CZ2007
Introduction to Databases

Querying Relational Databases using SQL
Part-3

Arijit Khan

Assistant Professor
School of Computer Science and Engineering
Nanyang Technological University, Singapore

arijit.khan@ntu.edu.sg

Schedule after Recess Week

SQL	**8 Lectures** - Week 8 (Oct 07-Oct 11) - Week 9 (Oct 14-Oct 18) - Week 10 (Oct 21-Oct 25) - Week 11 (Oct 28-Nov 01)
Semi-Structured Data, Quiz-2	**2 Lectures** - Week 12 (Nov 02-Nov 08) - Quiz during Tutorial session - Quiz syllabus: everything on SQL (Week 8, 9, 10 11)
Summary	- Week 13 (Nov 11-Nov 15)

Recap: Roadmap (SQL)

- Introduction to SQL
- Querying single relation

✓

Lecture-1

✓

- Ordering Tuples
- Multi-relation queries
- Subqueries

Lecture-2

Today's lecture: Chapter 6.3, 6.4 of the Book "Database Systems: The Complete Book; Hector Garcia-Molina Jeffrey D. Ullman, Jennifer Widom

- **Set operations**
- **Bag semantics**
- **Join expressions**
- **Aggregation**

Lectures-3 & 4

Recap: Roadmap (SQL)

- Groupings
- Creation of tables
- Database modifications
- Constraints
- Views

Lecture-5 & 6

- Triggers
- Indexes

Lecture-7 & 8

That would be all about Quiz-2!!

Today's Lecture

- Set operations
- Bag semantics
- Join expressions
- Aggregation

Study-at-Home slides at the end of every lecture

- They will be in the syllabus of Quiz-2 and Final Exam
- More examples and cases
- Study them at home
- If any questions, ask me !!

Questions?

The important thing is not to stop questioning.

Albert Einstein

Set Operations in SQL: UNION, INTERSECT, EXCEPT

- They are generally used to combine the results of two separate SQL queries.

- UNION, INTERSECT, EXCEPT

Syntax

- (subquery) UNION (subquery)
- (subquery) INTERSECT (subquery)
- (subquery) EXCEPT (subquery)

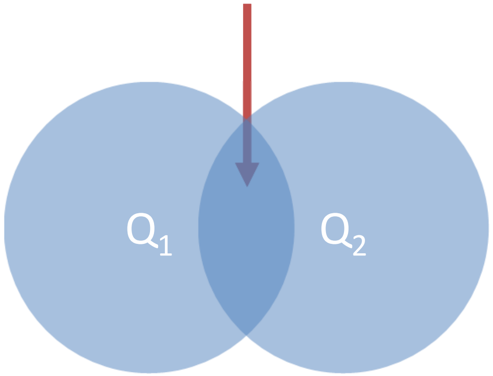
INTERSECT

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A

INTERSECT

SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \cap \{r.A \mid r.A = t.A\}$$



INTERSECT

Company(name, hq_city)
Product(pname, maker, factory_loc)

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name AND factory_loc = ‘US’

INTERSECT

SELECT hq_city
FROM   Company, Product
WHERE  maker = name AND factory_loc = ‘China’
```

✗

“Find Headquarters of companies which make gizmos in US ***AND*** *China”*

This is incorrect. **What is wrong?**

INTERSECT

Company(name, hq_city)
AS C

Product(pname, maker, factory_loc)
AS P

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'

INTERSECT

SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='China'
```

X Co. has a factory in the US (but not China)

Y Inc. has a factory in China (but not US)

INTERSECT

Company(name, hq_city)
AS C

Product(pname, maker, factory_loc)
AS P

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'

INTERSECT

SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='China'
```

X Co. has a factory in the US (but not China)

Y Inc. has a factory in China (but not US)

But Seattle is returned by the query!

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	Y	Y Inc.	China

INTERSECT

Company(name, hq_city)
AS C

Product(pname, maker, factory_loc)
AS P

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'

INTERSECT

SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='China'
```

X Co. has a factory in the US (but not China)

Y Inc. has a factory in China (but not US)

But Seattle is returned by the query!

We did the INTERSECT on the wrong attributes!

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	Y	Y Inc.	China

INTERSECT

Company(name, hq_city)
Product(pname, maker, factory_loc)

```
SELECT hq_city, name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'US'

INTERSECT

SELECT hq_city, name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'China'
```

*"Find Headquarters of companies which make gizmos in US **AND** China"*

This is okay.

[But, the output also contains "name" in addition to "hq_city"]

Solution – SELECT INTO

NANYANG TECHNOLOGICAL UNIVERSITY

```
Company(name, hq_city)
Product(pname, maker, factory_loc)
```

```
SELECT hq_city, name
INTO   HQ_Name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'US'

INTERSECT

SELECT hq_city, name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'China';
```

```
SELECT DISTINCT hq_city
FROM   HQ_Name;
```

*"Find Headquarters of companies which make gizmos in US **AND** China"*

✓

This is the solution – but it requires **two** SQL queries.

Solution – SELECT INTO

Company(name, hq_city)
Product(pname, maker, factory_loc)

```
SELECT hq_city, name
INTO   HQ_Name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'US'

INTERSECT

SELECT hq_city, name
FROM   Company, Product
WHERE  maker = name AND factory_loc = 'China';
```

```
SELECT DISTINCT hq_city
FROM   HQ_Name;
```

*"Find Headquarters of companies which make gizmos in US **AND** China"*

- SELECT INTO creates a new table.

- A physical table is created

This is the solution – but it requires **two** SQL queries.

Alternative Solution using Subquery

```
Company(name, hq_city)
Product(pname, maker, factory_loc)
```

```
SELECT DISTINCT hq_city
FROM   Company, Product
WHERE  maker = name
       AND name IN (
                        SELECT maker
                        FROM   Product
                        WHERE  factory_loc = ‘US’)
              AND name IN (
                        SELECT maker
                        FROM   Product
                        WHERE  factory_loc = ‘China’)
```

*“Headquarters of companies which make gizmos in US **AND** China”*

✓

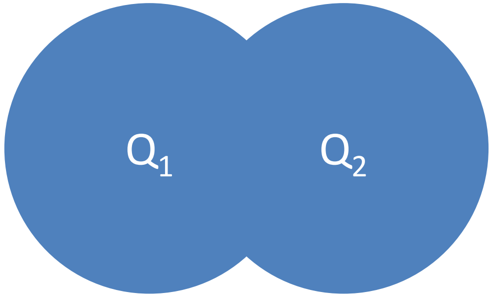
Union

```
SELECT  R.A
FROM    R, S
WHERE  R.A=S.A

UNION

SELECT R.A
FROM    R, T
WHERE  R.A=T.A
```

$\{r.A \mid r.A = s.A\} \cup \{r.A \mid r.A = t.A\}$



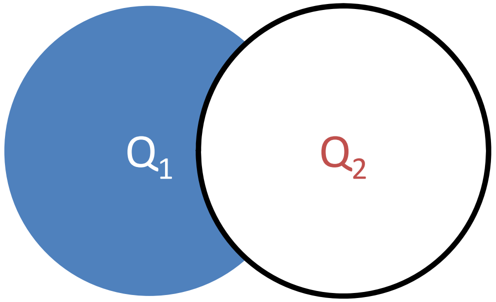
Except

```
SELECT R.A
FROM   R, S
WHERE  R.A=S.A

EXCEPT

SELECT R.A
FROM   R, T
WHERE  R.A=T.A
```

$$\{r.A \mid r.A = s.A\} \setminus \{r.A \mid r.A = t.A\}$$



More Example: Union

From relations Likes(drinker, beer), Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:

- The drinker likes the beer,

or

- The drinker frequents at least one bar that sells the beer.

```
(SELECT                  *
 FROM                    Likes)
UNION
( SELECT                 drinker, beer
  FROM                   Sells, Frequent
  WHERE                  Frequents.bar = Sells.bar);
```

Questions?

Bag Semantics vs. Set Semantics

- Set semantics → No duplicates, each item appears only once
- Bag semantics → Duplicates allowed, i.e., a multiset

- Default for SELECT-FROM-WHERE is **bag**
- Default for UNION, INTERSECT, and EXCEPT is **set**

How to change the default?

- Force set semantics with **DISTINCT** after SELECT
- Force bag semantics with **ALL** after UNION, etc.

DISTINCT: Change Bag Semantics to Set Semantics

```
SELECT DISTINCT Category
FROM   Product
```

Category
Gadgets
Photography
Household

Versus

```
SELECT Category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

ALL: Change Set Semantics to BAG Semantics

Likes

Drinker	Beer
Sally	Heineken
Sean	Bud
Melissa	Tiger

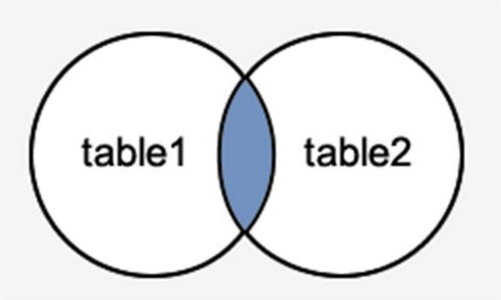
Buys

Drinker	Beer
Sally	Heineken
Sally	Bud
Melissa	Heineken
Melissa	Tiger

```
(SELECT      *
 FROM        Likes)
UNION ALL
( SELECT      *
  FROM       Buys);
```

Drinker	Beer
Sally	Heineken
Sally	Bud
Melissa	Heineken
Melissa	**Tiger**
Sally	Heineken
Sean	Bud
Melissa	**Tiger**

Join ($\bowtie$)



- Joins multiple tables

- Already did some examples while answering queries from multiple tables

Join (⋈)

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Example: Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
   AND Country=‘Japan’
   AND Price <= 200
```

```
SELECT PName, Price
FROM   Product
JOIN   Company ON Manufacturer = Cname
             AND Country=‘Japan’
WHERE  Price <= 200
```

Several equivalent ways to write a basic join in SQL.

A few more ways later on…

Join (⋈) - Example

Product

PName	Price	Category	Manuf
Gizmo	$19	Gadgets	GWorks
Powergizmo	$29	Gadgets	GWorks
SingleTouch	$149	Photography	Canon
MultiTouch	$203	Household	Hitachi

Company

Cname	Stock	Country
GWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT PName, Price
FROM   Product, Company
WHERE  Manufacturer = CName
       AND Country='Japan'
       AND Price <= 200
```

⇩

PName	Price
SingleTouch	$149.99

Meaning (Semantics) of Join

```
SELECT $x_1.a_1$, $x_1.a_2$, …, $x_n.a_k$
FROM   $R_1$ AS $x_1$, $R_2$ AS $x_2$, …, $R_n$ AS $x_n$
WHERE  Conditions($x_1$,…, $x_n$)
```

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        .....
            for $x_n$ in $R_n$ do
                if Conditions($x_1$,…, $x_n$)
                    then Answer = Answer $\cup$ {($x_1.a_1$, $x_1.a_2$, …, $x_n.a_k$)}
return Answer
```

Note: this is a *multiset* union (bag semantics)

Meaning (Semantics) of Join – An Example

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

Output

A
3
3

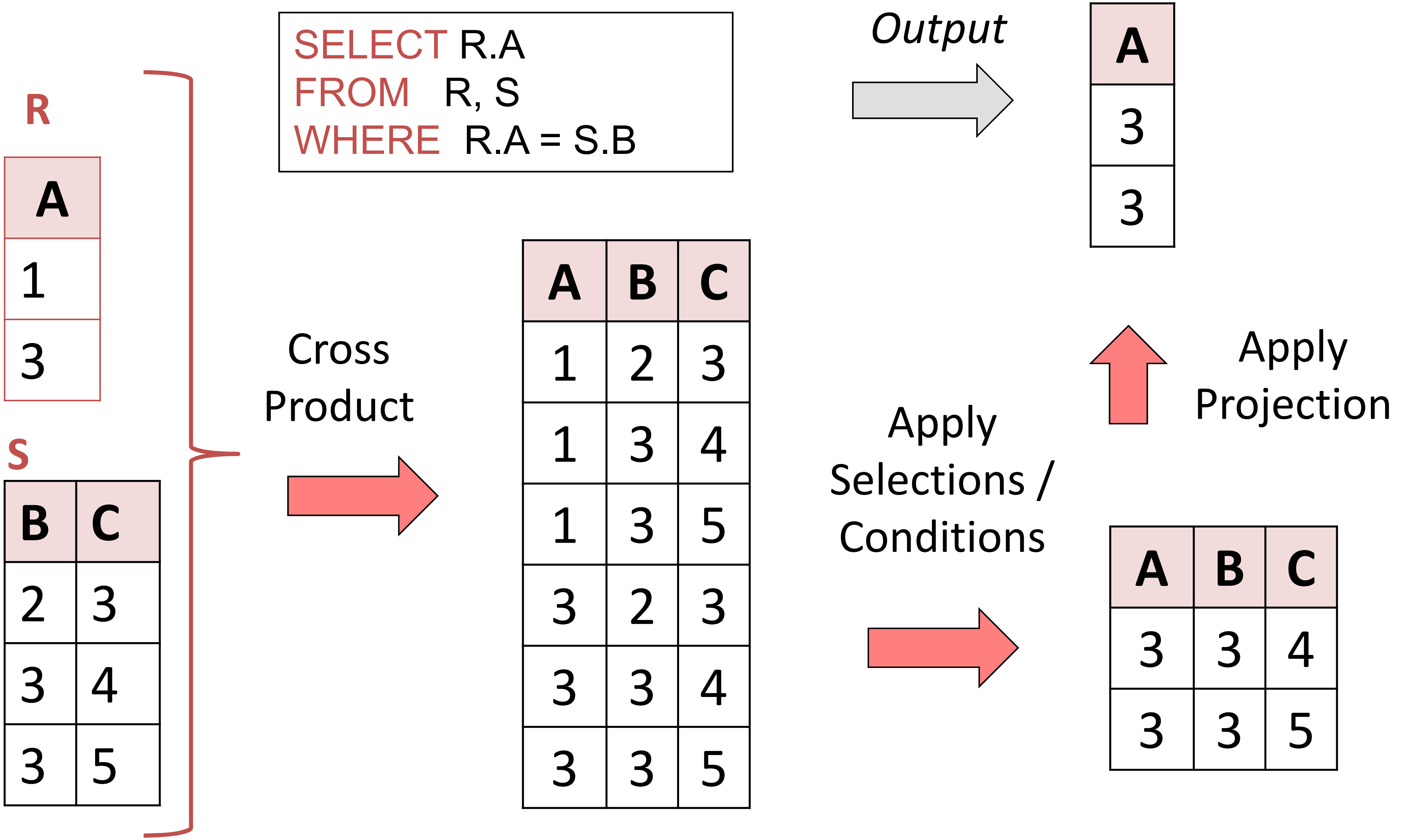
R

A
1
3

S

B	C
2	3
3	4
3	5

Meaning (Semantics) of Join – An Example



```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

R

A
1
3

S

B	C
2	3
3	4
3	5

Cross Product

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Output

A
3
3

Meaning (Semantics) of Join

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

1. Take **cross product**:

$$X = R \times S$$

2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A == r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,)\ for\ y \in Y$$

= Returning only *some* attributes
(Bag semantics)

Remembering this order is critical

How Join is Actually Executed in a Database System?

- The preceding slides show what a join means (i.e., semantics)

- Not actually how the DBMS executes it under the covers

We shall not study it in this course – will be discussed in CZ 4031

Join – More Examples

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Find all countries that manufacture some product in the ‘Gadgets’ category.

✗

```
SELECT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category=‘Gadgets’
```

Join – More Examples

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Find all countries that manufacture some product in the ‘Gadgets’ category.

```
SELECT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category=‘Gadgets’
```

✗

```
SELECT DISTINCT Country
FROM   Product, Company
WHERE  Manufacturer=CName AND Category=‘Gadgets’
```

✓

Join – A Difficult Example

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

What if S = ϕ?

```
SELECT DISTINCT R.A
FROM   R, T
WHERE  R.A=T.A
```

✗

Go back to the Join semantics! – the correct answer is ϕ

Join – A Difficult Example

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

- Recall the semantics!
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

- If S = {}, then the cross product of R, S, T = {}, and the query result = {}!

✓

Join – A Difficult Example

```
SELECT DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A OR R.A=T.A
```

- Recall the semantics!
  1. Take cross-product
  2. Apply selections / conditions
  3. Apply projection

```
output = {}

for r in R:
   for s in S:
      for t in T:
         if r['A'] == s['A'] or r['A'] == t['A']:
            output.add(r['A'])
return list(output)
```

- If S = {}, then the cross product of R, S, T = {}, and the query result = {}! ✓

Questions?

- Set operations ✓
- Bag semantics ✓
- Join expressions ✓
- **Aggregation**

SQL Aggregates

```
SELECT AVG(price)
FROM   Product
WHERE  maker = “Toyota”
```

```
SELECT COUNT(*)
FROM   Product
WHERE  year > 1995
```

- SQL supports several **aggregation** operations:
    - SUM, COUNT, MIN, MAX, AVG

Except COUNT, all aggregations apply to a single attribute

SQL Aggregates

- COUNT applies to duplicates, unless otherwise stated

```
SELECT COUNT(category)
FROM   Product
WHERE  year > 1995
```

We probably want:

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

SQL Aggregates

More Rules

- COUNT, MAX, and MIN apply to all types of fields
- SUM and AVG apply to only numeric fields.
- Except for COUNT(*) all functions ignore nulls.
- COUNT(*) returns the number of rows in the table.
- Use DISTINCT to eliminate duplicates.

More Examples on COUNT

Table

Beer(beer, manufacturer)

```
SELECT    COUNT(manufacturer)
FROM      Beer
```

NULL manufacturers will be ignored

- Duplicate manufacturers will be counted

```
SELECT    COUNT(*)
FROM      Beer
```

NULL manufacturers will be counted

More Examples on COUNT

Table

Beer(beer, manufacturer)

```
SELECT       COUNT(DISTINCT manufacturer)
FROM         Beer;
```

- Number of distinct manufacturers
- Nulls are ignored

```
SELECT       DISTINCT COUNT(manufacturer)
FROM         Beer ;
```

- Number of not-null manufacturers
- Nulls are ignored

Questions?

Summary

- Set operations ✓
- Bag semantics ✓
- Join expressions ✓
- Aggregation ✓

Study-at-Home

Types of SQL Joins (Slides 44-50)

Will be in the syllabus of Quiz-2 and Final Exam

Types of SQL Join

- Theta Join
- Inner Join
- Natural Join
- Left Outer Join
- Right Outer Join
- Full Outer Join
- ….. (Equi-Join, Product Join, Semi-Join, etc.)

Theta Join

Syntax

- R JOIN S ON <condition>
- A theta-join using <condition> for selection.

Example

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Example: Find all products manufactured in Japan, and stock price more than $300; return their names and prices.

```
SELECT PName, Price
FROM   Product
JOIN     Company ON Manufacturer = Cname  AND Country='Japan'
                              AND StockPrice >= 300
```

Theta Join

Syntax

- R JOIN S ON <condition>
- A theta-join using <condition> for selection.

Example

Product(PName, Price, Category, Manufacturer)

Company(CName, StockPrice, Country)

Example: Find all products manufactured in Japan, and stock price more than $300; return their names and prices.

```
SELECT PName, Price
FROM   Product
JOIN     Company ON Manufacturer=Cname  AND Country='Japan'
                       AND StockPrice >= 300
```

Any Boolean condition

Inner Join

Syntax

- R INNER JOIN S USING (<attribute list>)
- R INNER JOIN S ON *R.column_name = S.column_name*

Example

TableA

Column1	Column2
1	2

TableB

Column1	Column3
1	3

The INNER JOIN of **TableA** and **TableB** on Column1 will return:

TableA.Column1	TableA.Column2	TableB.Column1	TableB.Column3
1	2	1	3

SELECT * FROM TableA INNER JOIN TableB USING (Column1)

SELECT * FROM TableA INNER JOIN TableB ON TableA.Column1 = TableB.Column1

Natural Join

Syntax

- R NATURAL JOIN S

Example

TableA

Column1	Column2
1	2

TableB

Column1	Column3
1	3

The NATURAL JOIN of **TableA** and **TableB** will return:

Column1	Column2	Column3
1	2	3

```
SELECT * FROM TableA NATURAL JOIN TableB
```

- The repeated columns are avoided.
- One can not specify the joining columns in a natural join.

Outer Join

Syntax

- R **OUTER JOIN** S is the core of an outerjoin expression.

Different Variants

- Optional **NATURAL** in front of **OUTER**.
- Optional **ON** <condition> after **JOIN**.
- Optional **LEFT, RIGHT,** or **FULL** before **OUTER**.
  - **LEFT** = pad dangling tuples of R only.
  - **RIGHT** = pad dangling tuples of S only.
  - **FULL** = pad both; this choice is the default.

Example

- Loan(loanNo,branch,amount), Borrower(cName, loanNo)

```
•Loan LEFT OUTER JOIN Borrower ON
Loan.loanNo = Borrower.loanNo;
```

Questions ??

Thank You !