

More Relational Algebra and SQL Subqueries

Week 3

Union, Intersection, Set Difference in Relational Algebra

Relational Algebra Operators

There are eight operators

- π project ←
- σ select ←
- \cup union
- \cap intersection
- $-$ difference
- \times cross product ←
- \bowtie join ←
- \div division
- renaming (to provide names for the relation & attributes of answer)

Four operators
that you've already
seen

Relational Algebra Operators

There are eight operators

- π project ←
- σ select ←
- \cup union ←
- \cap intersection ←
- $-$ difference ←
- \times cross product ←
- \bowtie join ←
- \div division
- renaming (to provide names for the relation & attributes of answer)

Three operators
from set theory

Union in set theory vs. relational algebra

- In set theory, the elements of a set can be all different types
 $S = \{ 'a', 7053, (1, 2, 'Smith'), (3, 4, 5, 6, 7, 8, 9) \}$
(atomic values as well as tuples of different lengths)
- In set theory, you can take the union (or intersection or difference) of any two sets.

$$A = \{1, (3, 4, 'a'), 5.3\} \quad B = \{7, 1, (2, 3)\}$$

$$A \cup B = \{1, (3, 4, 'a'), 5.3, 7, (2, 3)\}$$

$$A \cap B = \{1\}$$

$$A - B = \{(3, 4, 'a'), 5.3\}$$

But in relational algebra, relations must have the “same shape” (be *union-compatible*) before you can take \cap , \cup , $-$.

Intersection, Union, Difference of Relations

\cup UNION

\cap INTERSECTION

$-$ SET DIFFERENCE

These operators can only be used with relations that are “union-compatible”.

Union Compatible

- Two relations are *union-compatible* if they have the same degree (i.e., the same number of attributes) and the corresponding attributes are defined on the same domains. (this is imprecise because domains/datatypes may not be precise; the type may simply be integer, for example)
- Suppose we have these relations:

Checking-Account (c-num, c-owner, c-balance)

Savings-Account (s-num, s-owner, s-balance)

These are *union-compatible* relations.

∪ Union in Relational Algebra

Consider this query: **Checking-account ∪ Savings-account**

Checking-account	c-num	c-owner	c-balance
------------------	-------	---------	-----------

101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
-----------------	-------	---------	-----------

103	J. Smith	5000.00
-----	----------	---------

	c-num	c-owner	c-balance
--	-------	---------	-----------

101	J. Smith	1000.00
102	W. Wei	2000.00
104	M. Jones	1000.00
105	H. Martin	10,000.00
103	J. Smith	5000.00

Notice: attributes names are from the 1st relation in query.

\cap Intersection in Relational Algebra (1/4)

Consider this query: **Checking-account** \cap **Savings-account**

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

What's the answer to this query?

⊖ Intersection in Relational Algebra (2/4)

What is the answer to this query:

Checking-account ⊖ **Savings-account**

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

What's the answer to this query?

It's empty. There are no tuples that are in both relations.

⊓ Intersection in Relational Algebra (3/4)

What's the answer to this query?

$(\pi_{c\text{-owner}} \text{Checking-account}) \cap (\pi_{s\text{-owner}} \text{Savings-account})$

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

∩ Intersection in Relational Algebra (4/4)

Intermediate query answers

$(\pi_{c\text{-owner}} \text{Checking-account}) \cap (\pi_{s\text{-owner}} \text{Savings-account})$

	c-owner	∩		s-owner
	J. Smith			J. Smith
	W. Wei			
	M. Jones			
	H. Martin			

Query answer is (using attribute name from Checking-account):

	c-owner
	J. Smith

— Set Difference in Relational Algebra (1/4)

Consider this query: **Checking-account** — **Savings-account**

Find all the tuples (rows) that are in the Checking-account relation that are not in the Savings-account relation.

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

What is the answer?

— Set difference operator in Rel. Alg. (2/4)

Consider this query: **Checking-account** — **Savings-account**

Find all the tuples (rows) that are in the Checking-account relation that are not in the Savings-account relation.

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

What is the answer? **All of the rows in the checking-account table.**

Set difference operator in Rel. Alg. (3/4)

$(\pi_{c\text{-owner}} \text{Checking-account}) - (\pi_{s\text{-owner}} \text{Savings-account})$

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

Compute the intermediate query answers. Then, what is the final query answer?

Set difference operator in Rel. Alg. (4/4)

$(\pi_{c\text{-owner}} \text{Checking-account}) - (\pi_{s\text{-owner}} \text{Savings-account})$

Checking-account	c-num	c-owner	c-balance
	101	J. Smith	1000.00
	102	W. Wei	2000.00
	104	M. Jones	1000.00
	105	H. Martin	10,000.00

Savings-account	s-num	s-owner	s-balance
	103	J. Smith	5000.00

Compute the intermediate query answers. Then, what is the final query answer?

	c-owner
	W. Wei
	M. Jones
	H. Martin

Some definitions of union-compatible require that all corresponding attributes have the same name

Graduate-student (id, name, GPA, phone)

Undergrad-student (id, name, GPA, phone)

These tables are union-compatible; we can issue the following queries:

1. Graduate-student \cap Undergrad-student
2. Graduate-student \cup Undergrad-student
3. Undergrad-student --- Graduate-student

What do these queries compute, described in English?

Union, Intersect, Except in SQL

SQL – for union, intersection, and difference

SELECT...
FROM...
WHERE...
GROUP BY
HAVING
ORDER BY

This shows the clauses that can appear in an SQL statement.

(SELECT...FROM...WHERE...)
UNION
(SELECT...FROM...WHERE...)

**UNION, INTERSECT,
EXCEPT**
work on two SQL query expressions (subqueries).

This is the first of many ways that you can use subqueries.

SQL UNION

```
(SELECT      C.Name  
FROM        Customer C  
WHERE       C.Name LIKE "B%")
```

UNION

```
(SELECT      S.Name  
FROM        Salesperson S  
WHERE       S.Name LIKE "B%");
```

Two
complete
queries -
with
the
UNION
operator
in between.

Note: the parentheses around the subqueries are optional.

SQL INTERSECTION

```
(SELECT      C.Name  
FROM        Customer C)
```

INTERSECT

```
(SELECT      S.Name  
FROM        Salesperson S);
```

Two
complete
queries -
with
the
INTERSECT
operator
in between.

SQL EXCEPT (or MINUS) (set difference)

```
(SELECT      S.Number  
FROM        Salesperson S)
```

EXCEPT

```
(SELECT      C.Salesperson  
FROM        Customer C);
```

Two complete subqueries with the EXCEPT operator in between. (MINUS was used in older versions of SQL.)

Sets vs. Bags (also called Multi-sets)

Consider sets $S1=\{a,b\}$, $S2=\{a,b,c,d\}$, $S3=\{b,d\}$

- What is $S1 \cup S2$?
- What is $S3 - S1$?
- What is $S2 \cap S3$?

Now consider bags $B1=\{a,a,b,b,b\}$, $B2=\{a,b,b,c,c,d,d\}$,
 $B3=\{b,b,b,b,d,d,d\}$

- What is $B1 \cup B2$?
- What is $B3 - B1$?
- What is $B2 \cap B3$?

Is there any correspondence between set and bag results?

Sets vs. Bags (also called Multi-sets)

Consider sets $S1=\{a,b\}$, $S2=\{a,b,c,d\}$, $S3=\{b,d\}$

- What is $S1 \cup S2$? $\{a, b, c, d\} = S2$
- What is $S3 - S1$? $\{d\}$
- What is $S2 \cap S3$? $\{b, d\} = S3$

Now consider bags $B1=\{a,a,b,b,b\}$, $B2=\{a,b,b,c,c,d,d\}$,
 $B3=\{b,b,b,b,d,d,d\}$

- What is $B1 \cup B2$? $\{a, a, a, b, b, b, b, b, c, c, d, d\}$
- What is $B3 - B1$? $\{b, d, d, d\}$
- What is $B2 \cap B3$? $\{b, b, d, d\}$

Is there any correspondence between set and bag results?

In SQL, the ALL keyword is for bag semantics

Set semantics (default)	Bag semantics (with ALL)
UNION	UNION ALL
INTERSECT	INTERSECT ALL
EXCEPT	EXCEPT ALL

Union, Intersect, Except (without ALL)

- If you don't specify ALL, then the answer is computed on sets.
- You can think of (set semantics, without ALL) like this:
 - Eliminate duplicates from first table
 - Eliminate duplicates from second table
 - THEN ... compute union, intersect, or difference
 - Eliminate duplicates from query answer table

Using the ALL keyword

- UNION ALL:
 - A has 3 copies of a row, B has 5
 - Answer has $N + M$ copies ($A \cup B$ has 8 copies of the row)
- INTERSECT ALL:
 - A has 3 copies of a row, B has 5
 - Answer has $\text{MIN}(N, M)$ copies ($A \cap B$ has 3 copies of the row)
- EXCEPT ALL:
 - A has 3 copies of a row, B has 5
 - $\text{MAX}(N - M, 0)$ ($A - B$ has 0 copies of the row; $B - A$ has 2)

Exercise using bags of tuples

fund1

Name	Amount
Ng	5
Ng	5
Apt	3
Apt	5
Apt	6
Fry	7

fund2

Name	Amount
Ng	4
Ng	5
Apt	3
Apt	3
Apt	3
Fry	2

SELECT * from fund1
UNION ALL
SELECT * from fund2

SELECT * from fund1
INTERSECT ALL
SELECT * from fund2

SELECT * from fund1
EXCEPT ALL
SELECT * from fund2

Set Operations Implementation

	SQL Server 2000	SQL Server 2005	Oracle 10g	PostgreSQL	MySQL 5.1
UNION	X	X	X	X	X
UNION ALL	X	X	X	X	X
INTERSECT		X	X	X	
INTERSECT ALL				X	
EXCEPT		X	X	X	
EXCEPT ALL				X	

Review our Sample Database

We use the following database for sample SQL queries:

Credit Rating **Credit Amount** **Credit Balance**



Customer (Number, Name, Address, CRating, CAmount, CBalance,
Salesperson)

**Foreign key: Customer.Salesperson
references Salesperson.Number**



Salesperson (Number, Name, Address, Office)

Subqueries in SQL

- We can write SQL queries – surround them by parentheses and use them in various places in an SQL statement.
- The query inside the parentheses is called a nested subquery.
- The query answer (from the nested subquery) is computed in the ordinary way. The query answer is then substituted into the query – in place.

Consider the query answers (from subqueries)

- How many rows? It could be:
 - Many
 - One
 - Zero
- How many columns? Whatever the query writer chooses. It could be:
 - One
 - Many
- If query answer has one row and one column, it's a **scalar query**. The query answer is an **atomic value**.
- Otherwise, the query answer is a **table**.

Where can we use scalar subqueries in SQL?

- A scalar subquery can appear anywhere a constant can appear:
 - In the WHERE clause
compare `a.agent_salary = 5300`
versus `a.agent_salary = (select max(salary) from agent)`
 - In the HAVING clause
compare `HAVING count(*) > 5`
versus `HAVING COUNT(*) > (select ... from ...)`
 - In the SELECT clause (to introduce a constant)
compare `SELECT sname, 'good sailor' as rate FROM ...`
versus `SELECT sname, (select ... from ...) as rate FROM ...`

Where can we use arbitrary subqueries in SQL?

- In a FROM clause FROM (select ... from ...) as t
compare: SELECT sid
 FROM sailors

versus: SELECT sid
 FROM (select sid from reserves) as x
- In the WHERE clause – with new predicates
(as introduced on the next few slides)

Scalar subquery in the where clause (1/3)

```
SELECT C1.Number, C1.Name  
FROM      Customer C1  
WHERE C1.CRating =
```

SELECT..FROM..WHERE query that returns
one row with one column (a constant).

```
(SELECT MAX (C2.CRating)  
FROM      Customer C2);;
```

The circled expression is called the Inner Query. The rest is the Outer Query.

How would you evaluate this query?

What does this query compute?

Note: the subquery must return JUST ONE value; otherwise the equality condition can't be evaluated.

Scalar subquery in the where clause (2/3)

```
SELECT C1.Number, C1.Name  
FROM      Customer C1  
WHERE C1.CRating =
```


This is a complete
SELECT..FROM..WHERE
query.

```
(SELECT MAX (C2.CRating)  
FROM      Customer C2)
```

The comparator can be any of the six standard comparators: =, >, <, >=, <=, <> (not equal)

Scalar subquery in the where clause (3/3)

```
SELECT      C1.Number, C1.Name
FROM        Customer C1
WHERE       C1.CRating =
            (SELECT MAX (C2.CRating)
             FROM    Customer C2)
```



Notice that the inner query doesn't mention any attributes from the outer query. That is, C1 is not mentioned in the inner query.

In this case, the query processor only needs to evaluate the inner query once – because nothing changes when each tuple from the outer query is evaluated.

ALL or SOME before a subquery (works with arbitrary (non-scalar) subquery)

Syntax:

<attribute-name> <comparator> **SOME | ALL** <subquery>

can appear in the WHERE clause

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE  S.Name = SOME (SELECT C.Name
                       FROM   Customer)
```

How many rows will the subquery return?

How will this query be evaluated?

Does the subquery mention any attributes from outer query?

ALL used in a predicate in the WHERE clause

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number = ALL (SELECT C.Salesperson
                        FROM    Customer C
                        WHERE   C.CRating = 3)
```


What is the meaning of this query?

ALL – with non-correlated subquery

```
SELECT S.Number, S.Name
```

```
FROM Salesperson S
```

```
WHERE S.Number = ALL (SELECT C.Salesperson  
                       FROM Customer C  
                       WHERE C.CRating = 3)
```



Notice that the inner query doesn't mention any attributes from the outer query. That is, S is not mentioned in the inner query.

In this case, you only need to evaluate the inner query once – because nothing changes when each tuple from the outer query is evaluated.

Meaning of SOME and ALL

```
SELECT S.Number, S.Name
FROM   Salesperson S
WHERE  S.Name = SOME (SELECT C.Name
                        FROM   Customer);
```

- For **SOME**, the expression must be true for **at least one row** in the subquery answer
 - **ANY** is an older form of **SOME**
- For **ALL**, the expression must be true for **all rows** in the subquery answer.

IN or NOT IN before a subquery

```
SELECT C1.Number, C1.Name  
FROM      Customer C1  
WHERE C1.Name IN
```



```
(SELECT Name  
FROM Salesman)
```

Any SQL query:



IN and NOT IN can appear in these forms:

<attribute-name> IN (subquery)

(<attrib-name₁>, ..., <attrib-name_n>) IN (subquery)

or

<attribute-name> NOT IN (subquery)

(<attrib-name₁>, ..., <attrib-name_n>) NOT IN (subquery)

This is a Correlated Subquery

```
SELECT  S.Number, S.Name
FROM    Salesperson S
WHERE   S.Number IN (SELECT  C.Salesperson
                    FROM    Customer C
                    WHERE   C.Name = S.Name)
```

Because the subquery mentions an attribute from a table in the outer query

The subquery must be re-evaluated every time the WHERE clause from the outer query is evaluated.

You should look at the use of correlation names to figure out whether it is a correlated subquery.

Subquery with “IN” – can be equivalent to a join

```
SELECT S.Number, S.Name
FROM Salesperson S
WHERE S.Number IN      (SELECT C.Salesperson
                        FROM Customer C
                        WHERE C.Name = S.Name)
```

```
SELECT DISTINCT S.Number, S.Name
FROM Salesperson S, Customer C
WHERE S.Number = C.Salesperson AND C.Name = S.Name
```

Are these two queries equivalent?

Do we need to use the DISTINCT clause in the second query in order for these two queries to be equivalent?

“IN” and “= SOME”

You can substitute = **SOME** for **IN**, and vice versa, to make an equivalent query, e.g.,

```
SELECT C.Number, C.Name
FROM   Customer C
WHERE  C.address IN
        (SELECT S.address
         FROM   Salesman S)
```

```
SELECT C.Number, C.Name
FROM   Customer C
WHERE  C.address = SOME
        (SELECT S.address
         FROM   Salesman S)
```

Is this a correlated subquery? (query repeated from slide 6)

```
SELECT C1.Number, C1.Name  
FROM Customer C1  
WHERE C1.CRating IN
```

```
(SELECT MAX (C2.CRating)  
FROM Customer C2)
```

What is the advantage of a subquery that is NOT correlated?


EXISTS before a subquery in a WHERE clause

```
SELECT  C.Name
FROM    Customer C
WHERE   EXISTS (SELECT *
                FROM    Salesperson S
                WHERE   S.Number = C.Salesperson
                        AND S.Name = C.Name)
```

If the answer to the subquery is not empty -
then the EXISTS predicate returns TRUE

Subqueries

```
SELECT C.Name
FROM   Customer C
WHERE  EXISTS (SELECT *
              FROM   Salesperson S
              WHERE  S.Number = C.Salesperson
                  AND S.Name = C.Name)
```



Four predicates that can be applied to a subquery in SQL:

EXISTS (subquery) - is the subquery answer not empty?

NOT EXISTS (subquery) – is the subquery answer empty?

UNIQUE (subquery) – does the subquery answer have any duplicates?

NOT UNIQUE (subquery) – does the subquery answer have duplicate rows?

A Subquery is an SQL query in parentheses

- May return exactly one row and one column; then it is **scalar subquery**. (Treated as *null* value when query answer is empty.)
- May return exactly one row (any number of columns); then it is a **row-valued subquery**.
- May return more than one row (any number of columns); then it is a **table-valued subquery**.
- May or may not be correlated (May or may need to be executed more than once.)
- May appear in the SELECT, FROM, WHERE, or HAVING clause
- Must be scalar in SELECT clause.
- Must be non-correlated in the FROM clause.
- Must be scalar in SELECT for =, >, <, <=, >=, <>, IN

Relational Algebra: Divide Operator

Suppose we have this extra table, in the Bank database:

Account-types	Type
	checking savings

Suppose we would like to know which customers have at least one account of each type of account. That is, we want to know who has accounts of ALL the types.

We can use the Divide operator in Rel. Alg.

$(\pi_{\text{Owner, Type}} \text{Account}) \div \text{Account-types}$

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Account-types	Type
	checking
	savings

	Owner
	J. Smith

Find account owners
who have ALL types of
accounts.

Divide Operator

For $R \div S$ where $R(r_1, r_2, r_3, r_4)$ and $S(s_1, s_2)$

Since S has two attributes, there must be two attributes in R (say r_3 and r_4) that are defined on the same domains, respectively, as s_1 and s_2 . We could say that (r_3, r_4) is union-compatible with (s_1, s_2) .

The query answer has the remaining attributes (r_1, r_2) . And the answer has a tuple, (r_1, r_2) , in the answer if the (r_1, r_2) value appears with *every* S tuple in R .

How does divide work?

$$(\pi_{\text{Owner, Type}} \text{Account}) \div \text{Account-types}$$

$\pi_{\text{Owner, Type}}$ Account

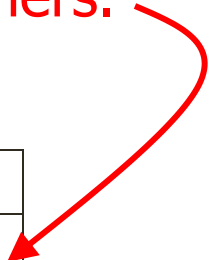
Owner	Type
J. Smith	checking
W. Wei	checking
J. Smith	savings
M. Jones	checking
H. Martin	checking

Can we find an owner where there are enough tuples in this table for that owner so that we can match EVERY tuple in Account-types?

List all such owners.

Account-types	Type
	checking
	savings

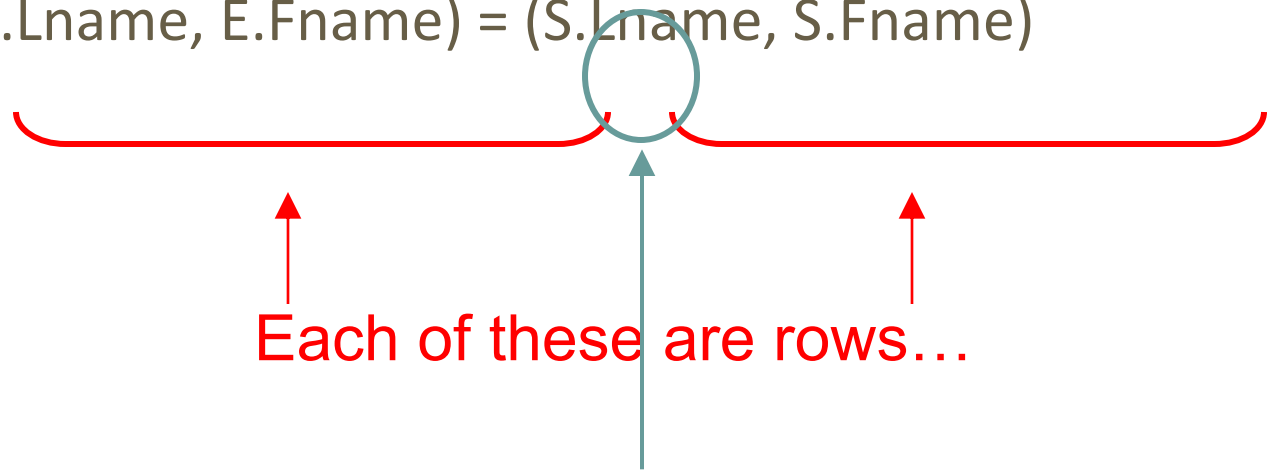
Owner



row value constructors in SQL:1999

- A row value constructor allows you to create a row “on the fly” for example:

WHERE (E.Lname, E.Fname) = (S.Lname, S.Fname)



and the equality comparison is doing a pair-wise comparison on the two rows.

table value constructors in SQL:1999

- You can also create a table “on the fly”

VALUES row-value-expr, ..., row-value-expr

Example:

```
INSERT INTO movie_stars  
VALUES (“Rocky Horror Picture Show”, 1977, “Curry, Tim”),  
      (“Rocky V”, 1984, “Stallone, Sylvester”);
```

- The part shown in RED font is a table value constructor

What about row/table constructors in relational algebra?

If you need to have either a row or a relation, you can just define it and then use it in relational algebra.

You could say something like:

Let $R = \{(\text{"John"}, 5, \text{"male"}), (\text{"Sue"}, 6, \text{"female"})\}$

or let R be the table

Name	Age	Gender
John	5	male
Sue	6	female

and then use R in expressions ... like $R \times \text{Student}$...or whatever

Why do we use Relational Algebra?

Because:

- It is mathematically defined (where relations are sets)
- We can prove that two relational algebra expressions are equivalent.

For example:

$$\begin{aligned} \sigma_{\text{cond1}} (\sigma_{\text{cond2}} R) &\equiv \sigma_{\text{cond2}} (\sigma_{\text{cond1}} R) \equiv \sigma_{\text{cond1 AND cond2}} R \\ &\equiv (\sigma_{\text{cond1}} R) \cap (\sigma_{\text{cond2}} R) \end{aligned}$$

$$R1 \bowtie_{\text{cond}} R2 \equiv \sigma_{\text{cond}} (R1 \bowtie R2)$$

“AND”, “OR”, and “NOT”

$$\sigma_{\text{cond1 OR cond2}} R \equiv (\sigma_{\text{cond1}} R) \cup (\sigma_{\text{cond2}} R)$$

$$\sigma_{\text{cond1 AND cond2}} R \equiv (\sigma_{\text{cond1}} R) \cap (\sigma_{\text{cond2}} R)$$

$$\sigma_{\text{cond1 AND NOT cond2}} R \equiv (\sigma_{\text{cond1}} R) - (\sigma_{\text{cond2}} R)$$

The WHERE clause (and the predicate for the σ operator) may contain AND, OR, as well as NOT.

Uses of Relational Algebra Equivalences

- To help query writers – they can write queries in several different ways
- To help query optimizers – they can choose among different ways to execute the query

and in both cases **we know for sure** that the two queries (the original and the replacement) are identical – that they will produce the same answer on all database instances