

ADVANCED SQL II

CS 564 - Spring 2025

ACKs: Dan Suciu, Jignesh Patel, AnHai Doan

WHAT IS THIS LECTURE ABOUT

- **SQL: Set Operators**
 - UNION/EXCEPT/INTERSECT
 - duplicates in SQL
- **SQL: Nulls**
- **SQL: Outer Joins**

SET AND MULTISSET OPERATORS

SET OPERATORS: REFRESHER

$$R = \{1, 2, 3\}$$

$$S = \{1, 2, 4, 5\}$$

- Intersection: $R \cap S = \{1, 2\}$
- Union: $R \cup S = \{1, 2, 3, 4, 5\}$
- Difference:
 $R - S = \{3\}$
 $S - R = \{4, 5\}$

SET OPERATORS IN SQL

SQL supports set operations between the outputs of subqueries:

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

SET OPERATORS: INTERSECT

SELECT A FROM R

INTERSECT

SELECT A FROM S;

R	A	S	A	output	A
	1		1		1
	1		1		
	1		2		
	2		2		2
	3		4		
			5		

Returns the tuples that belong
in **both** subquery results

SET OPERATORS: UNION

SELECT A FROM R

UNION

SELECT A FROM S;

R

A
1
1
1
2
3

S

A
1
1
2
2
4
5

output

A
1
2
3
4
5

Returns the tuples that belong
in **either** subquery results

SET OPERATORS: EXCEPT

SELECT A FROM R

EXCEPT

SELECT A FROM S;

R	A	S	A	output	A
	1		1		3
	1		1		
	1		2		
	2		2		
	3		4		
			5		

Returns the tuples that belong in the first and **not** the second subquery result

SEMANTICS

- When using set operators, SQL eliminates all duplicate tuples
- We can modify the semantics by using the keyword **ALL** (e.g. **UNION ALL**)
- When using **ALL**, the operators are evaluated using **multiset** (or **bag**) semantics

SET OPERATORS: UNION ALL

SELECT A FROM R
UNION ALL
SELECT A FROM S;

output

R	A
	1
	1
	1
	2
	3

S	A
	1
	1
	2
	2
	4
	5

The number of copies of each tuple is the **sum** of the number of copies in the subqueries

A
1
1
1
1
1
2
2
2
3
4
5

SET OPERATORS: INTERSECT ALL

SELECT A FROM R
INTERSECT ALL
SELECT A FROM S;

R	A	S	A	output	A
	1		1		1
	1		1		1
	1		2		2
	2		2		
	3		4		
			5		

The number of copies of each tuple is the **minimum** of the number of copies in the subqueries

SET OPERATORS: EXCEPT ALL

SELECT A FROM R

EXCEPT ALL

SELECT A FROM S;

R

A
1
1
1
2
3

S

A
1
1
2
2
4
5

output

A
1
3

The number of copies of each tuple is the **difference** (if positive) of the number of copies in the subqueries

NULL VALUES

NULL VALUES

- tuples in SQL relations can have **NULL** as a value for one or more attributes
- The meaning depends on context:
 - **Missing value**: *e.g.* we know that Greece has some population, but we don't know what it is
 - **Inapplicable**: *e.g.* the value of attribute *spouse* for an unmarried person

NULL PROPAGATION

- When we do arithmetic operations using **NULL**, the result is again a **NULL**
 - $(10 * x) + 5$ returns **NULL** if $x = \text{NULL}$
 - **NULL**/0 also returns **NULL**!
- String concatenation also results in **NULL** when one of the operands is **NULL**
 - 'Wisconsin' || **NULL** || '-Madison' returns **NULL**

COMPARISONS WITH NULL

- The logic of conditions in SQL is **3-valued logic**:
 - **TRUE** = 1
 - **FALSE** = 0
 - **UNKNOWN** = 0.5
- When any value is compared with a **NULL**, the result is **UNKNOWN**
 - *e.g.* $x > 5$ is **UNKNOWN** if $x = \text{NULL}$
- A query produces a tuple in the answer **only if** its truth value in the **WHERE** clause is **TRUE** (1)

3-VALUED LOGIC

The truth value of a **WHERE** clause is computed using the following rules:

- **C1 AND C2** ----> $\min\{ \text{value}(C1), \text{value}(C2) \}$
- **C1 OR C2** ----> $\max\{ \text{value}(C1), \text{value}(C2) \}$
- **NOT C** ----> $1 - \text{value}(C)$

3-VALUED LOGIC: EXAMPLE

SELECT *

FROM R

WHERE $(R.A > 0)$ AND $((R.B < 5)$ OR $(\text{NOT } R.C = 3))$;

$\underbrace{\hspace{1.5cm}}$

1

$\underbrace{\hspace{1.5cm}}$

0.5

$\underbrace{\hspace{1.5cm}}$

0.5

$\underbrace{\hspace{2.5cm}}$

0.5 (1-0.5)

$\underbrace{\hspace{4.5cm}}$

0.5 (max{0.5, 0.5})

$\underbrace{\hspace{6.5cm}}$

0.5 (min{0.5, 1})

tuple (1, NULL, NULL)

the expression is **UNKNOWN!**

COMPLICATIONS

What will happen in the following query?

```
SELECT COUNT(*)  
FROM Country  
WHERE IndepYear > 1990 OR IndepYear <= 1990 ;
```

It will not count the rows with NULL!

TESTING FOR NULL

We can test for **NULL** explicitly:

- **x IS NULL**
- **x IS NOT NULL**

```
SELECT COUNT(*)  
FROM Country  
WHERE IndepYear > 1990 OR IndepYear <= 1990  
OR IndepYear IS NULL;
```

OUTER JOINS

INNER JOINS

The joins we have seen so far are **inner joins**

```
SELECT C.Name AS Country, MAX(T.Population) AS N
FROM Country C, City T
WHERE C.Code = T.CountryCode
GROUP BY C.Name;
```

Alternative syntax:

```
SELECT C.Name AS Country, MAX(T.Population) AS N
FROM Country C
INNER JOIN City T ON C.Code = T.CountryCode
GROUP BY C.Name;
```

We can simply also write **JOIN**

LEFT OUTER JOINS

A **left outer join** includes tuples from the left relation even if there's no match on the right! It fills the remaining attributes with NULL

```
SELECT C.Name AS Country, MAX(T.Population)
FROM Country C
LEFT OUTER JOIN City T
    ON C.Code = T.CountryCode
GROUP BY C.Name ;
```

LEFT OUTER JOIN: EXAMPLE

**SELECT A, C
FROM R LEFT OUTER JOIN S
ON R.B = S.B**

R

A	B
a	2
a	5
b	5
c	6

S

B	C
2	100
3	200
5	300
7	400



A	C
a	100
a	300
b	300
c	NULL

OTHER OUTER JOINS

- **Left outer join:**
 - include the left tuple even if there is no match
- **Right outer join:**
 - include the right tuple even if there is no match
- **Full outer join:**
 - include the both left and right tuples even if there is no match