SQL:
Structured Query Language
Data Manipulation Language (DML) – Part II

CSC343, Introduction to Databases
Nosayba El-Sayed (based on slides from Diane Horton)
Fall 2015

DCS50

SQL Recap

```
SELECT sID, AVG(grade) as stdavg
FROM  Took
WHERE  sID > 22222
GROUP BY  sID
HAVING  AVG(grade) > 70
ORDER BY  AVG(grade)  DESC
```

```
Order of execution:
1. FROM
2. ON
3. JOIN
4. OUTER
5. WHERE
6. GROUP BY
7. HAVING
8. SELECT
9. DISTINCT
10.ORDER BY
```

If you're not using **DISTINCT** or **GROUPING**, you can **ORDER BY UN-SELECTED** attributes.

SQL Recap

```
SELECT sID
FROM Student
WHERE sID > 22222
ORDER BY cgpa DESC
```

```
Order of execution:
1. FROM
2. ON
3. JOIN
4. OUTER
5. WHERE
6. GROUP BY
7. HAVING
8. SELECT
9. DISTINCT
10.ORDER BY
```

If you're not using **DISTINCT** or **GROUPING**, you can **ORDER BY UN-SELECTED** attributes.

SQL Joins

The joins you know from RA

Expression within SQL statements	Meaning (RA)
`R, S`	$R \times S$
`R cross join S`	
`R natural join S`	$R \bowtie S$
`R join S on <Condition>`	$R \bowtie_{condition} S$
`R natural left [outer] join S`	$R \ltimes S$
`R natural right [outer] join S`	$R \rtimes S$
`R natural full [outer] join S`	$R \ltimes\rtimes S$

The joins you know from RA (Cont.)

Expression within SQL statements	Meaning (RA)
`R left [outer] join S on <cond>`	$R \ltimes_{condition} S$
`R right [outer] join S on <cond>`	$R \rtimes_{condition} S$
`R full [outer] join S on <cond>`	$R \ltimes\!\rtimes_{condition} S$

Dangling tuples

- With joins that require some attributes to match, tuples lacking a match are left out of the results.
- We say that they are "dangling".
- An outer join preserves dangling tuples by padding them with `NULL` in the other relation.
- A join that doesn't pad with `NULL` is called an inner join.

Example

R

A	B
1	2
4	5

S

B	C
2	3
6	7

R NATURAL LEFT JOIN S

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

Example

R

A	B
1	2
4	5

S

B	C
2	3
6	7

R NATURAL RIGHT JOIN S

A	B	C
1	2	3
NULL	6	7

Example

R

A	B
1	2
4	5

S

B	C
2	3
6	7

R NATURAL FULL JOIN S

A	B	C
1	2	3
4	5	NULL
NULL	6	7

In practice natural join is dangerous

- Attributes with matching names don't necessarily mean matching meanings!
- Having implicit comparisons impairs readability.
- Also: if the schema changed, a query that *looks* fine may actually be broken, without being able to tell.
- Best practise: Don't use natural join.

Summary of join expressions

- Cartesian product

  A cross join B — same as A, B

- Theta-join

  A join B on C — no padding of tuples

  A {left|right|full} join B on C — padding

- Natural join

  A natural join B — no padding of tuples

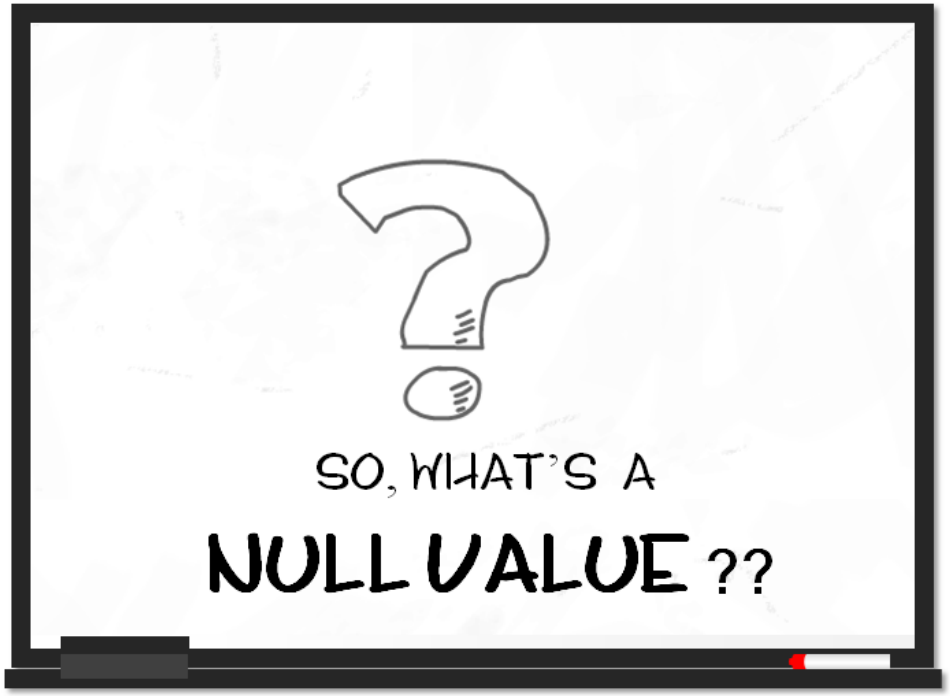
  A natural {left|right|full} join B — padding

Keywords INNER and OUTER

- There are keywords `INNER` and `OUTER`, but you never need to use them.

- Your intentions are clear anyway:
  - You get an OUTER join iff you use the keywords `LEFT`, `RIGHT`, or `FULL`.
  - If you don't use the keywords `LEFT`, `RIGHT`, or `FULL` you get an INNER join.

Impact of having null values



Missing Information

- Two common scenarios:
  - Missing value.

    E.g., we know a student has some *email* address, but we don't know what it is.
  - Inapplicable attribute.

    E.g., the value of attribute *spouse* for an unmarried person.

Representing missing information

- One possibility: use a special value as a placeholder. E.g.,
  - If age unknown, use -1.
  - If StNum unknown, use 999999999.
- Pros and cons?
- Better solution: use a value not in any domain. We call this a null value.
- Tuples in SQL relations can have `NULL` as a value for one or more components.

Checking for null values

- You can compare an attribute value to `NULL` with
  - `IS NULL`
  - `IS NOT NULL`
- Example:

```
SELECT *
FROM Course
WHERE breadth IS NULL;
```

- *Note: do not use* `WHERE breadth = NULL;`

Impact of null values on SQL expressions?

Assume: x is NULL

- Arithmetic expressions?
  - Result is always NULL
  - Example: (x + grade) → NULL
  - Even if 'grade' is 0!
    i.e. (x * 0) → NULL
  - Also: (x - x) → NULL
- Comparison operators? (> , < , = , …)
  - (x < 32) → UNKNOWN
  - Result is UNKNOWN
  - This UNKNOWN is a *truth-value*!
  - Truth-values in SQL are: (TRUE, FALSE, UNKNOWN)
  - What if we have: WHERE (x<32 OR name='Alex')?

Evaluating Logic Expressions with UNKNOWN

- Logic: TRUE, FALSE, UNKNOWN
  - UNKNOWN OR FALSE → UNKNOWN
  - UNKNOWN OR TRUE → TRUE
  - UNKNOWN AND TRUE → UNKNOWN
  - UNKNOWN AND FALSE → FALSE
  - NOT UNKNOWN → UNKNOWN

- In SQL
  - A tuple is in a query result iff the result of the WHERE clause is TRUE
  - *Demo: where-null*

Ternary logic tricks:

TRUE = 1
FALSE = 0
UNKNOWN = ½

AND = min(...)
OR = max(...)
NOT = 1-x

Thinking of the truth-values as numbers

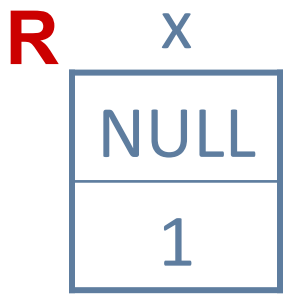
A	B	as nums	A and B	min	A or B	max
T	T	1, 1	T	1	T	1
TF or FT		1, 0	F	0	T	1
F	F	0, 0	F	0	F	0
TU or UT		1, 0.5	U	0.5	T	1
FU or UF		0, 0.5	F	0	U	0.5
U	U	0.5, 0.5	U	0.5	U	0.5

Impact of null values on aggregation

- "Aggregation ignores `NULL`."
- `NULL` never contributes to a sum, average, or count, and can never be the minimum or maximum of a column (unless *every* value is `NULL`).
- If ALL values are `NULL` in a column, then the result of the aggregation is `NULL`.
  - Exception: `COUNT` of an empty set is 0.

Impact of null values on aggregation

- COUNT()
  - COUNT(R.*) = 2
  - COUNT(S.*) = 1
  - COUNT(T.*) = 0
  - COUNT(R.x) = 1
  - COUNT(S.x) = 0
  - COUNT(T.x) = 0



- Other aggregations (e.g. MIN/MAX)
  - MIN(R.x) = 1
  - MIN(S.x) = NULL
  - MIN(T.x) = NULL
  - MAX(R.x) = 1
  - MAX(S.x) = NULL
  - MAX(T.x) = NULL

Summary

	Some nulls in A	All nulls in A
`min(A)`	ignore the nulls	NULL
`max(A)`		
`sum(A)`		
`avg(A)`		
`count(A)`		0
`count(*)`	all tuples count	

*Demo

Subqueries

Subqueries in a FROM clause

- Instead of a relation name in the FROM clause, we can use a subquery.
- The subquery must be parenthesized.
- Must name the result, so you can refer to it in the outer query.

Example

- What does this do?

```
SELECT sid, dept||cnum as course, grade
FROM Took,
   (SELECT *
    FROM Offering
    WHERE instructor='Horton') Hoffering
WHERE Took.oid = Hoffering.oid;
```

- This FROM is analogous to:

  Took × $\rho_{Hoffering}$ («*subquery*»)

- Can you suggest another version?

Subquery as a value in a WHERE

- If a subquery is guaranteed to produce exactly one tuple, then the subquery can be used as a *value*.
- Simplest situation: that one tuple has only one attribute.

Example

- Find all students with a cgpa greater than that of student 99999.

```
SELECT sid, surname
FROM Student
WHERE cgpa >
      (SELECT cgpa
       FROM Student
       WHERE sid = 99999);
```

Special cases

- What if the subquery returns `NULL`?
  - Evaluates to UNKNOWN, tuple not returned
- What if the subquery could return more than one value?
- When a subquery can return multiple values, we can make comparisons using a quantifier:
  - cgpa > at least one of them (ANY)
  - cgpa > all of them (ALL)

The Operator ANY

- Syntax:

  x *«comparison»* ANY (*«subquery»*)

  or equivalently

  x *«comparison»* SOME (*«subquery»*)

- Semantics:

  Its value is true iff the comparison holds for at least one tuple in the subquery result, i.e.,

  $\exists\ y \in$ *«subquery results»* | x *«comparison»* y

- x can be a list of attributes,

  but this feature is not supported by psql.

The Operator ALL

- Syntax:

  x *«comparison»* ALL (*«subquery»*)

- Semantics:
  Its value is true iff the comparison holds for every tuple in the subquery result, i.e.,
  $\forall\ y \in$ *«subquery results»* $|$ x *«comparison»* y

- x can be a list of attributes,
  but this feature is not supported by psql.

- *Demo: any and all*

The Operator IN

- Syntax:

  x IN (*«subquery»*)

- Semantics:

  Its value is true iff x equals at least one of the tuples in the subquery result.

- x can be a list of attributes, and psql does support this feature.

Example – *Q2 in Class Exercises*

What does this do?

```
SELECT sid, dept||cnum AS course, grade
FROM Took NATURAL JOIN Offering
WHERE
   grade >= 80 AND
   (cnum, dept) IN (
        SELECT cnum, dept
        FROM Took NATURAL JOIN Offering
                  NATURAL JOIN Student
        WHERE surname = 'Lakemeyer');
```

Q3 in Class Exercises

Suppose we have tables R(a, b) and S(b, c).

1. What does this query do?

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

2. Can we express this query without using IN?

The Operator EXISTS

- Syntax:
  EXISTS (*«subquery»*)
- Semantics:
  Its value is true iff the subquery has at least one tuple.

Example (Q4) : NOT EXISTS

What does this do?

Instructors who have exactly one offering!

```
SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
   SELECT *
   FROM Offering
   WHERE
        oid <> Off1.oid AND
        instructor = Off1.instructor );
```

Scope

- Queries are evaluated from the inside out.
- If a name might refer to more than one thing, use the most closely nested one.
- If a subquery refers only to names defined inside it, it can be evaluated once and used repeatedly in the outer query.
- If it refers to any name defined outside of itself, it must be evaluated once for each tuple in the outer query.
  These are called correlated subqueries.

Renaming can make scope explicit

```
 SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
   SELECT *
   FROM Offering Off2
   WHERE
     Off2.oid <> Off1.oid AND
     Off2.instructor = Off1.instructor );
```

Summary: where subqueries can go

- As a *relation* in a FROM clause.
- As a *value* in a WHERE clause.
- With ANY, ALL, IN or EXISTS in a WHERE clause.
- As operands to UNION, INTERSECT or EXCEPT.
- Reference: textbook, section 6.3.

Q5

- For each course find the instructor who has taught the most offerings of it. If there are ties, include them all.
  Report the course (eg csc343), instructor, and number of offerings of the course by that instructor.
  Use one or more *views* to hold intermediate steps.

ROSI Schema

Students(sID, surName, campus)

Courses(dept, cNum, name, breadth)

Offerings(oID, dept, cNum, term, instructor)

Took(sID, oID, grade)