Query Processing

政治大學
資訊科學系
沈錳坤

Outline

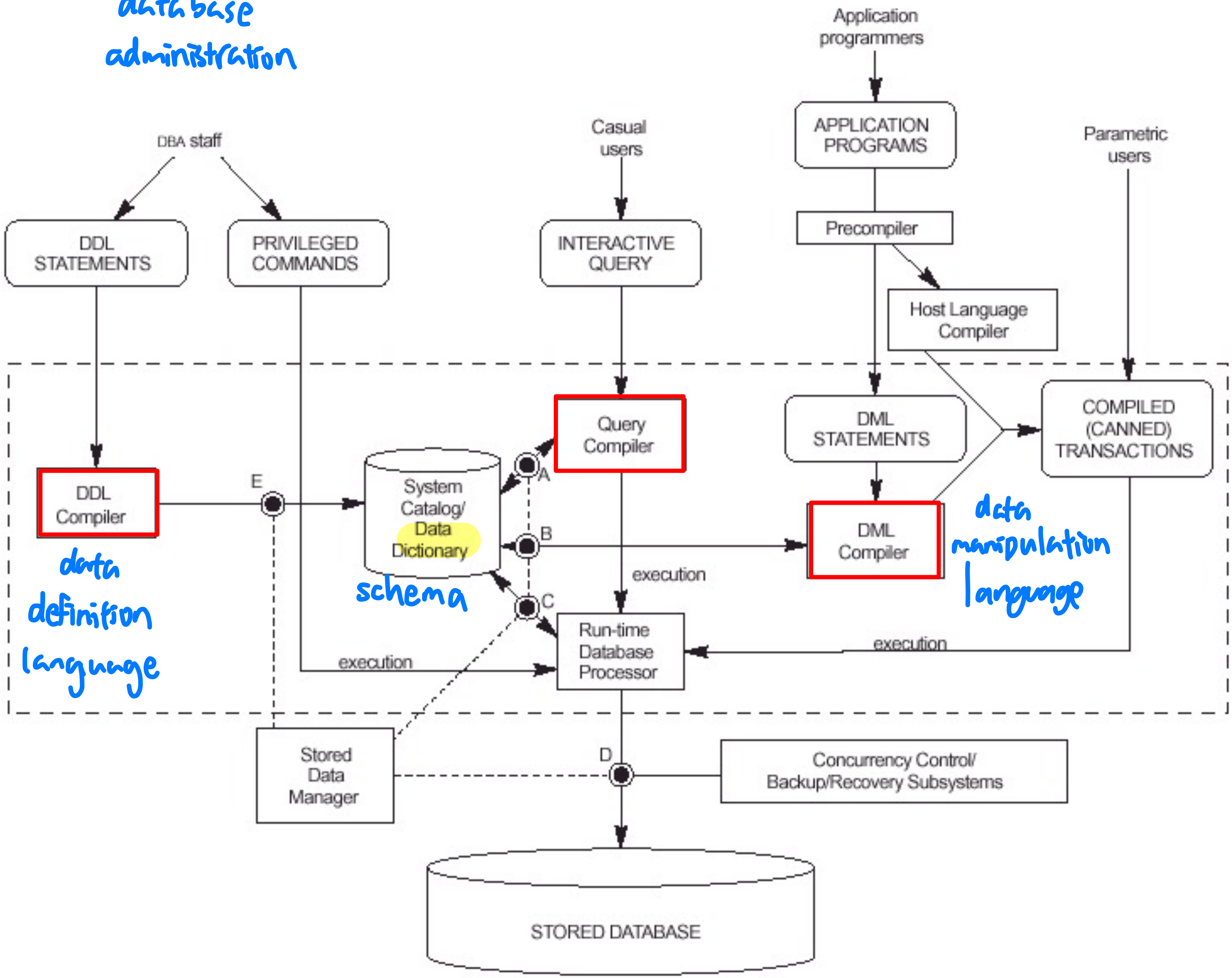
1. Query Processing
2. Transforming SQL Queries to Relational Algebra
3. Basic Algorithms for Executing Query Operations

database administration

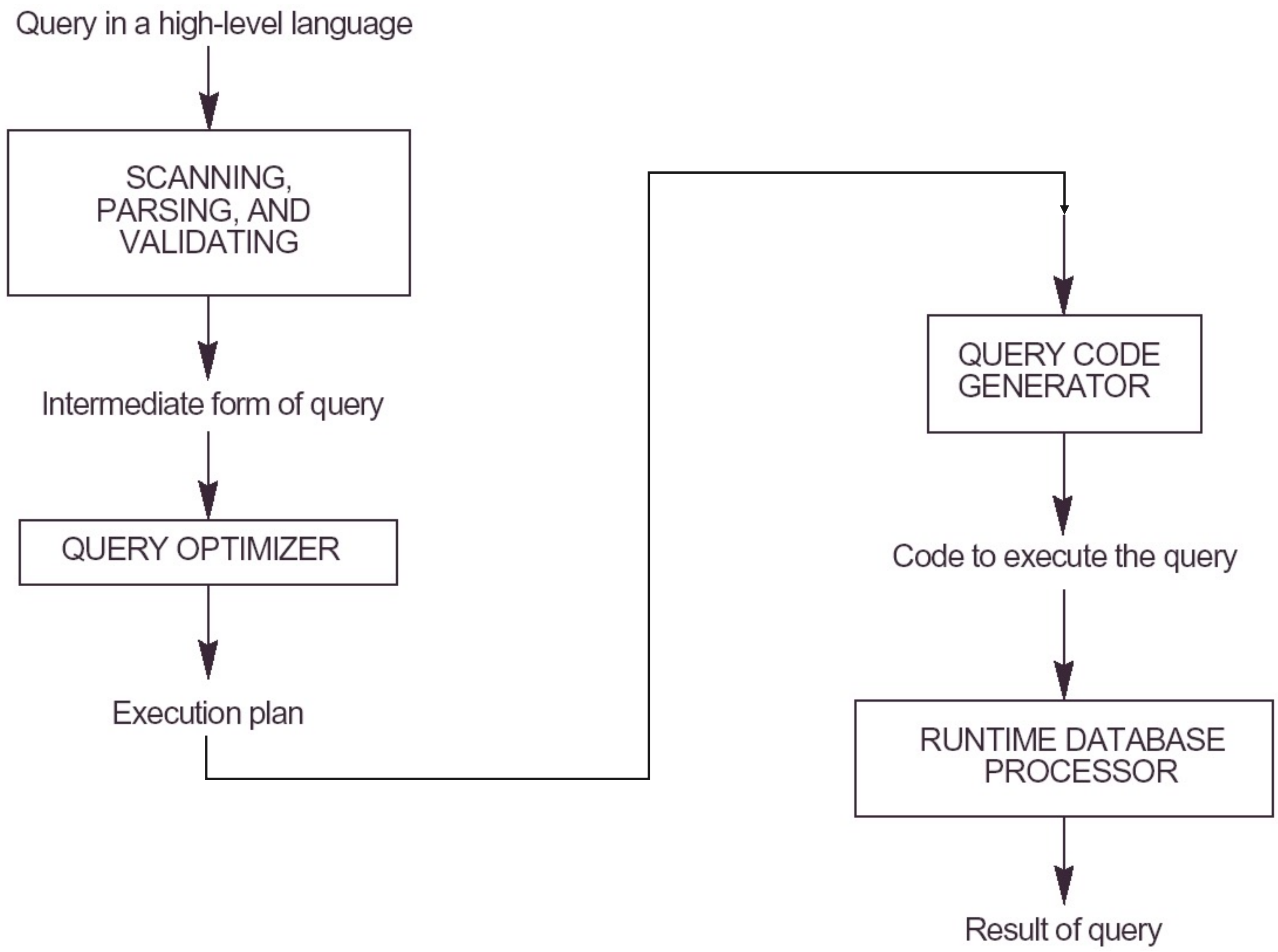
1

2

3



Query Processing



Query Processing (cont.)

- Scanner: identifies language tokens
- Parser: check query syntax
- Validate: check attributes & relation names are valid & semantically meaningful names in schema of queried DB
- Intermediate form: query tree or query graph (relational algebra)
- Query optimization: process of choosing a suitable execution strategy (execution plan)
- Code generator: generate the code to execute the plan
- Runtime DB processor: running the code (in compiled or interpreted) to produce query result

Outline

1. Query Processing
2. Transforming SQL Queries to Relational Algebra
3. Basic Algorithms for Executing Query Operations

Transforming SQL to Relational Algebra

- SQL query ➔ query tree
  - For uncorrelated query
    - SQL query decomposed to query blocks
    - Query optimizer choose an execution plan for each query block

```
SELECT Lname, Fname
FROM Employee
WHER Salary > (SELECT MAX(Salary)
               FROM Employee
               WHERE DNo =5)
```

outer block

```
SELECT Lname, Fname
FROM Employee
WHERE Salary > c
```

inner block

```
SELECT MAX (Salary)
FROM Employee
WHERE Dno=5
```

Transforming SQL to Relational Algebra (cont.)

- For correlated nested query
  - It's much harder to optimize

```
Select E.Fname, E.Lname
From Employee as E
Where E.SSN in (
        Select ESSN
        From Dependent as D
        Where E.Fname = D.Dependent_Name
                        and E.Sex = D.Sex);
```

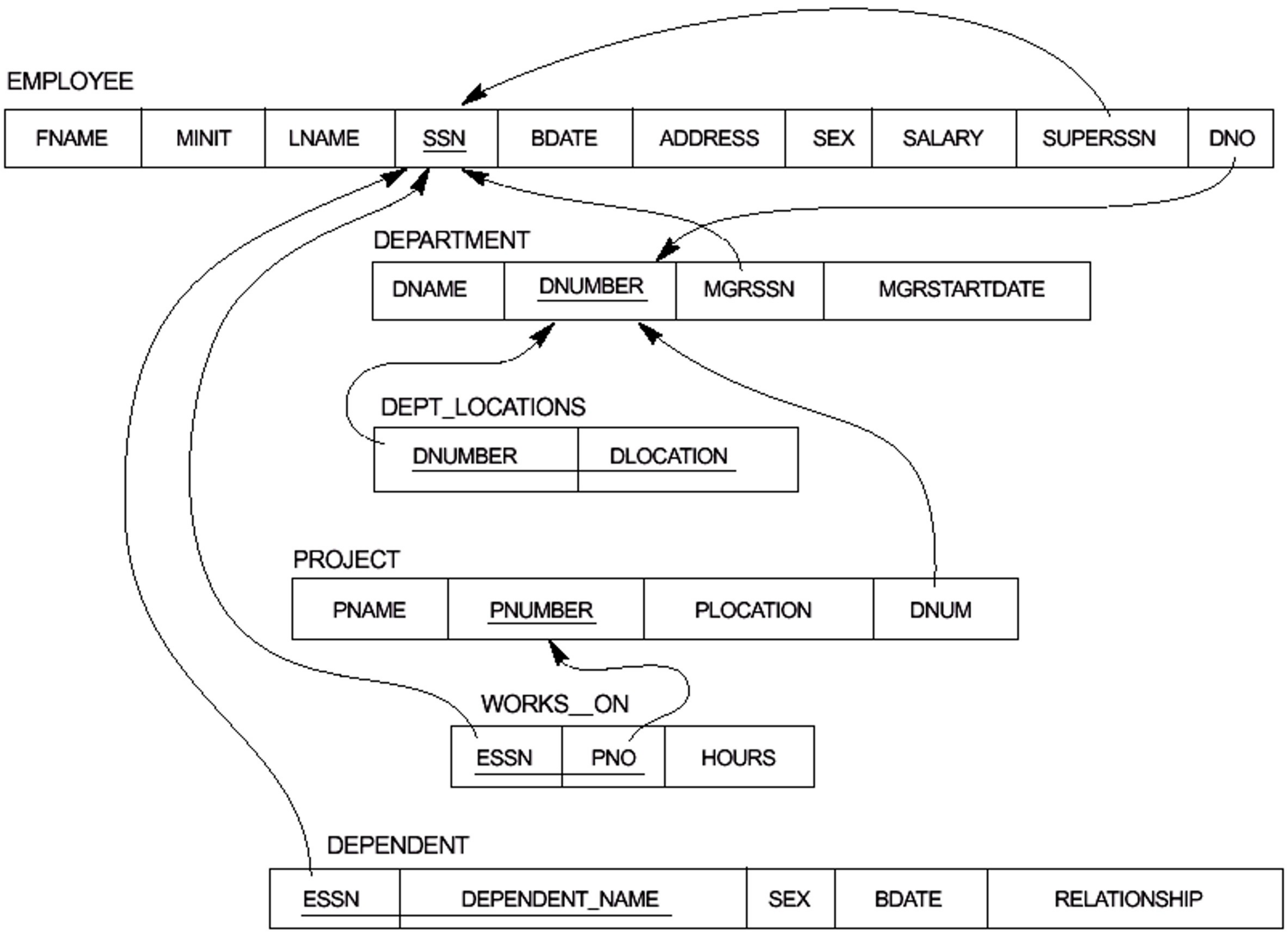
Transforming SQL to Relational Algebra (cont.)

For every project located in 'Stafford', retrieve the project number, the controlling department number, and the department manager's last name, address, and birthdate.

Select P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
From Project as P, Department as D, Employee as E
Where P.Dnum=D.Dnumber and D.Mgrssn=E.SSN and P.Plocation='Stafford';

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}} (((\sigma_{\text{Plocation='Stafford'}}(\text{Project})) \otimes_{\text{Dnum=Dnumber}}(\text{Department})) \otimes_{\text{Mgrssn=SSN}}(\text{Employee}))$$

For every project located in ‘Stafford’, retrieve the project number, the controlling department number, and
the department manager’s last name, address, and birthdate.



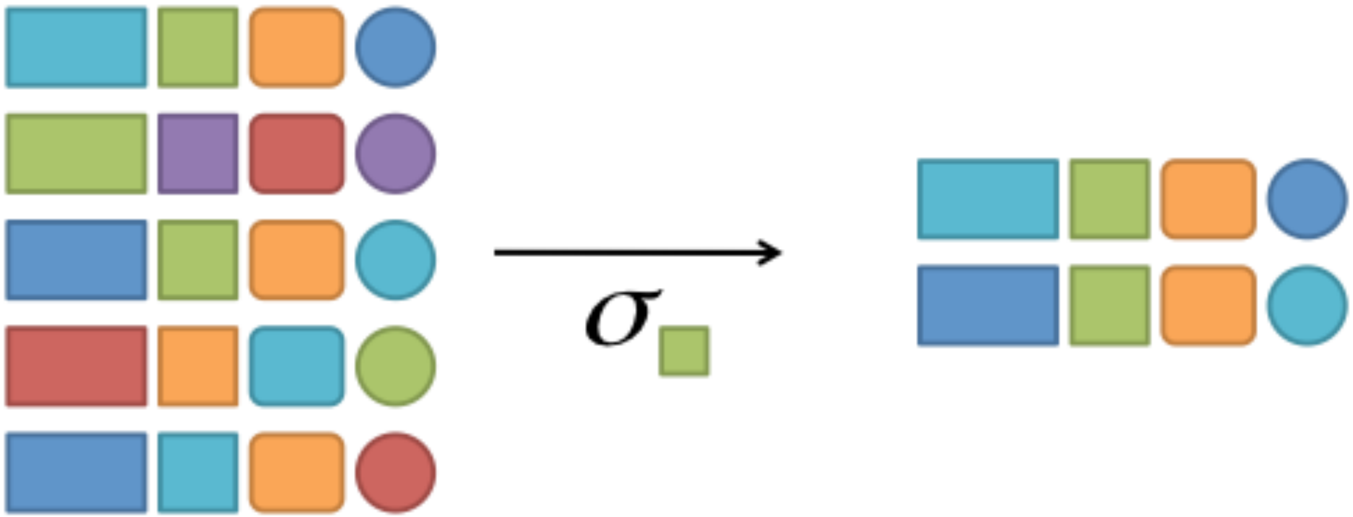
Outline

1. Query Processing
2. Transforming SQL Queries to Relational Algebra
3. Basic Algorithms for Executing Query Operations

Basic Algorithms for Executing Query Operations

- Relational algebra
  - Specially for RDB Operations
    - Select
    - Project
    - Join
  - Set operations from mathematical set theory
    - Union
    - Intersection
    - Set Difference
    - Cartesian Product
- Ordered by, Aggregation (Group by)

SELECTION



SELECTION (cont.)

- Search methods for simple selection
  - Linear search
  - Binary search: ordered file on search attribute
  - Using hash for equality search: hashed file, hash index
  - Using a primary index for equality search, range query
  - Using a clustering index for equality search, range query
  - Using a secondary index for equality search, range query
  - Using a bitmap index for query involving a set of values for an attribute

not suitable for range query i.e.

$a = 5$

$\cancel{3 < a < 10}$

SELECTION (cont.)

- Search methods for conjunctive selection

  SELECT SSN, DeptName

  FROM Employee

  WHERE EmployeeName='Brown" AND DeptNumber=5;

  – Using an individual index for conjunctive selection
    - using primary index on EmployeeName, then check DeptNumber
  – Using an composite index for conjunctive selection
    - Multidimensional index on (EmployeeName, DeptNumber) e.g. R tree
  – Using intersection of record pointers for conjunctive selection

    Step 1: Using primary index on Employeename

    Step 2: Using secondary index on DeptNumber

    Step 3: Intersection of results of Step 1 & Step 2

SELECTION (cont.)

- Query optimization for conjunctive selection
  - When more than one of attributes involved in condition have an index
  - Optimizer must choose the access path that retrieves the fewest records

SELECTION (cont.)

- Query optimization for disjunctive selection

  SELECT SSN, DeptName

  FROM Employee

  WHERE EmployeeName='Brown" OR DeptNumber=5;

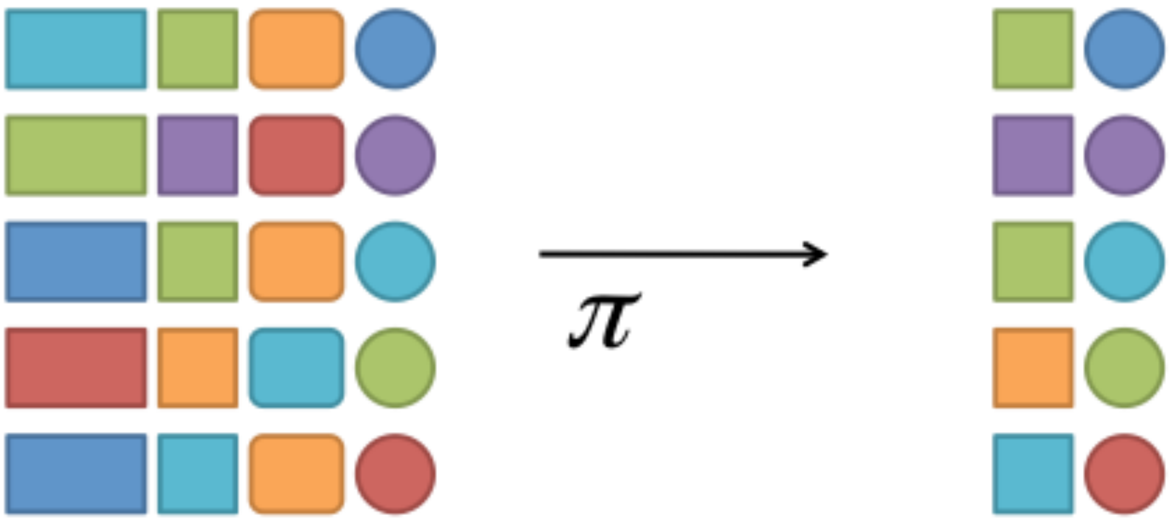
  – When any one of the condition does not have an index, use linear or binary search

  – Only if an index exists on each simple condition, use index and applying union operation

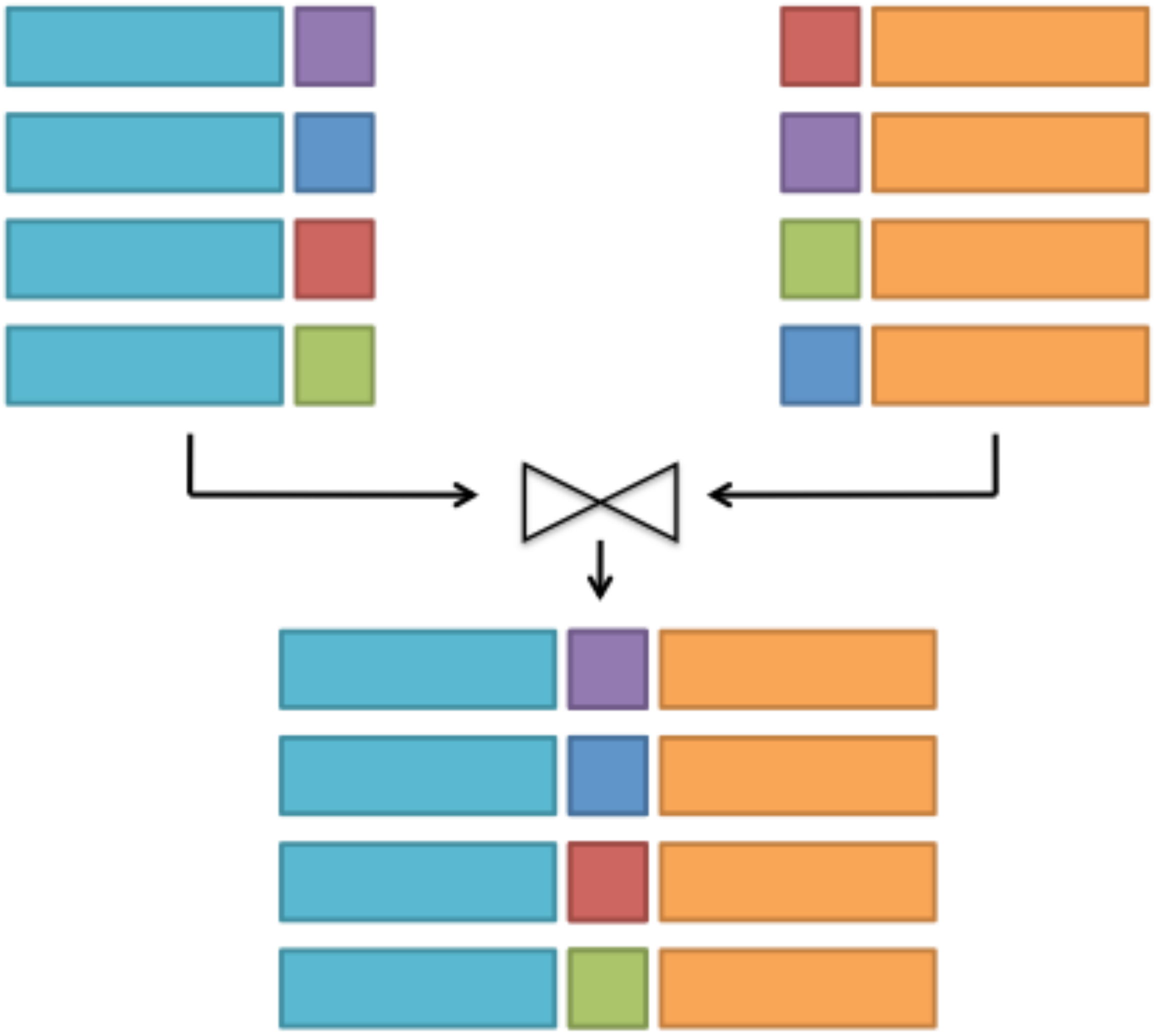
∵ the one that does not have an index needs to do linear / binary search anyway

PROJECTION

- Duplicate tuples must be eliminated
  - Sorting & eliminate duplicate records which appear consecutively after sorting
  - Hashing



JOIN



JOIN (cont.)

- Joins
  - Nested loop join
  - Indexed-based nested-loop join using an index structure
  - Sort-merge join
    - R & S are physically sorted by joined attribute
    - Both files are scanned concurrently in order of joined attributes
  - Hash join
    - Partitioning phase: the smaller relation hashes records to memory buckets
    - Probing phase: a single pass thru the other file hashes each record to probe appropriate bucket

JOIN: Index-based Nested-loop

- Join(*R*, *S*)
  - An index exists for join attribute of relation *S*
  - For each record *t* in *R*
    - Retrieve all matching records *s* from S using the index

JOIN: Sort-merge

- Join(*R*, *S*)
  - Similar to sort-merge operation in merge sort
  - Assume *R* and *S* are physically sorted on join attributes
  - Both *R* and *S* are scanned concurrently
    - Matching records that have same value of join attributes
  - If *R* and/or *S* are not sorted, may be sorted first by external sorting

Merge Operation in Merge Sort

0	1	2	3	4	5	6	7
7	8	11	14	4	6	8	23

p_1 p_2

0	1	2	3	4	5	6	7
4	6	7	8	8	11	14	23

p_3

Sort-Merge Operation in Join

R

20	30	30	40
w	x	y	z

p_1

S

10	20	20	20	30	30
A	B	C	D	E	F

p_2

20	20	20	30	30	30	30
w	w	w	x	x	y	y
B	C	D	E	F	E	F

p_3

```
sort the tuples in R on attribute A;          (* assume R has n tuples (records) *)
sort the tuples in S on attribute B;          (* assume S has m tuples (records) *)
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do {   if R(i) [A] > S(j) [B]
       then   set j ← j + 1
    elseif R(i) [A] < S(j) [B]
       then   set i ← i + 1
    else {    (* R(i) [A] = S(j) [B], so we output a matched tuple *)
              output the combined tuple <R(i), S(j)> to T;

              (* output other tuples that match R(i), if any *)
              set l ← j + 1;
              while (l ≤ m) and (R(i) [A] = S(l) [B])
              do {   output the combined tuple <R(i), S(l)> to T;
                     set l ← l + 1
              }

          (* output other tuples that match S(j), if any *)
          set k ← i + 1;
          while (k ≤ n) and (R(k) [A] = S(j) [B])
          do {   output the combined tuple <R(k), S(j)> to T;
                 set k ← k + 1
          }
          set i ← k, j ← l
    }
}
```

mismatch

match

JOIN: Partition-Hash

- Join(R, S)
  - R & S are partitioned into M partitions separately using the same partitioning hash function.
  - Records in partition R_i only need to be joined with records in partition S_i
  - 2 phases
    - Partitioning phase: partitioning hash on join attributes
    - Probing phase

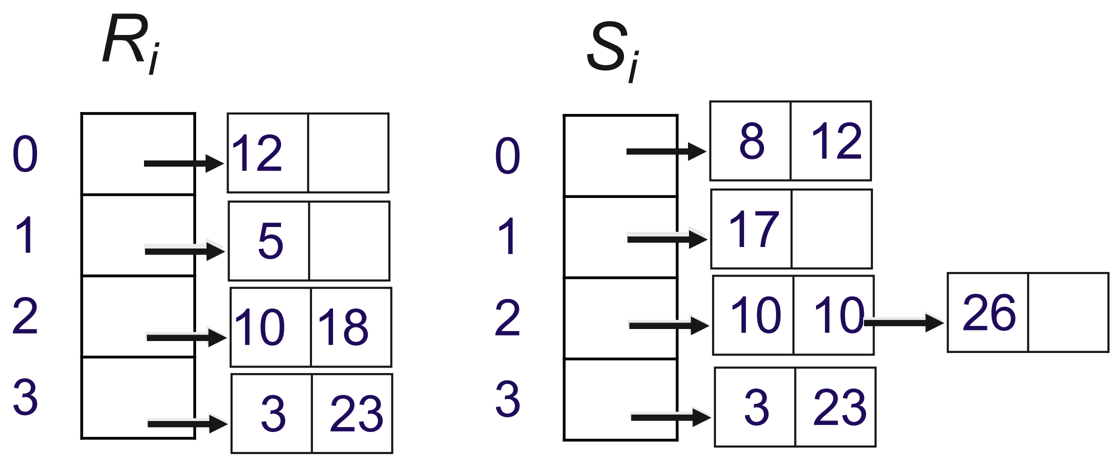
```
for i = 0 to M-1
    for each block j of partition S_i
            for each record of block j
        probe partition R_i for matching records
```

R

3	t
5	u
10	v
12	w
18	x
23	y

Hash : *K* mod 4

block factor = 2



S

3	A
8	B
10	C
10	D
12	E
17	F
23	G
26	H

R join *S*

12	w	E
10	v	C
10	v	D
3	t	A
23	y	G

CARTESIAN PRODUCT

- Query optimizer try to avoid the Cartesian Product operation

Union, Intersection: Sort-Merge

- Union
  - Scanning & merge both sorted files concurrently
  - When ever same tuple exists in both relation, only one is kept
- Intersection
  - Keep in merged result only those tuples that appear in both relations

Union: Sort-Merge Algorithm

```
sort the tuples in R and S using the same unique sort attributes;
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do {   if R(i) > S(j)
          then  {  output S(j) to T;
                   set j ← j + 1
          }
       elseif R(i) < S(j)
          then  {  output R(i) to T;
                   set i ← i + 1
          }
       else set  j ← j + 1                 (* R(i)=S(j), so we skip one of the duplicate tuples *)
}
if (i ≤ n) then add tuples R(i) to R(n) to T;
if (j ≤ m) then add tuples S(j) to S(m) to T;
```

Intersection: Sort-Merge Algorithm

```
sort the tuples in R and S using the same unique sort attributes;
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do {   if R(i) > S(j)
            then   set j ← j + 1
       elseif  R(i) < S(j)
            then   set i ← i + 1
       else   {   output R(j) to T;                  (* R(i)=S(j), so we output the tuple *)
                  set i ← i + 1, j ← j + 1
       }
}
```

Union, Intersection: Hashing

- Union(R, S)
  - Hash the records R
  - Probe the record of the other relations S, but not insert duplicate records in the buckets to the result
- Intersection(R, S)
  - Hash the records R
  - Probe the record of the other relations S, add identical records in the buckets to the result

External Sorting *

- Sorting used in
  - ORDER BY-clause
  - Sort-merge algorithms used for JOIN (UNION, INTERSECTION)
  - Duplicate elimination algorithms for PROJECT
- External sorting
  - Sort-merge strategy

Aggregation Operations

- Q: For each department, retrieve the department number, the number of employees in the department, and their average salary

aggregate function

SELECT DNo, COUNT (*), AVG (Salary)
FROM Employee
GROUP BY DNo

Aggregation Operations (cont.)

- Aggregate operators applied to entire table
- Can be computed by a table scan or by using an appropriate index
- Index could also be used for count, average, sum if it is a dense index
- Index could also be used for count distinct
- Group by
  - Sorting or hashing technique
  - Clustering index

Conclusions

- Query processing
- Transforming SQL into Relational Algebra
- Algorithms for executing relational algebra
  - Selection σ
  - Project π
  - Join $\bowtie$
  - Union, Intersection $\cup$, $\cap$
  - Aggregation
  - Ordered-By
- Indexing, Hash, Sort-merge