COMP9120

Week 5: Advanced SQL

Semester 1, 2022

Dr Mohammad Polash
School of Computer Science

THE UNIVERSITY OF SYDNEY

Slides developed by Dr Lijun Chang

Reminder

- **Quiz** next week (week 6)
  - Released on Thursday, 31 March at 23:59
  - Due date Friday, 1 April at 23:59
  - Quiz duration → 1.5 hours
    - Only one attempt is allowed
  - 10 MCQ questions (on week1 – week 5 contents)
  - 3 essay questions
    - ERD to logical design
    - Relational Algebra
    - SQL
  - Practice quiz is available in Canvas to familiarize yourself with the system

Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal People of the Eora nation and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.

Copyright warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

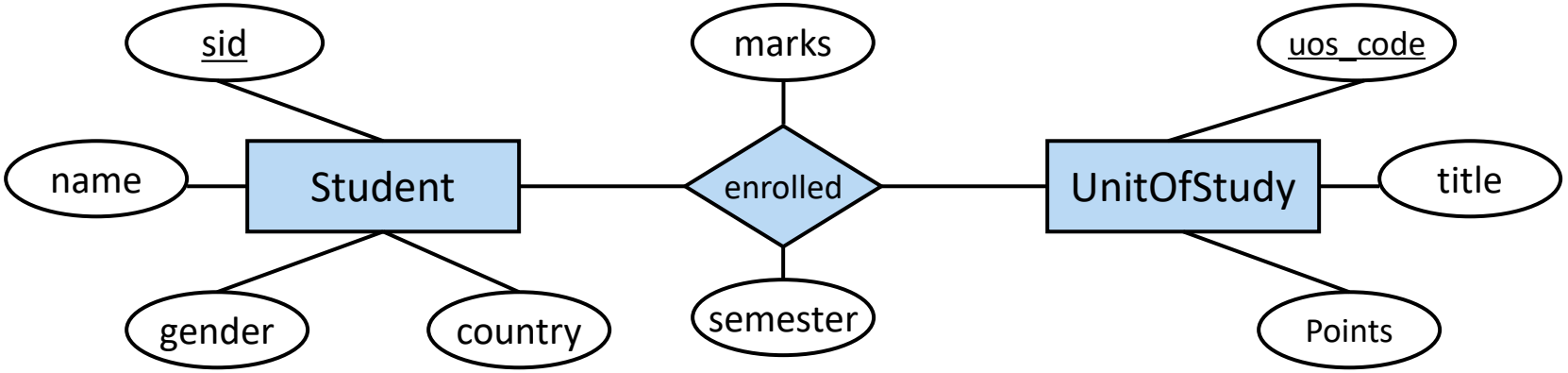
WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Running Example



Student			
sid	**name**	**gender**	**country**
1001	Adam	M	AUS
1002	Bob	M	ROK
1003	Lily	F	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Adam	M	GER

Enrolled			
sid	**uos_code**	**semester**	**marks**
1001	COMP5138	2020-S2	72
1002	COMP5702	2020-S2	85
1003	COMP5138	2020-S2	67
1006	COMP5318	2020-S2	94
1003	ISYS3207	2020-S1	78
1006	ISYS3207	2020-S2	40

UnitOfStudy		
uos_code	**title**	**points**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

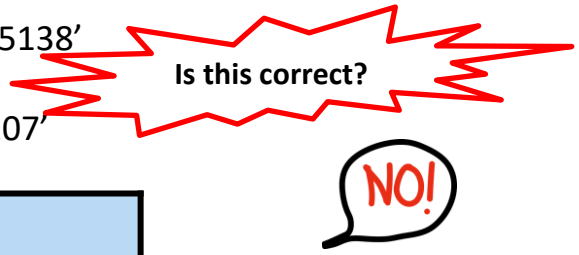
Review: Set Operations

› Find id of all students who are enrolled in both ‘COMP5138’ and ‘ISYS3207’.

```
SELECT sid FROM Enrolled WHERE uos_code='COMP5138'
INTERSECT
SELECT sid FROM Enrolled WHERE uos_code='ISYS3207'
```

› How about listing the **names** of students who are enrolled in both ‘COMP5138’ and ‘ISYS3207’?

```
SELECT name FROM Student NATURAL JOIN Enrolled WHERE uos_code='COMP5138'
INTERSECT
SELECT name FROM Student NATURAL JOIN Enrolled WHERE uos_code='ISYS3207'
```



Student			
sid	**name**	**gender**	**country**
1001	Adam	M	AUS
1002	Bob	M	ROK
1003	Lily	F	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Adam	M	GER

Enrolled			
sid	**uos_code**	**Semester**	**marks**
1001	COMP5138	2020-S2	72
1002	COMP5702	2020-S2	85
1003	COMP5138	2020-S2	67
1006	COMP5318	2020-S2	94
1003	ISYS3207	2020-S1	78
1006	ISYS3207	2020-S2	40

Outline

- **Nested Queries**
- Aggregation and Grouping
- NULL Values and Three-valued Logic

Nested Queries

› How about listing the names of students who enrolled in both ‘COMP5138’ and ‘ISYS3207’?

```
SELECT name FROM Student NATURAL JOIN Enrolled WHERE uos_code='COMP5138'
INTERSECT
SELECT name FROM Student NATURAL JOIN Enrolled WHERE uos_code='ISYS3207'
```

- Correct SQL using nested queries

```
SELECT name
  FROM Student
 WHERE sid IN (
            SELECT sid FROM Enrolled WHERE uos_code='COMP5138'
            INTERSECT
            SELECT sid FROM Enrolled WHERE uos_code='ISYS3207'
            )
```

This subquery lists the id of all students who are enrolled in both ‘COMP5138’ and ‘ISYS3207’

Nested Queries

- SQL provides a mechanism for the nesting of queries to formulate complex queries
  - SQL is **compositional**: everything (inputs/outputs) is represented as multisets, the output of one query can thus be used as the input to another (nesting!)

- A Select-From-Where expression that is nested within another query is called a **subquery**
  - Typically appear in the **WHERE** clause
  - But also, can appear in the **FROM** clause, **SELECT** clause, or **HAVING** clause

- A common use of subqueries is to perform tests for *set membership*, *set comparisons*, and *set cardinality*.

Example: Nested Queries

› Find the names of students who are enrolled in 'COMP5138'?

```
SELECT name
  FROM Student
 WHERE sid IN ( SELECT sid
                FROM Enrolled
                WHERE uos_code='COMP5138' )
```

The **IN** operator will test to see if the sid value of a row is included in the list returned from the subquery

Subquery is embedded in parentheses. In this case it returns a list that will be used in the **WHERE** clause of the outer query

Set Comparison Operators in SQL

v set-comparison R: v is a value in the outer query, R is the result of a subquery

- v **[NOT] IN** R
  - tests whether v is in (or not in, if **NOT** is specified) R: **true** $\Leftrightarrow$ v $\in$ R
- **[NOT] EXISTS** R
  - tests whether a set R is not (is, if **NOT** is specified) empty: **true** $\Leftrightarrow$ R $\neq \emptyset$ (**true** $\Leftrightarrow$ R = $\emptyset$)
- v op **ALL** R
  - op can be $<$, $\leq$, $>$, $\geq$, $=$, $\neq$
  - tests whether a predicate is true for the whole set: **true** $\Leftrightarrow \forall$ t $\in$ R : (v op t)
- v op **SOME** R
  - the same as v op **ANY** R
  - tests whether a predicate is true for at least one set element: **true** $\Leftrightarrow \exists$ t $\in$ R : (v op t)

Example: Nested Queries

› Find the id of the student with the highest mark

```
SELECT sid
FROM Enrolled
WHERE marks >= ALL ( SELECT marks
                     FROM Enrolled )
```

› Find name of the students who did not enroll in 2020-S2 semester.

```
SELECT name
FROM Student
WHERE sid NOT IN( SELECT sid
                  FROM Enrolled
                  WHERE semester = '2020-S2')
```

Views

- A view is a virtual table
  - Defined through an SQL query, used as a table in other queries
  - Normally evaluated on each use
    - Not evaluated once and stored
  - Convenient way of encapsulating queries as tables

```
CREATE VIEW student_enrollment AS
        SELECT sid, name, title, semester
        FROM student NATURAL JOIN Enrolled NATURAL JOIN unitofstudy

SELECT *
FROM student_enrollment
ORDER BY name;
```

Exercise

› Find male students name who are enrolled in units that has the lowest credit point

Solution: [Without view and step-by-step nested query]

```
SELECT name
FROM  Student
WHERE gender = 'M' AND sid IN (SELECT sid
                               FROM  Enrolled
                               WHERE uos_code IN (SELECT uos_code
                                                  FROM UnitOfStudy
                                                  WHERE points <= ALL (SELECT points
                                                                       FROM UnitOfStudy)));
```

Exercise

› Find male students name who are enrolled in units that has the lowest credit point

Solution: [Using View]

```
CREATE VIEW MaleStudents AS
        SELECT sid, name
        FROM  Student
        WHERE gender = 'M';
```

```
CREATE VIEW LowestCreditPointUnit AS
        SELECT uos_code
        FROM UnitOfStudy
        WHERE points <= ALL (SELECT points
                             FROM UnitOfStudy);
```

```
SELECT name
FROM  MaleStudents
WHERE sid IN (SELECT sid
              FROM  Enrolled
              WHERE uos_code IN (SELECT uos_code
                                 FROM LowestCreditPointUnit));
```

Outline

- Nested Queries
- **Aggregation and Grouping**
- NULL Values and Three-valued Logic

Aggregation

- Besides retrieving data, SQL supports several **aggregation** operations
  - **COUNT**, **SUM**, **AVG**, **MAX**, **MIN** (also called aggregate functions)
  - Except **COUNT**, all aggregations apply to a single attribute
  - These operations apply to duplicates, unless **DISTINCT** is specified

Example

- How many courses are there?
  - **SELECT COUNT**(*) **FROM** unitofstudy
- Find the highest mark for ‘COMP5138’?
  - **SELECT MAX**(marks) **FROM** Enrolled

    WHERE uos_code = ‘COMP5138’
- Find the average mark of ‘COMP5138’?
  - **SELECT AVG**(marks) **FROM** Enrolled

    WHERE uos_code = ‘COMP5138’
- How many students are enrolled?
  - **SELECT COUNT**(**DISTINCT** sid) **FROM** Enrolled

Enrolled			
sid	**uos_code**	**Semester**	**marks**
1001	COMP5138	2020-S2	72
1002	COMP5702	2020-S2	85
1003	COMP5138	2020-S2	67
1006	COMP5318	2020-S2	94
1003	ISYS3207	2020-S1	78
1006	ISYS3207	2020-S2	40

UnitOfStudy		
uos_code	**title**	**points**
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

Example

› Find the id of the student who has got the highest mark in COMP5138

Enrolled			
sid	**uos_code**	**Semester**	**marks**
1001	COMP5138	2020-S2	72
1002	COMP5702	2020-S2	85
1003	COMP5138	2020-S2	67
1006	COMP5318	2020-S2	94
1003	ISYS3207	2020-S1	78
1006	ISYS3207	2020-S2	40

```
SELECT sid
FROM enrolled
WHERE uos_code = 'COMP5138' AND
        marks = (SELECT MAX(marks)
                 FROM enrolled
                 WHERE uos_code = 'COMP5138');
```

Example

› Find the name of the student who has got the highest mark in ‘Relational DBMS’ course

```
SELECT name FROM student
WHERE sid IN (
        SELECT sid FROM enrolled NATURAL JOIN unitofstudy
        WHERE title = ‘Relational DBMS’ and marks =(SELECT MAX(marks)
                                                    FROM enrolled NATURAL JOIN unitofstudy
                                                    WHERE title = ‘Relational DBMS’))
```

Student

sid	name	gender	country
1001	Adam	M	AUS
1002	Bob	M	ROK
1003	Lily	F	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Adam	M	GER

Enrolled

sid	uos_code	Semester	marks
1001	COMP5138	2020-S2	72
1002	COMP5702	2020-S2	85
1003	COMP5138	2020-S2	67
1006	COMP5318	2020-S2	94
1003	ISYS3207	2020-S1	78
1006	ISYS3207	2020-S2	40

UnitOfStudy

uos_code	title	points
COMP5138	Relational DBMS	6
COMP5318	Data Mining	6
INFO6007	IT Project Management	6
SOFT1002	Algorithms	12
ISYS3207	IS Project	4
COMP5702	MIT Research Project	18

Grouping

› Instead of aggregating all (qualifying) tuples into a single value, sometimes we want to apply aggregation to each of several *groups* of tuples.

› Example: Find the total sales amount of each company

Sales Table

id	company	amount
1	IBM	5500
2	DELL	4500
3	IBM	6500

SELECT company, **SUM**(amount)
FROM Sales

company	amount
IBM	16500
DELL	16500
IBM	16500

SELECT company, **SUM**(amount)
FROM Sales
GROUP BY company

company	amount
IBM	12000
DELL	4500

Group By Queries

- In SQL, we can "partition" a relation into *groups* according to the value(s) of one or more attributes:

SELECT target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification

SELECT company, **SUM**(amount)
FROM Sales
GROUP BY company

- A *group* is a set of tuples that have the same value for all attributes in grouping-list.
- NOTE: Attributes in the **SELECT** / **HAVING** clause must be either in aggregate functions or from the grouping-list
  - Intuitively, each result tuple corresponds to a *group,* and these attributes must have a single value per group.

Group By Overview

```
SELECT company, SUM(amount)
FROM Sales
GROUP BY company
```

Sales Table

id	company	amount
1	IBM	5500
2	DELL	4500
3	IBM	6500

company	amount
IBM	5500
IBM	6500
DELL	4500

company	amount
IBM	12000
DELL	4500

Exercise

- Find the total number of student from each country

```
SELECT country, COUNT(*)
FROM student
GROUP BY country;
```

- Find the total number of Male and Female student

```
SELECT gender, COUNT(*)
FROM student
GROUP BY gender;
```

Student			
sid	**name**	**gender**	**country**
1001	Adam	M	AUS
1002	Bob	M	ROK
1003	Lily	F	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Adam	M	GER

Filtering Groups with HAVING Clause

- Group By Example:
  - List courses and their average marks

```
SELECT uos_code, AVG(marks)
FROM enrolled
GROUP BY uos_code
```

- **HAVING** clause can further filter groups to fulfil a predicate
  - Example: Find courses that have average mark > 60, and their average marks

```
SELECT uos_code, AVG(mark)
FROM enrolled
GROUP BY uos_code
HAVING AVG(mark) > 60
```

  - NOTE: Predicates in the **HAVING** clause are applied after the formation of groups, whereas predicates in the **WHERE** clause are applied before forming groups

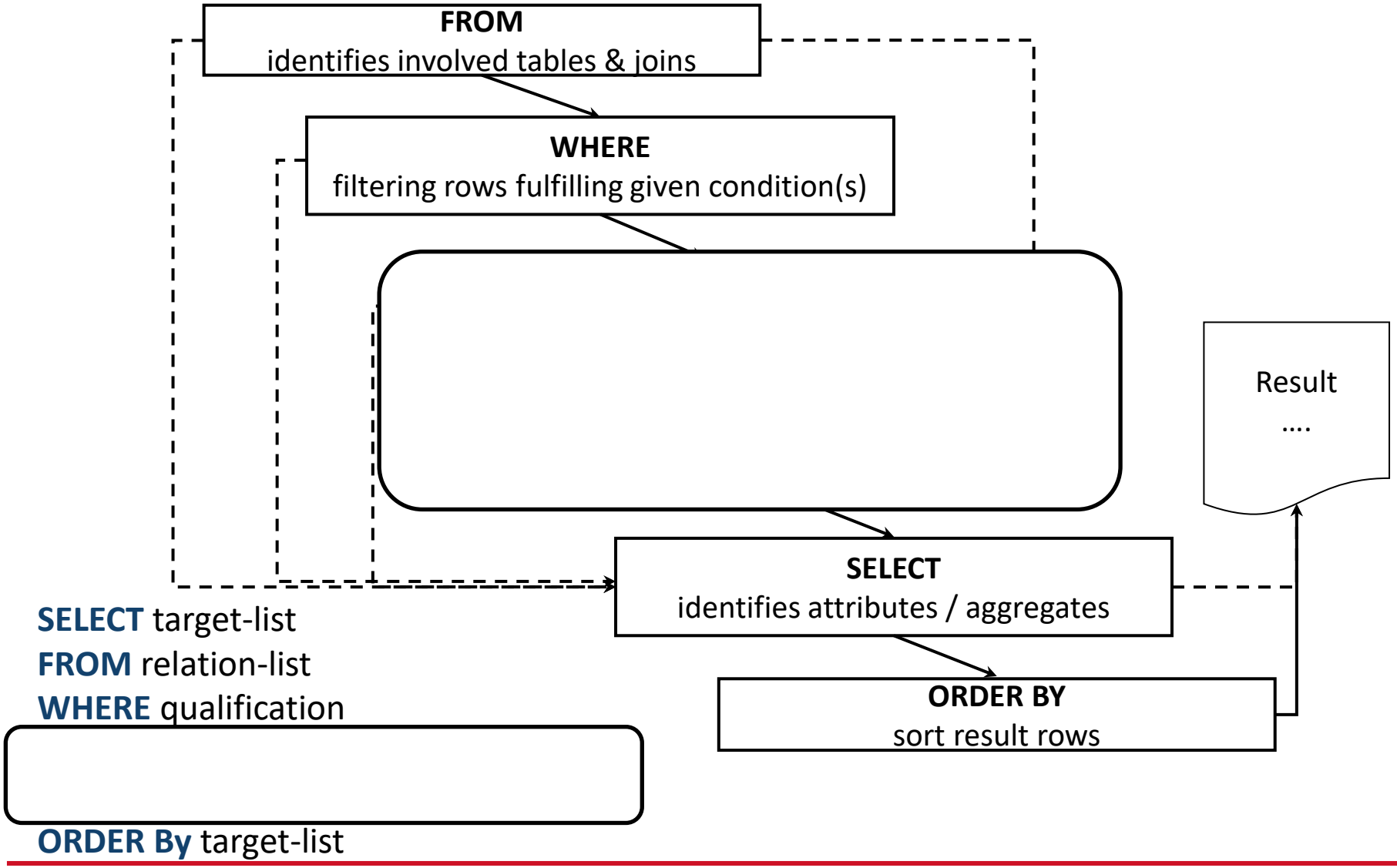
Exercise

› Select the country name with more than 1 student

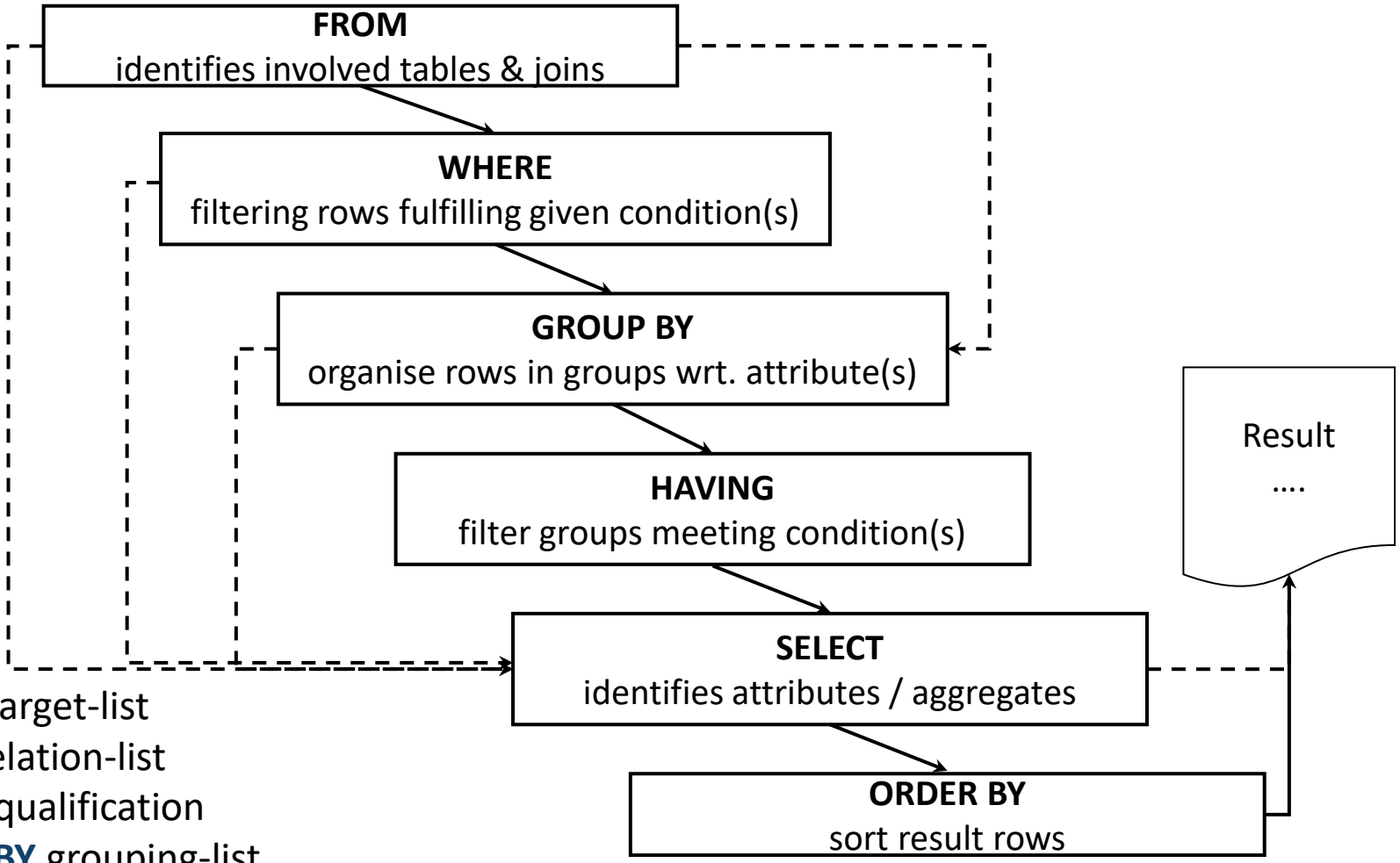
```
SELECT country
 FROM student
 GROUP BY country
 HAVING COUNT(*) > 1;
```

Student			
sid	**name**	**gender**	**country**
1001	Adam	M	AUS
1002	Bob	M	ROK
1003	Lily	F	AUS
1004	Simon	M	GBR
1005	Jesse	F	CHN
1006	Adam	M	GER

Query-Clause Evaluation Order



Query-Clause Evaluation Order



SELECT target-list
FROM relation-list
WHERE qualification
GROUP BY grouping-list
HAVING group-qualification
ORDER By target-list

Evaluation Example

› Find the maximum marks of 6-credit point courses with at least 2 students

```
SELECT uos_code, MAX(marks)
FROM enrolled NATURAL JOIN UnitOfStudy
WHERE points = 6
GROUP BY uos_code
HAVING COUNT(*) >= 2
```

1. enrolled and UnitOfStudy are joined

   FROM enrolled **NATURAL JOIN** UnitOfStudy

2. Tuples that fail the **WHERE** condition are discarded

   WHERE credit_points = 6

	uos_code character (8)	sid integer	semester character varying	marks integer	title character varying (30)	points integer
1	COMP5138	1001	2020-S2	72	Relational DBMS	6
~~2~~	~~COMP5702~~	~~1002~~	~~2020-S2~~	~~85~~	~~MIT Research Project~~	~~18~~
3	COMP5138	1003	2020-S2	67	Relational DBMS	6
4	COMP5318	1006	2020-S2	94	Data Mining	6
~~5~~	~~ISYS3207~~	~~1003~~	~~2020-S2~~	~~78~~	~~IS Project~~	~~4~~
~~6~~	~~ISYS3207~~	~~1006~~	~~2020-S1~~	~~40~~	~~IS Project~~	~~4~~

Evaluation Example (cont'd)

3. Remaining tuples are partitioned into groups by the value of attributes in the grouping-list (uos_code).

GROUP BY uos_code

4. Groups which fail the **HAVING** condition are discarded.

HAVING COUNT(*) >= 2

uos_code character (8)	sid integer	semester character varying	marks integer	title character varying (30)	points integer
COMP5138	1001	2020-S2	72	Relational DBMS	6
COMP5138	1003	2020-S2	67	Relational DBMS	6
~~COMP5318~~	~~1006~~	~~2020-S2~~	~~94~~	~~Data Mining~~	~~6~~

5. ONE result tuple is generated per group

SELECT uos_code, **MAX**(mark)

uos_code	MAX(mark)
COMP5138	72

Exercise

› Find the uos code in which students have got the lowest average mark

Solution:

```
SELECT uos_code
FROM enrolled
GROUP BY uos_code
HAVING AVG(marks) <= ALL (SELECT AVG (marks)
                          FROM enrolled
                          GROUP BY uos_code);
```

NOTE: aggregate function calls cannot be nested

Outline

- Nested Queries
- Aggregation and Grouping
- **NULL Values and Three-valued Logic**

NULL Values

- It is possible for tuples to have a null value, denoted by **NULL**, for some of their attributes
  - **NULL** signifies that a value *does not exist* or *not applicable*, it does *not mean "0" or "blank"*!
  - Integral part of SQL to handle missing / unknown information

- The predicate **IS NULL** or **IS NOT NULL** can be used to check for null values
  - e.g. Find students who don't have a mark for an assessment yet.

```
SELECT sid
FROM enrolled
WHERE marks IS NULL
```

NULL Values

- The result of any arithmetic expression involving NULL is NULL
  - e.g. 5 + NULL returns NULL

- Any comparison with NULL returns **unknown**
  - e.g. 5 < NULL or NULL <> NULL or NULL = NULL

- Result of **WHERE** clause predicate is treated as **false** if it evaluates to **unknown**
  - e.g: **SELECT** sid **FROM** enrolled **WHERE** marks < 50
    ignores all students without a mark so far

Three-valued Logic

- Three-valued logic for Boolean operations
  - OR: (**unknown** OR **true**) = **true**, (**unknown** OR **false**) = **unknown**
    (**unknown** OR **unknown**) = **unknown**
  - AND: (**true** AND **unknown**) = **unknown**, (**false** AND **unknown**) = **false**,
    (**unknown** AND **unknown**) = **unknown**
  - NOT: (NOT **unknown**) = **unknown**

- It is equivalent to the following

true = 1	B1 AND B2 = min(B1, B2)
unknown = 0.5	B1 OR B2 = max(B1, B2)
false = 0	NOT B1 = 1 – B1

Unexpected Behavior of NULL Values

```
SELECT *
FROM enrolled
WHERE marks < 25 OR marks >= 25
```

The students whose marks are unknown will not be returned!

```
SELECT *
FROM enrolled
WHERE marks < 25 OR marks >= 25 OR marks IS NULL
```

This SQL now lists all students!

NULL Values and Aggregation

- Aggregate functions ignore NULL values on the aggregated attributes
- Result of an aggregate function is NULL if there is no non-null rows
- Examples:
  - Minimum mark of all courses

```
SELECT MIN(marks)      -- ignores tuples with nulls on mark
FROM enrolled
```

  - Number of all courses

```
SELECT COUNT(*)        -- counts all tuples
FROM enrolled
```

  - Number of all courses with a mark

```
SELECT COUNT(marks)    -- ignores tuples with nulls on mark
FROM enrolled
```

You should now be able to…

- **…formulate even complex SQL Queries**
  - Including multiple joins with correct join conditions
  - Aggregate functions
  - Grouping and Having conditions
  - Handling NULL values

References

- Kifer/Bernstein/Lewis (2nd edition)
  - **Chapter 5**
- Ramakrishnan/Gehrke (3rd edition - the ‘Cow’ book)
  - Chapter 5
- Ullman/Widom (3rd edition)
  - Sections 6.3 and 6.4
- Silberschatz/Korth/Sudarshan (5th edition - ‘sailing boat’)
  - Sections 3.1-3.6
- Elmasri/Navathe (5th edition)
  - Sections 8.4 and 8.5.1

Next Week

- Database Application Development
  - SQL in Client Code
  - Call-level Database APIs
  - Server-Side Application Development with Stored Procedures

- Readings:
  - Kifer/Bernstein/Lewis book, Chapter 8
  - Ramakrishnan/Gehrke (Cow book), Chapter 6
  - Ullman/Widom, Chapter 9

See you next Week!