CS 220
Database Systems

Fall 2019

Lecture 4

Relational Algebra

Relational Algebra

Relational Algebra has two types of operators

Set Operators

Union, Intersection, Difference & Product

Relational Operators

Select, Project, Rename, Join & Division

Set Operators - Union

Each operator uses two relations as operands and produces a new relation as a result.

A relation is represented as **R**

Cardinality of a relation is represented as **|R|**

Union compatible relations:

they have the same number of attribute types defined on the same domains

The UNION operator applied to two union-compatible relations P and Q results into a new relation $R = P \cup Q$

$$R = \{t \mid t \in P \text{ OR } t \in Q\} \text{ with } \max(|P|, |Q|) \leq |R| \leq |P| + |Q|$$

Set Operators - Union

Resulting relation **R** has tuples from P or Q or both

Duplicate tuples are eliminated

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

SUPNR	SUPNAME	SUPSTATUS
10	Victor	80
16	Bart	90
22	Catherine	85
24	Sophie	75
26	Laura	92

P ∪ Q

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80
10	Victor	80
22	Catherine	85
24	Sophie	75
26	Laura	92

Set Operators - Intersection

The INTERSECTION operator applied to two union-compatible relations P and Q results into a new relation $R = P \cap Q$

Resulting relation **R** has tuples from P and Q

$$R = \{t \mid t \in P \text{ AND } t \in Q\}$$

$$\text{with } 0 \leq |R| \leq | \min(|P|, |Q|)$$

Set Operators - Intersection

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

SUPNR	SUPNAME	SUPSTATUS
10	Victor	80
16	Bart	90
22	Catherine	85
24	Sophie	75
26	Laura	92

P ∩ Q

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90

Set Operators - Difference

The DIFFERENCE operator applied to two union-compatible relations P and Q results into a new relation R = P - Q

$$R = \{t \mid t \in P \text{ AND } t \notin Q\} \quad \text{with } 0 \leq |R| \leq |P|$$

The resulting relation R consists of the tuples that belong to P but not to Q

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

SUPNR	SUPNAME	SUPSTATUS
10	Victor	80
16	Bart	90
22	Catherine	85
24	Sophie	75
26	Laura	92

P - Q

SUPNR	SUPNAME	SUPSTATUS
18	Wilfried	60
20	Seppe	80

Set Operators - Product

The PRODUCT operator is also known as CARTESIAN PRODUCT, CROSS PRODUCT or CROSS JOIN operator
The Cartesian product of both relations consists of all possible combinations of tuples from P and Q.

The resulting relation R = P × Q consists of a set of tuples r = pq such that

$$R = \{r = pq \mid p \in P \text{ and } q \in Q\} \text{ with } |R| = |P| \times |Q|$$

Set Operators - Product

P

SUPNR	SUPNAME	SUPSTATUS
16	Bart	90
18	Wilfried	60
20	Seppe	80

Q

PRODNR	PRODNAME	PRODCOLOR
100	Hiking Shoes	black
200	Backpack	blue

P × Q

SUPNR	SUPNAME	SUPSTATUS	PRODNR	PRODNAME	PRODCOLOR
16	Bart	90	100	Hiking Shoes	black
16	Bart	90	200	Backpack	blue
18	Wilfried	60	100	Hiking Shoes	black
18	Wilfried	60	200	Backpack	blue
20	Seppe	80	100	Hiking Shoes	black
20	Seppe	80	200	Backpack	blue

Relational Operators

SELECT, PROJECT, RENAME, JOIN, and DIVISION

- SELECT, PROJECT and RENAME are unary operators
  - They use one operand.
- JOIN and DIVISION operators are binary operators
  - They use two operands.

Select Operator

The SELECT operator is used to return a subset of tuples of a relation based on a selection condition

Conditions combined using Boolean operators

AND, OR, NOT

The result of applying a SELECT operator with a selection condition S on a relation P can be represented as:

$R=\sigma_S(P)$ with SELECT operator σ

Select Operator

SELECT operator to a relation results in a horizontal subset of the relation

P

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

$$R=\sigma_{SUPCITY=San\ Francisco}(P)$$

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90

Project Operator

The PROJECT operator is used to project a relation on a list of attribute types.

The result of applying a PROJECT operator with a list L on a relation P can be represented as:

$R=\pi_L(P)$ with PROJECT operator π

Project Operator

PROJECT operator to a relation results in a vertical subset of the relation with |R| ≤ |P| since duplicate tuples will be eliminated

The result of $\pi SUPNR, SUPNAME$(P)

P

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

SUPNR	SUPNAME
10	Victor
16	Bart
22	Catherine
24	Sophie
26	Laura

$\pi_{SUPNR,\ SUPNAME}$(P)

Rename Operator

RENAME operator is used to rename an attribute of a relation

RENAME operator on a relation P to rename attribute type B to A can be represented as: $R=\rho_{A|B}(P)$ with RENAME operator ρ

The result of R=ρSUPLEVEL|SUPSTATUS(P) then becomes

SUPNR	SUPNAME	SUPCITY	SUPLEVEL
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

Join Operator

JOIN operator allows to combine tuples of two relations based on specific conditions , called join conditions

The result of applying a JOIN operator to two relations P and Q with join condition j can be represented as follows:

$$R = P \bowtie_j Q$$ with JOIN operator $\bowtie$ and join condition j

The resulting relation R consists of a set of combinations of pq tuples such that a combined tuple r ∈ R is characterized by:

$$R = \{r = pq \mid p \in P \text{ AND } q \in Q \text{ AND } j\}$$ with $|R| \leq |P| \times |Q|$

Join condition j could be that two tuples should have the same values for two corresponding attribute types

Join Operator

The relation P is a SUPPLIER relation, and relation Q is a SUPPLIES relation indicating which suppliers can supply what products.

P

SUPNR	SUPNAME	SUPCITY	SUPSTATUS
10	Victor	San Francisco	80
16	Bart	San Francisco	90
22	Catherine	Las Vegas	85
24	Sophie	New York	75
26	Laura	Orlando	92

Q

SUPNR	PRODNR
10	100
10	200
22	100
22	120
22	180
26	100
26	160

$P \bowtie_{P.SUPNR=Q.SUPNR} Q$

SUPNR	SUPNAME	SUPCITY	SUPSTATUS	PRODNR
10	Victor	San Francisco	80	100
10	Victor	San Francisco	80	200
22	Catherine	Las Vegas	85	100
22	Catherine	Las Vegas	85	120
22	Catherine	Las Vegas	85	180
26	Laura	Orlando	92	100
26	Laura	Orlando	92	160

Suppose we want a list of supplier data along with the products they can supply. This can be obtained with this join

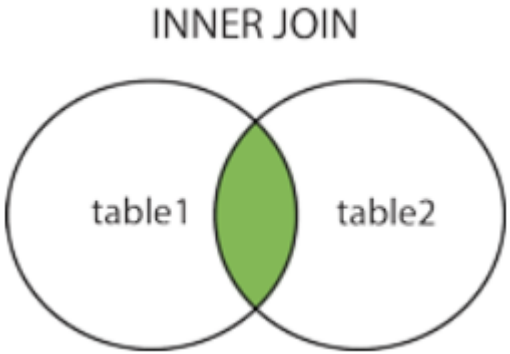
$$R = P \bowtie_{P.SUPNR=Q.SUPNR} Q$$

Theta (θ) -Join

- Theta join is a derivative of the Cartesian product. Instead of taking all combinations of tuples from R and S, we only take a subset of those tuples that match a given condition F
- The Theta-Join is a general join in that it allows any expression in the condition F.
- However, there are more specialized joins that are frequently used:
  - A **equijoin** only contains the equality operator (=) in formula F
  - The **NATURAL-JOIN** is a variant of the EQUI-JOIN in which one of the shared join-attribute types is removed from the result.

Inner-Join

➔ Theta Join, Equijoin, and Natural Join are called **inner joins**. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation.



Outer Joins

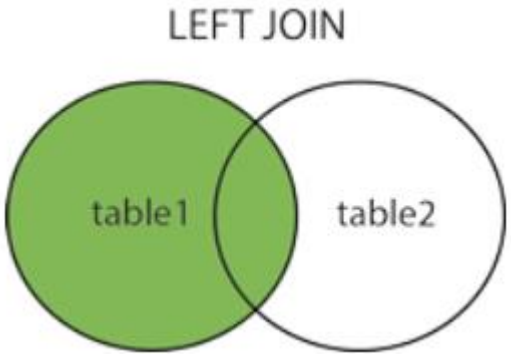
➔INNER-JOINS do not include tuples with lacking corresponding join values

➔Outer joins are types of joins that include all the tuples from the participating relations in the resulting relation.

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Joins

A LEFT-OUTER-JOIN includes all tuples from the left relation (P) in the result, either matched with a corresponding tuple from the right relation (Q) based on the join condition or augmented with NULL values in case no match with a tuple from Q can be made.



$$R = P \overset{\text{left}}{\bowtie}_j Q$$

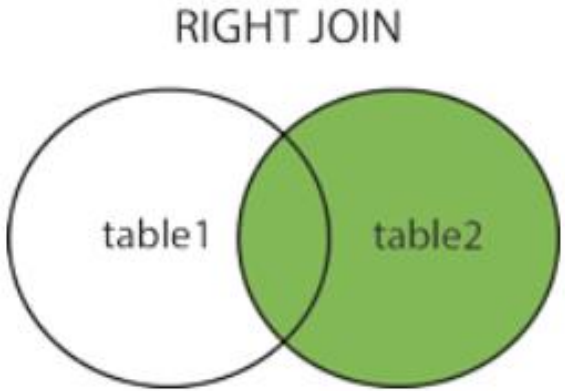
$P \overset{\text{left}}{\bowtie}_{P.SUPNR=Q.SUPNR} Q$

SUPNR	SUPNAME	SUPCITY	SUPSTATUS	PRODNR
10	Victor	San Francisco	80	100
10	Victor	San Francisco	80	200
16	Bart	San Francisco	90	NULL
22	Catherine	Las Vegas	85	100
22	Catherine	Las Vegas	85	120
22	Catherine	Las Vegas	85	180
24	Sophie	New York	75	NULL
26	Laura	Orlando	92	100
26	Laura	Orlando	92	160

Right Outer Joins

$$R = P \bowtie\!\!\!\!\!\!\subset_j Q$$

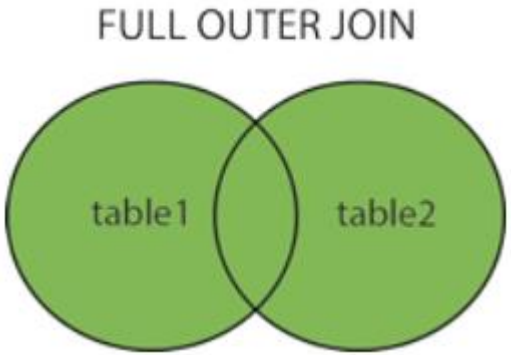
A RIGHT-OUTER-JOIN includes all tuples from the right relation (Q) in the result, either matched with a corresponding tuple from the left relation (P) based on the join condition or augmented with NULL values in case no match with a tuple from P can be made.



Full Outer Joins

$$R = P \mathbin{⟗_j} Q$$

A FULL-OUTER-JOIN includes all tuples from P and Q in the result either matched with the counterparts according to the join condition or augmented with NULL values in case no match can be found.



SQL - Joins

SQL - Join

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Syntax

```
SELECT column_list
    FROM table_1
        JOIN table_2 ON join_condition;
```

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
    FROM Orders
        JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Example Schema

Relations:

emp (eno, ename, bdate, title, salary, supereno, dno)
proj (pno, pname, budget, dno)
dept (dno, dname, mgreno)
workson (eno, pno, resp, hours)

Foreign keys:

emp: emp.supereno to emp.eno, emp.dno to dept.dno
proj: proj.dno to dept.dno
dept: dept.mgreno to emp.eno
workson: workson.eno to emp.eno, workson.pno to proj.pno

emp

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1975-01-05	EE	30000	E2	null
E2	M. Smith	1966-06-04	SA	50000	E5	D3
E3	A. Lee	1966-07-05	ME	40000	E7	D2
E4	J. Miller	1950-09-01	PR	20000	E6	D3
E5	B. Casey	1971-12-25	SA	50000	E8	D3
E6	L. Chu	1965-11-30	EE	30000	E7	D2
E7	R. Davis	1977-09-08	ME	40000	E8	D1
E8	J. Jones	1972-10-11	SA	50000	null	D1

workson

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

proj

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

dept

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

Implicit Join using Where Clause

Multiple tables can be queried in a single SQL statement by listing them in the **FROM** clause. Join condition can be specified in **Where** clause

ename
R. Davis
J. Jones

```
SELECT ename  FROM emp, dept
    WHERE dname = 'Management' and emp.dno = dept.dno
```

**This method is Not Recommended mainly due to readability*

Implicit Join - Examples

- Return the department names and the projects in each department:

```
SELECT dname, pname FROM dept, proj
    WHERE dept.dno = proj.dno
```

- Return the employees and the names of their department:

```
SELECT ename, dname FROM emp, dept
    WHERE emp.dno=dept.dno
```

- Return all projects who have an employee working on them whose title is 'EE':

```
SELECT pname FROM emp, proj, workson
    WHERE emp.title = 'EE'
        and workson.eno=emp.eno
        and workson.pno = proj.pno
```

Explicit Join Syntax

You can Specify a join condition directly in the **FROM** clause instead of the **WHERE.**

Example #1: Return the employees who are assigned to the 'Management' department

```
SELECT ename
    FROM emp JOIN dept ON emp.dno = dept.dno
    WHERE dname = 'Management'
```

Example #2: Return all projects who have an employee working on them whose title is 'EE':

```
SELECT pname
    FROM emp E JOIN workson W ON E.eno = W.eno
    JOIN proj AS P ON W.pno = P.pno
    WHERE E.title = 'EE'
```

Left Outer Join - Examples

```
Select `film`.film_id, `film`.title , `language`.name
    from `film` left outer join `language`
    on film.language_id= `language`.language_id;
```

	film_id	title	name
984	984	WONDERFUL DROP	English
985	985	WONDERLAND CHRIST	English
986	986	WONKA SEA	English
987	987	WORDS HUNTER	English
988	988	WORKER TARZAN	English
989	989	WORKING MICROCOSI	English
990	990	WORLD LEATHERNECK	English
991	991	WORST BANGER	English
992	992	WRATH MILE	English
993	993	WRONG BEHAVIOR	English
994	994	WYOMING STORM	English
995	995	YENTL IDAHO	English
996	996	YOUNG LANGUAGE	English
997	997	YOUTH KICK	English
998	998	ZHIVAGO CORE	English
999	999	ZOOLANDER FICTION	English
1000	1,000	ZORRO ARK	English

```
Select `film`.film_id, `film`.title , `language`.name
    from `language` left outer join `film`
    on  `language`.language_id = film.language_id;
```

	film_id	title	name
989	989	WORKING MICROCOSMOS	English
990	990	WORLD LEATHERNECKS	English
991	991	WORST BANGER	English
992	992	WRATH MILE	English
993	993	WRONG BEHAVIOR	English
994	994	WYOMING STORM	English
995	995	YENTL IDAHO	English
996	996	YOUNG LANGUAGE	English
997	997	YOUTH KICK	English
998	998	ZHIVAGO CORE	English
999	999	ZOOLANDER FICTION	English
1000	1,000	ZORRO ARK	English
1001	[NULL]	[NULL]	Italian
1002	[NULL]	[NULL]	Japanese
1003	[NULL]	[NULL]	Mandarin
1004	[NULL]	[NULL]	French
1005	[NULL]	[NULL]	German

Right Outer Join - Examples

```
Select `film`.film_id, `film`.title , `language`.name
    from `film` right outer join `language`
    on film.language_id= `language`.language_id;
```

	film_id	title	name
989	989	WORKING MICROCOSM	English
990	990	WORLD LEATHERNECK	English
991	991	WORST BANGER	English
992	992	WRATH MILE	English
993	993	WRONG BEHAVIOR	English
994	994	WYOMING STORM	English
995	995	YENTL IDAHO	English
996	996	YOUNG LANGUAGE	English
997	997	YOUTH KICK	English
998	998	ZHIVAGO CORE	English
999	999	ZOOLANDER FICTION	English
1000	1,000	ZORRO ARK	English
1001	[NULL]	[NULL]	Italian
1002	[NULL]	[NULL]	Japanese
1003	[NULL]	[NULL]	Mandarin
1004	[NULL]	[NULL]	French
1005	[NULL]	[NULL]	German

```
Select `film`.film_id, `film`.title , `language`.name
    from `language` right outer join `film`
    on  `language`.language_id = film.language_id;
```

	film_id	title	name
984	984	WONDERFUL DROP	English
985	985	WONDERLAND CHRIST	English
986	986	WONKA SEA	English
987	987	WORDS HUNTER	English
988	988	WORKER TARZAN	English
989	989	WORKING MICROCOSM	English
990	990	WORLD LEATHERNECK	English
991	991	WORST BANGER	English
992	992	WRATH MILE	English
993	993	WRONG BEHAVIOR	English
994	994	WYOMING STORM	English
995	995	YENTL IDAHO	English
996	996	YOUNG LANGUAGE	English
997	997	YOUTH KICK	English
998	998	ZHIVAGO CORE	English
999	999	ZOOLANDER FICTION	English
1000	1,000	ZORRO ARK	English

Inner Join vs Full Outer Join - Example

```
select first_name, last_name,
order_date, order_amount
from customers c inner join orders
o on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/4/1776	$234.56
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	03/14/1760	$78.50
Thomas	Jefferson	09/03/1790	$65.50

```
select first_name, last_name,
order_date, order_amount
from customers c full join orders o
on c.customer_id = o.customer_id
```

first_name	last_name	order_date	order_amount
George	Washington	07/04/1776	$234.56
Thomas	Jefferson	03/14/1760	$78.50
John	Adams	05/23/1784	$124.00
Thomas	Jefferson	09/03/1790	$65.50
NULL	NULL	07/21/1795	$25.50
NULL	NULL	11/27/1787	$14.40
James	Madison	NULL	NULL
James	Monroe	NULL	NULL

More SQL

SQL Operators - Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Operators - Comparison Operators

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.

SQL Operators - Comparison Operators

Operator	Description	Example
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Operators - Logical Operators

- AND
- OR
- NOT
- EXISTS
- IN
- BETWEEN
- LIKE
- IS NULL
- DISTINCT

SQL Clauses - Group By

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

```
SELECT COMPANY, COUNT(*)

FROM PRODUCT_MAST

GROUP BY COMPANY;
```

SQL Clauses - Group By with Join

Orders

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Shippers

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

```
SELECT Shippers.ShipperName,
COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

Output

ShipperName	NumberOfOrders
Federal Shipping	68
Speedy Express	54
United Package	74

SQL Clauses - Having

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

```
SELECT COMPANY, COUNT(*)
FROM PRODUCT_MAST
GROUP BY COMPANY
HAVING COUNT(*)>2;
```

SQL Clauses - Order By

- The ORDER BY clause sorts the result-set in ascending or descending order.
- It sorts the records in ascending order by default. DESC keyword is used to sort the records in descending order.

```
SELECT *
FROM CUSTOMER
ORDER BY NAME;
```

```
SELECT *
FROM CUSTOMER
ORDER BY NAME DESC;
```

SQL Clauses - Google Time

What is the use of **Limit** clause?