Query Processing and Optimization

Dr. L.M. Jenila Livingston

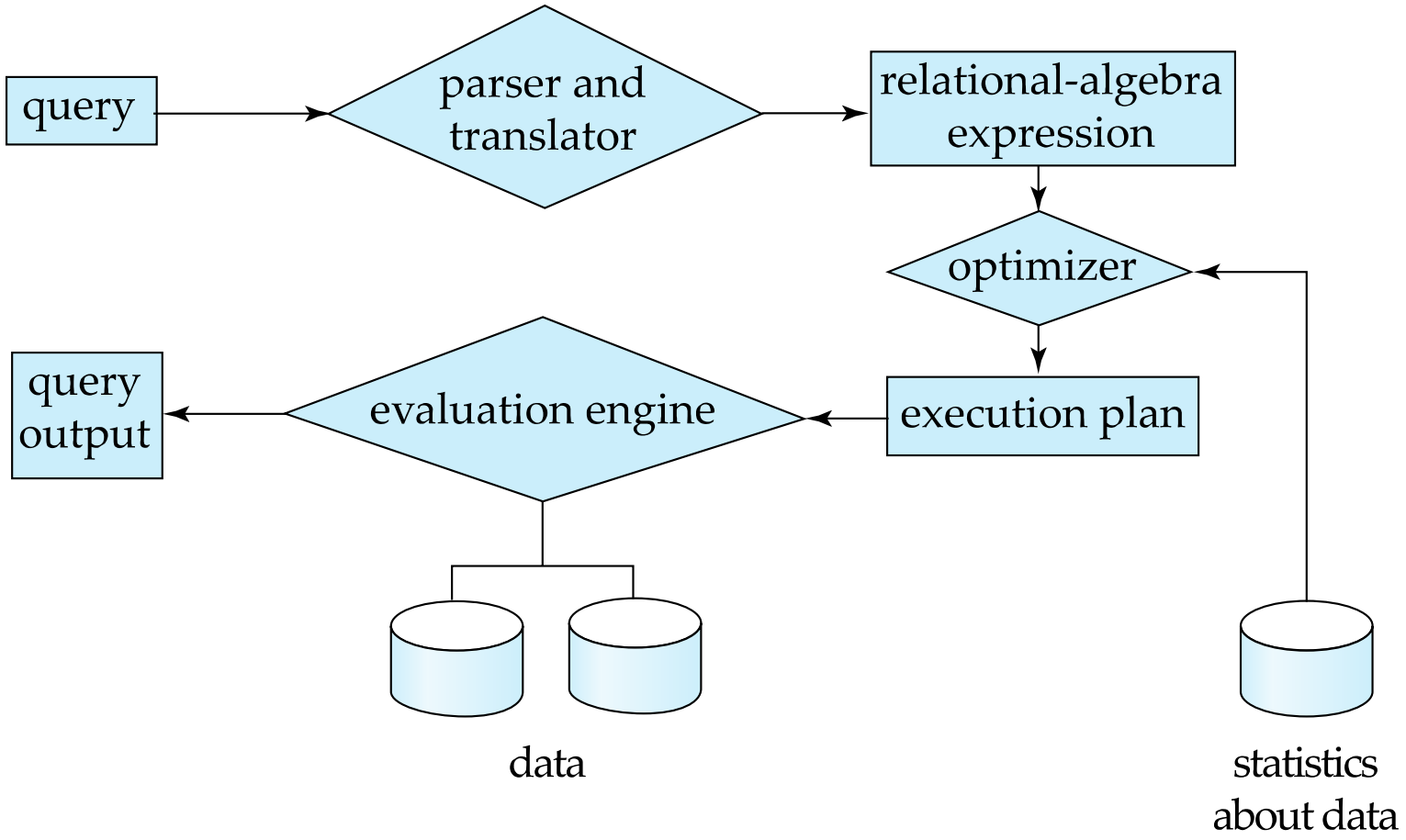
Introduction to Query Processing

- **Query optimization**: the process of choosing a suitable execution strategy for processing a query.

- Internal representation of a query
  - **Query Tree**

Basic Steps in Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



$R \cup S$	union
$R \cap S$	intersection
$R \setminus S$	set difference
$R \times S$	Cartesian product
$\pi_{A1, A2, ..., An}(R)$	projection
$\sigma_{F}(R)$	selection
$R \bowtie S$	natural join
$R \bowtie_{\theta} S$	theta-join
$R \div S$	division
$\rho\ [A1\ B1, .., An\ Bn]$	rename

https://www.coursera.org/lecture/data-manipulation/relational-algebra-operators-theta-join-ppEF5

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.
- Aggregate operators in SQL must be included in the extended algebra.

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
```

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

$\pi g_{\text{MAX(SALARY)}} (\sigma_{\text{DNO=5}} (\text{EMPLOYEE}))$

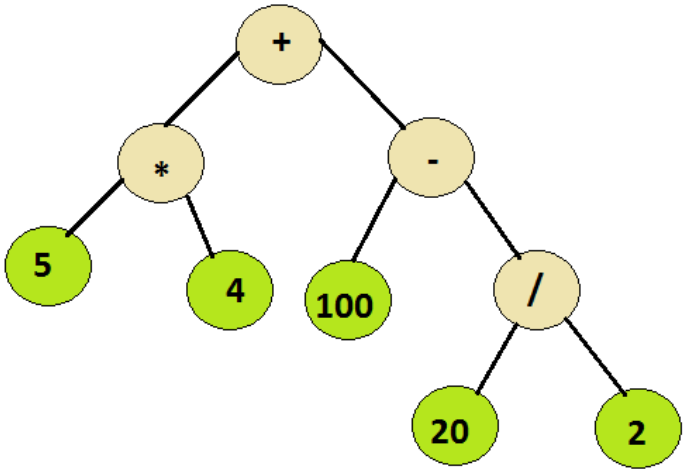
Using Heuristics in Query Optimization

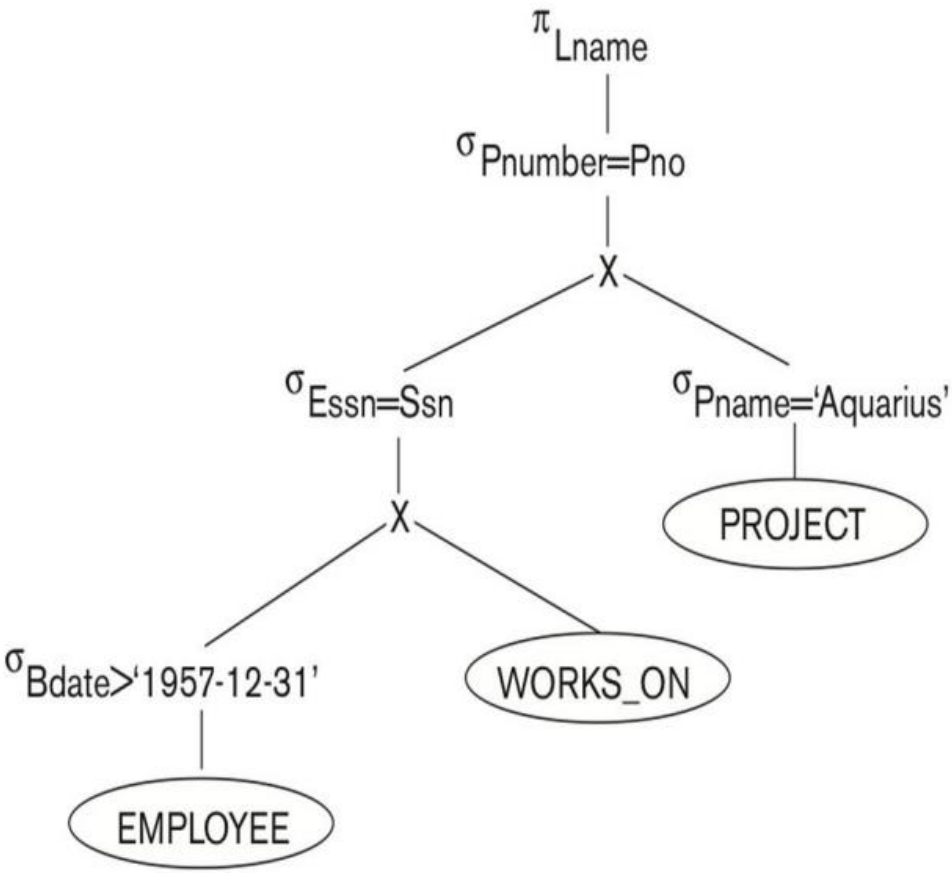
- ***query tree*** is used to represent a ***relational algebra*** or extended relational algebra expression, whereas a ***query graph*** is used to represent a *relational calculus expression*.

Using Heuristics in Query Optimization

- **Query tree**: a tree data structure that corresponds to a relational **algebra expression**. It represents the input relations of the query as ***leaf nodes*** of the tree, and represents the relational algebra operations as ***internal nodes***.
- An execution of the query tree consists of executing an internal node operation whenever its operands are available and then replacing that internal node by the relation that results from executing the operation.

Expression Tree Vs Query Tree



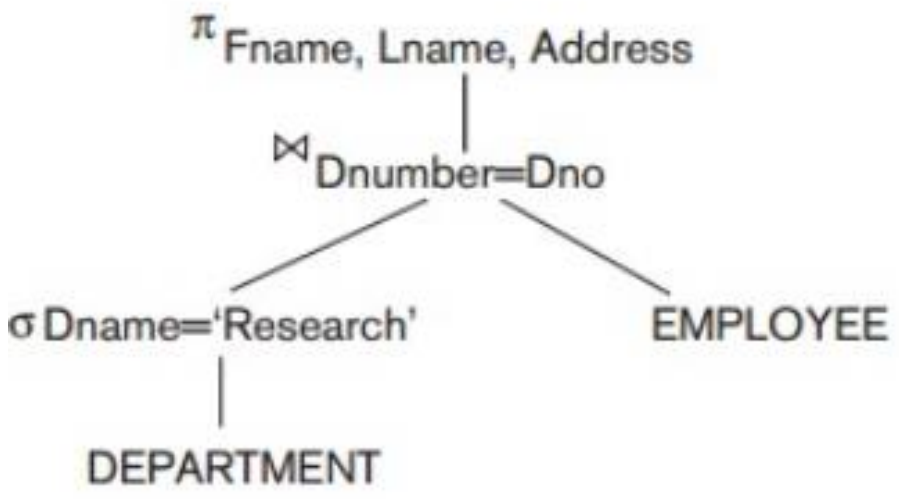


Query Tree

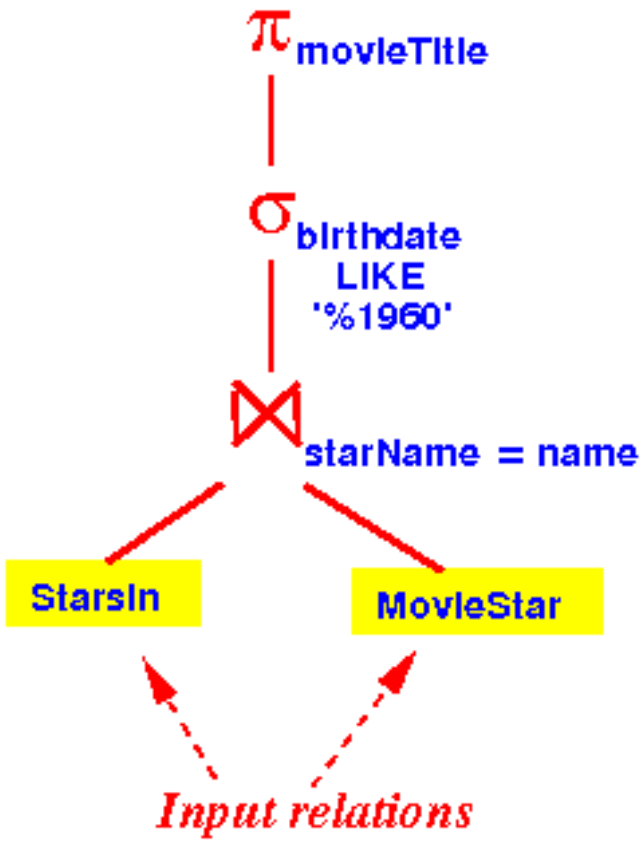
Relational algebra

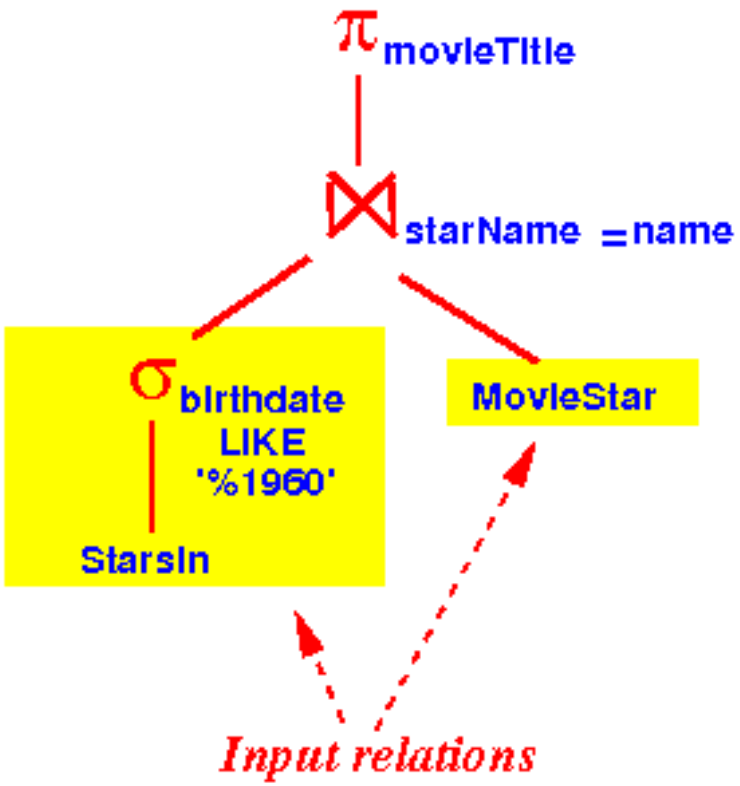
$$\pi_{\text{Fname, Lname, Address}}(\sigma_{\text{Dname='Research'}}(\text{DEPARTMENT}) \bowtie_{\text{Dnumber=Dno}} \text{EMPLOYEE})$$

Corresponding Query Tree



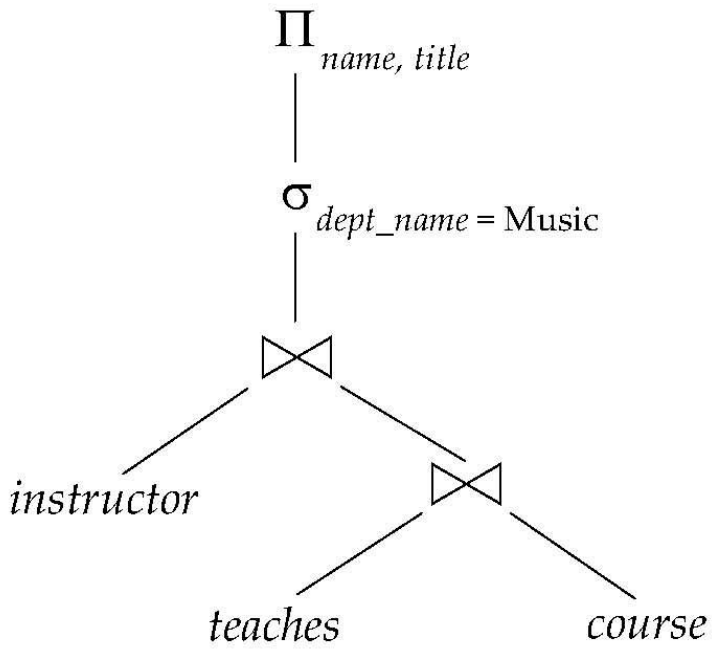
Query Tree

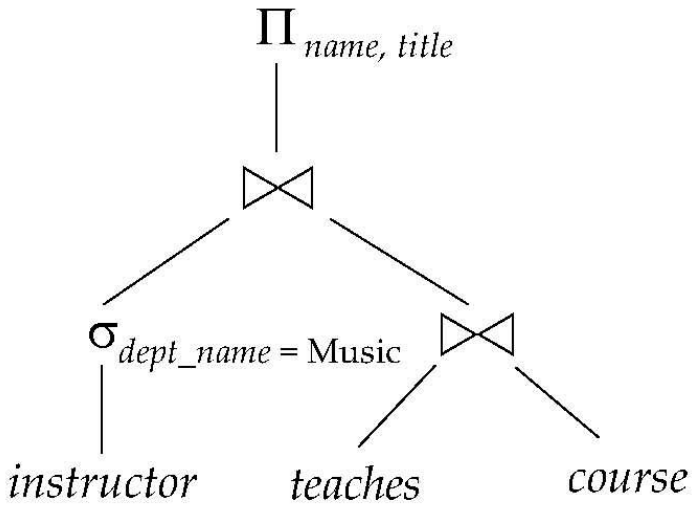

π movieTitle
σ birthdate LIKE '%1960'
⋈ starName = name
StarsIn
MovieStar
Input relations


π movieTitle
⋈ starName =name
σ birthdate LIKE '%1960'
StarsIn
MovieStar
Input relations

Query Tree

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation





Using Heuristics in Query Optimization

General Transformation Rules for Relational Algebra Operations:

1. **Cascade of σ:** A **conjunctive selection** condition can be broken up **into a cascade** (sequence) of individual s operations:

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

2. **Commutativity of σ:** The σ operation is commutative:

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

3. **Cascade of π:** In a cascade (sequence) of π operations, all but the last one can be ignored:

   $$\pi_{List1}(\pi_{List2}(\ldots(\pi_{Listn}(R))\ldots)) = \pi_{List}(R)$$

4. **Commuting σ with π:** If the selection condition c involves only the attributes A1, ..., An in the projection list, the two operations can be commuted:

   $$\pi_{A1, A2, \ldots, An}(\sigma_c(R)) = \sigma_c(\pi_{A1, A2, \ldots, An}(R))$$

Using Heuristics in Query Optimization (2)

General Transformation Rules for Relational Algebra

5. **Commutativity of $\bowtie$ (and x):** The $\bowtie$ operation is commutative as is the x operation:

   $R \bowtie_C S = S \bowtie_C R;$ $\qquad R \times S = S \times R$

6. **Commuting σ with $\bowtie$ (or x):** 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 follows ($_{c0}$ is join condition) :

   $\sigma_c (R \bowtie_{c0} S) = (\sigma_c (R)) \bowtie_{c0} S$

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

   $\sigma_c (R \bowtie_{c0} S) = (\sigma_{c1}(R)) \bowtie_{c0} (\sigma_{c2} (S))$

Using Heuristics in Query Optimization (3)

General Transformation Rules for Relational Algebra Operations (cont.):

7. **Commuting π with $\bowtie$ (or x):** Suppose that the projection list is L = {A1, ..., An, B1, ..., Bm}, where A1, ..., An are attributes of R and B1, ..., Bm are attributes of S. If the join condition c involves only attributes in L, the two operations can be commuted as follows:

$$\pi_L (R \bowtie_C S) = (\pi_{A1, ..., An} (R)) \bowtie_C (\pi_{B1, ..., Bm} (S))$$

If the join condition c contains additional attributes not in L, these must be added to the projection list, and a final π operation is needed.

Using Heuristics in Query Optimization (4)

General Transformation Rules for Relational Algebra Operations (cont.):

8. **Commutativity of set operations:** The set operations ∪ and ∩ are commutative but – is not.

$$R \cup S = S \cup R; \qquad R \cap S = S \cap R$$

8. **Associativity of ⋈ , x, ∪, and ∩ :** These four operations are individually associative; that is, if θ stands for any one of these four operations (throughout the expression), we have $(R \theta S) \theta T = R \theta (S \theta T)$

9. **Commuting σ with set operations:** The σ operation commutes with ∪ , ∩ , and –. If θ stands for any one of these three operations, we have

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

Using Heuristics in Query Optimization (5)

General Transformation Rules for Relational Algebra Operations (cont.):

11. **The π operation commutes with U.**

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

12. **Converting a (σ, x) sequence into $\bowtie$:** If the condition c of a σ that follows a x Corresponds to a join condition, convert the (σ, x) sequence into a $\bowtie$ as follows:

$$(\sigma_C (R \times S)) = (R \bowtie_C S)$$

Using Heuristics in Query Optimization (6)

Outline of a Heuristic Algebraic Optimization Algorithm:

1. Using rule 1, break up any select operations with conjunctive conditions into a cascade of select ops.
2. Using rules 2, 4, 6, and 10 concerning the commutativity 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.
3. Using rule 9 concerning associativity of binary operations, rearrange the leaf nodes of the tree so that the leaf node relations with the most restrictive select operations are executed first in the query tree rep.

Using Heuristics in Query Optimization (7)

Outline of a Heuristic Algebraic Optimization Algorithm (cont.)

4. Using Rule 12, combine a Cartesian product operation with a subsequent select operation in the tree into a join operation.
5. Using rules 3, 4, 7, and 11 concerning the cascading of project and the commuting of project with other operations, break down and **move lists of projection attributes down the tree as far as possible** by creating new project operations as needed.

Using Heuristics in Query Optimization (8)

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. **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**.)

Transformation Algorithm Outline

- Transform a query represented in relational algebra to an equivalent one (generates the same result.)
- **Step 1:** Design the initial canonical tree (initial query tree) of the query
- **Step 2:** Decompose SELECT(σ) operations and move the SELECT operation down the query tree
- **Step 3:** Apply more restrictive SELECT (σ) operation first. If the two relations are residing at same site, they will be handled first.
- **Step 4:** Replace CARTESIAN PRODUCT(x) and SELECT(σ) with THETA JOIN (⋈) operation
- **Step 5:** Move PROJECT (π) operations down the query tree

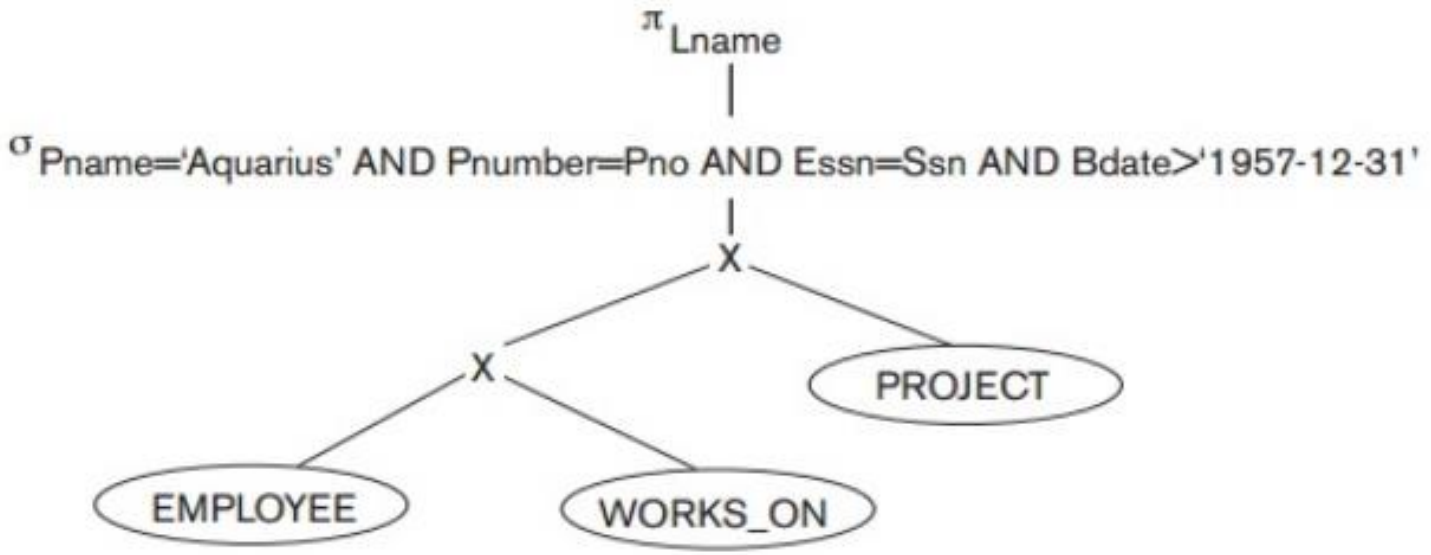
Example 1:

```
DEPARTMENT(Dname,Dnumber,Mgr_ssn,Mgr_startdate)
DEPT_LOCATIONS(Dnumber,Dlocation)
EMPLOYEE(Fname,Minit,Lname,Ssn,Bdate,Dno,Salary)
PROJECT(Pname,Pnumber,Plocation,Dnum)
WORKS_ON(Essn,Pno,Hours)
```

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

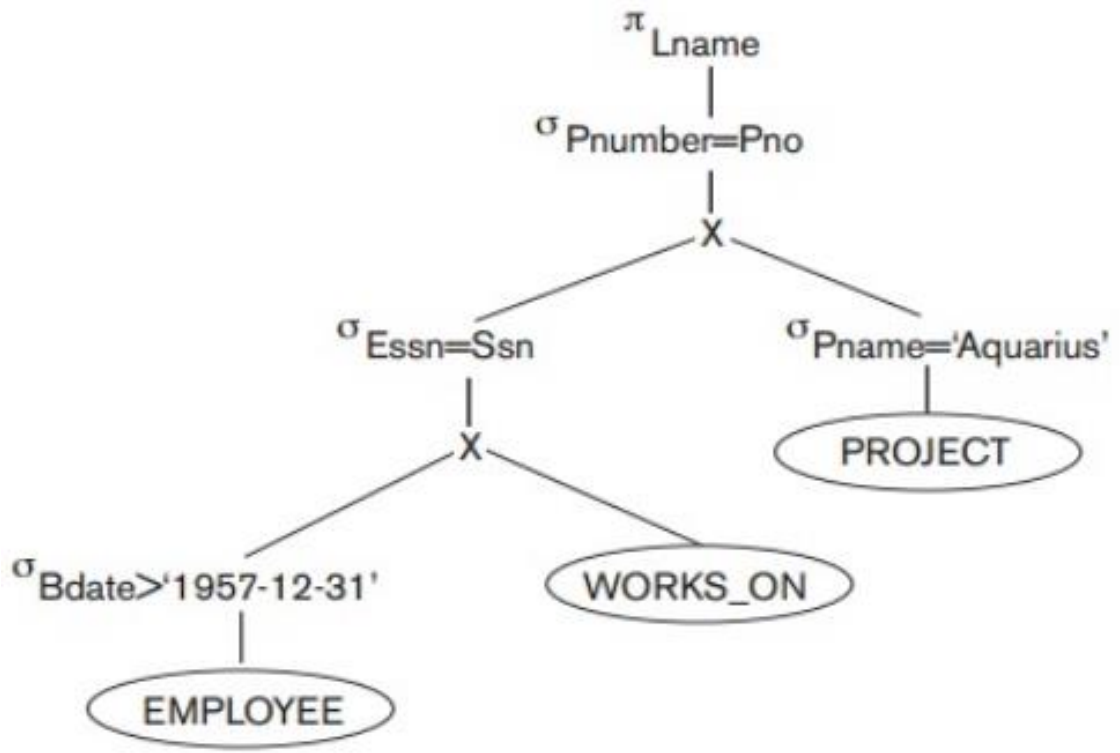
Heuristic optimization process (1)

- Step 1. Initialization canonical query tree



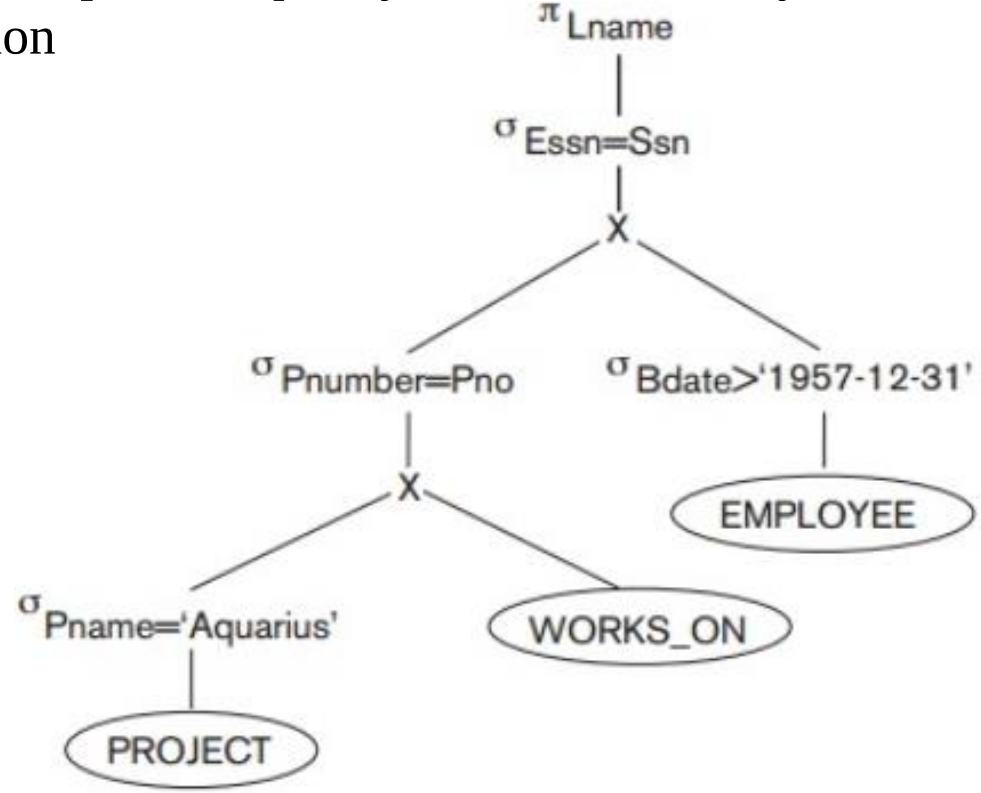
Heuristic optimization process (2)

- Step 2. Decompose SELECT(σ) operations and move the SELECT operation down the query tree



Heuristic optimization process (3)

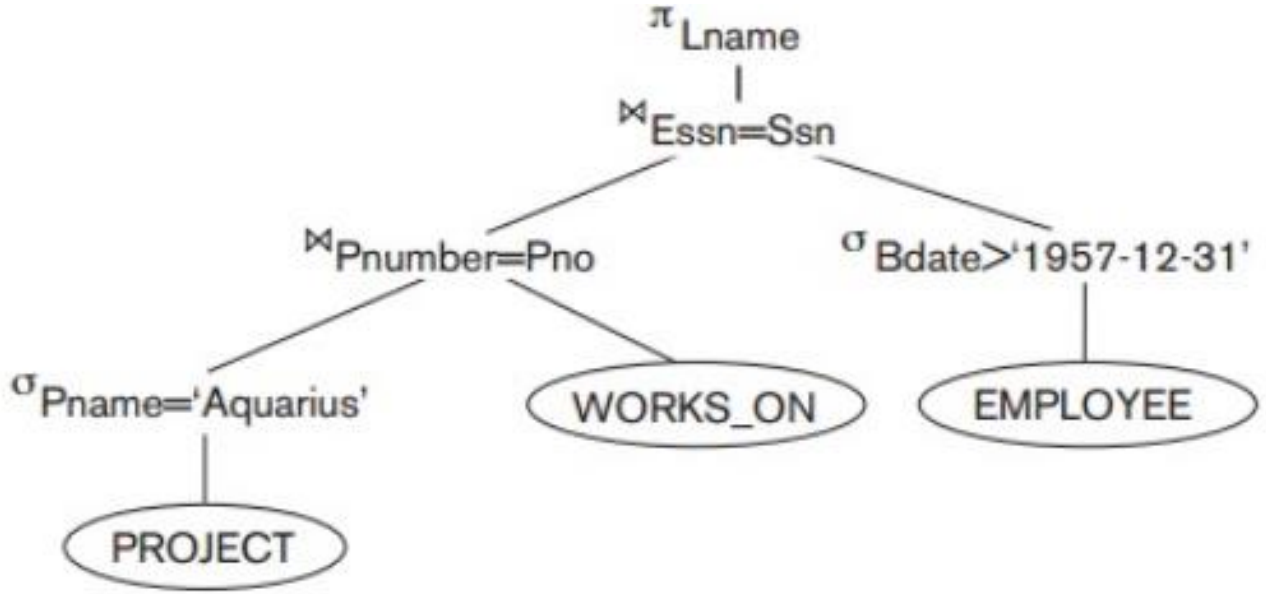
- **Step 3. Apply more restrictive SELECT operation first**
  - **most restrictive**—that is, result in relations with the fewest tuples or with the smallest absolute size—should be executed before other similar operations.
  - Using the associativity rules, rearrange the leaf nodes so that the most restrictive selection conditions are executed first. And if the two relations are residing at same site, they will be handled first.
  - For example, an equality condition is likely more restrictive than an inequality condition



Pnumber is a key attribute and with = operator will retrieve a single record only.

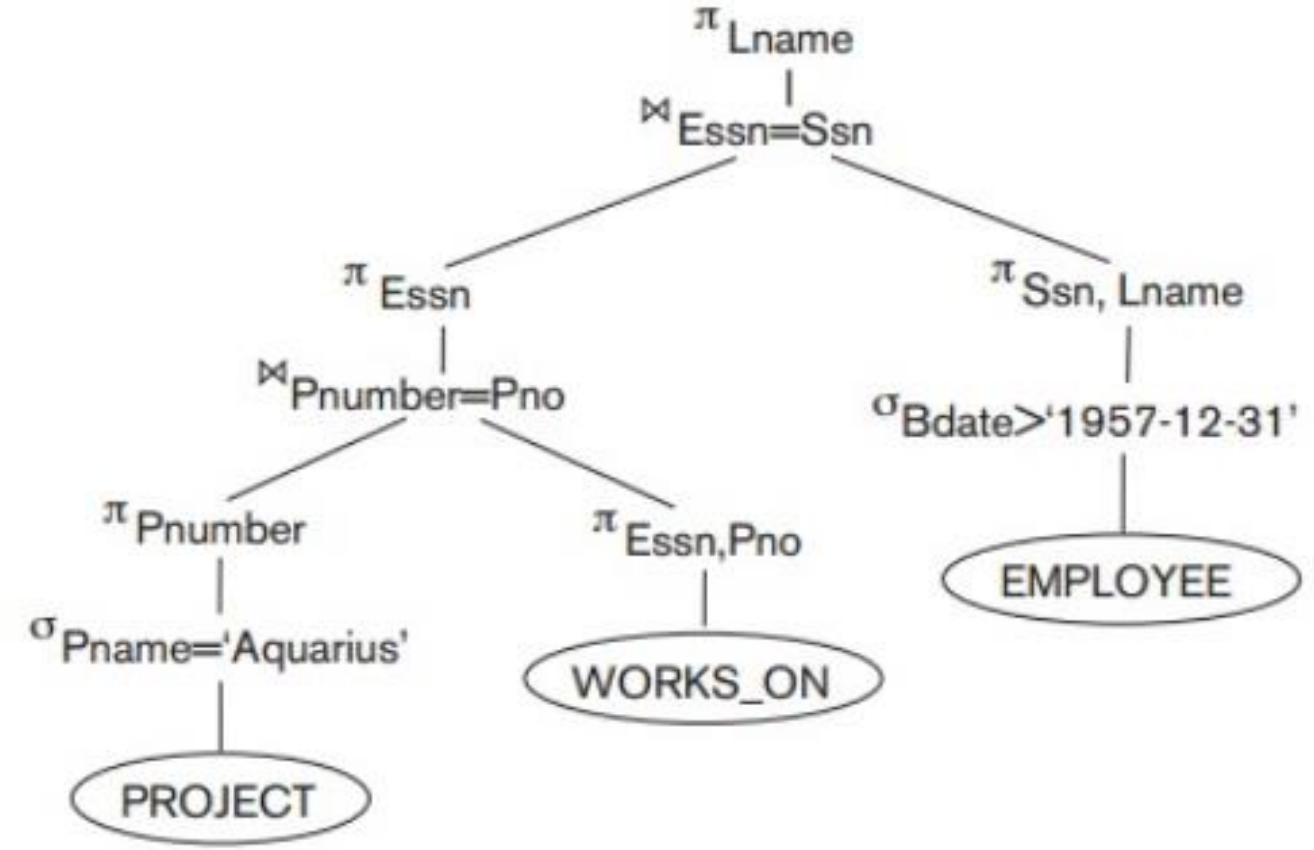
Heuristic optimization process (4)

- Step 4. Replace CARTESIAN PRODUCT and SELECT with THETA JOIN operation



Heuristic optimization process (5)

- Step 5. Move PROJECT operations down the query tree



Example 2:

- Four tables:
- Table1: SmallBusiness

SBId	Name	Address	Phone	ContactPerson

- Table2: User

UserId	UserName	FullName	password	SBId

- Table3: Journal

JournalId	JournalNo	Date	Description	Evidence	SBId	UserId

- Table 4: JournalDetail

JournalDetailId	JournalId	AccountNo	Debet	Credit

- SELECT Journal.JournalNo, Journal.Date, Journal.Description, Journal.Evidence, JournalDetail.Debet, JournalDetail.Credit, SmallBusiness.Name, User.FullName

- FROM Journal, JournalDetail, SmallBusiness, User

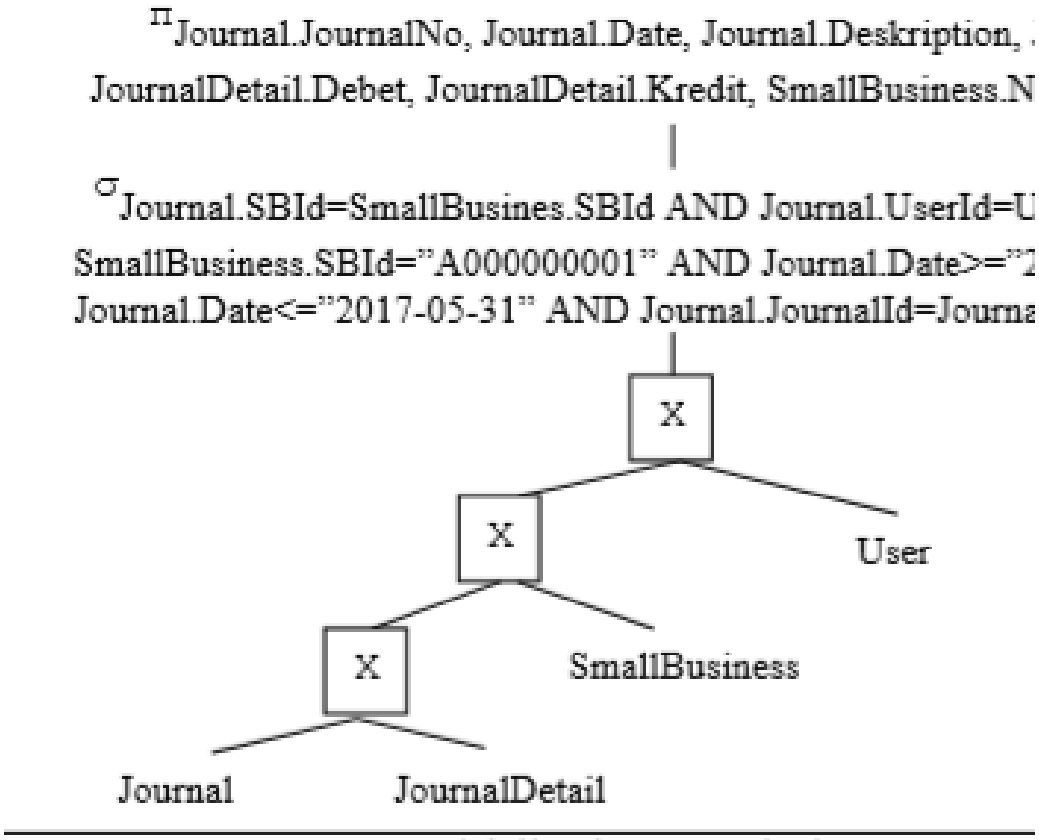
- WHERE Journal.SBId = SmallBusiness.SBId AND Journal.UserId = User.UserId AND SmallBusiness.SBId = ”A000000001” AND Journal.Date >= ”2019-05-01” AND

  Journal.Date <= ”2019-05-31” AND

  Journal.JournalNo = JournalDetail.JournalNo

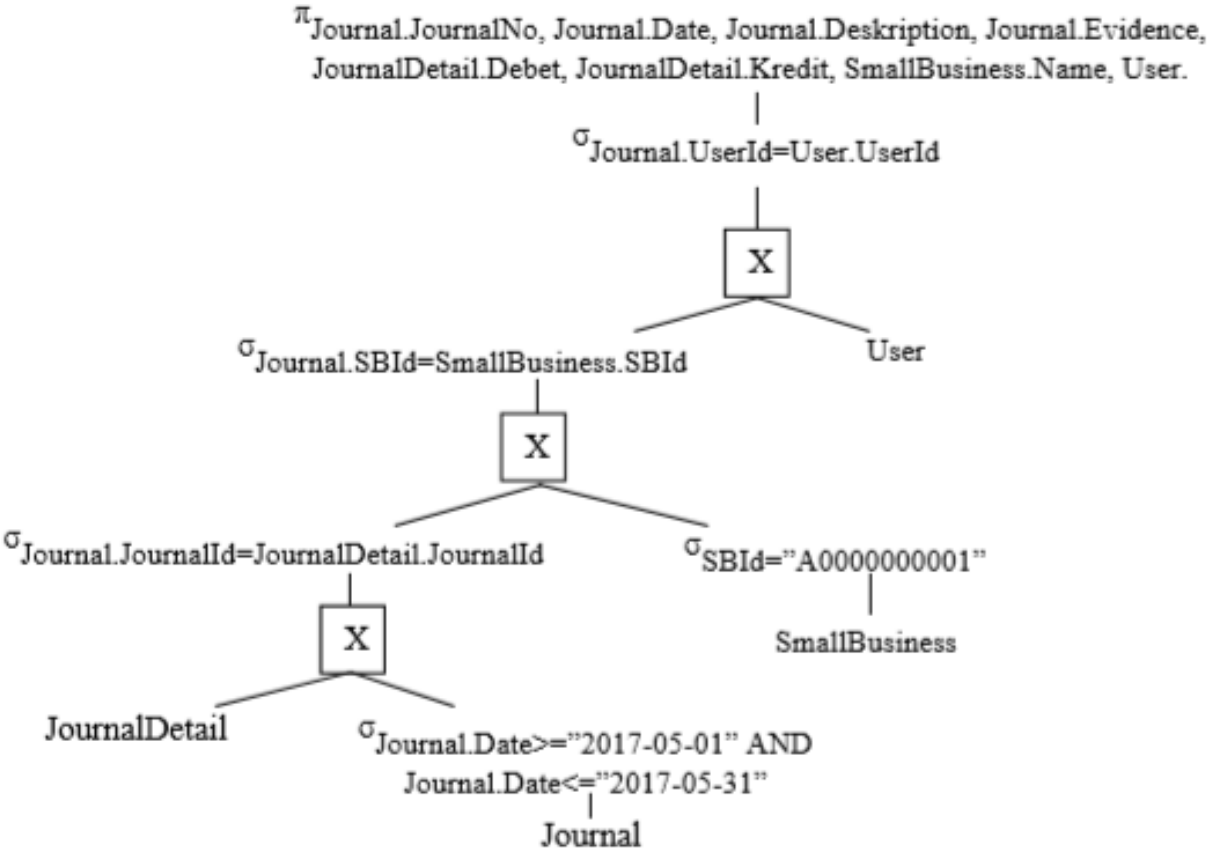
Heuristic optimization process (1)

- Step 1. Initialization canonical query tree



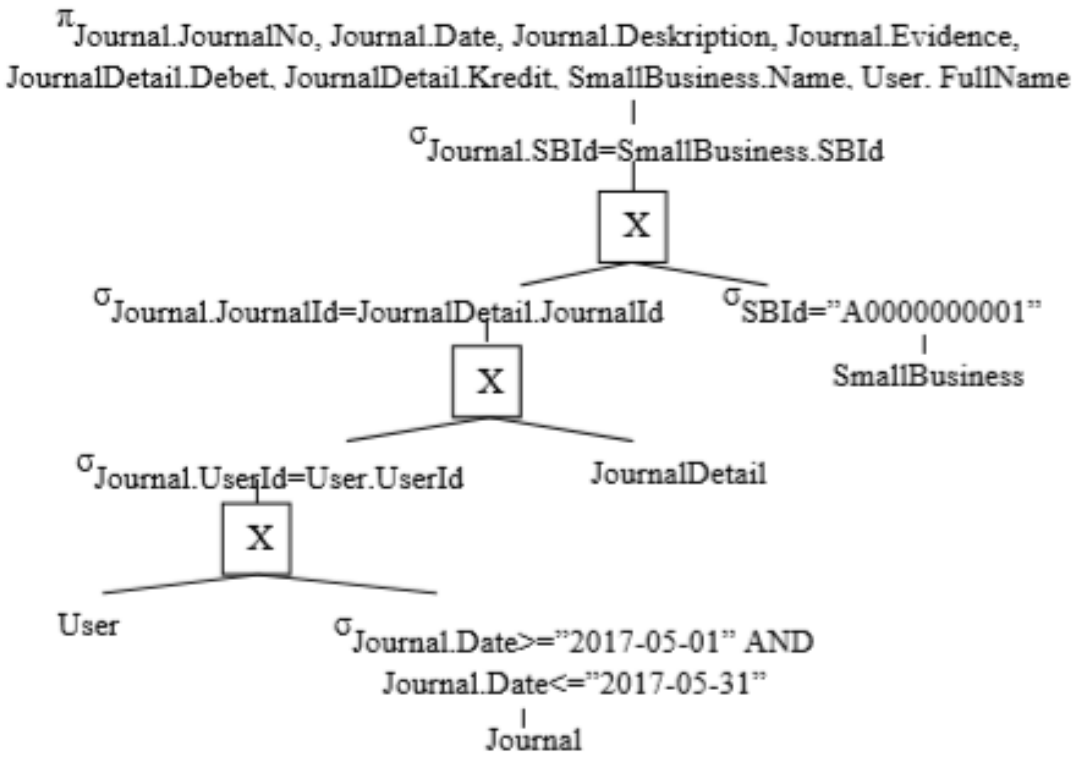
Heuristic optimization process (2)

- Step 2. Decompose SELECT(σ) operations and move the SELECT operation down the query tree



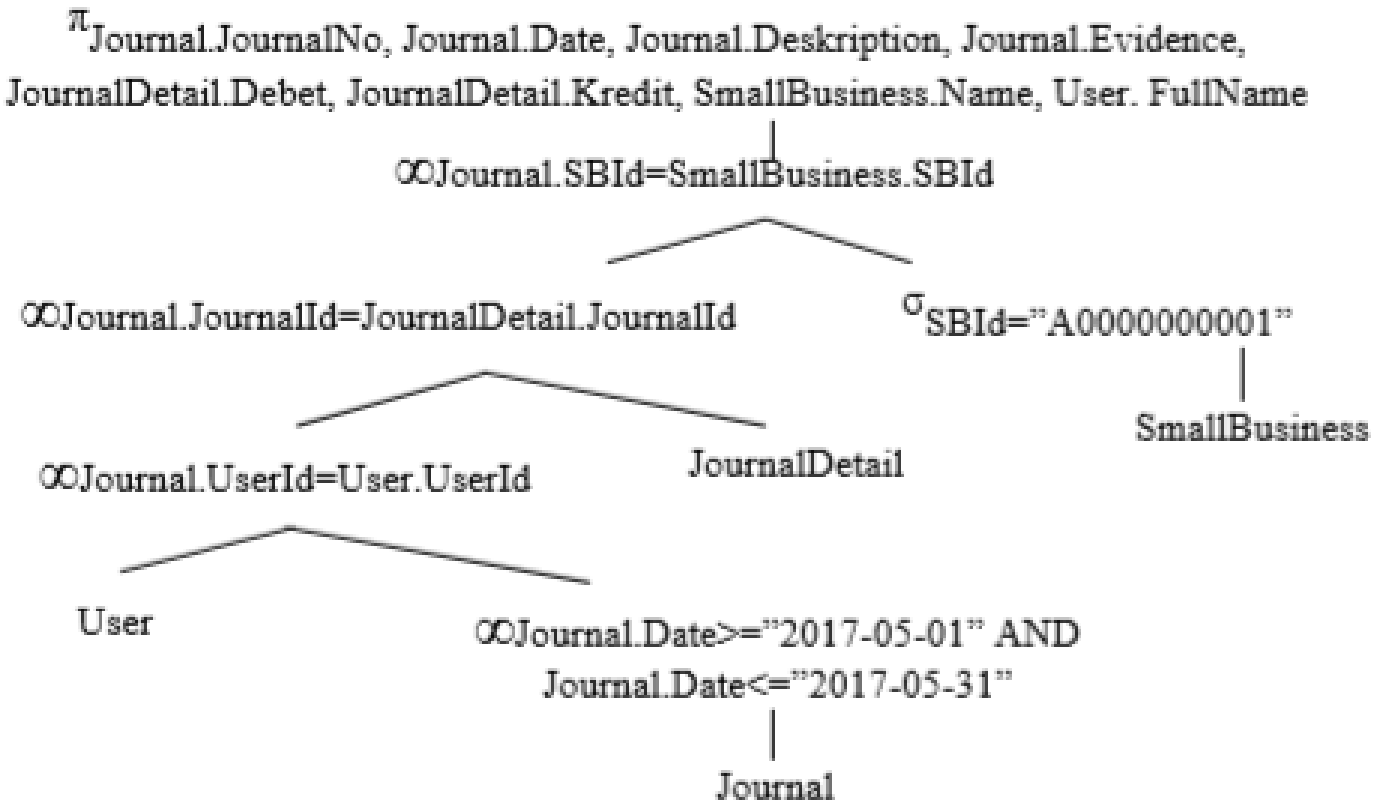
Heuristic optimization process (3)

- Step 3. Apply more restrictive SELECT operation first



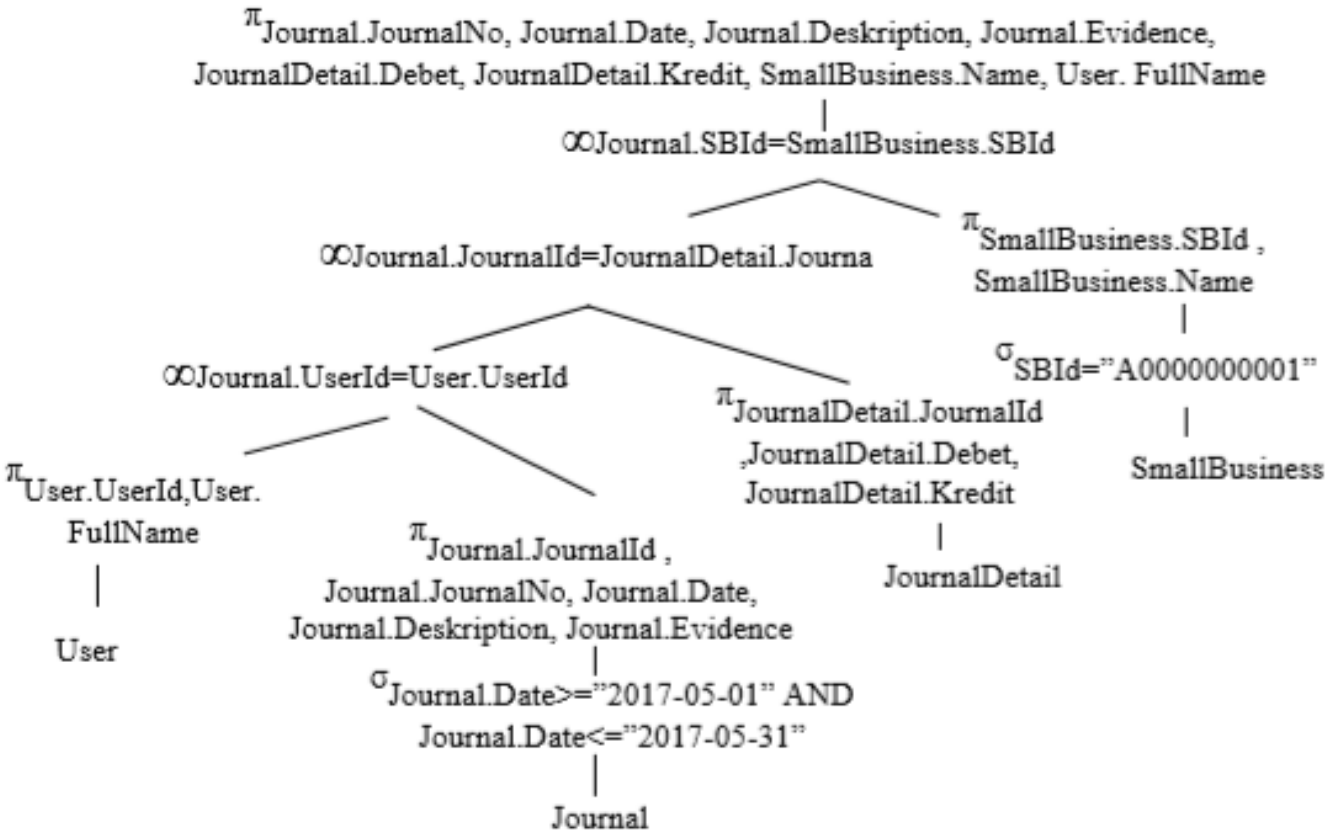
Heuristic optimization process (4)

- Step 4. Replace CARTESIAN PRODUCT and SELECT with JOIN operation



Heuristic optimization process (5)

- Step 5. Move PROJECT operations down the query tree



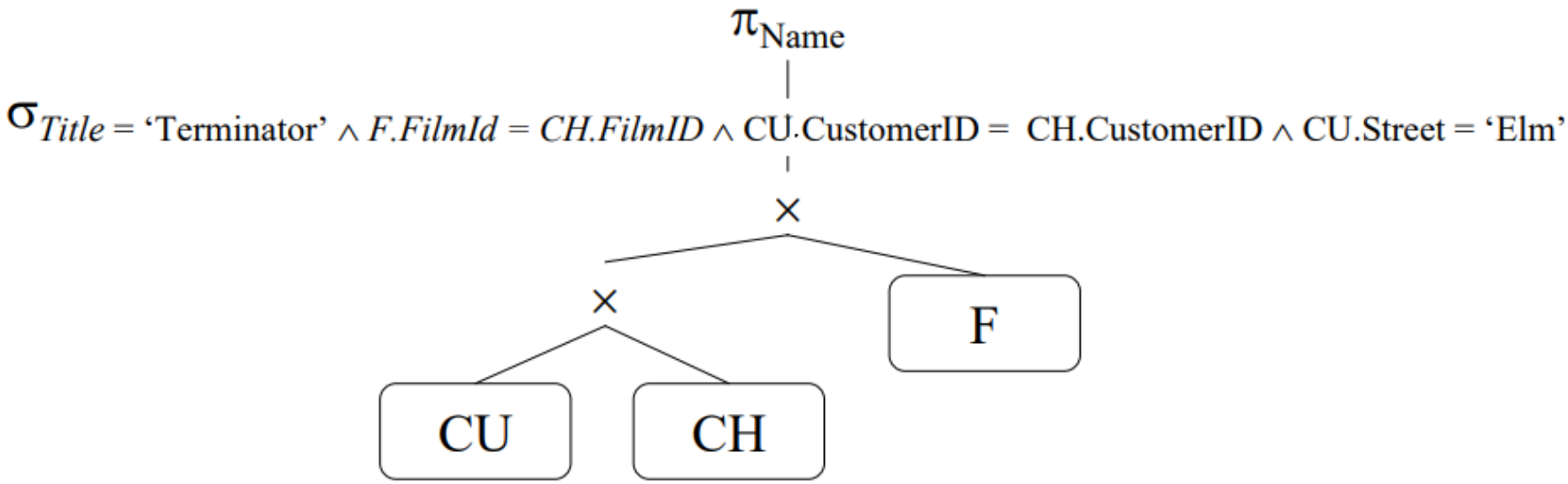
Query Tree Optimization Example 3

- What are the names of customers living on Elm Street who have checked out “Terminator”?

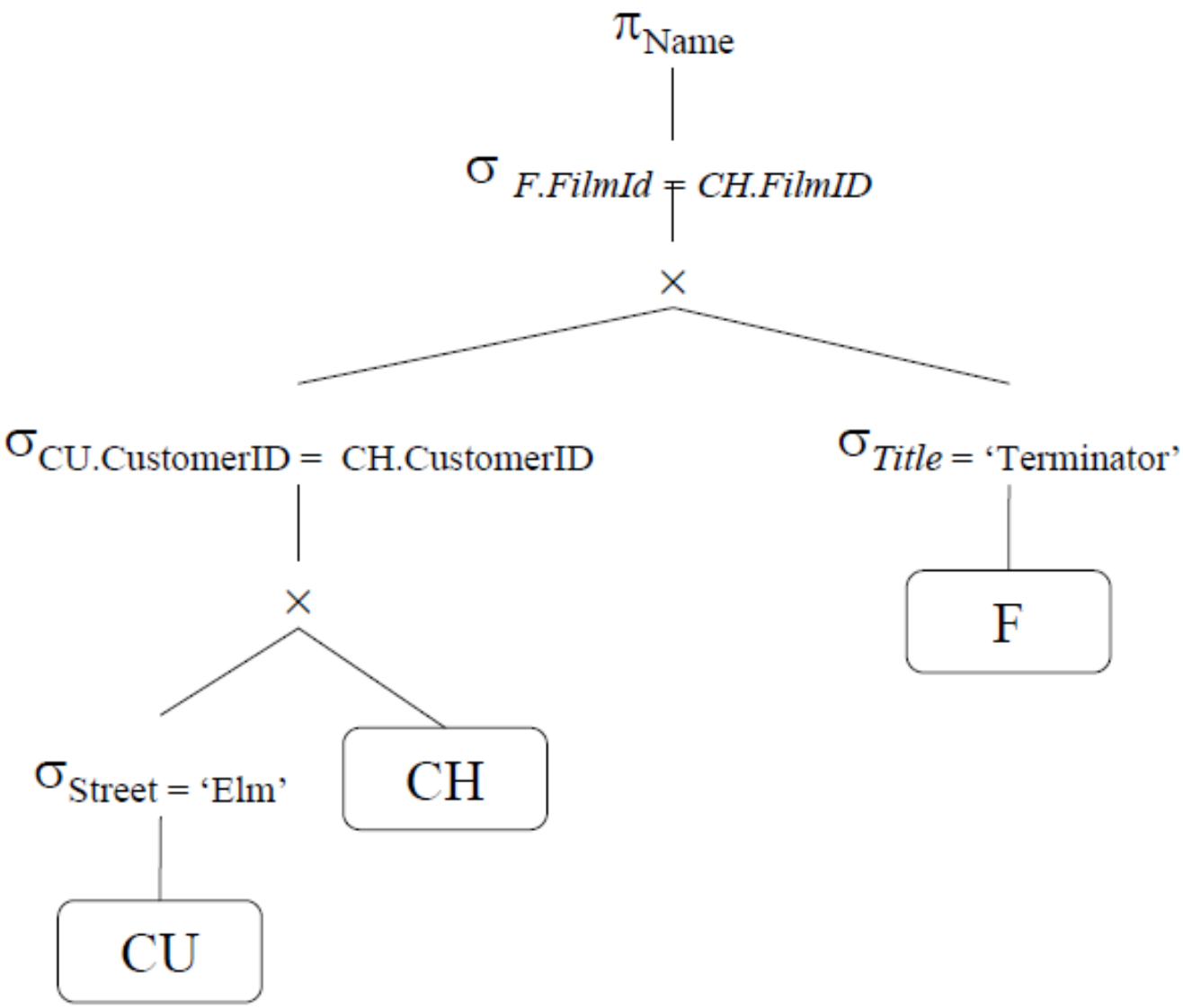
- SQL query:

```
SELECT Name
FROM   Customer CU, CheckedOut CH, Film F
WHERE  Title = 'Terminator' AND F.FilmId = CH.FilmID
AND CU.CustomerID = CH.CustomerID AND CU.Street = 'Elm'
```

Step 1. Initialization canonical query tree

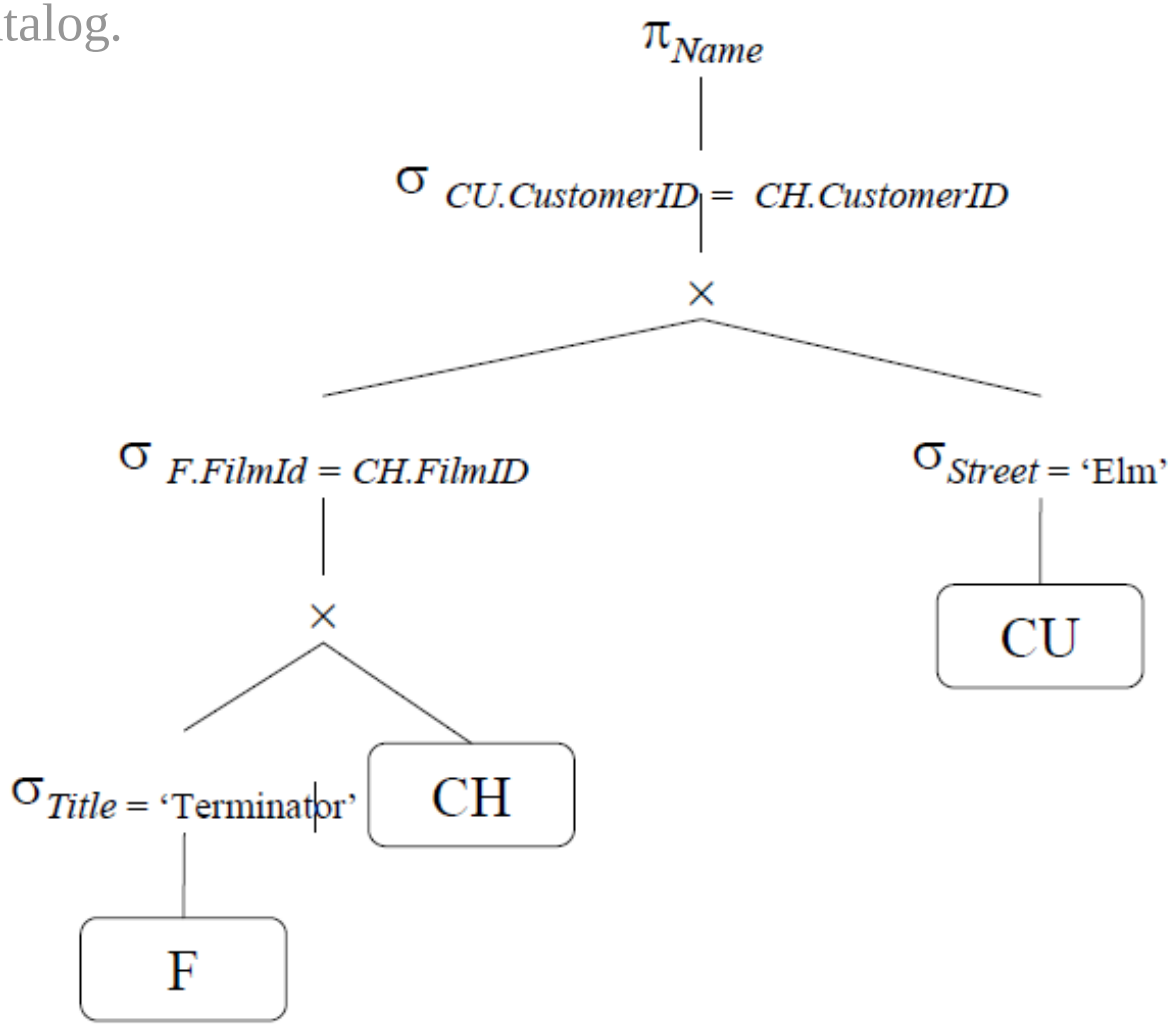


Step2: Apply Selections Early

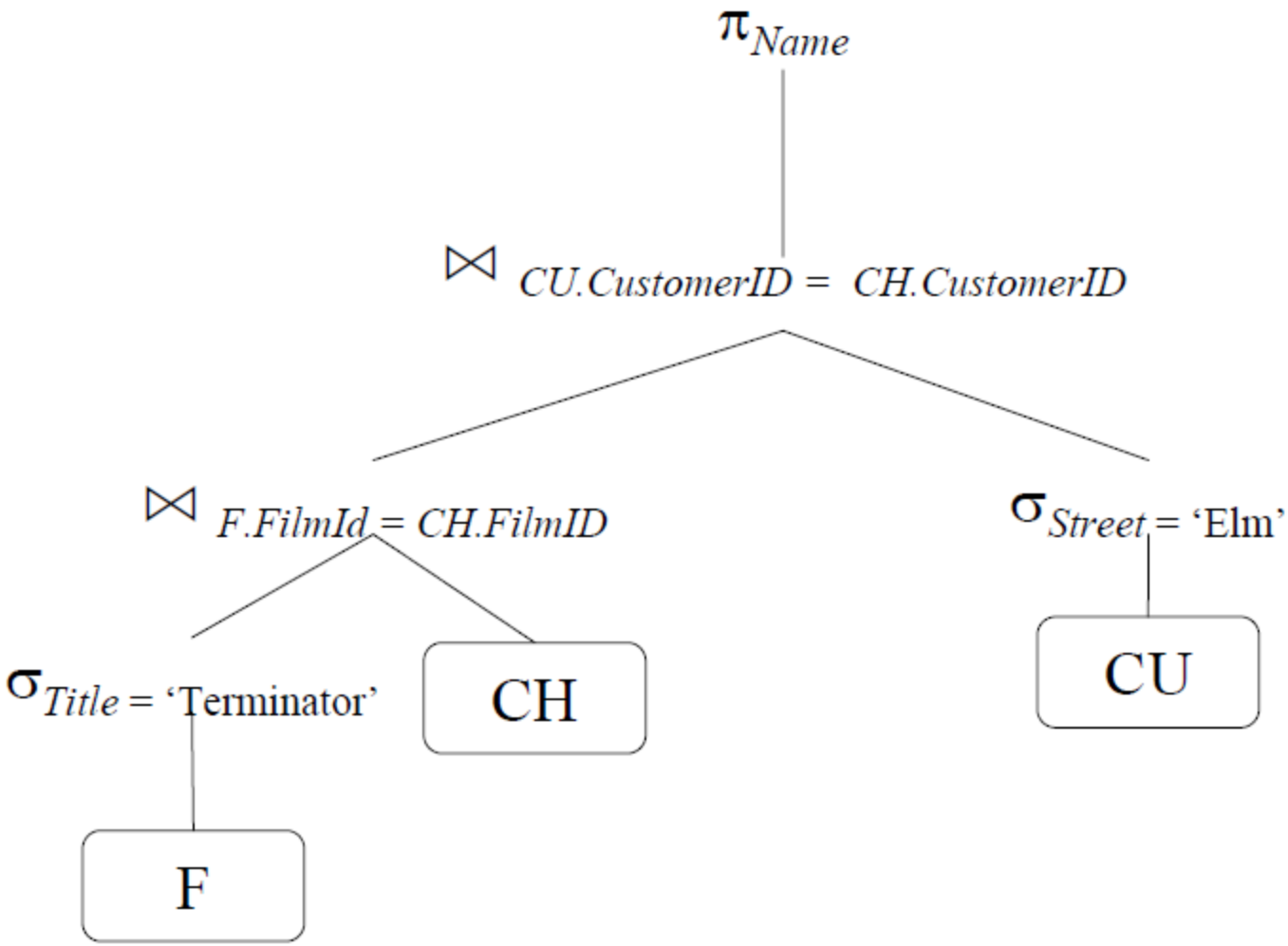


Step3: Apply More Restrictive Selections Early

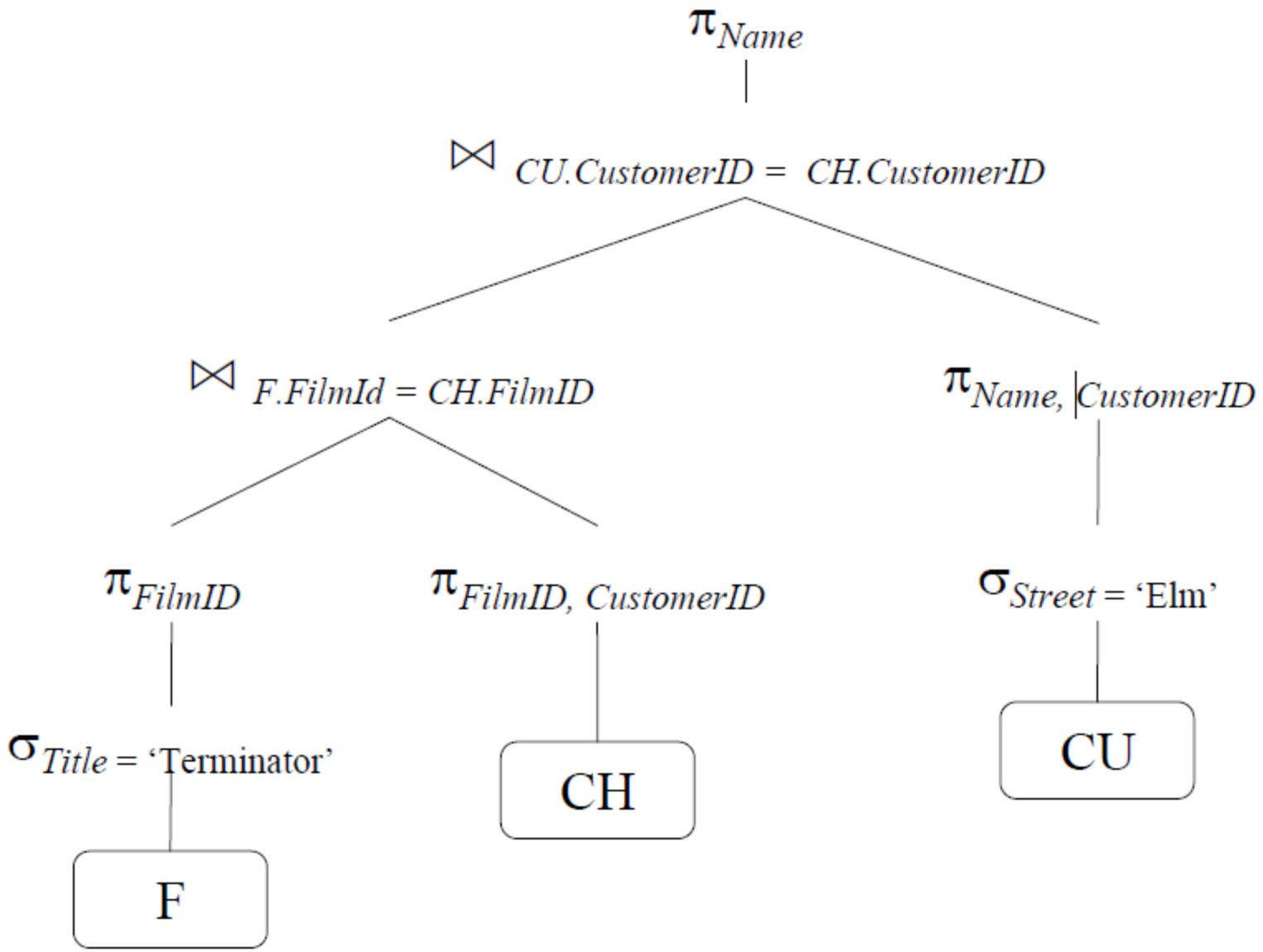
The definition of *most restrictive* SELECT can mean either the ones that produce a relation with the fewest tuples or with the smallest absolute size. this is more practical because estimates of selectivity are often available in the DBMS catalog.



Step 4: Form Joins



Step 5: Apply Projections Early



Relational Algebra Operators

Presentation by

Dr. Jenila Livingston L.M.

VIT Chennai

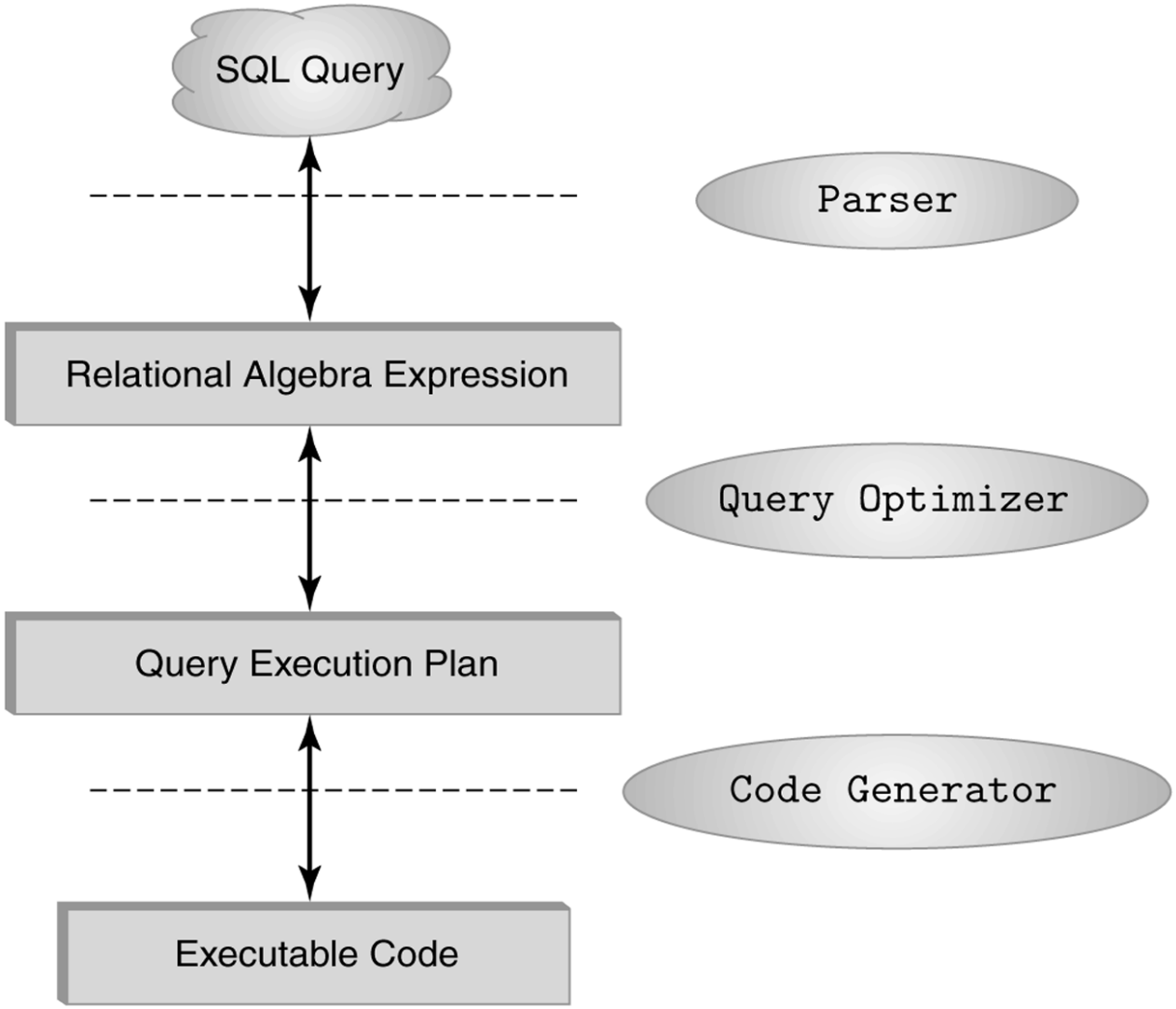
Formal Query Languages

- Specialized languages for asking questions or queries
- Theoretical languages
- Developed before SQL. SQL is based on these languages
  - Relational Algebra
  - Relational Calculus

Relational Algebra

- ***Introduced by E. F. Codd in 1970.***
- **More operational, very useful for representing execution plans.**
- **relational algebra**, a **procedural language** that defines database operations in terms of algebraic expressions.
- Similar to normal algebra (as in 2+3*x-y), except we use relations as values instead of numbers/ operands , and the operations and operators are different.
- We need to know about relational algebra to understand query execution and optimization in a relational DBMS.

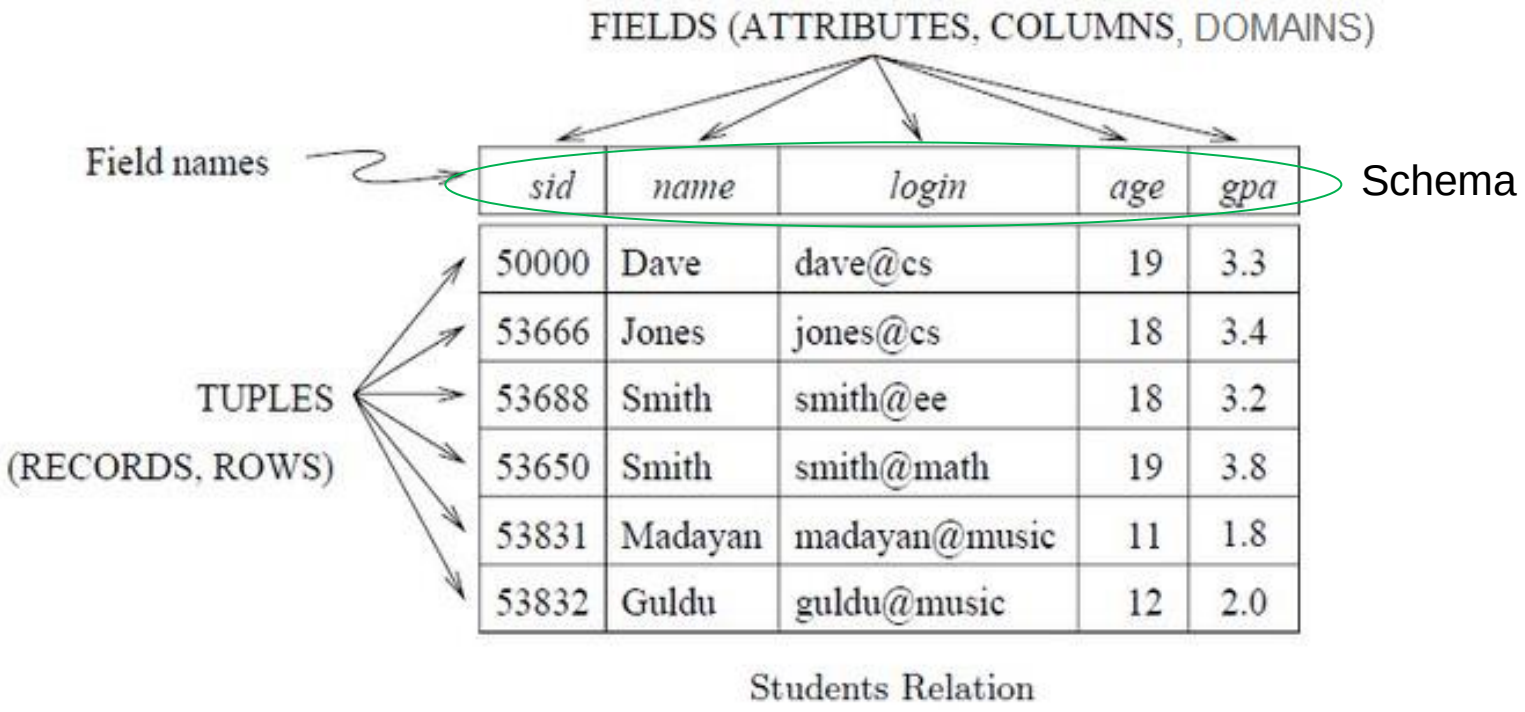
The Role of Relational Algebra in a DBMS



Relational Algebra

1. Selection of Tuples
2. Selection of Attributes
3. Set Operations
4. Join operations
5. Rename
6. Assignment
7. Aggregation

Table or Relation



sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Students Relation

Degree: No. of attributes = 5
Cardinality: No. of tuples = 6

1. Selection of tuples

Returns all tuples which satisfy a condition

- Relation r
- **Selection (σ)**

Notation: $\sigma_c(r)$

The condition c can be =, <, ≤, >, ≥, <>

- Select all attributes/ tuples from r

  $\sigma(r)$

- Select tuples with A=B and D > 5

  $\sigma_{A=B \text{ and } D > 5}(r)$

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

A	B	C	D
α	α	1	7
β	β	23	10

Select Operator

- Produce table containing subset of rows of argument table satisfying condition

$$\sigma_{condition}(relation)$$

- Example: List the persons where hobby is stamp collection

$$\sigma_{Hobby='stamps'}(Person)$$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition - Examples

- σ *Id*>3000 OR *Hobby*='hiking' (Person)
- σ *Id*>3000 AND *Id* <3999 (Person)
- σ NOT(*Hobby*='hiking') (Person)
- σ *Hobby*$\neq$'hiking' (Person)

2. Selection of Columns (Attributes)

Retains only wanted ***columns*** from relation (vertical).

- Relation *r*:
- **Projection (Π)**

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

Notation: $\Pi_{A1,\ldots,An}(r)$

- Select A and C
  - $\Pi_{A, C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operator

- Produces table containing subset of columns of argument table

$$\pi_{attribute\ list}(relation)$$

- Example: Display name and hobby of persons

$$\pi_{Name,Hobby}(\text{Person})$$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Name	*Hobby*
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator

- Example:

$\pi_{Name,Address}(Person)$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Result is a table (no duplicates); can have fewer tuples than the original

Expressions

$\pi_{Id, Name} (\sigma_{Hobby='stamps' \text{ OR } Hobby='coins'} (Person))$

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Person

Id	*Name*
1123	John
9876	Bart

Result

3. Set Operations

- Union (∪)
- Intersection (∩)
- Set Difference (-)
- Division (/)
- Cartesian Product / Cross Product

Union of two relations

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Union

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

Set Intersection of two relations

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Intersection

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r – s:*

A	B
α	1
β	1

r-S means Tuples in r1, but not in r2.

Set Difference

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

$S2 - S1$

4. Division

- Denoted by ÷
- Binary Operation
- Used in queries that include the phrase “for all”.

EXAMPLES OF DIVISION

A

sno	*pno*
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Exercise

- Selects tuples from books where subject is 'database'.
- Selects tuples from books where subject is 'database' and 'price' is 450.
- Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010
- Selects subject and author from the relation Books.

- Projects the names of the authors who have either written a book or an article or both
- Provides the name of authors who have written books but not articles.
- Yields a relation, which shows all the books and articles written by bala.

- $\sigma_{subject = "database"}(Books)$
- $\sigma_{subject = "database" \text{ and } price = "450"}(Books)$
- $\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$
- $\prod_{subject, author}(Books)$

- $\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)
- $\prod_{author}$ (Books) − $\prod_{author}$ (Articles)
- $\sigma_{author = \text{'bala'}}$(Books X Articles)

Group by and Order by

- Grouping γ
- Sorting τ

4. Joining two relations
Cartesian Product

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian Product - Example

r

A	B
a1	2
a2	4

s

C	D	E
4	d1	e1
3	d2	e1
5	d3	e2

rx s

A	B	C	D	E
a1	2	4	d1	e1
a1	2	3	d2	e1
a1	2	5	d3	e2
a2	4	4	d1	e1
a2	4	3	d2	e1
a2	4	5	d3	e2

Cartesian Product: $R \times S$

R

A	B	C
a1	b1	c3
a2	b1	c5
a3	b4	c7

S

E	F
e1	f1
e2	f5

$R \times S$

A	B	C	E	F
a1	b1	c3	e1	f1
a1	b1	c3	e2	f5
a2	b1	c5	e1	f1
a2	b1	c5	e2	f5
a3	b4	c7	e1	f1
a3	b4	c7	e2	f5

Inner Join

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Natural Join
  - r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Outer Join

- Left Outer Join ()
- Right Outer Join ()
- Full Outer Join ()

Join – Example

- Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

- Inner Join

$$loan \bowtie Borrower$$

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- Left Outer Join

loan ⟕ *Borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join – Example

- Right Outer Join

  loan ⟖ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- Full Outer Join

  loan ⟗ *borrower*

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

5. Rename operation - ρ

The operation $\rho_x(r)$

- means relation r renamed to x.

6. Assignment

- The assignment operation (←) provides a convenient way to express complex queries.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
$$result = temp1 - temp2$$

  - The result to the right of the ← is assigned to the relation variable on the left of the ←.
  - May use variable in subsequent expressions

7. Aggregate

- Functions on more than one tuple
- Samples:
  - Sum
  - Count-distinct
  - Max
  - Min
  - Count
  - Avg
- Use “as” to rename

$$\Pi_{branchname,\ \mathcal{G}\ \textbf{sum}(balance)\ \textbf{as}\ totalbalance}(account)$$

Aggregate Operation – Example

- Relation *account* grouped by *branch-name*:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$\Pi_{branch\_name}\, g_{\mathbf{sum}(balance)}\,(account)$$

branch_name	**sum**(*balance*)
Perryridge	1300
Brighton	1500
Redwood	700

Summary

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

EXAMPLES OF ALGEBRA QUERIES

In the rest of this section we shall illustrate queries using the following relations

Sailors, Reserves and Boats.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

QUERY Q1

(Q1) Find the names of sailors who have reserved boat 103

Two relations involved here are

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

$\pi_{sname}((\sigma_{bid=103}\ Reserves) \bowtie Sailors)$

The answer is thus the following relational instance

{<Dustin>, <Lubber>, <Horatio>}

QUERY Q1 (cont'd)

There are of course several ways to express Q1 in relational algebra.

Here is another:

$$\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie \text{Sailors}))$$

Which of these expressions should we use?

That is a question of optimization. Indeed, when we describe how to state queries in SQL, we can leave it to the optimizer in the DBMS to select the nest approach.

QUERY Q2

(Q2) Find the names of sailors who have reserved a red boat.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\pi_{sname}((\sigma_{color='red'}\text{Boats}) \bowtie \text{Reserves} \bowtie \text{Sailors})$

QUERY Q3

(Q3) Find the colors of boats reserved by Lubber.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

QUERY Q3

(Q3) Find the colors of boats reserved by Lubber.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\pi_{color}((\sigma_{sname='Lubber'}Sailors) \bowtie Reserves \bowtie Boats)$

QUERY Q4

(Q4) Find the names of Sailors who have reserved at least one boat

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

QUERY Q4

(Q4) Find the names of Sailors who have reserved at least one boat

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

π_{sname}(Sailors $\bowtie$ Reserves)

QUERY Q5

(Q5) Find the names of sailors who have reserved a red or a green boat.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

QUERY Q5

(Q5) Find the names of sailors who have reserved a red or a green boat.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\rho(Tempboats, (\sigma_{color='red'}Boats) \cup (\sigma_{color='green'}Boats))$

$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$

QUERY Q6

(Q6) Find the names of Sailors who have reserved a red and a green boat.

It seems tempting to use the expression used in Q5, replacing simply ∪ by ∩. However, this won't work, for such an expression is requesting the names of sailors who have requested a boat that is both red and green! The correct expression is as follows:

$$\rho(Tempred, \pi_{sid}((\sigma_{color='red'}Boats) \bowtie Reserves))$$

$$\rho(Tempgreen, \pi_{sid}((\sigma_{color='green'}Boats) \bowtie Reserves))$$

$$\pi_{sname}\ ((Tempred \cap Tempgreen) \bowtie Sailors)$$

QUERY 7

(Q7) Find the sids of sailors with age over 20 who have not reserved a red boat.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\pi_{sid}(\sigma_{age>20}Sailors) - \pi_{sid}((\sigma_{color='red'}Boats) \bowtie Reserves \bowtie Sailors)$

QUERY 8

(Q) Find the names of sailors who have reserved all boats.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\rho(Tempsids, (\pi_{sid,bid}Reserves) / (\pi_{bid}Boats))$

$\pi_{sname}(Tempsids \bowtie Sailors$

QUERY Q9

(Q9) Find the names of sailors who have reserved all boats called Interlake.

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$\rho(Tempsids, (\pi_{sid,bid}Reserves)/(\pi_{bid}(\sigma_{bname='Interlake'}Boats)))$

$\pi_{sname}(Tempsids \bowtie Sailors)$

Examples

- *Person* (name, age, gender)
  Frequents (name, pizzeria)
  Eats (name, pizza)
  Serves (pizzeria, pizza, price)

Examples

- a. Find all pizzerias frequented by at least one person under the age of 18.

  b. Find the names of all females who eat either mushroom or pepperoni pizza (or both).

  c. Find the names of all females who eat both mushroom and pepperoni pizza.

a. $\pi_{pizzeria}(\sigma_{age<18}(Person) \bowtie Frequents)$

b. $\pi_{name}\left(\sigma_{gender='female' \wedge (pizza='mushroom' \vee pizza='pepperoni')}(Person \bowtie Eats)\right)$

c. $\pi_{name}(\sigma_{gender='female' \wedge pizza='mushroom'}(Person \bowtie Eats)) \cap$
$\pi_{name}(\sigma_{gender='female' \wedge pizza='pepperoni'}(Person \bowtie Eats))$

- d. Find all pizzerias that serve at least one pizza that Amy eats for less than $10.00.

  e. Find all pizzerias that are frequented by only females or only males.

  f. For each person, find all pizzas the person eats that are not served by any pizzeria the person frequents. Return all such person (name) / pizza pairs.

d. $\pi_{pizzeria}\left(\sigma_{name='\text{Amy}'}(Eats) \bowtie \sigma_{price<10}(Serves)\right)$

e. $$\left(\begin{array}{l}\pi_{pizzeria}(\sigma_{gender='\text{female}'}(Person) \bowtie Frequents) \; - \\ \pi_{pizzeria}(\sigma_{gender='\text{male}'}(Person) \bowtie Frequents)\end{array}\right) \cup \left(\begin{array}{l}\pi_{pizzeria}(\sigma_{gender='\text{male}'}(Person) \bowtie Frequents) \; - \\ \pi_{pizzeria}(\sigma_{gender='\text{female}'}(Person) \bowtie Frequents)\end{array}\right)$$

f. $Eats - \pi_{name,pizza}(Frequents \bowtie Serves)$

- g. Find the names of all people who frequent only pizzerias serving at least one pizza they eat.

  h. Find the names of all people who frequent every pizzeria serving at least one pizza they eat.

  i. Find the pizzeria serving the cheapest pepperoni pizza. In the case of ties, return all of the cheapest-pepperoni pizzerias.

g. $\pi_{name}(Person) - \pi_{name}\left(Frequents - \pi_{name,pizzeria}(Eats \bowtie Serves)\right)$

h. $\pi_{name}(Person) - \pi_{name}\left(\pi_{name,pizzeria}(Eats \bowtie Serves) - Frequents\right)$

i. $$\pi_{pizzeria}(\sigma_{pizza='\text{pepperoni}'} Serves) \; - \; \pi_{pizzeria}\left(\sigma_{price>price2}\left(\begin{array}{l}\pi_{pizzeria,price}(\sigma_{pizza='\text{pepperoni}'} Serves) \\ \quad \times \\ \rho_{pizzeria2,price2}\left(\pi_{pizzeria,price}(\sigma_{pizza='\text{pepperoni}'} Serves)\right)\end{array}\right)\right)$$

Thank You!

SQL

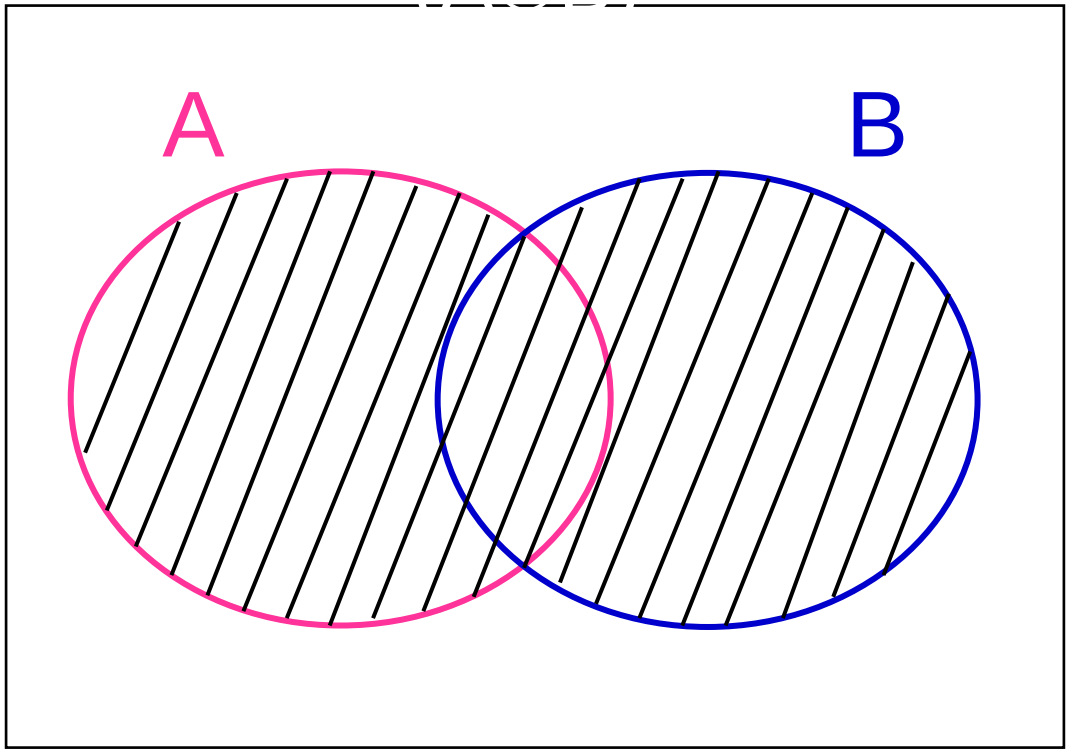
Structured Query Language

Dr. Jenila Livingston L.M.
SCSE

Set Operators

- Union
- Intersection
- Minus

union



A table containing all the rows from **A** and **B**.

UNION

```
SELECT ..... FROM ..... WHERE ..... ;
UNION
SELECT ..... FROM ..... WHERE .....;
```

eg. The two clubs want to hold a joint party. Make a list of all students. (Union)

Result

```
SELECT * FROM bridge
UNION
SELECT * FROM chess
ORDER BY class, name INTO TABLE party
```

- SELECT supplier_id FROM suppliers

UNION

SELECT supplier_id FROM orders ORDER BY supplier_id;

SELECT supplier_id, supplier_name FROM suppliers WHERE supplier_id > 2000

UNION

SELECT company_id, company_name FROM companies WHERE company_id > 1000 ORDER BY 1;

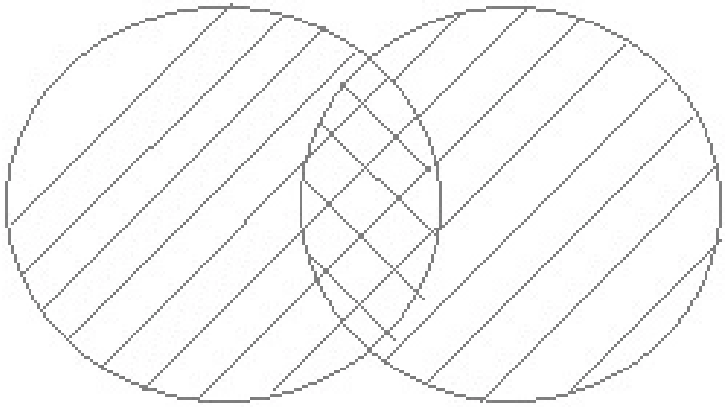
supplier_id	supplier_name
1000	Microsoft
2000	Oracle
3000	Apple
4000	Samsung

company_id	company_name
1000	Microsoft
3000	Apple
7000	Sony
8000	IBM

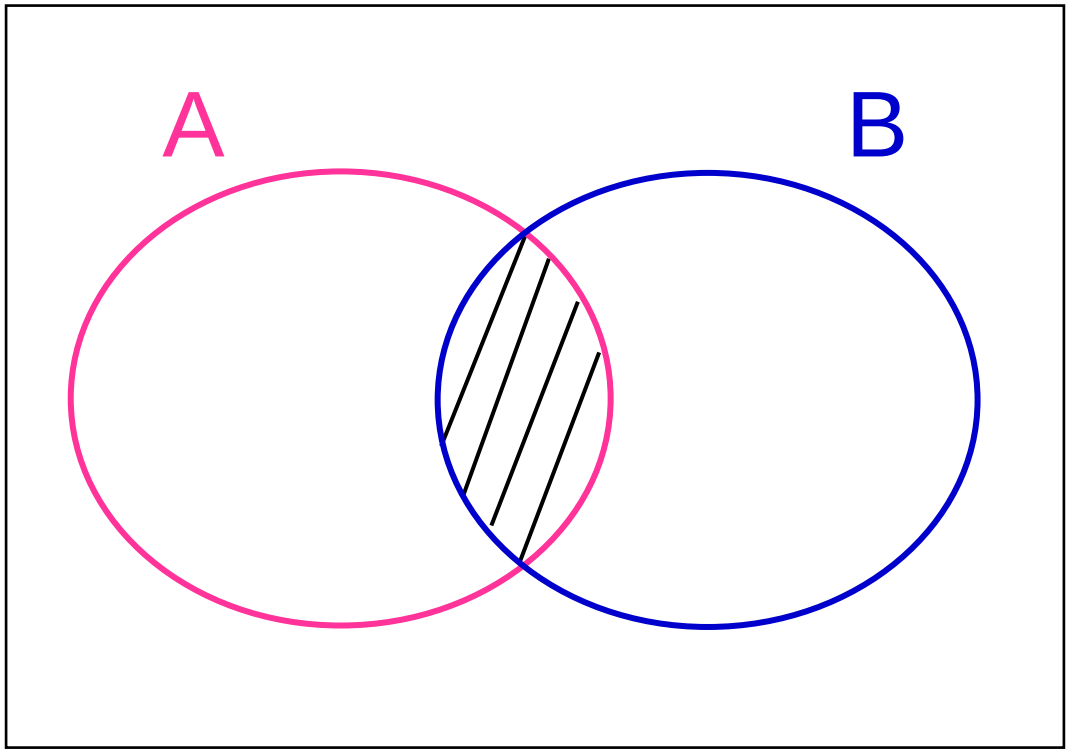
ID_Value	Name_Value
3000	Apple
4000	Samsung
7000	Sony
8000	IBM

UNION ALL

- This operation is similar to Union. But it also shows the duplicate rows.



intersection



A table containing only rows that appear in both **A** and **B**.

INTERSECTION

```
SELECT ..... FROM table1 ;
WHERE col IN  ( SELECT col FROM table2 )
```

eg. list the students who are members of both clubs. (Intersection)

Result

```
SELECT * FROM bridge
WHERE id IN ( SELECT id FROM chess ) ;
```

INTERSECT CLAUSE

- SELECT column1 [, column2] FROM tables [WHERE condition]

  INTERSECT

  SELECT column1 [, column2] FROM tables [WHERE condition]

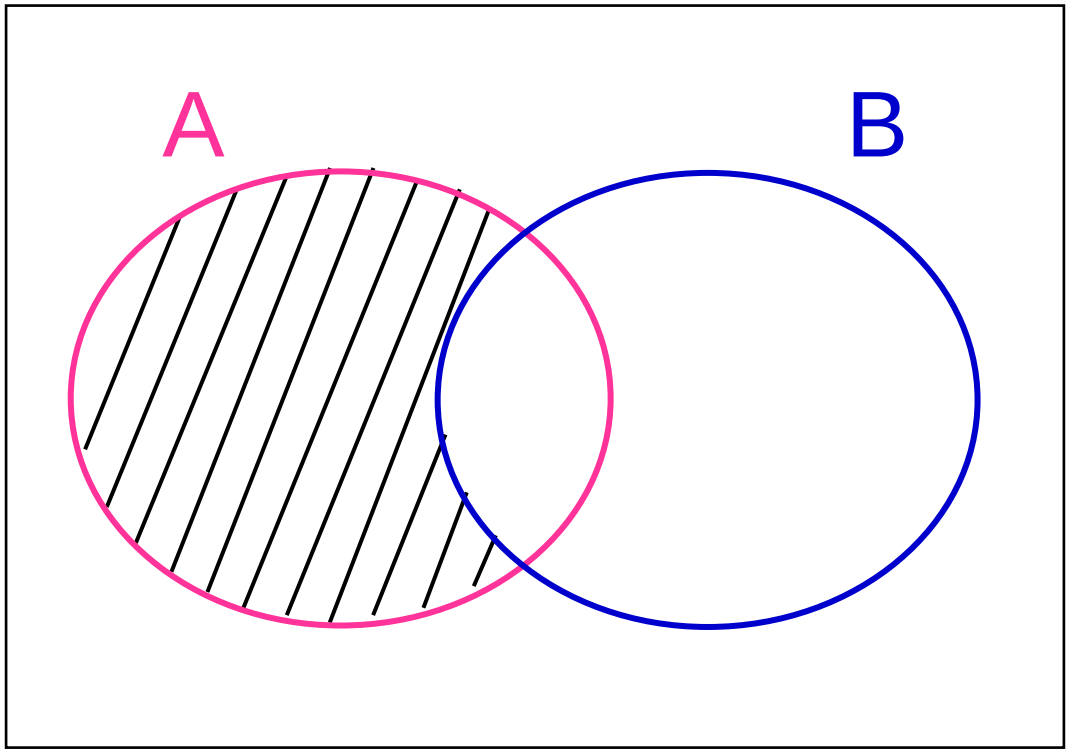
Data types of the column list must be compatible

INTERSECT - EXAMPLE

- SELECT supplier_id FROM suppliers
  INTERSECT
  SELECT supplier_id FROM orders;
- SELECT supplier_id FROM suppliers WHERE supplier_id > 78
  INTERSECT
  SELECT supplier_id FROM orders WHERE quantity <> 0;

MINUS

difference



A table containing rows that appear in **A** but not in **B**.

MINUS

SELECT ..... FROM *table1* ;
WHERE *col* **NOT IN** (SELECT *col* FROM *table2*)

eg. list the students who are members of the Bridge Club but not Chess Club. (Difference)

Result

```
SELECT * FROM bridge
WHERE id NOT IN ( SELECT id FROM chess );
```

MINUS CLAUSE

- SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions]

  MINUS

  SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions];

SELECT supplier_id FROM suppliers

MINUS

SELECT supplier_id FROM orders;

8. SQL Joins

- A SQL JOIN clause combines records from two or more tables in a database.
- It creates a set that can be saved as a table or used as is.
- A JOIN is a means for combining fields from two tables by using values common to each.

JOIN OPERATION

- A join can be specified in the FROM clause which list the **two input relations** and the WHERE clause which **lists the join condition**.

Example:

Emp

ID	State
1000	CA
1001	MA
1002	TN

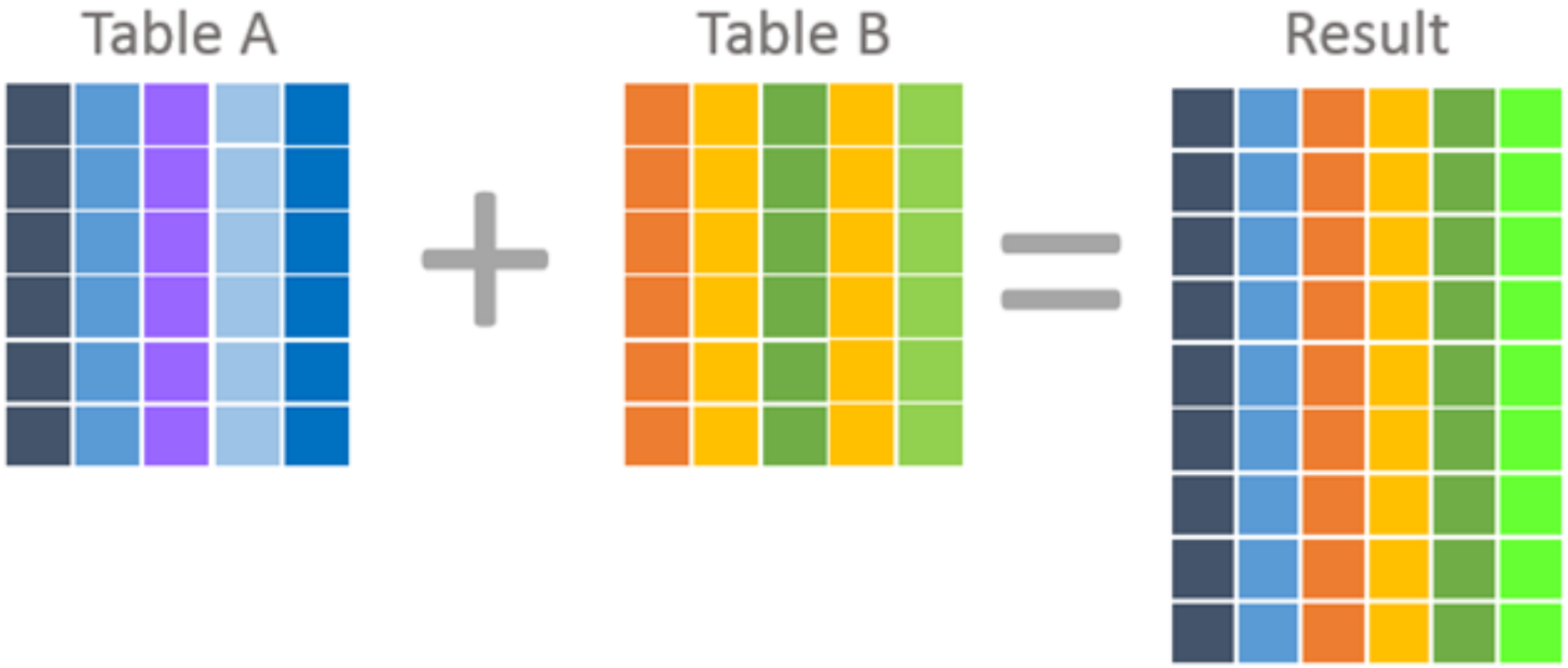
Dept

ID	Division
1001	IT
1002	Sales
1003	Biotech

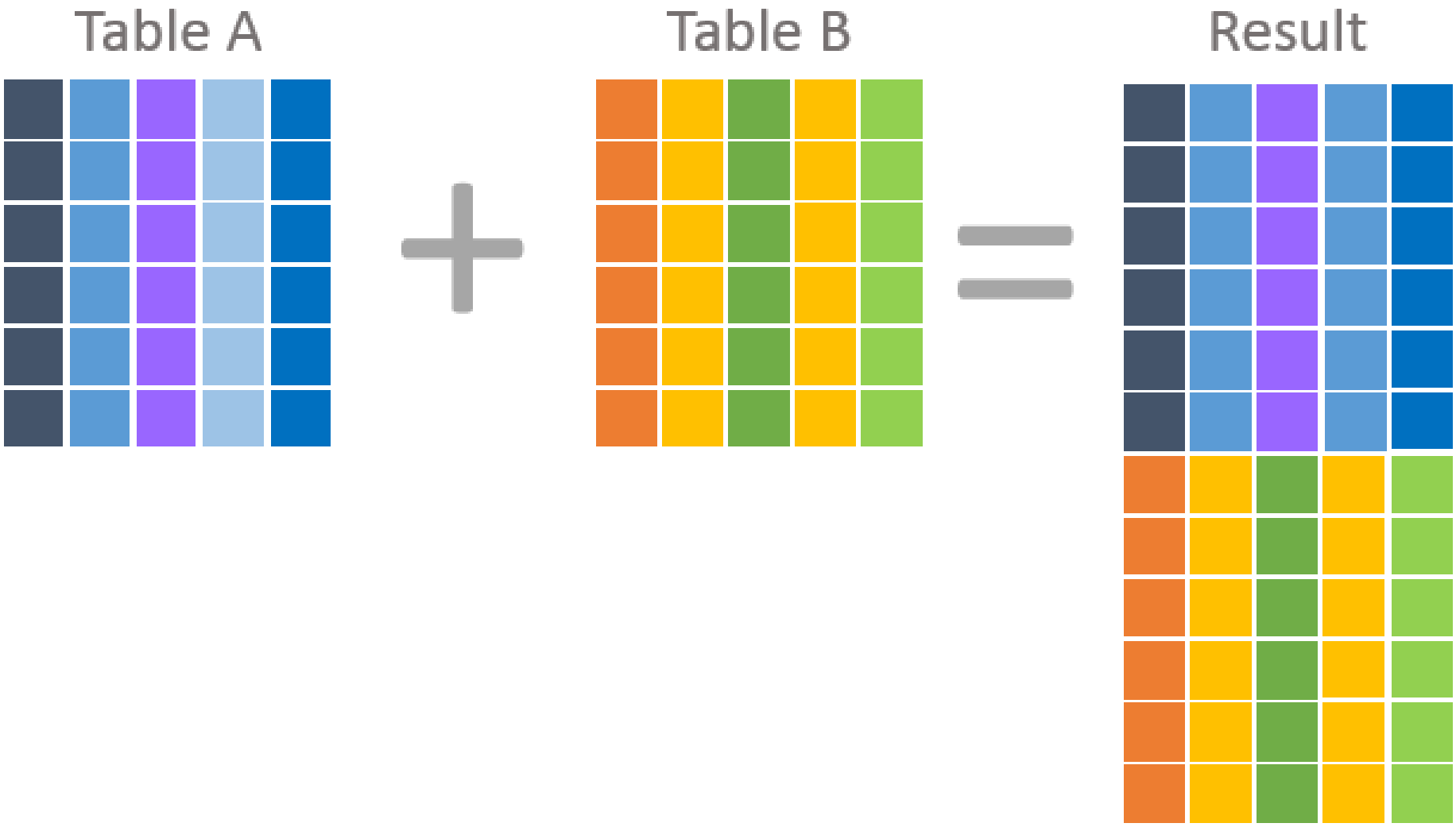
Join Vs Union

- Joins and Unions can be used to combine data from one or more tables. The difference lies in how the data is combined.
- In simple terms, **joins combine data into new columns**. If two tables are joined together, then the data from the first table is shown in one set of column alongside the second table's column in the same row.
- **Unions combine data into new rows.** If two tables are "union-ed" together, then the data from the first table is in one set of rows, and the data from the second table in another set. The rows are in the same result.

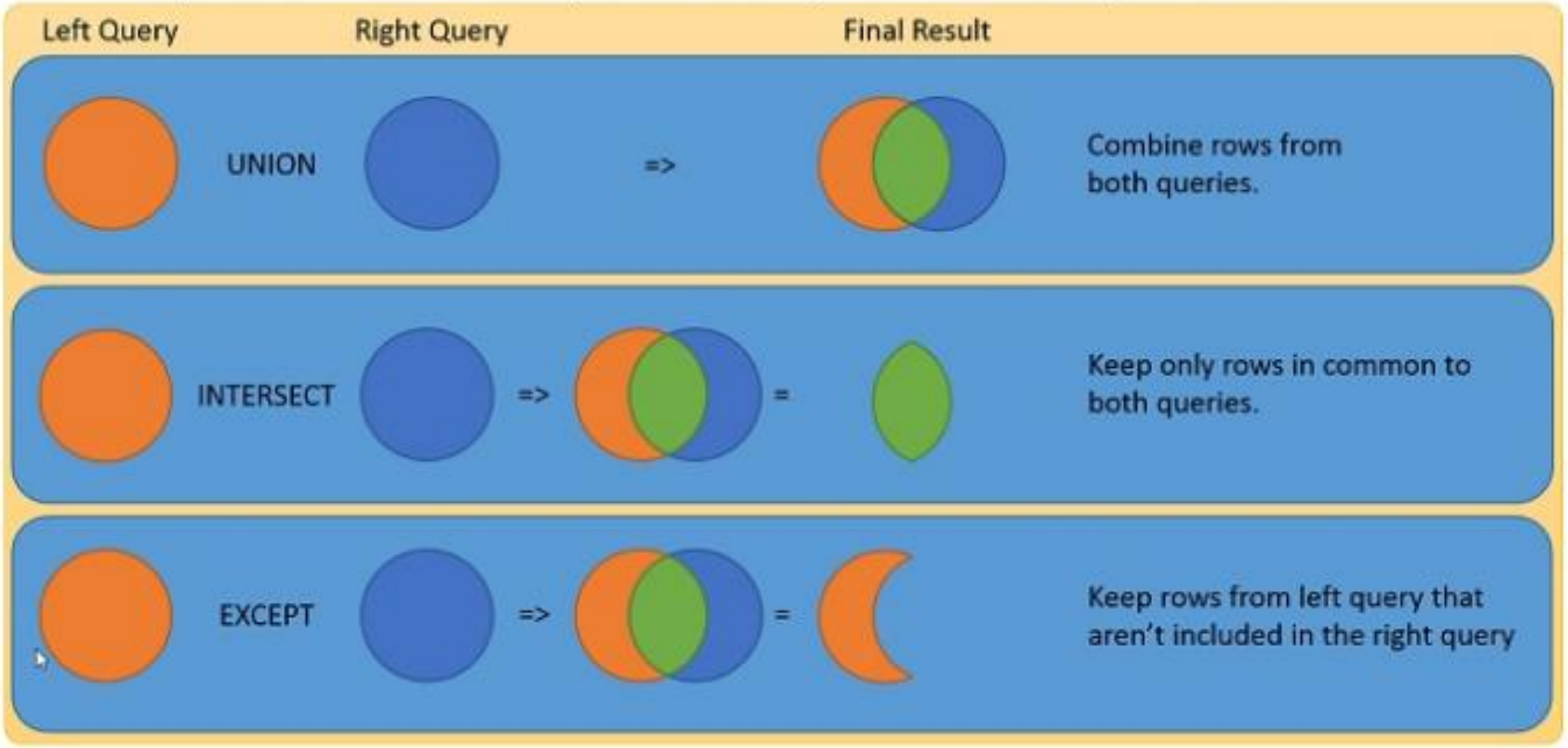
JOIN



UNION



Visual Explanation of UNION, INTERSECT, and EXCEPT operators



Types of Joins

- Equi-join
  - Inner join and Natural Join
  - Outer joins
    - Left outer join
    - Right outer joins
    - Full outer join
- Non Equi-join
- Cross join
- Self-join

EQUI JOIN Vs NON EQUI JOIN

- The SQL EQUI JOIN is a simple sql join uses the equal sign(=) as the comparison operator for the condition. It has two types - SQL Outer join and SQL Inner join

- The **SQL NON EQUI JOIN** is a join uses comparison operator other than the equal sign like >, <, >=, <= with the condition

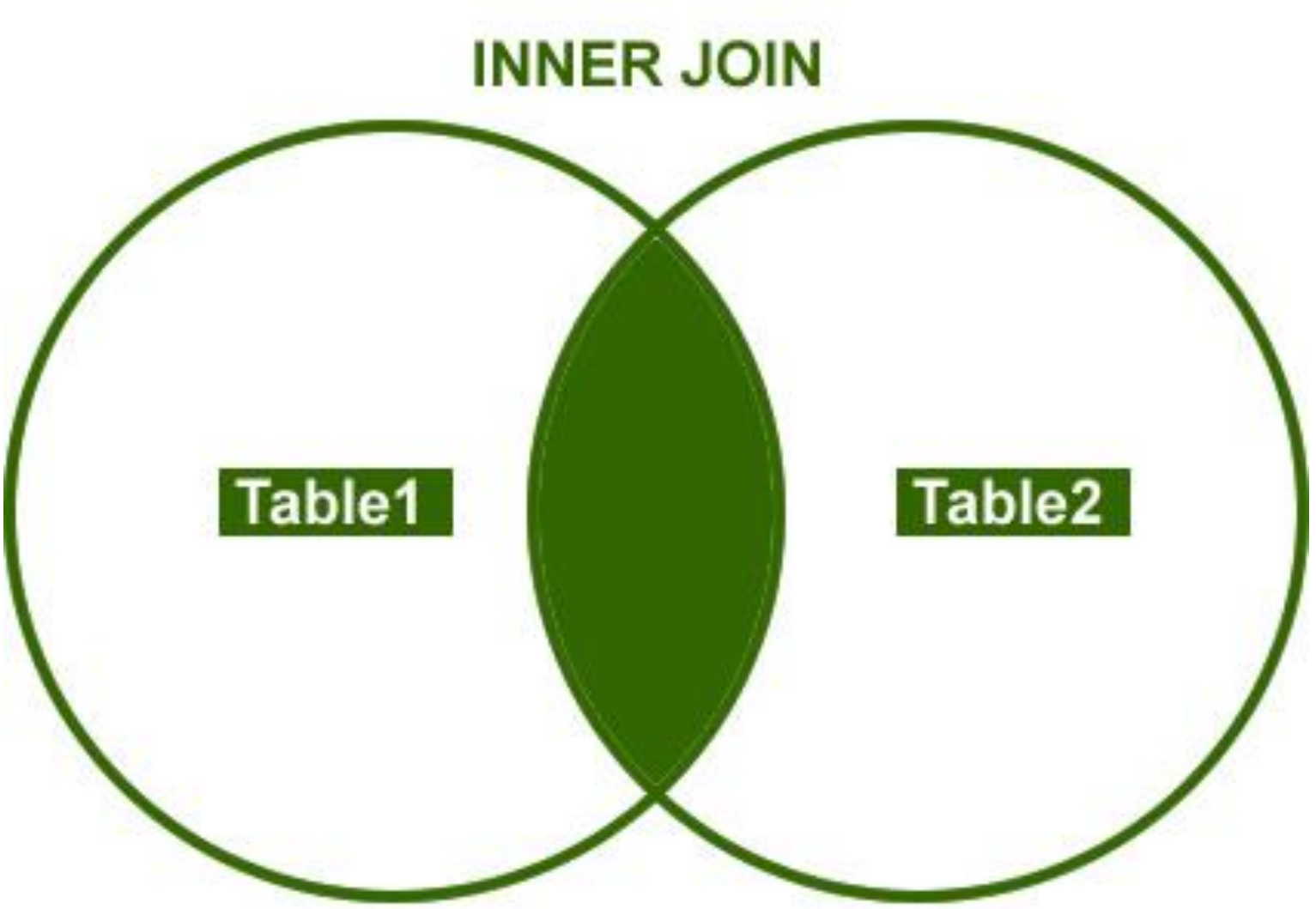
INNER JOIN Vs OUTER JOIN

SQL INNER JOIN

- This type of EQUI JOIN returns all rows from tables where the key record of one table is equal to the key records of another table.

SQL OUTER JOIN

- This type of EQUI JOIN returns all rows from one table and only those rows from the secondary table where the joined condition is satisfying i.e. the columns are equal in both tables.




```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
        ON t1.Col1 = t2.Col1
```

JOIN DEFAULT: INNER JOIN

Inner Join

```
SELECT *
FROM table1, table2
WHERE table1.comcol = table2.comcol
```

```
SELECT a.comcol, a.col1, b.col2, expr1, expr2
FROM table1 a, table2 b
WHERE a.comcol = b.comcol
```

```
SELECT a.comcol, a.col1, b.col2, expr1, expr2
FROM table1 a INNER JOIN table2 b
ON a.comcol = b.comcol
```

INNER JOIN

- SELECT * FROM student S INNER JOIN Marks M ON S.Roll_No = M.Roll_No;

Student Table

Roll_No	Name
1	A
2	B
3	C

Marks Table

Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50

NATURAL JOIN

- Natural Join joins two tables based on same attribute name and datatypes. The resulting table will contain all the attributes of both the table but keep only one copy of each common column.

- SELECT * FROM Student S NATURAL JOIN Marks M;

Natural Join Results

Student Table

Roll_No	Name
1	A
2	B
3	C

Marks Table

Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Marks
2	B	70
3	C	50

select * from student s natural join marks m;

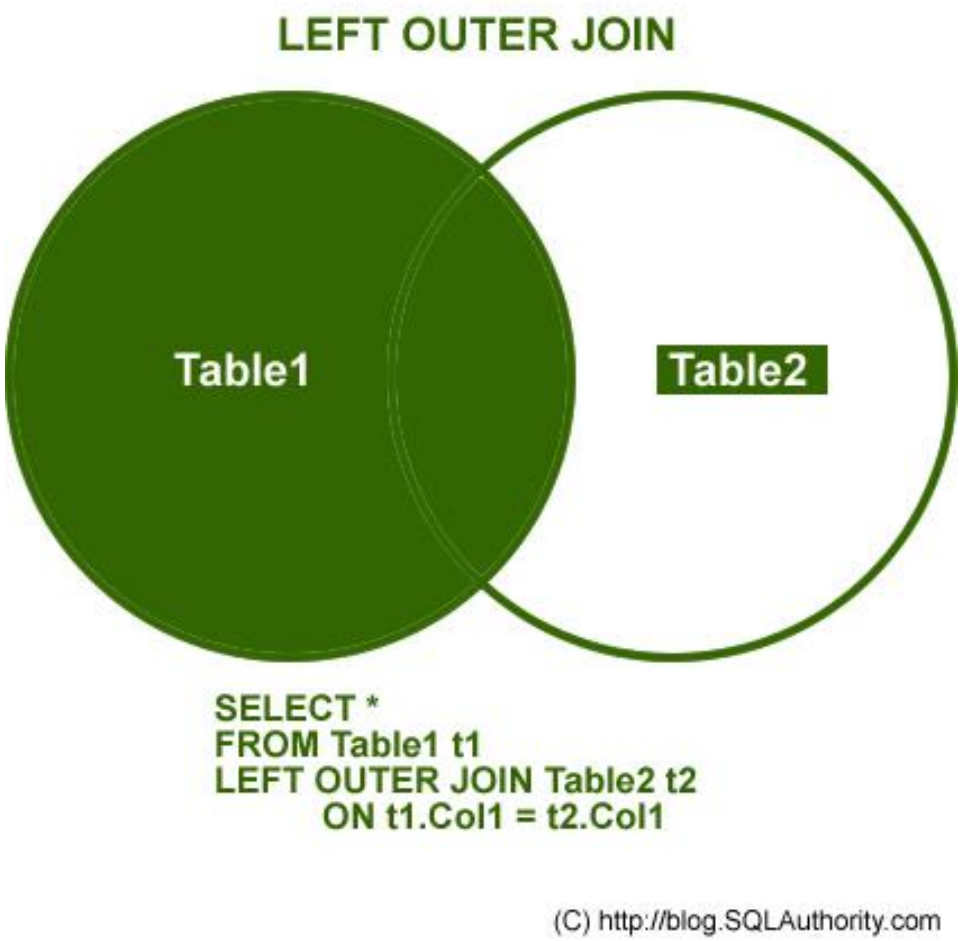
Difference between Natural JOIN and INNER JOIN in SQL

SR.NO.	NATURAL JOIN	INNER JOIN
1.	Natural Join joins two tables based on same attribute name and datatypes.	Inner Join joins two table on the basis of the column which is explicitly specified in the ON clause.
2.	In Natural Join, The resulting table will contain all the attributes of both the tables but keep only one copy of each common column	In Inner Join, The resulting table will contain all the attribute of both the tables including duplicate columns also
3.	In Natural Join, If there is no condition specifies then it returns the rows based on the common column	In Inner Join, only those records will return which exists in both the tables
4.	SYNTAX: SELECT * FROM table1 NATURAL JOIN table2;	SYNTAX: SELECT * FROM table1 INNER JOIN table2 ON table1.Column_Name = table2.Column_Name;

Outer Join

- Left outer join
- Right outer joins
- Full outer join

SQL Joins



- **left outer join** = `left join`

```
SELECT *
FROM emp left join dept
on emp.id = dept.id;
```

The **LEFT JOIN** keyword returns all records from the **left** table (Table1), and the matching records from the right table (Table2).

SELECT * FROM STUDENT S LEFT JOIN MARKS M ON S.ROLL_NO=M.ROLL_NO;

Student Table

Roll_No	Name
1	A
2	B
3	C

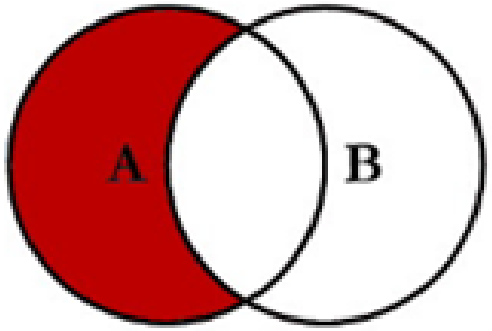
Marks Table

Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50
1	A		

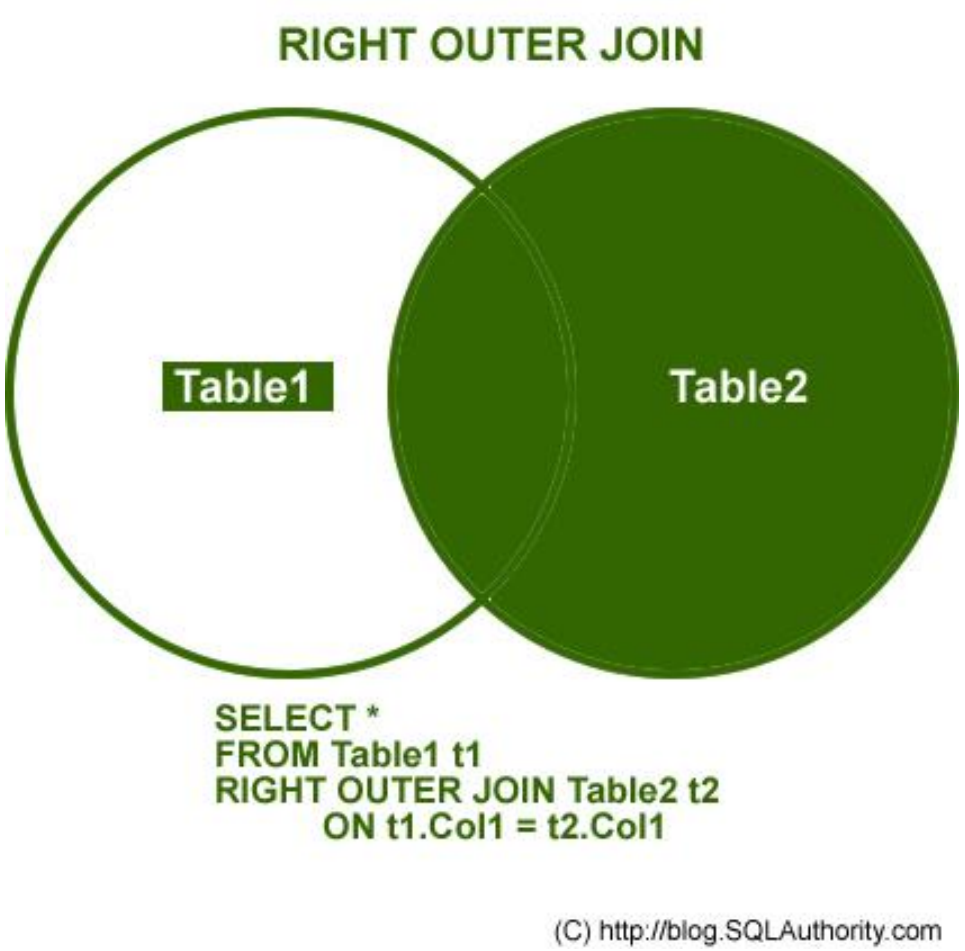
returns all rows of table on left side of join.
The rows for which there is **no matching row on right side**, result contains NULL in the right side.

SELECT * FROM STUDENT S LEFT JOIN MARKS M ON S.ROLL_NO=M.ROLL_NO
WHERE M.ROLL_NO **IS NULL**;



Roll_No	Name	Roll_No	Marks
1	A		

SQL JOINS



- **Right outer join** = `right join`

```
SELECT *
FROM emp right join dept
on emp.id = dept.id;
```

The **RIGHT JOIN** keyword returns all records from the **right** table (Table2), and the matching records from the left table (Table1).

SELECT * FROM STUDENT S RIGHT JOIN MARKS M ON S.ROLL_NO=M.ROLL_NO;

Student Table

Roll_No	Name
1	A
2	B
3	C

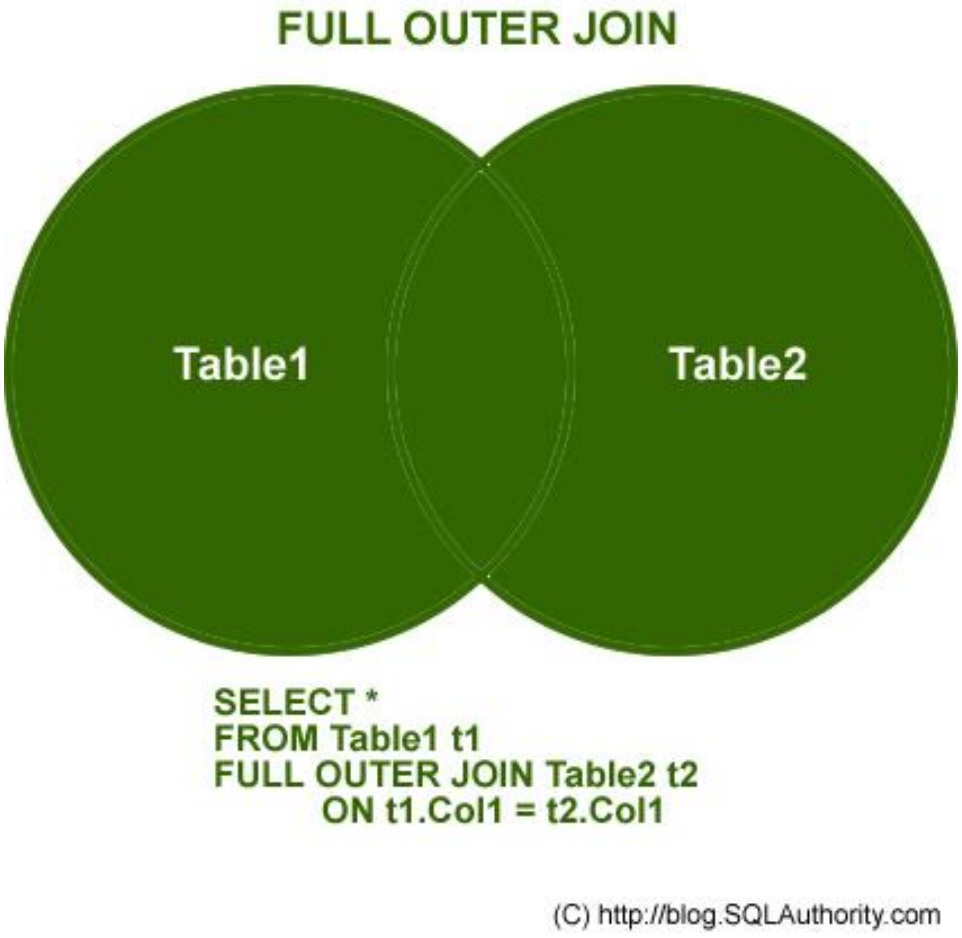
Marks Table

Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50
		4	85

returns all rows of table on right side of join.
The rows for which there is **no matching row on left side**, result contains NULL in the left side.

SQL Joins




```
SELECT *
FROM emp Full join dept
on emp.id = dept.id;
```

SELECT * FROM STUDENT S FULL JOIN MARKS M ON S.ROLL_NO=M.ROLL_NO;

Student Table

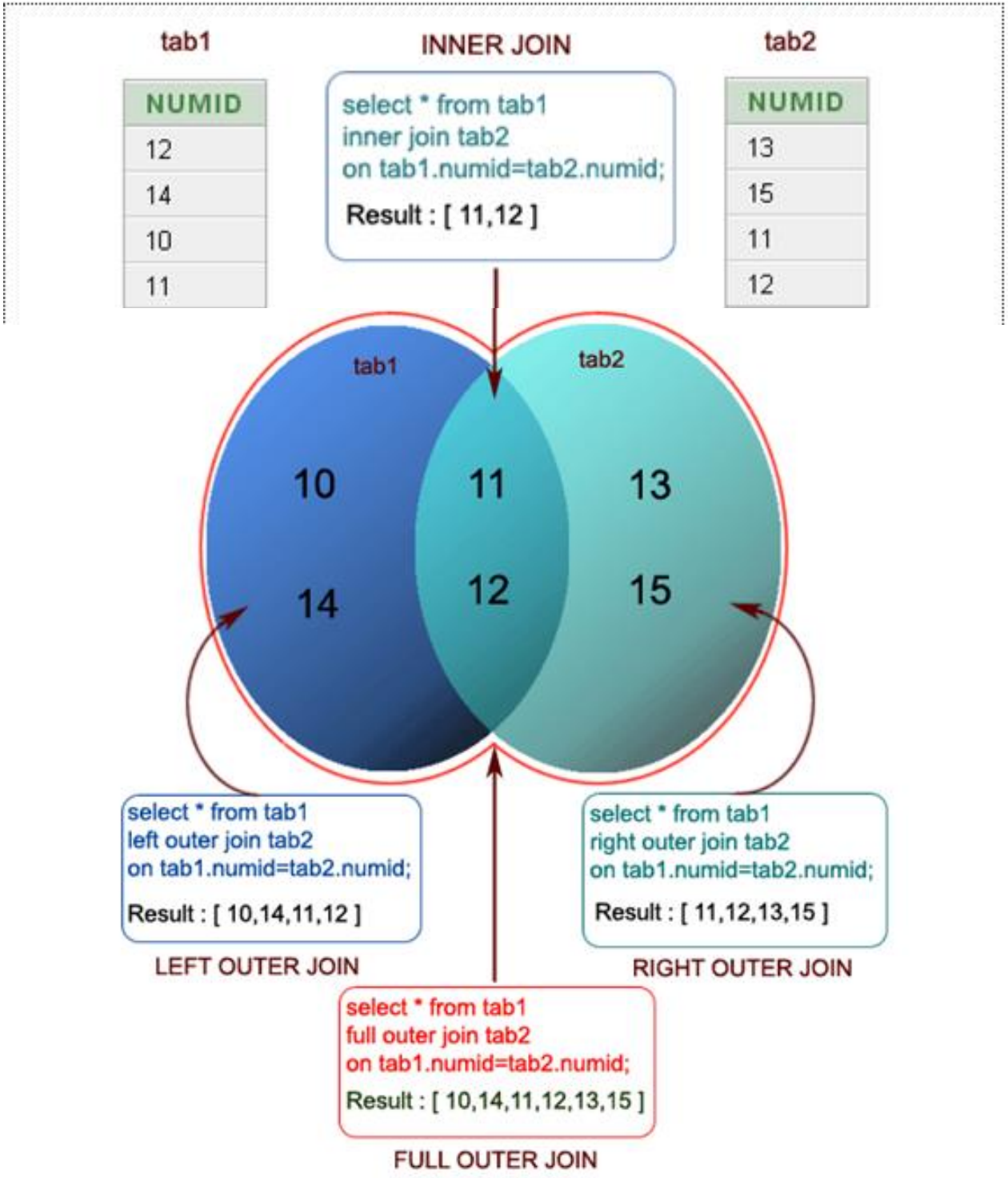
Roll_No	Name
1	A
2	B
3	C

Marks Table

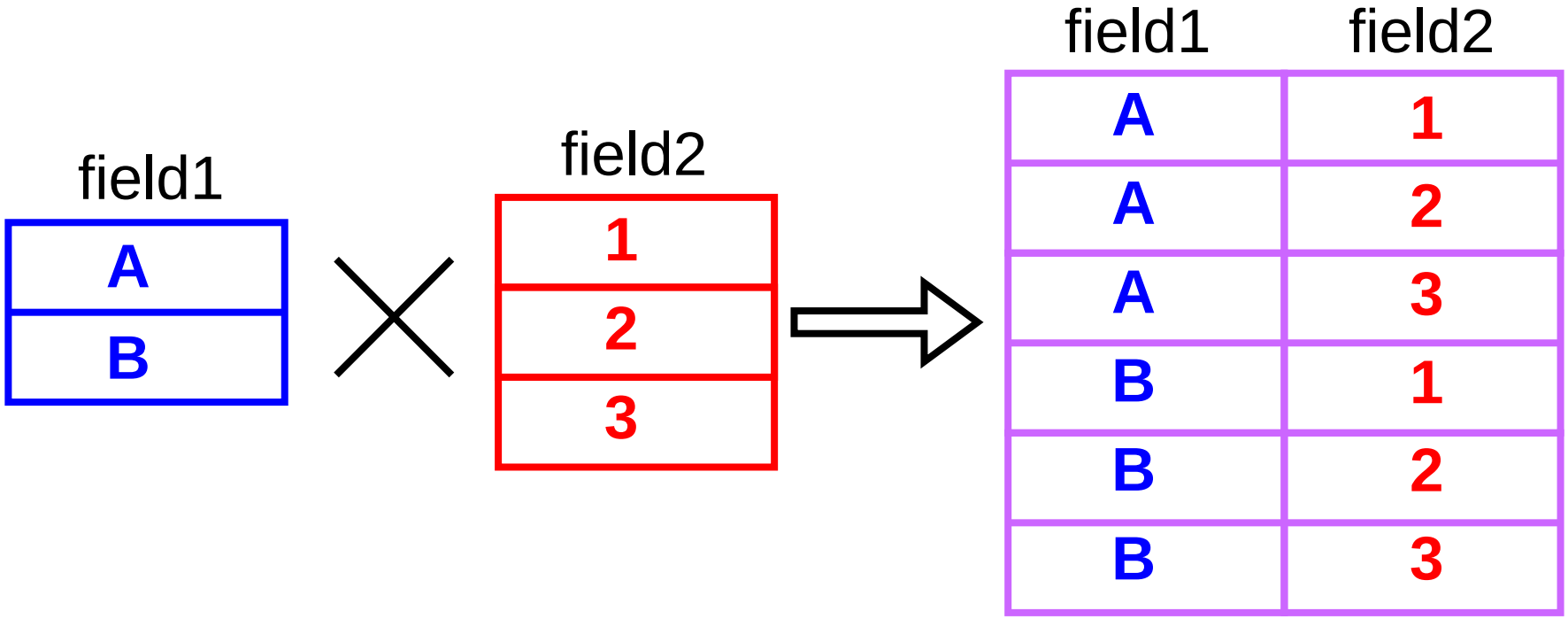
Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50
		4	85
1	A		

Contains results of both Left and Right outer joins


tab1
NUMID
12
14
10
11
INNER JOIN
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
Result : [11,12]
tab2
NUMID
13
15
11
12
tab1
tab2
10
14
11
12
13
15
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
Result : [10,14,11,12]
LEFT OUTER JOIN
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
Result : [11,12,13,15]
RIGHT OUTER JOIN
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
Result : [10,14,11,12,13,15]
FULL OUTER JOIN

Cross Join



SELECT * FROM STUDENT S CROSS JOIN MARKS M

Student Table

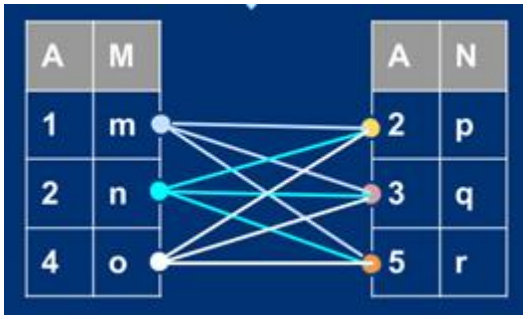
Roll_No	Name
1	A
2	B
3	C

Marks Table

Roll_No	Marks
2	70
3	50
4	85

Roll_No	Name	Roll_No	Marks
1	A	2	70
1	A	3	50
1	A	4	85
2	B	2	70
2	B	3	50
2	B	4	85
3	C	2	70
3	C	3	50
3	C	4	85

CROSS JOIN



Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG

```
SELECT * FROM
   Student CROSS JOIN
   Enrolment
```

ID	Name	ID	Code
123	John	123	DBS
124	Mary	123	DBS
125	Mark	123	DBS
126	Jane	123	DBS
123	John	124	PRG
124	Mary	124	PRG
125	Mark	124	PRG
126	Jane	124	PRG
123	John	124	DBS
124	Mary	124	DBS

Self Join

- **Joining the table itself** called self join.
- Self join is used to retrieve the records having some relation or **similarity with other records in the same table**.
- Here we need to use aliases for the same table to set a self join between single table and retrieve records satisfying the condition in where clause.

Self Join - Example

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

- Employees getting salary less than that of an employee
- SELECT a.ID, b.NAME, a.SALARY FROM EMP a, EMP b

  WHERE **a.SALARY < b.SALARY**;

ID	NAME	SALARY
2	Ramesh	1500.00
2	kaushik	1500.00
1	Chaitali	2000.00
2	Chaitali	1500.00
3	Chaitali	2000.00
6	Chaitali	4500.00
1	Hardik	2000.00
2	Hardik	1500.00
3	Hardik	2000.00
4	Hardik	6500.00
6	Hardik	4500.00
1	Komal	2000.00
2	Komal	1500.00
3	Komal	2000.00
1	Muffy	2000.00
2	Muffy	1500.00
3	Muffy	2000.00
4	Muffy	6500.00
5	Muffy	8500.00
6	Muffy	4500.00

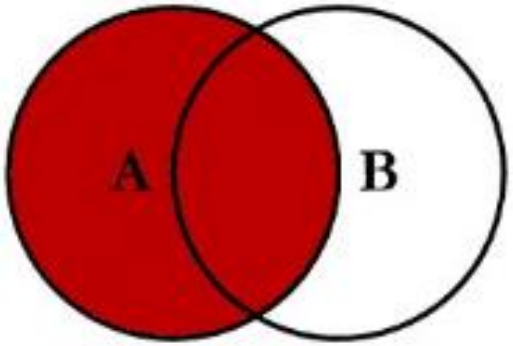
```
SELECT
A.ROLL_NO,B.ROLL_NO,A.MARKS,B.MARKS,
A.MARKS-B.MARKS AS DIFF FROM MARKS A,
MARKS B WHERE A.MARKS>B.MARKS;
```

Marks Table

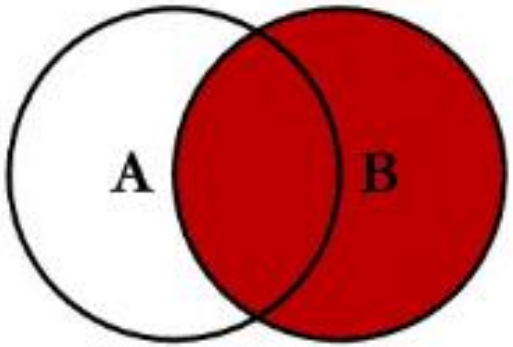
Roll_No	Marks
2	70
3	50
4	85

ROLL_NO	ROLL_NO	MARKS	MARKS	DIFF
2	3	70	50	20
4	2	85	70	15
4	3	85	50	35

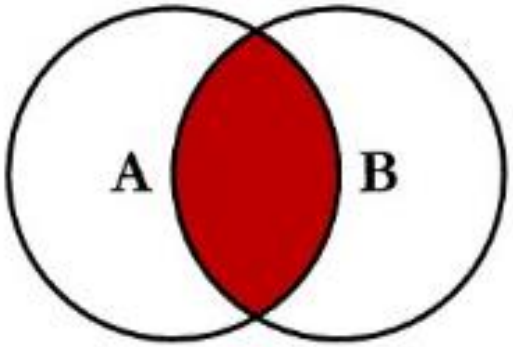
SQL JOINS



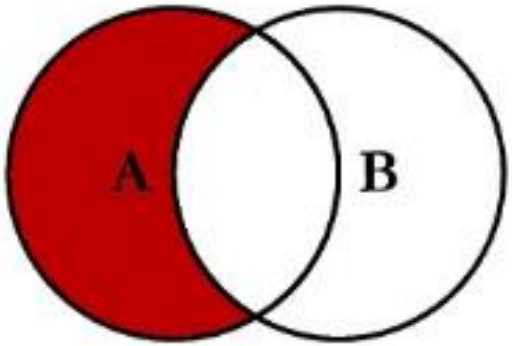

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



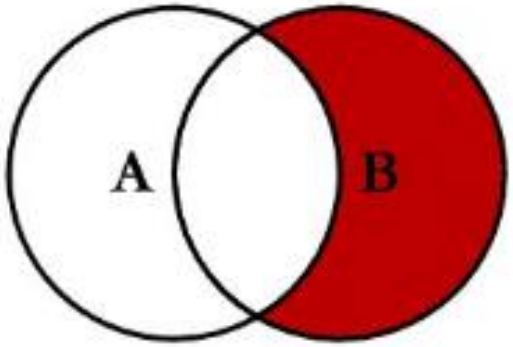

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



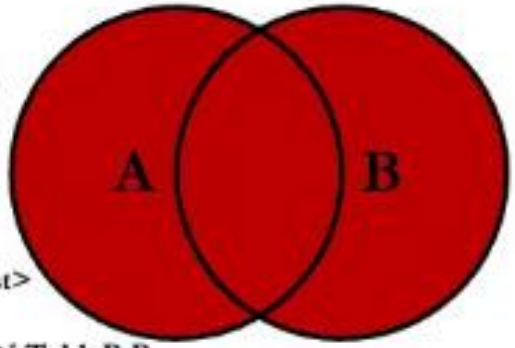

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



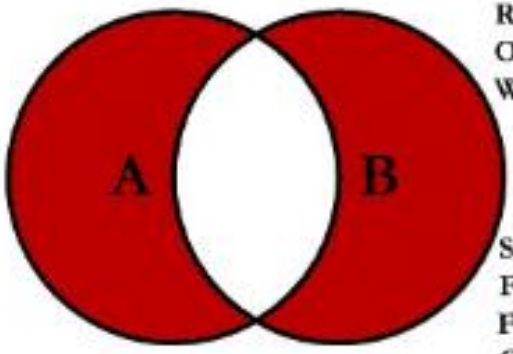

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

THANK YOU

Transactions

Dr. L.M. Jenila Livingston

VIT Chennai

Transactions

- Transaction Concept
- ACID Properties
- Transaction States

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g. transaction to transfer $50 from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

ACID Properties

A **transaction** is a unit of program execution that accesses and possibly updates various data items.To preserve the integrity of data the database system must ensure:

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
  - That is, for every pair of transactions T_i and T_j, it appears to T_i that either T_j, finished execution before T_i started, or T_j started execution after T_i finished.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Example of Fund Transfer

- Transaction to transfer $50 from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- **Atomicity requirement**
  - if the transaction fails after step 3 and before step 6, money will be “lost” leading to an inconsistent database state
    - Failure could be due to software or hardware
  - the system should ensure that updates of a partially executed transaction are not reflected in the database.

Example of Fund Transfer (Cont.)

- Transaction to transfer $50 from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)

Constraint: The sum of A+B must be the same
Let A=150, B=50 A+B=200

A= 150-50=100
B=50+50=100
} =200, consistent

- **Consistency requirement** in above example:
  - the sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
  - Explicitly specified integrity constraints such as primary keys and foreign keys
  - Implicit integrity constraints
    - e.g. sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
  - A transaction must see a consistent database.
  - During transaction execution the database may be temporarily inconsistent.
  - When the transaction completes successfully the database must be consistent

Example of Fund Transfer (Cont.)

- **Isolation requirement** — if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

T1	T2
1. **read**(A)	
2. $A := A - 50$	
3. **write**(A)	
	read(A), read(B), print(A+B)
4. **read**(B)	
5. $B := B + 50$	
6. **write**(B	

- Isolation can be ensured trivially by running transactions **serially**
  - that is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, as we will see later.

Example of Fund Transfer (Cont.)

- **Durability requirement** — once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures

Demonstrating ACID in short

Transaction to transfer $50 from account *A* to account *B*:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B)*

Atomicity: if transaction fails after 3 and before 6, 3 should not affect db

Consistency: total value A+B, unchanged by transaction

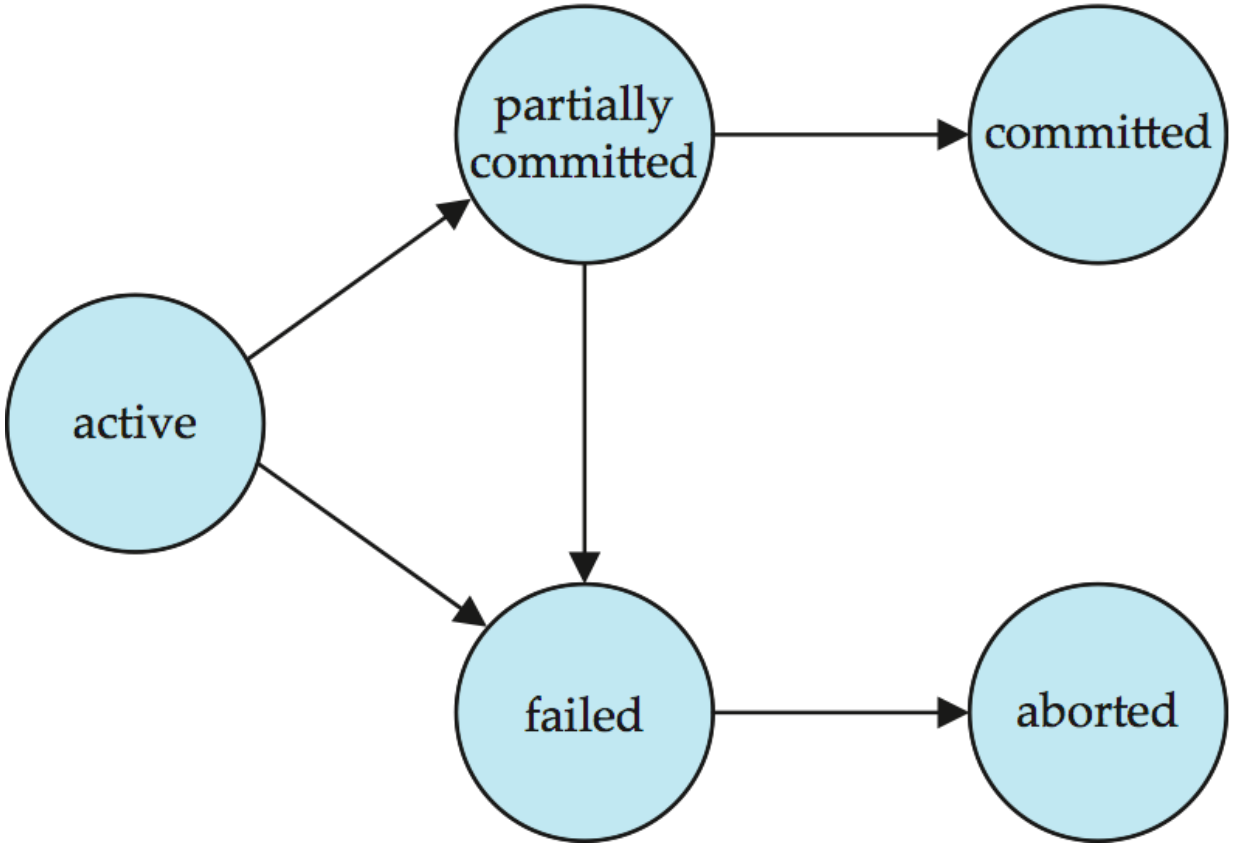
Isolation: other transactions should not be able to see A, B between steps 3-6

Durability: once user notified of transaction commit, updates to A,B should be done even in case of system failure

Transaction State

- **Active** – the initial state; the transaction stays in this state while it is executing
- **Partially committed** – after the final statement has been executed.
- **Failed** -- after the discovery that normal execution can no longer proceed.
- **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
  - restart the transaction
    - can be done only if no internal logical error
  - kill the transaction
- **Committed** – after successful completion.

Transaction State (Cont.)



References

- Abraham Silberschatz, Henry F. Korth and S. Sudarshan, *Database System Concepts*, 6th Edition, McGraw-Hill