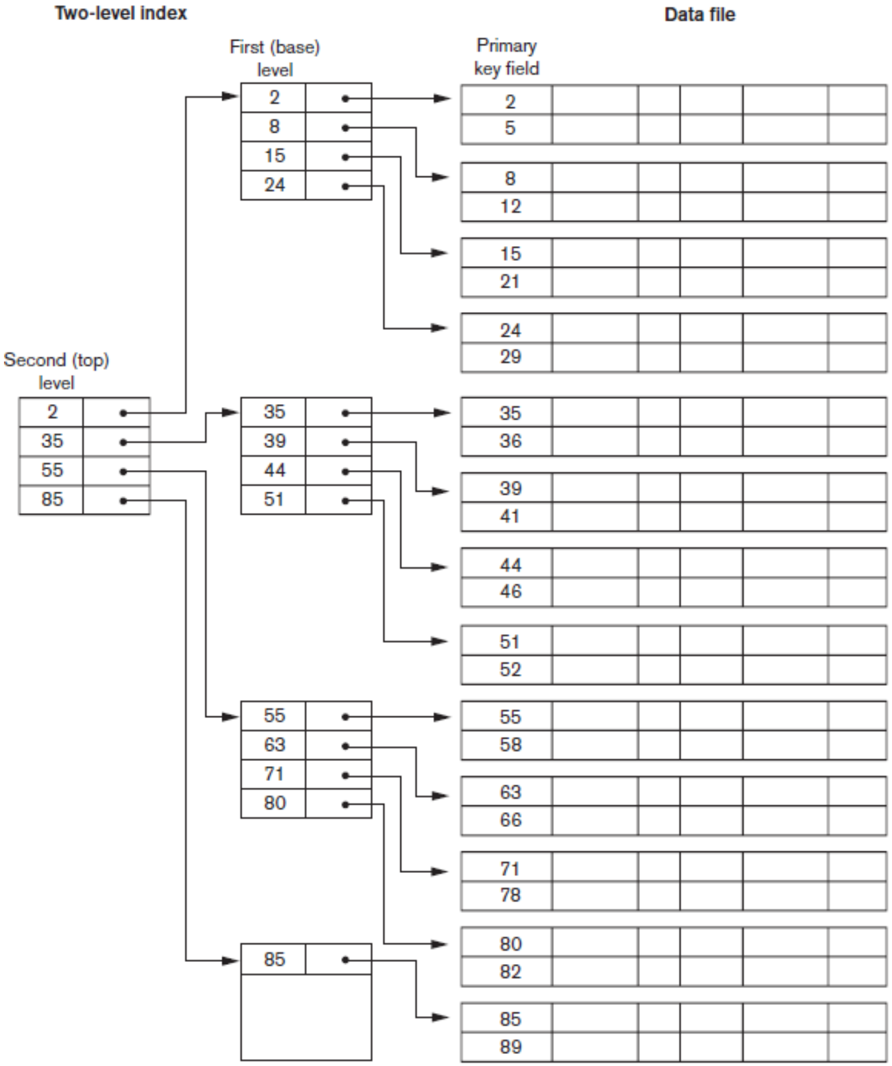


Secondary Indexes

- A secondary index provides a secondary means of accessing a data file for which some primary access already exists. - Indexes based on other key values which are not the PK
- The records in this case are not sorted across the blocks based on the secondary key fields since they are sorted using the PK.



Multilevel Indexes



Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- B-trees and B+-trees are special cases of the well-known search data structure known as a tree.
- Please check the additional slides for the details.

Downsides of Indexes

- ✓ Extra space
- ✓ Index creation
- ✓ Index maintenance

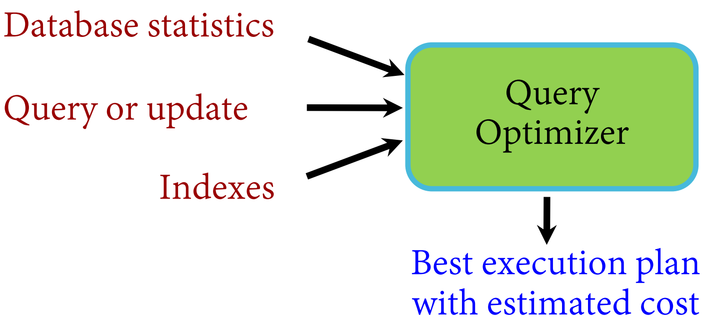
Picking which indexes to create

Benefit of an index depends on:

- ✓ Size of table
- ✓ Data distributions
- ✓ Query vs. update load

Physical Design Advisors

Input: database (statistics) and workload
Output: recommended indexes



SQL Syntax Indexes

Create Index IndexName on T(A)

Create Index IndexName on T(A1,A2,…,An)

Create Unique Index IndexName on T(A)

Drop Index IndexName

Summary

- ✓ File Organization
- ✓ Introduction to Indexing
- ✓ Single-Level Ordered Indexes
    - Primary Indexes
    - Clustering Indexes
    - Secondary Indexes
- ✓ Multilevel Indexes
    - Two-Level Primary Indexing
- ✓ Dynamic Multilevel Indexes

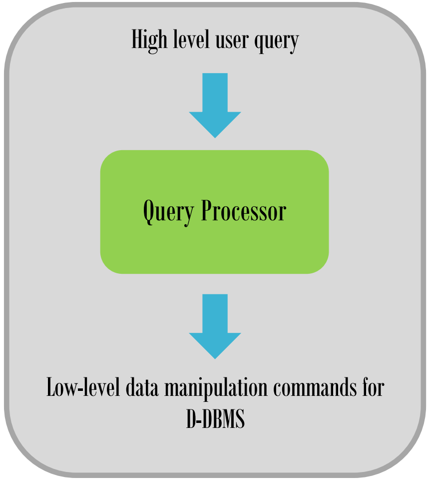
Additional Material

Indexing and Query optimization - Part II

Query Processing

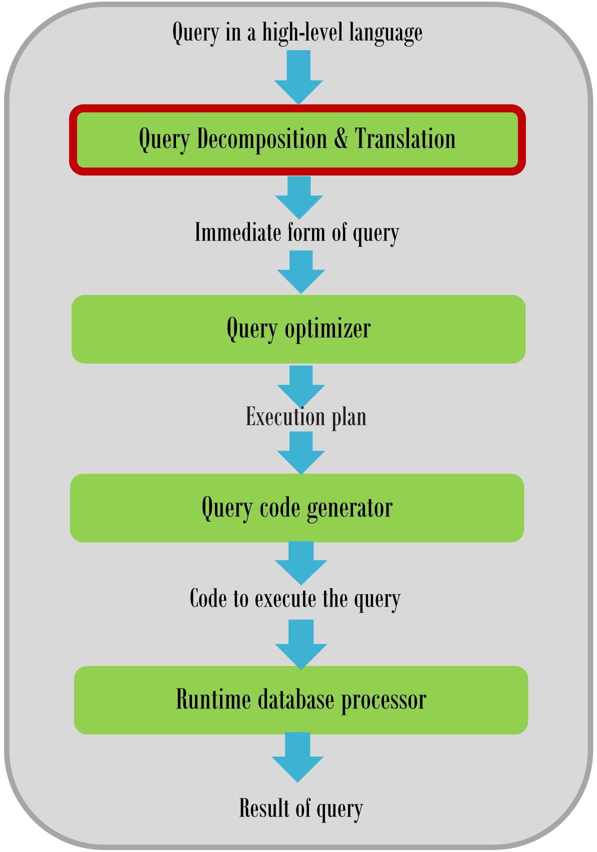
✓ **Aim**:

Transform a query written in a high-level language, typically SQL, into a correct and efficient execution strategy expressed in a low-level language (implementing the relational algebra), and to execute the strategy to retrieve the required data.



An important aspect of query processing is query optimization.

Query Processing



Transform a high-level query into a relational algebra query and to check whether the query is syntactically and semantically correct.

The typical stages:

- ✓ Analysis,
- ✓ Normalization,
- ✓ Semantic analysis,
- ✓ Simplification, and
- ✓ Query restructuring.

Query Decomposition

✓ **Analysis:**

```
Select staffNumber
From Staff
Where position > 10;
```

Staff

StaffNo	fName	lName	Position	Sex	DOB	Salary	BranchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

This query would be rejected on two grounds:

(1) In the select list, the attribute staffNumber is not defined for the Staff relation (should be staffNo).

(2) In the WHERE clause, the comparison ">10" is incompatible with the data type position, which is a variable character string.

Query Decomposition

Normalization:

- ✓ The normalization stage of query processing converts the query into a normalized form that can be more easily manipulated.
- ✓ Predicate can be converted into one of two forms:
  - **Conjunctive normal form:** (position= 'Manager' v salary >20000) ^ branchNo = 'B003'
  - **Disjunctive normal form:** (position = 'Manager' ∧ branchNo = 'B003') ∨ (salary >20000 ∧ branchNo = 'B003')

Semantic analysis:

- ✓ The objective of semantic analysis is to reject normalized queries that are incorrectly formulated or contradictory.
- ✓ For example:

the predicate (position = 'Manager' ∧ position = 'Assistant') on the Staff relation → **contradictory**

Query Decomposition

Simplification:

- ✓ detect redundant qualifications,
- ✓ eliminate common subexpressions, and
- ✓ transform the query to a semantically equivalent but more easily and efficiently computed form.
- ✓ Access restrictions, view definitions, and integrity constraints are considered at this stage.
    - introduce redundancy.
- ✓ Assuming user has appropriate access privileges, first apply well-known idempotency rules of Boolean algebra.

$p \wedge (p) \equiv p$	$p \vee (p) \equiv p$
$p \wedge \text{false} \equiv \text{false}$	$p \vee \text{false} \equiv p$
$p \wedge \text{true} \equiv p$	$p \vee \text{true} \equiv \text{true}$
$p \wedge (\sim p) \equiv \text{false}$	$p \vee (\sim p) \equiv \text{true}$
$p \wedge (p \vee q) \equiv p$	$p \vee (p \wedge q) \equiv p$

Query Decomposition

For example, consider the following integrity constraint,:

```
CREATE ASSERTION OnlyManagerSalaryHigh
CHECK ((position <> ‘Manager’ AND salary< 20000)
OR (position= ‘Manager’ AND salary> 20000));
```

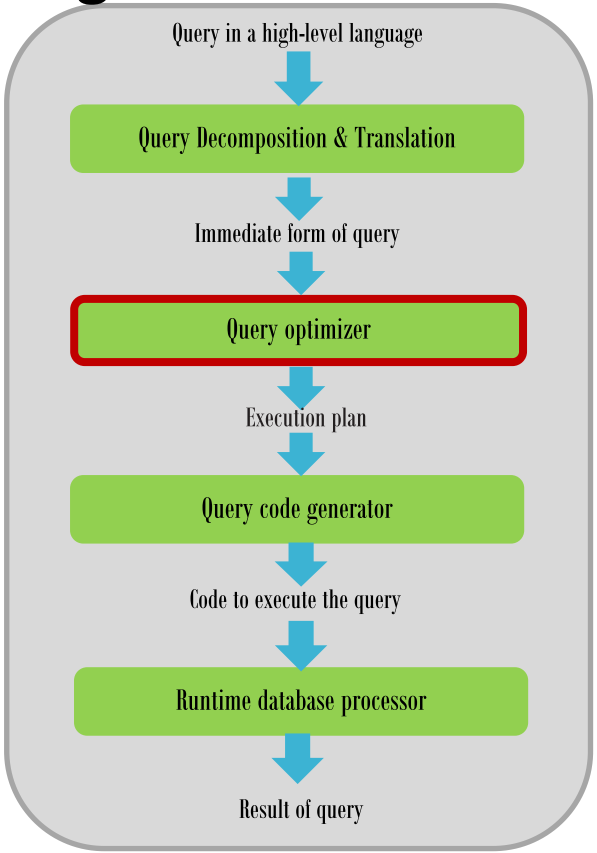
and consider the effect on the query:

```
SELECT *
FROM Staff
WHERE (position 5 ‘Manager’ AND salary , 15000);
```

A contradiction of the integrity constraint so there can be no tuples that satisfy this predicate.

Query restructuring: the query is restructured to provide a more efficient implementation

Query Processing



Aim:

As there are many equivalent transformations of the same high-level query, choose the one that minimizes resource usage.

There are two main techniques for query optimization.

- ✓ **Heuristic rules**
- ✓ Systematically estimating

Query optimization

Heuristic rules:

- ✓ Uses **transformation rules** to convert one relational algebra expression into an equivalent form that is known to be more efficient.

Transformation Rules for the Relational Algebra Operations

1. **Conjunctive Selection operations can cascade into individual Selection operations (and vice versa).**

$$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

Sometimes referred to as cascade of Selection.

$$\sigma_{branchNo='B003' \wedge salary>15000}(Staff) = \sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff))$$

Proofs of the rules can be found in Aho et al. (1979).

Transformation Rules for the Relational Algebra Operations

2. **Commutativity of Selection operations.**

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

For example:

$$\sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff)) = \sigma_{salary>15000}(\sigma_{branchNo='B003'}(Staff))$$

Transformation Rules for the Relational Algebra Operations

3. **In a sequence of Projection operations, only the last in the sequence is required.**

$$\Pi_L \Pi_M \ldots \Pi_N(R) = \Pi_L (R)$$

For example:

$$\Pi_{lName}\Pi_{branchNo,\ lName}(Staff) = \Pi_{lName} (Staff)$$

Transformation Rules for the Relational Algebra Operations

4. **Commutativity of Selection and Projection.**

If predicate p involves only attributes in projection list, Selection and Projection operations commute:

$$\Pi_{Ai, \ldots, Am}(\sigma_p(R)) = \sigma_p(\Pi_{Ai, \ldots, Am}(R)) \qquad \text{where } p \in \{A_1, A_2, \ldots, A_m\}$$

For example:

$$\Pi_{fName, lName}(\sigma_{lName='Beech'}(Staff)) = \sigma_{lName='Beech'}(\Pi_{fName,lName}(Staff))$$

Transformation Rules for the Relational Algebra Operations

5. **Commutativity of Theta join (and Cartesian product).**

$$R \bowtie_p S = S \bowtie_p R$$

$$R \times S = S \times R$$

Rule also applies to Equijoin and Natural join

For example:

$$\text{Staff} \bowtie_{\text{staff.branchNo=branch.branchNo}} \text{Branch} = \text{Branch} \bowtie_{\text{staff.branchNo=branch.branchNo}} \text{Staff}$$

Transformation Rules for the Relational Algebra Operations

6. **Commutativity of Selection and Theta join (or Cartesian product).**

If the selection predicate involves only attributes of one of the relations being joined, then the Selection and Join (or Cartesian product) operations commute:

$$\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$$

$$\sigma_p(R \times S) = (\sigma_p(R)) \times S \qquad \text{where } p \in \{A_1, A_2, \ldots, A_n\}$$

If selection predicate is conjunctive predicate having form $(p \wedge q)$, where p only involves attributes of R, and q only attributes of S, Selection and Theta join operations commute as:

$$\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$$

$$\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$$

For example:

$$\sigma_{position='Manager' \wedge city='London'}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$$

$$(\sigma_{position='Manager'}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'} (Branch))$$

Branch

BranchNo	Street	City	Postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX

Staff

StaffNo	fName	lName	Position	Sex	DOB	Salary	BranchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003

Transformation Rules for the Relational Algebra Operations

7. **Commutativity of Projection and Theta join (or Cartesian product).**

If projection list is of form $L = L_1 \cup L_2$, where L_1 only has attributes of R, and L_2 only has attributes of S, provided join condition only contains attributes of L, Projection and Theta join commute:

$$\Pi_{L1 \cup L2}(R \bowtie_r S) = (\Pi_{L1}(R)) \bowtie_r (\Pi_{L2}(S))$$

If join condition contains additional attributes not in L ($M = M_1 \cup M_2$ where M_1 only has attributes of R, and M_2 only has attributes of S), a final projection operation is required:

$$\Pi_{L1 \cup L2}(R \bowtie_r S) = \Pi_{L1 \cup L2}((\Pi_{L1 \cup M1}(R)) \bowtie_r (\Pi_{L2 \cup M2}(S)))$$

For example:

$$\Pi_{position,city,branchNo}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) = (\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\Pi_{city,\ branchNo} (Branch))$$

and using the latter rule:

$$\Pi_{position,\ city}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) = \Pi_{position,\ city} ((\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\Pi_{city,\ branchNo} (Branch)))$$

Transformation Rules for the Relational Algebra Operations

8. **Commutativity of Union and Intersection (but not set difference).**

$$R \cup S = S \cup R$$

$$R \cap S = S \cap R$$

9. **Commutativity of Selection and set operations (Union, Intersection, and Set difference).**

$$\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$$

$$\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$$

$$\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$$

10. **Commutativity of Projection and Union.**

$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

Transformation Rules for the Relational Algebra Operations

11. **Associativity of Theta join (and Cartesian product).**

Cartesian product and Natural join are always associative:

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$(R \times S) \times T = R \times (S \times T)$$

If join condition q involves attributes only from S and T, then Theta join is associative:

$$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$$

For example:

$$(Staff \bowtie_{Staff.staffNo=PropertyForRent.staffNo} PropertyForRent) \bowtie_{ownerNo=Owner.ownerNo \wedge staff.lName=Owner.lName} Owner = Staff \bowtie_{staff.staffNo = PropertyForRent.staffNo \wedge staff.lName=lName} (PropertyForRent \bowtie_{ownerNo} Owner)$$

Transformation Rules for the Relational Algebra Operations

12. **Associativity of Union and Intersection (but not Set difference).**

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

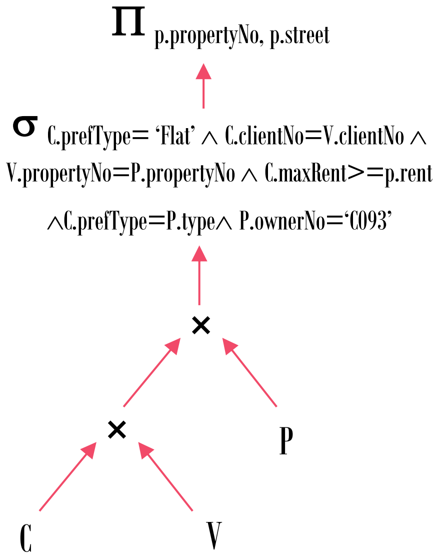
Example : Use of Transformation Rules

For prospective renters of flats, find properties that match requirements and owned by C093.

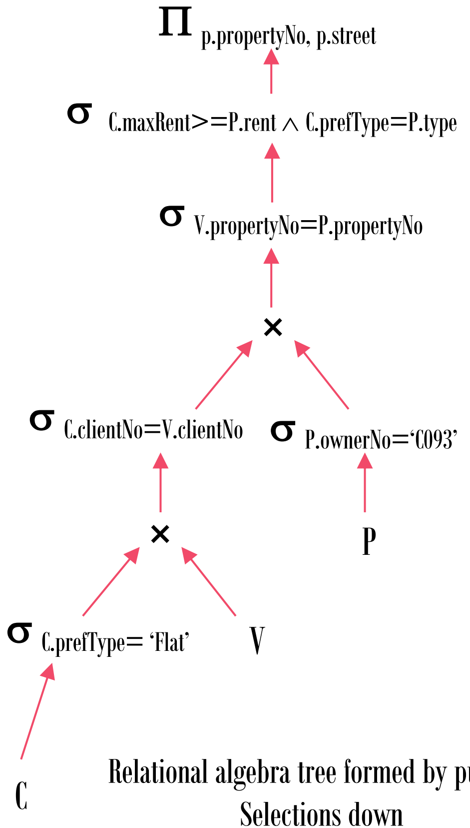
```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE   c.prefType = 'Flat' AND
            c.clientNo = v.clientNo AND
            v.propertyNo = p.propertyNo AND
            c.maxRent >= p.rent AND
            c.prefType = p.type AND
            p.ownerNo = 'C093';
```


$$\Pi_{p.propertyNo,\ p.street}\left(\sigma_{c.prefType='Flat' \wedge c.clientNo=v.clientNo \wedge v.propertyNo=p.propertyNo \wedge c.maxRent>=p.rent \wedge c.prefType=p.type \wedge p.ownerNo='C093'}((c \times v) \times p)\right)$$

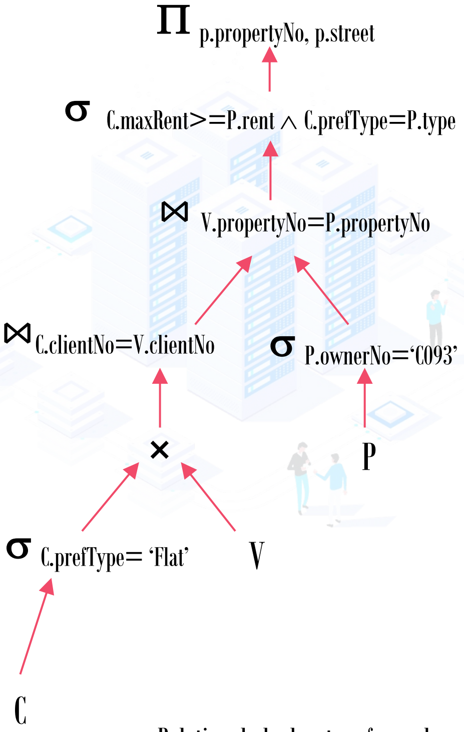
$$\Pi_{\text{p.propertyNo, p.street}}\left(\sigma_{\text{c.prefType= 'Flat'} \wedge \text{c.clientNo=v.clientNo} \wedge \text{v.propertyNo=p.propertyNo} \wedge \text{c.maxRent>=p.rent} \wedge \text{c.prefType=p.type} \wedge \text{p.ownerNo='C093'}}\left((c \times v) \times p\right)\right)$$



Canonical relational algebra tree

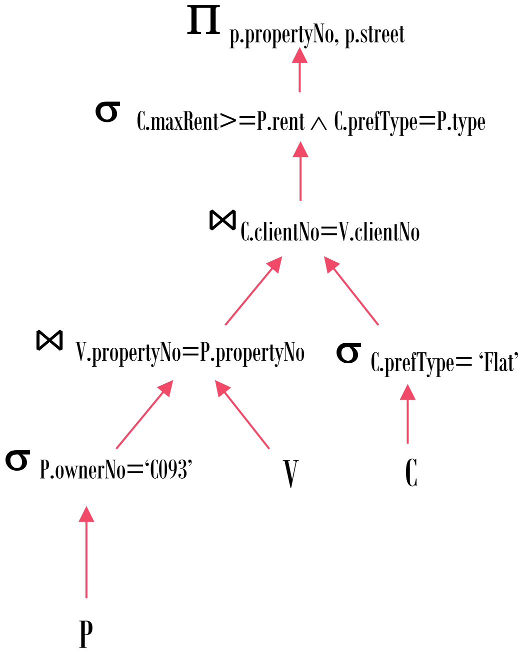


Relational algebra tree formed by pushing Selections down

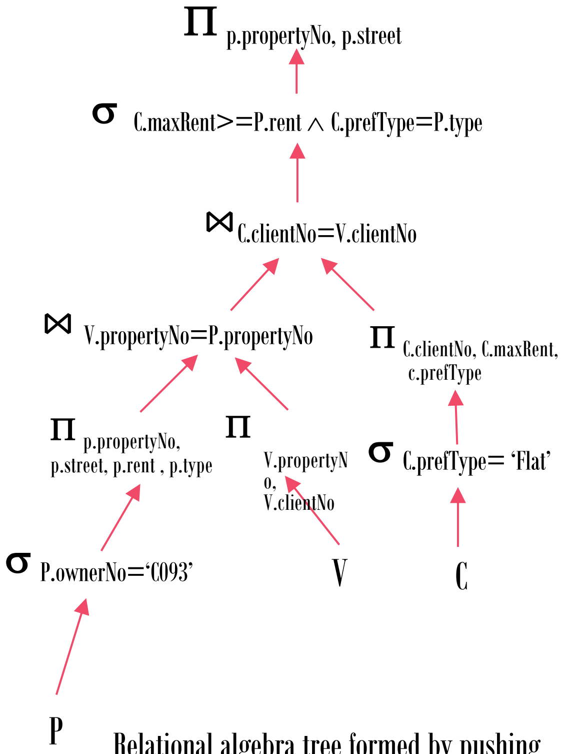


Relational algebra tree formed
by changing Selection/Cartesian products to Equijoins;

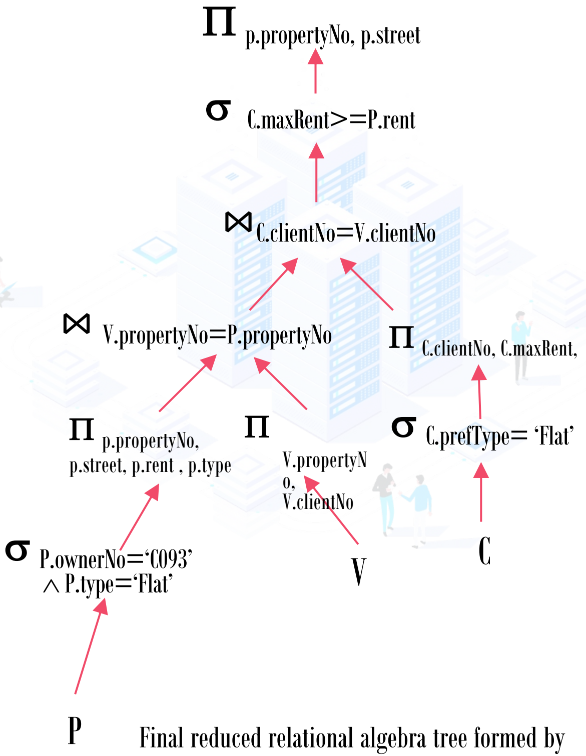
$$\Pi_{p.propertyNo,\ p.street}(\sigma_{c.prefType=\text{'Flat'} \wedge c.clientNo=v.clientNo \wedge v.propertyNo=p.propertyNo \wedge c.maxRent>=p.rent \wedge c.prefType=p.type \wedge p.ownerNo=\text{'C093'}}((c \times v) \times p))$$



Relational algebra tree formed using associativity of Equijoins;



Relational algebra tree formed by pushing Projections down;



Final reduced relational algebra tree formed by substituting c.prefType = 'Flat' in Selection on p.type and pushing resulting Selection down tree.

Heuristical Processing Strategies

1. Perform Selection operations as early as possible.
   - ✓ Keep predicates on same relation together.
2. Combine Cartesian product with subsequent Selection whose predicate represents join condition into a Join operation.

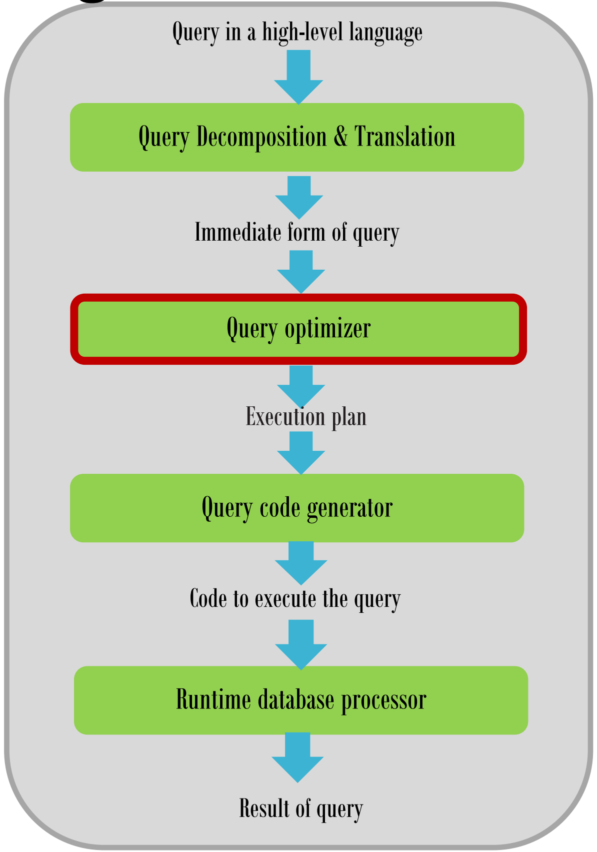
$$\sigma_{R.a\ \theta\ S.b}(RXS) = R \bowtie_{R.a\ \theta\ S.b} (S)$$

3. Use associativity of binary operations to rearrange leaf nodes so leaf nodes with most restrictive Selection operations executed first.

$$(R \bowtie_{R.a\ \theta\ S.b} S) \bowtie_{S.c\ \theta\ T.d} T$$

4. Perform Projection as early as possible.
   - ✓ Keep projection attributes on same relation together.
5. Compute common expressions once.
   - ✓ If common expression appears more than once, and result not too large, store result and reuse it when required.
   - ✓ Useful when querying views, as same expression is used to construct view each time.

Query Processing



Aim:

As there are many equivalent transformations of the same high-level query, choose the one that minimizes resource usage.

There are two main techniques for query optimization.

- ✓ Heuristic rules
- ✓ Systematically estimating

Cost Estimation for the Relational Algebra Operations

- ✓ Many different ways of implementing Relational Algebra (RA) operations.
- ✓ Aim of Query Optimization(QO) is to choose most efficient one.
  - Use formulae that estimate costs for a number of options and select one with lowest cost.
- ✓ Consider only cost of disk access, which is usually dominant cost in QP.
- ✓ Many estimates are based on cardinality of the relation, so need to be able to estimate this.

Database Statistics

The success of estimating the size and cost of intermediate relational algebra operations depends on the amount and currency of the statistical information that the DBMS holds.

For each base relation R:

- ✓ nTuples(R)- the number of tuples (records) in relation R (that is, its cardinality).
- ✓ bFactor(R)- the blocking factor of R (that is, the number of tuples of R that fit into one block).
- ✓ nBlocks(R)- the number of blocks required to store R.

For each attribute A of base relation R:

- ✓ $nDistinct_A(R)$- the number of distinct values that appear for attribute A in relation R.
- ✓ $min_A(R)$, $max_A(R)$- the minimum and maximum possible values for the attribute A in relation R.
- ✓ $SC_A(R)$—the selection cardinality of attribute A in relation R.

For each multilevel index I on attribute set A:

- ✓ $nLevels_A(I)$—the number of levels in I.
- ✓ $nLfBlocks_A(I)$—the number of leaf blocks in I.

Selection Operation ($S = \sigma_p(R)$)

Predicate may be simple or composite.

Number of different implementations, depending on file structure, and whether attribute(s) involved are indexed/hashed.

Main strategies are:

1. Linear Search (Unordered file, no index): $[nBlocks(R)/2]$, for equality condition on key attribute $nBlocks(R)$, otherwise
2. Binary Search (Ordered file, no index): $[\log_2 (nBlocks(R))]$, for equality condition on ordered attribute $[\log_2 (nBlocks(R))] + [SC_A(R)/bFactor(R)] - 1$, otherwise
3. Equality on hash key: 1, assuming no overflow
4. Equality condition on primary key: $nLevels_A(I) + 1$
5. Inequality condition on primary key: $nLevels_A(I) + [nBlocks(R)/2]$
6. Equality condition on clustering (secondary) index: $nLevels_A(I) + [SC_A(R)/bFactor(R)]$
7. Equality condition on a non-clustering (secondary) index: $nLevels_A(I) + [SC_A(R)]$
8. Inequality condition on a secondary B+-tree index: $nLevels_A(I) + [nLfBlocks_A(I)/2 + nTuples(R)/2]$ $nLevels_A(I) + 1$

Selection Operation ($S = \sigma_p(R)$)

Cost estimation for Selection operation:

We make the following assumptions about the Staff relation:

- There is a hash index with no overflow on the primary key attribute staffNo.
- There is a clustering index on the foreign key attribute branchNo.
- There is a B+-tree index on the salary attribute.
- The Staff relation has the following statistics stored in the system catalog:

nTuples(Staff) = 3000	
bFactor(Staff) = 30	→ nBlocks(Staff) = 100
$nDistinct_{branchNo}(Staff) = 500$	→ $SC_{branchNo}(Staff) = 6$
$nDistinct_{position}(Staff) = 10$	→ $SC_{position}(Staff) = 300$
$nDistinct_{salary}(Staff) = 500$	→ $SC_{salary}(Staff) = 6$
$min_{salary}(Staff) = 10,000$	$max_{salary}(Staff) = 50,000$
$nLevels_{branchNo}(I) = 2$	
$nLevels_{salary}(I) = 2$	$nLfBlocks_{salary}(I) = 50$

The estimated cost of a linear search on the key attribute staffNo is 50 blocks,
The cost of a linear search on a non-key attribute is 100 blocks.

Consider the following Selection operations:

S1: $\sigma_{staffNo='SG5'}(Staff)$

- ✓ Equality condition on the primary key. the attribute staffNo is hashed, estimate the cost as 1 block. The estimated cardinality of the result relation is $SC_{staffNo}(Staff) = 1$.

S2: $\sigma_{position='manager'}(Staff)$

- ✓ The attribute in the predicate is a non-key, non-indexed attribute, so we cannot improve on the linear search method, giving an estimated cost of 100 blocks. The estimated cardinality of the result relation is $SC_{position}(Staff) = 300$

Join Operation ($T = (R \bowtie_F S)$)

The most time-consuming operation to process.

The main strategies for implementing the Join operation.

- ✓ Block Nested Loop Join:

  nBlocks(R) 1 (nBlocks(R) * nBlocks(S)), if buffer has only one block for R and S

  nBlocks(R) 1 [nBlocks(S)*(nBlocks(R)/(nBuffer 2 2))], if (nBuffer 2 2) blocks for R

  nBlocks(R) 1 nBlocks(S), if all blocks of R can be read into database buffer

- ✓ Indexed Nested Loop Join:

  Depends on indexing method; for example:

  nBlocks(R) 1 nTuples(R)*(nLevelsA (I) 1 1), if join attribute A in S is the primary key

  Blocks(R) 1 nTuples(R)*(nLevelsA (I) 1 [SCA (R)/bFactor(R)]), for clustering index I on attribute A

- ✓ Sort-Merge Join:

  nBlocks(R)*[log2 (nBlocks(R)] 1 nBlocks(S)*[log2 (nBlocks(S)], for sorts

  nBlocks(R) 1 nBlocks(S), for merge

- ✓ Hash Join:

  3(nBlocks(R) 1 nBlocks(S)), if hash index is held in memory

  2(nBlocks(R) 1 nBlocks(S))*[lognBuffer–1 (nBlocks(S)) 2 1] 1 nBlocks(R) 1 nBlocks(S), otherwise

Projection Operation($\mathbf{S} = \Pi_{A1, A2, \ldots, Am}(\mathbf{R})$)

To implement projection, need to:

- ✓ Remove attributes that are not required;
- ✓ Eliminate any duplicate tuples produced from previous step.

Estimating the cardinality of the Projection operation:

When the Projection contains a key attribute: the cardinality of the Projection is: nTuples(S) = nTuples(R)

If the Projection consists of a single non-key attribute ($S = \Pi_A(R)$), we can estimate the cardinality of the Projection as: nTuples(S) = $SC_A(R)$

Two main approaches to eliminating duplicates:

- ✓ Sorting;
- ✓ Hashing.

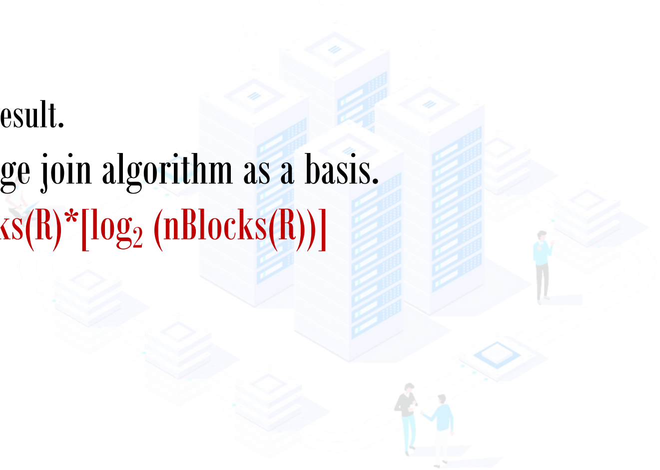
The Relational Algebra Set Operations($T = R \cup S$, $T = R \cap S$, $T = R - S$)

Implemented by

- ✓ sorting both relations on same attributes, and
- ✓ then scanning through each of sorted relations once to obtain desired result.

For all these operations, we could develop an algorithm using the sort–merge join algorithm as a basis.

The estimated cost in all cases is simply: $nBlocks(R) + nBlocks(S) + nBlocks(R)*[\log_2 (nBlocks(R))] + nBlocks(S)*[\log_2 (nBlocks(S))]$



Enumeration of Alternative Execution Strategies

Pipelining

Linear Trees

Physical Operators and Execution Strategies

Reducing the Search Space

Semantic Query Optimization

Alternative Approaches to Query Optimization:

- Simulated Annealing
- Iterative Improvement
- Two-Phase Optimization
- Genetic algorithms
- A* heuristic algorithm

Distributed Query Optimization