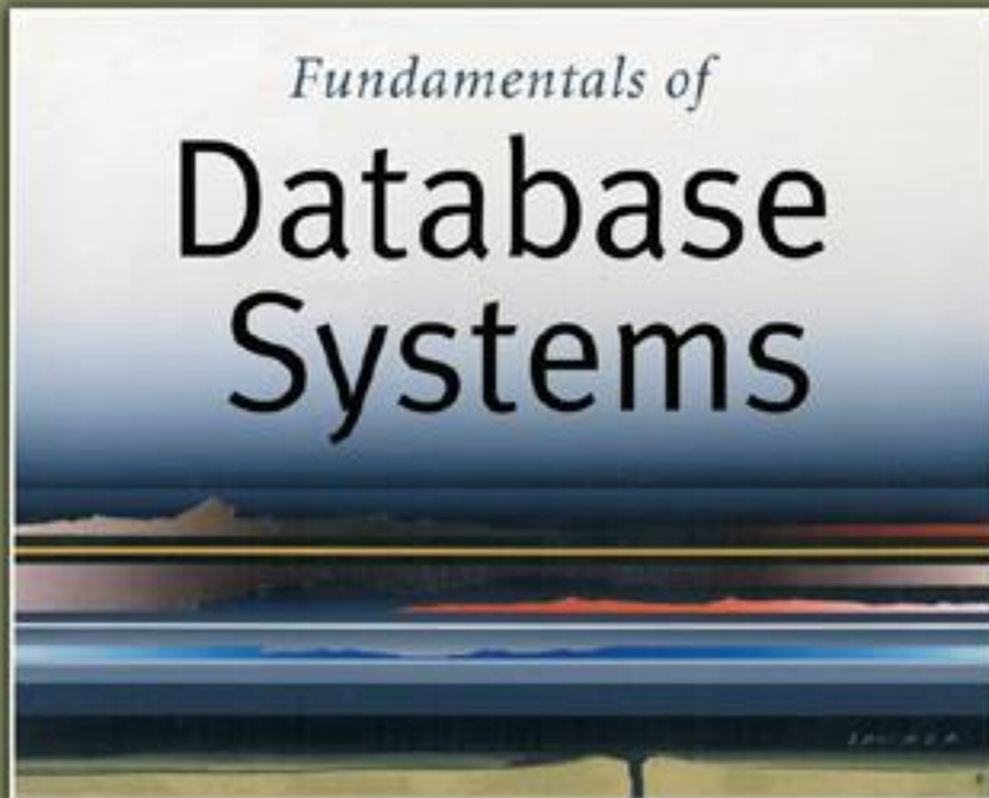
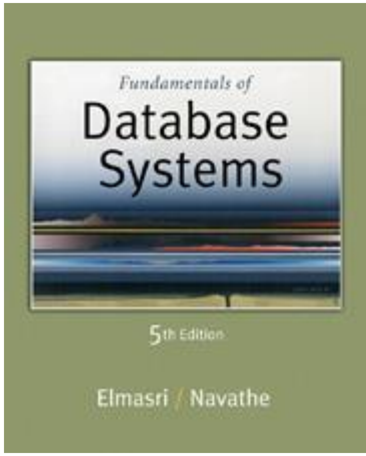


5th Edition

Elmasri / Navathe

Chapter 6

The Relational Algebra



Relational Algebra Overview

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a ***new relation***, which may have been formed from one or more *input* relations
  - This property makes the algebra "closed" (all objects in relational algebra are relations)

Relational Algebra Overview (continued)

- The **algebra operations** thus produce new relations
  - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

Relational Algebra Overview

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: **σ** (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, **–**)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
  - Additional Relational Operations
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Database State for COMPANY

- All examples discussed below refer to the COMPANY database shown here.

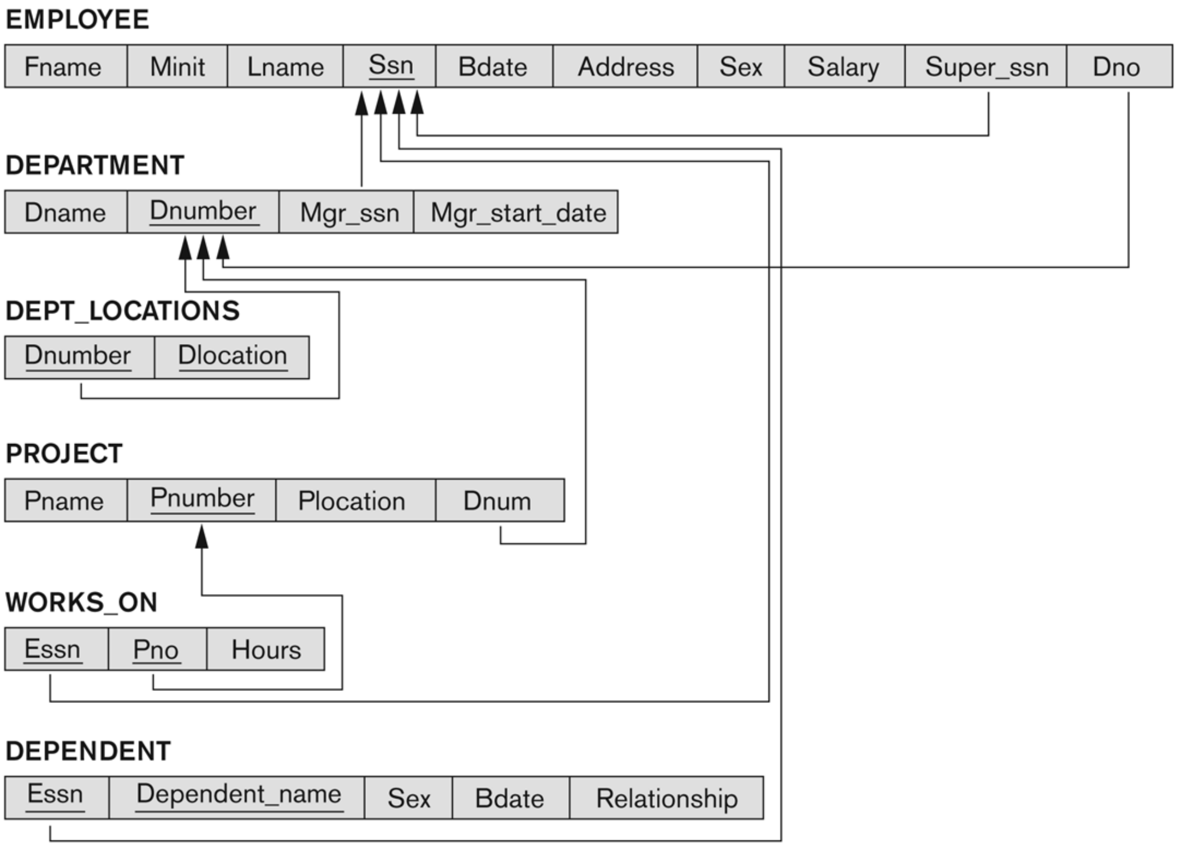


Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**.
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)

Unary Relational Operations: SELECT

- In general, the *select* operation is denoted by

  $\sigma_{\text{<selection condition>}}(R)$ where

  - the symbol **σ** (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition **true** are selected
    - appear in the result of the operation
  - tuples that make the condition **false** are filtered out
    - discarded from the result of the operation

- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$$\sigma_{\text{DNO} = 4}\,(\text{EMPLOYEE})$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{\text{SALARY} > 30{,}000}\,(\text{EMPLOYEE})$$

Unary Relational Operations: SELECT (contd.)

- SELECT Operation Properties
  - The SELECT operation $\sigma_{<\text{selection condition}>}(R)$ produces a relation S that has **the same schema (same attributes)** as R
  - SELECT σ is **commutative**:
    - $\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$
  - Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
    - $\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(\sigma_{<\text{cond1}>}(R)))$
  - A cascade of SELECT operations may be **replaced by a single selection with a conjunction** of all the conditions:
    - $\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R)) = \sigma_{<\text{cond1}>\ \text{AND}\ <\text{cond2}>\ \text{AND}\ <\text{cond3}>}(R)))$
  - The number of tuples in the result of a SELECT **is less than (or equal to)** the number of tuples in the input relation R

The following query results refer to this database state

Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by π (pi)
- This operation **keeps certain *columns*** (attributes) from a relation and **discards** the other columns.
  - PROJECT creates a **vertical partitioning**
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded

Unary Relational Operations: PROJECT (cont.)

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(R)$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.
- The project operation ***removes any duplicate tuples***
  - This is because the result of the *project* operation must be a *set of tuples*
    - Mathematical sets *do not allow* duplicate elements.

- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

Unary Relational Operations: PROJECT (contd.)

- PROJECT Operation Properties
  - The number of tuples in the result of projection $\pi_{<list>}(R)$ is **always less or equal** to the number of tuples in R
    - **If the list of attributes includes a *key* of R**, then the number of tuples in the result of PROJECT is ***equal*** to the number of tuples in R
  - PROJECT is ***not* commutative**
    - $\pi_{<list1>} (\pi_{<list2>} (R)) = \pi_{<list1>} (R)$ as long as <list2> contains the attributes in <list1>

Examples of applying SELECT and PROJECT operations

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(Dno=4\ \textbf{AND}\ Salary>25000)\ OR\ (Dno=5\ \textbf{AND}\ Salary>30000)}$ (EMPLOYEE). (b) $\pi_{Lname,\ Fname,\ Salary}$(EMPLOYEE). (c) $\pi_{Sex,\ Salary}$(EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a **single relational algebra expression** by nesting the operations, or
  - We can apply **one operation at a time** and create **intermediate result relations**.
- In the latter case, we **must give names** to the relations that hold the intermediate results.

Single expression versus sequence of relational operations (Example)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a *single relational algebra expression* as follows:
  - $\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
  - $\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$

Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to ***rename*** the **attributes** of a relation or the **relation name** or **both**
  - Useful when a query requires multiple operations
  - **Necessary in some cases (see JOIN operation later)**

Unary Relational Operations: RENAME (contd.)

- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S\ (B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..Bn
  - $\rho_S(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, …..Bn

Relational Algebra Operations from Set Theory: UNION

- UNION Operation
  - Binary operation, denoted by $\cup$
  - The result of $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S
  - **Duplicate tuples are eliminated**
  - The two operand relations R and S must be **"type compatible" (or UNION compatible)**
    - R and S must have **same number of attributes**
    - **Each pair of corresponding attributes must be type compatible** (have same or compatible domains)

Example of the result of a UNION operation

- UNION Example

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra Operations from Set Theory: UNION

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5} (EMPLOYEE)$$
$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$
$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$
$$RESULT \leftarrow RESULT1 \cup RESULT2$$

  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Example of the result of a UNION operation

- UNION Example

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra Operations from Set Theory: INTERSECTION

- INTERSECTION is denoted by $\cap$
- The result of the operation $R \cap S$, is a relation that **includes all tuples that are in both R and S**
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be **"type compatible"**

Relational Algebra Operations from Set Theory: SET DIFFERENCE (cont.)

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of **R – S, is a relation that includes all tuples that are in R but not in S**
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be **"type compatible"**

Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

Some properties of UNION, INTERSECT, and DIFFERENCE

- Notice that both union and intersection are ***commutative*** operations; that is
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- The **minus operation is not commutative**; that is, in general
  - $R - S \neq S - R$

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation X
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with **degree n + m attributes**:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have **$n_R * n_S$ tuples**.
  - **The two operands do NOT have to be "type compatible"**

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- Generally, CROSS PRODUCT is **not a meaningful operation**
  - Can become meaningful when **followed by other operations**
- Example (not meaningful):
  - FEMALE_EMPS ← σ $_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES ← π $_{FNAME,\ LNAME,\ SSN}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS ← EMPNAMES x DEPENDENT
- EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT
  - whether or not they are **actually related**

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows
- Example (meaningful):
  - FEMALE_EMPS ← $\sigma_{\text{SEX='F'}}$(EMPLOYEE)
  - EMPNAMES ← $\pi_{\text{FNAME, LNAME, SSN}}$ (FEMALE_EMPS)
  - **EMP_DEPENDENTS ← EMPNAMES x DEPENDENT**
  - **ACTUAL_DEPS ← $\sigma_{\text{SSN=ESSN}}$(EMP_DEPENDENTS)**
  - RESULT ← $\pi_{\text{FNAME, LNAME, DEPENDENT\_NAME}}$ (ACTUAL_DEPS)
- RESULT will now contain the name of female employees and their dependents

Example of applying CARTESIAN PRODUCT

Figure 6.5
The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

Binary Relational Operations: JOIN

- JOIN Operation (denoted by $\bowtie$)
  - The sequence of **CARTESIAN PRODECT followed by SELECT** is used quite commonly to identify and select **related tuples** from two relations
  - A special operation, called **JOIN combines this sequence into a single operation**
  - This operation is very important for any relational database with more than a single relation, because it allows us ***combine related tuples* from various relations**
  - The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$\mathbf{R} \bowtie_{\text{<join condition>}} \mathbf{S}$$

  - where R and S can be any relations that result from general *relational algebra expressions*.

Binary Relational Operations: JOIN (cont.)

- Example: Suppose that we want to retrieve the name of the manager of each department.
  - To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
  - We do this by using the join $\bowtie$ operation.
  - $DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE$
- **MGRSSN=SSN is the join condition**
  - Combines each department record with the employee who manages the department
  - The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

Example of applying the JOIN operation

DEPT_MGR

Dname	Dnumber	Mgr_ssn	. . .	Fname	Minit	Lname	Ssn	. . .
Research	5	333445555	. . .	Franklin	T	Wong	333445555	. . .
Administration	4	987654321	. . .	Jennifer	S	Wallace	987654321	. . .
Headquarters	1	888665555	. . .	James	E	Borg	888665555	. . .

Figure 6.6
Result of the JOIN operation

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE$$

Binary Relational Operations: NATURAL JOIN Operation

- **NATURAL JOIN** Operation *
  - Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the **second (superfluous) attribute** in an EQUIJOIN condition.
    - **because one of each pair of attributes with identical values is superfluous**
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, ***have the same name* in both relations**
  - If this is not the case, a **renaming operation is applied first**.

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that **cannot be expressed in the basic relational algebra** is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
  - These functions are used in simple statistical queries that **summarize information** from the database tuples.
- Common functions applied to collections of numeric values include
  - **SUM, AVERAGE, MAXIMUM, and MINIMUM**.
- The **COUNT** function is used for counting tuples or values.

Aggregate Function Operation

- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, without removing duplicates

Using Grouping with Aggregation

- The previous examples all summarized one or more attributes for a set of tuples
  - Maximum Salary or Count (number of) Ssn
- Grouping can be combined with Aggregate Functions
- Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY
- A variation of aggregate operation $\mathcal{F}$ allows this:
  - Grouping attribute placed to **left of symbol**
  - Aggregate functions to right of symbol
  - $_{\text{DNO}}\ \mathcal{F}\ _{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
- Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Examples of applying aggregate functions and grouping

Figure 6.10
The aggregate function operation.
(a) $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.
(b) $_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.
(c) $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

Examples of Queries in Relational Algebra : Procedural Form

- **Q1: Retrieve the name and address of all employees who work for the 'Research' department.**

RESEARCH_DEPT $\leftarrow \sigma_{\text{DNAME='Research'}}$ (DEPARTMENT)

RESEARCH_EMPS $\leftarrow$ (RESEARCH_DEPT $\bowtie_{\text{DNUMBER= DNOEMPLOYEE}}$ EMPLOYEE)

RESULT $\leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}}$ (RESEARCH_EMPS)

- **Q6: Retrieve the names of employees who have no dependents.**

ALL_EMPS $\leftarrow \pi_{\text{SSN}}$(EMPLOYEE)

EMPS_WITH_DEPS(SSN) $\leftarrow \pi_{\text{ESSN}}$(DEPENDENT)

EMPS_WITHOUT_DEPS $\leftarrow$ (ALL_EMPS - EMPS_WITH_DEPS)

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}$ (EMPS_WITHOUT_DEPS * EMPLOYEE)

Examples of Queries in Relational Algebra – Single expressions

As a single expression, these queries become:

- **Q1: Retrieve the name and address of all employees who work for the 'Research' department.**

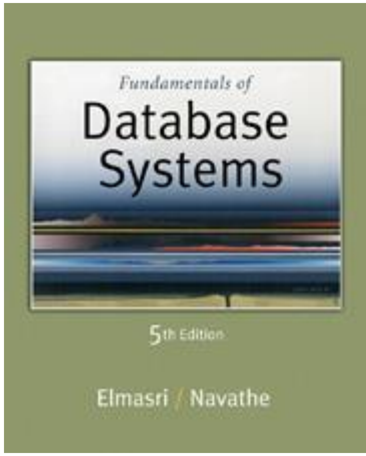
$\pi_{\text{Fname, Lname, Address}}$ ($\sigma_{\text{Dname= 'Research'}}$ (DEPARTMENT $\bowtie_{\text{Dnumber=Dno}}$(EMPLOYEE))

- **Q6: Retrieve the names of employees who have no dependents.**

$\pi_{\text{Lname, Fname}}((\pi_{\text{Ssn}}\ (\text{EMPLOYEE}) - \rho_{\text{Ssn}}\ (\pi_{\text{Essn}}\ (\text{DEPENDENT}))) * \text{EMPLOYEE})$

Chapter 15

Algorithms for Query Processing and Optimization



Query Processing

- A query expressed in a high-level query language such as SQL must first be **scanned, parsed, and validated**.
  - The **scanner** identifies the **query tokens**—such as SQL keywords, attribute names, and relation names.
  - The **parser** checks the **query syntax** to determine whether it is formulated according to the **syntax rules (rules of grammar)** of the query language.
  - The query must also be **validated** by checking that all attribute and relation names are **valid and semantically meaningful names** in the schema of the particular database being queried.

Query Optimization

- An **internal representation** of the query is **then** created, usually as a tree data structure called a **query tree**.
- The DBMS must then devise an **execution strategy or query plan** for retrieving the results of the query from the database files.
- A query typically has **many possible execution strategies**, and the process of **choosing a suitable one** for processing a query is known as **query optimization.**
- **Query optimization**:
  - The process of choosing a suitable execution strategy for processing a query.

Introduction to Query Processing

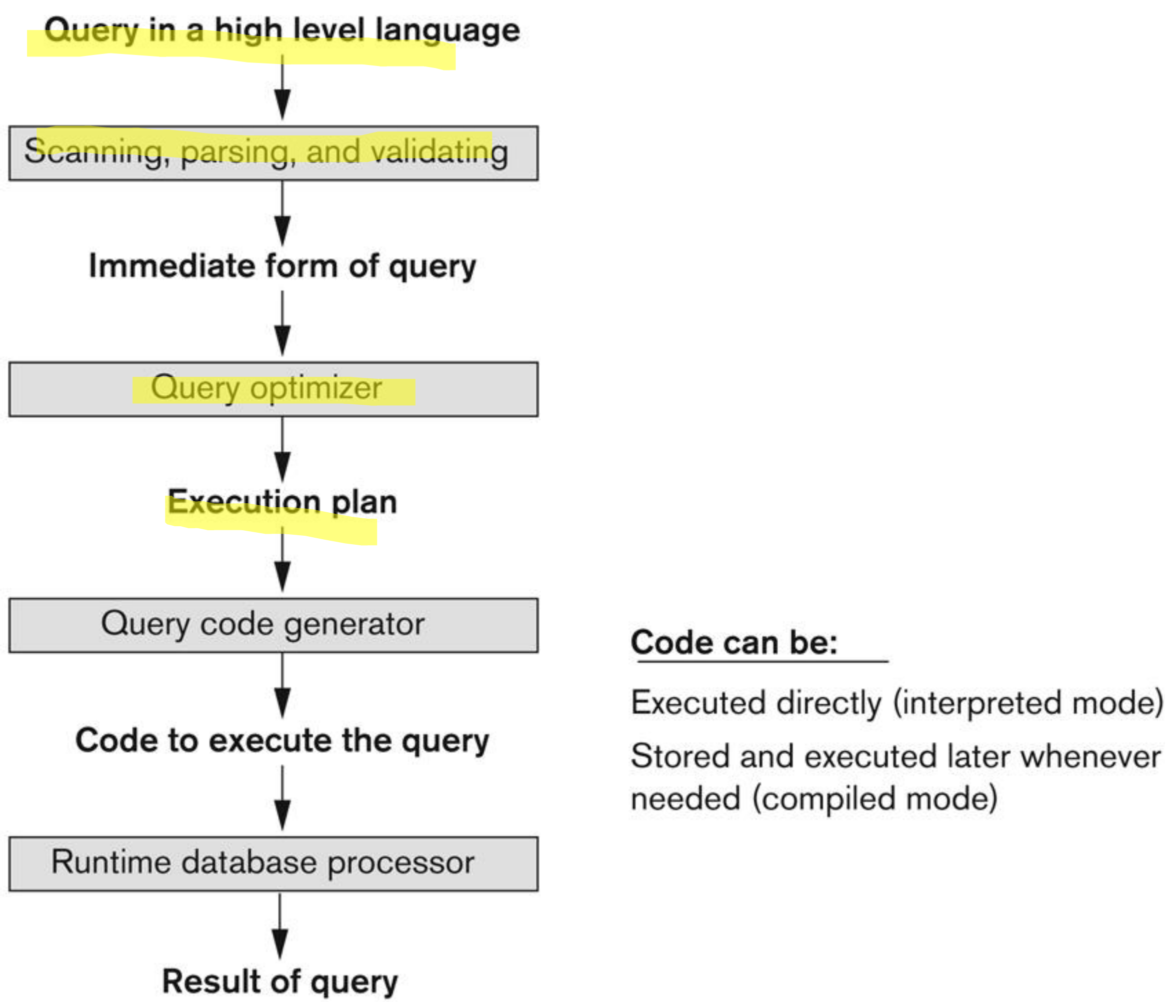


Figure 15.1
Typical steps when processing a high-level query.

Compiled vs. Interpreted Queries

- The **optimizer must limit the number of execution strategies** to be considered; otherwise, **too much time** will be spent making **cost** estimates for the many possible execution strategies.
- This approach is more suitable for **compiled queries** where the optimization is **done at compile time and the resulting execution strategy code is stored and executed directly at runtime.**
- For **interpreted queries,** where the entire process occurs at **runtime**, a full-scale optimization may slow **down the response time.**

Translating SQL Queries into Relational Algebra

- **Query block**:
  - The **basic unit** that can be translated into the algebraic operators and optimized.
- A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block.
- **Nested queries** within a query are identified as separate query blocks.
- In general, a query tree gives a good **visual representation and understanding of the query in terms of the relational operations**

Translating SQL Queries into Relational Algebra (2)

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > (   SELECT    MAX (SALARY)
                       FROM      EMPLOYEE
                       WHERE     DNO = 5);
```

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > C
```

```
SELECT    MAX (SALARY)
FROM      EMPLOYEE
WHERE     DNO = 5
```

$$\pi_{\text{LNAME, FNAME}} (\sigma_{\text{SALARY>C}}(\text{EMPLOYEE}))$$

$$\mathcal{F}_{\text{MAX SALARY}} (\sigma_{\text{DNO=5}} (\text{EMPLOYEE}))$$

Query Optimization

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

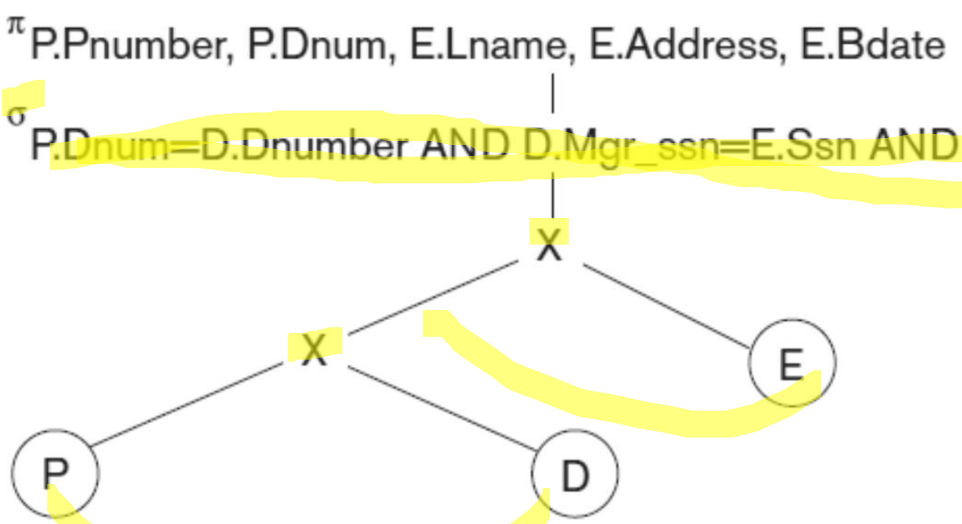
$$\pi_{PNUMBER,\ DNUM,\ LNAME,\ ADDRESS,\ BDATE}\ (((\sigma_{PLOCATION='STAFFORD'}(PROJECT)) \bowtie_{DNUM=DNUMBER}\ (DEPARTMENT)) \bowtie_{MGRSSN=SSN}\ (EMPLOYEE))$$

- SQL query:

```
Q2:     SELECT      P.NUMBER,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';
```

Initial Query Tree

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr\_ssn=Ssn}}(\text{EMPLOYEE}))$$



Improved Query Tree

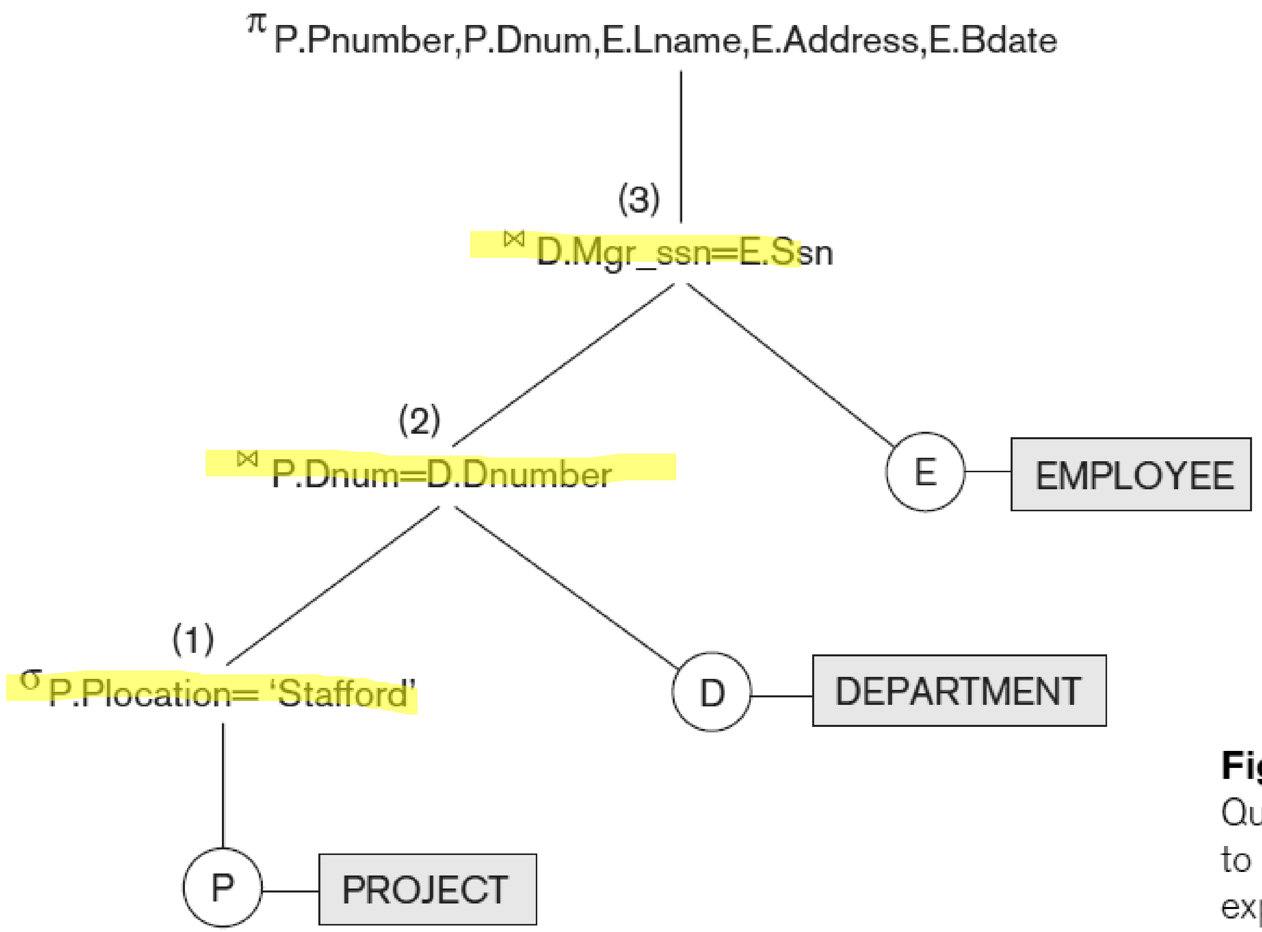


Figure 6.9
Query tree corresponding to the relational algebra expression for Q2.

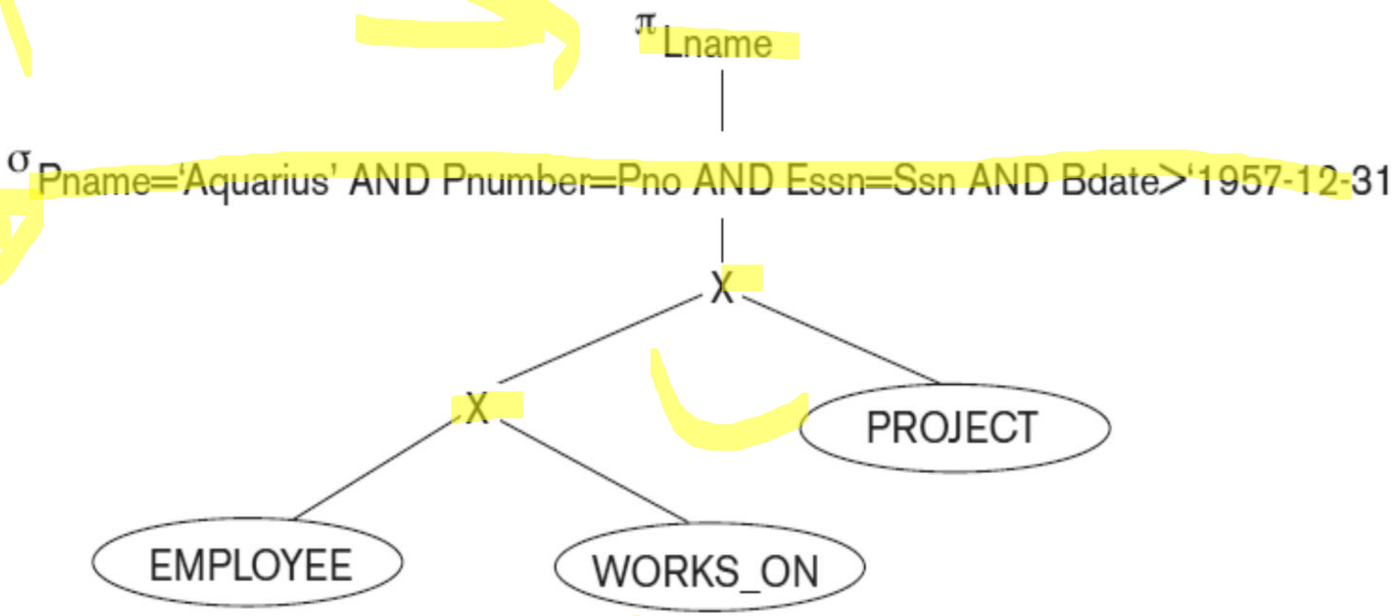
Example

- Find the last names of employees born after 1957 who work on a project named ‘Aquarius’.

- SELECT Lname
  FROM EMPLOYEE, WORKS_ON, PROJECT
  WHERE Pname=‘Aquarius’ AND Pnumber=Pno
  AND Essn=Ssn AND Bdate > ‘1957-12-31’;

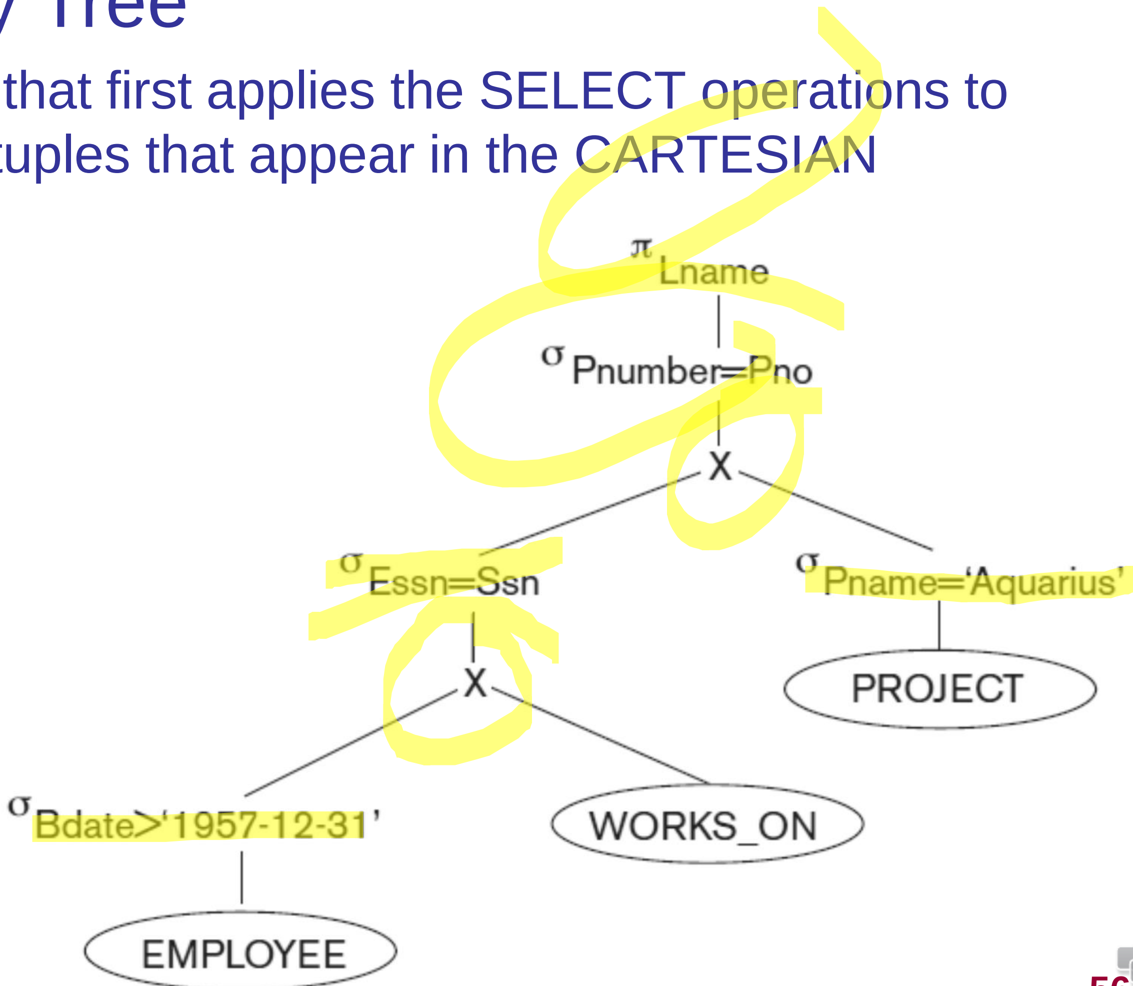
$\pi_{Lname}(\sigma_{Pname='Aquarius' \text{ AND } Pnumber=Pno \text{ AND } Essn=Ssn \text{ AND } Bdate>'1957-12-31'}$ (EMPLOYEE X WORKS_ON X PROJECT))

1- Initial Query Tree



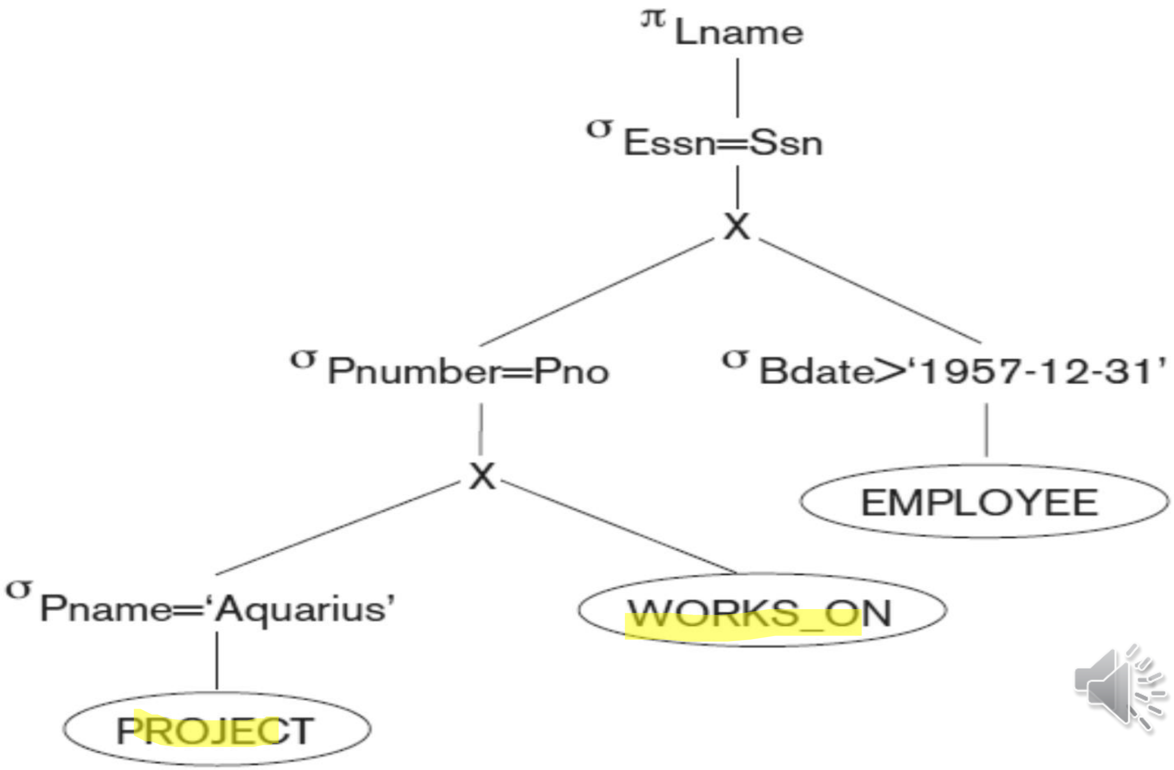
2- Improved Query Tree

An **improved query tree** that first applies the SELECT operations to reduce the number of tuples that appear in the CARTESIAN PRODUCT



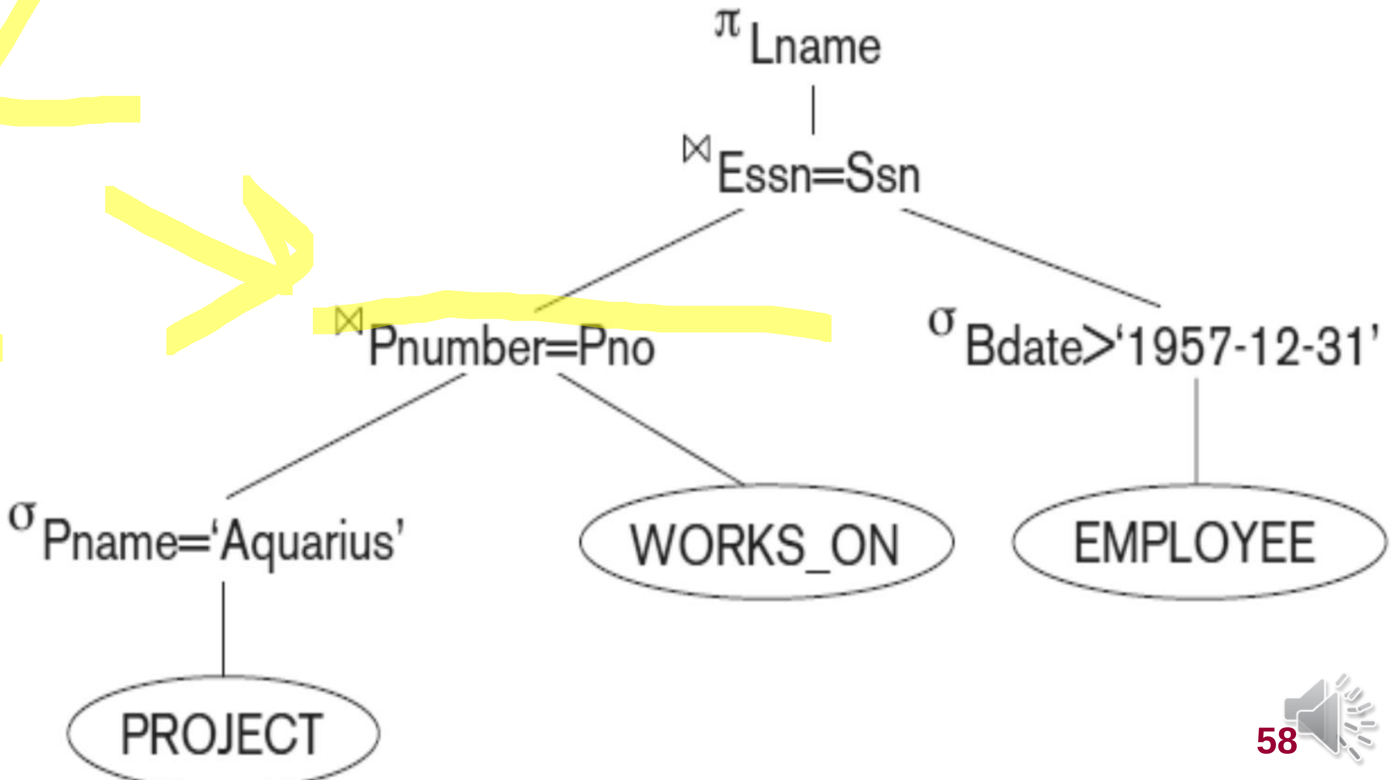
3- Improved Query Tree

A further improvement is achieved by **switching the positions** of the EMPLOYEE and PROJECT relations in the tree:



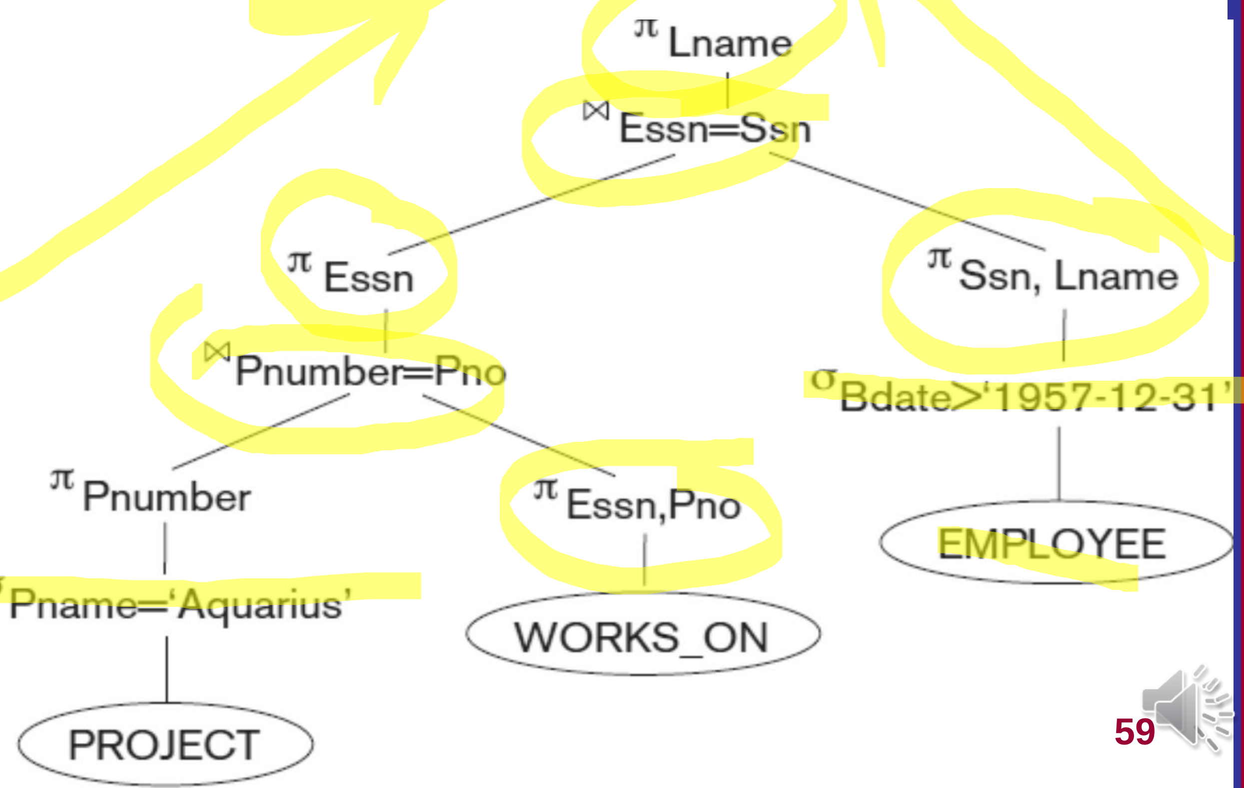
4- Improved Query Tree

We can further improve the query tree by **replacing any CARTESIAN PRODUCT operation that is followed by a join condition with a JOIN operation**:



5- Improved Query Tree

Another improvement is to **keep only the attributes needed** by subsequent operations in the intermediate relations, by **including PROJECT (π) operations as early as possible in the query tree**:



Rules

- 1. **Break up any SELECT** operations with conjunctive conditions into a cascade of SELECT operations. This permits a greater degree of freedom in moving SELECT operations down different branches of the tree.
- 2. **Move each SELECT** operation as far down the query tree as is permitted by the attributes involved in the select condition.
  - If the condition involves attributes from only one table, which means that it represents a selection condition, the operation is moved all the way to the leaf node that represents this table.
- 3. **Rearrange the leaf nodes** of the tree, position the leaf node relations with **the most restrictive SELECT** operations so they are executed first in the query tree representation. The definition of most restrictive SELECT can mean either **the ones that produce a relation with the fewest tuples or with the smallest absolute size**.

- 4. Combine a CARTESIAN PRODUCT operation with a subsequent SELECT operation in the tree **into a JOIN operation**, if the condition represents a join condition.
- 5. **Break down and move lists of projection attributes down** the tree as far as possible by creating new PROJECT operations as needed.

- Summary of Heuristics for Algebraic Optimization:
  1. The main heuristic is to apply first the operations that **reduce the size of intermediate results**.
  2. **Perform select operations as early as possible** to reduce the number of tuples and **perform project operations as early as possible** to reduce the number of attributes. (This is done by moving select and project operations as far down the tree as possible.)
  3. The select and join operations that are most restrictive should be executed before other similar operations. (This is done by reordering the leaf nodes of the tree among themselves and adjusting the rest of the tree appropriately.)

- Cost Components for Query Execution
  1. Access cost to secondary storage
  2. Computation cost
  3. Communication cost

- Note: Different database systems may focus on different cost components.
  - Large (1) vs. small (2) and distributed (3) databases