COMP9120

Week 6: Advanced SQL

Semester 1, 2025

Professor Athman Bouguettaya
School of Computer Science

Slides developed by Dr Lijun Chang

Warming up

Acknowledgement of Country

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

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.

New In-Person/Zoom Drop-in Helpdesk Sessions

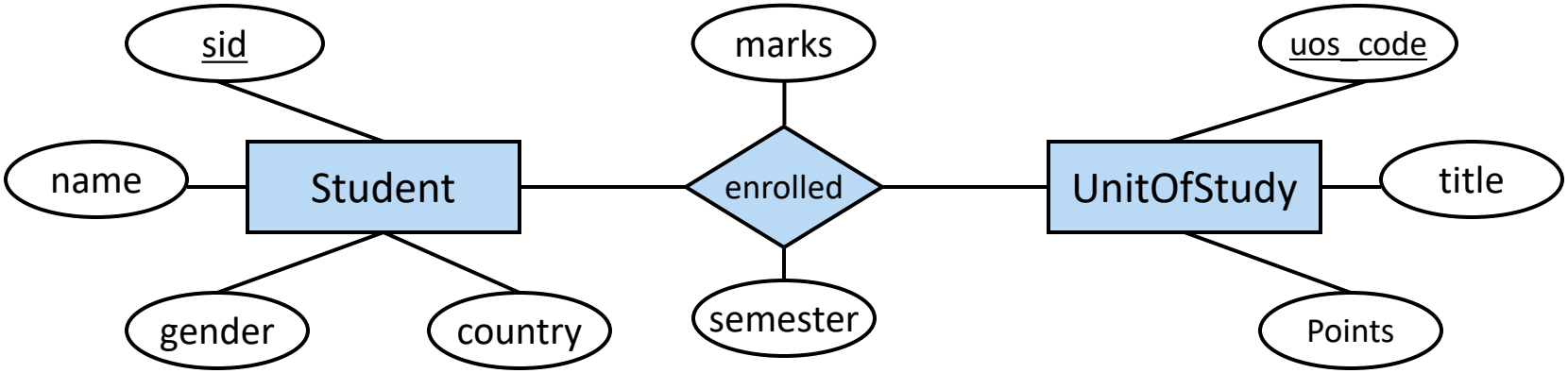
- Due to the very low attendance of live ed sessions, I am replacing them with a combination of live and zoom drop-in helpdesk sessions. The new drop-in sessions started yesterday! The schedule can be found on Canvas @ https://canvas.sydney.edu.au/courses/63042/files/42180612?module_item_id=2681519
- Note the (online) zoom session links can be accessed following the Zoom link on canvas.

Week	Date	Weekday	FROM	TO	Name	Type	Room
6	2-Apr-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
6	3-Apr-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
6	4-Apr-25	FRI	10am	12pm	Dipankar	in-person	J02.03.315.PNR Building.PNR Learning Studio 315
6	2-Apr-25	WED	10am	11am	Abbey	online	
6	2-Apr-25	WED	8pm	9pm	Abbey	online	
6	3-Apr-25	THUR	3pm	4pm	Abbey	online	
6	4-Apr-25	FRI	10am	11am	Abbey	online	
7	9-Apr-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
7	11-Apr-25	FRI	10am	12pm	Dipankar	in-person	J02.03.315.PNR Building.PNR Learning Studio 315
8	16-Apr-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
8	17-Apr-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
9	30-Apr-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
9	30-Apr-25	WED	8pm	9pm	Abbey	online	
9	1-May-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
9	2-May-25	FRI	10am	12pm	Dipankar	in-person	F07.03.352.Carslaw Building.Carslaw Learning Studio 352
10	7-May-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
10	8-May-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
10	9-May-25	FRI	10am	12pm	Dipankar	in-person	F07.03.352.Carslaw Building.Carslaw Learning Studio 352
10	7-May-25	WED	10am	11am	Abbey	online	
10	7-May-25	WED	8pm	9pm	Abbey	online	
10	8-May-25	THUR	3pm	4pm	Abbey	online	
10	9-May-25	FRI	10am	11am	Abbey	online	
11	14-May-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
11	15-May-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
11	16-May-25	FRI	10am	12pm	Dipankar	in-person	F07.03.352.Carslaw Building.Carslaw Learning Studio 352
11	14-May-25	WED	10am	11am	Abbey	online	
11	14-May-25	WED	8pm	9pm	Abbey	online	
11	15-May-25	THUR	3pm	4pm	Abbey	online	
11	16-May-25	FRI	10am	11am	Abbey	online	
12	21-May-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
12	23-May-25	FRI	10am	12pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
13	28-May-25	WED	10am	12pm	Dipankar	in-person	J12.01.110-VR.The School of Information Technologies.SIT Computer Lab 110-VR
13	29-May-25	THUR	3pm	5pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331
13	30-May-25	FRI	10am	12pm	Dipankar	in-person	A27.03.330.Edward Ford Building.Edward Ford Computer Lab 330/331

Call for a Volunteer for a “Lightning” Talk Next Week!

- If you wish to volunteer for a 5 minute “lightning” talk in Week 7 (next week!) and have professional experience using databases and want to share it with the class, please send me an email or talk to me after class. Hurry up, first come first serve!

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	2023-S2	72
1002	COMP5702	2023-S2	85
1003	COMP5138	2023-S2	67
1006	COMP5318	2023-S2	94
1003	ISYS3207	2023-S1	78
1006	ISYS3207	2023-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	Thesis	18

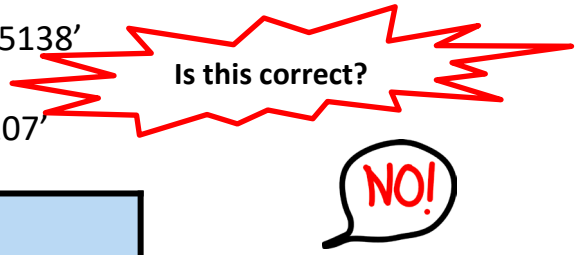
Review: Set Operations

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

```
SELECT id FROM Enrolled WHERE uos_code='COMP5138'
INTERSECT
SELECT id 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	2023-S2	72
1002	COMP5702	2023-S2	85
1003	COMP5138	2023-S2	67
1006	COMP5318	2023-S2	94
1003	ISYS3207	2023-S1	78
1006	ISYS3207	2023-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’

Set Comparison Operators in SQL

Format: *Value v* **set-comparison** *R*: ***v*** is a value in the *outer query* and ***R*** *is the result of a subquery*

- v **[NOT] IN** R
  - v **IN** R: tests whether v is in R: **true** $\Leftrightarrow$ v $\in$ R
  - v **NOT IN** R: tests whether v is *not* in R: **true** $\Leftrightarrow$ v $\notin$ R
- **[NOT] EXISTS** R
  - **EXISTS** R: tests whether a set R is *not* empty: **true** $\Leftrightarrow$ R $\neq \emptyset$
  - **NOT EXISTS** R: tests whether a set R is empty: **true** $\Leftrightarrow$ R = $\emptyset$
- v op **ALL** R
  - op can be **<, ≤, >, ≥, =, ≠**
  - tests whether a predicate (i.e., **op**) is true for the whole set: **true** $\Leftrightarrow \forall$ t $\in$ R : (v op t)
- v op **SOME** R
  - *Same* as v op **ANY** R
  - tests whether a predicate (i.e., **op**) 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 students 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 2023-S2 semester.

```
SELECT name
FROM Student
WHERE sid NOT IN ( SELECT sid
                   FROM Enrolled
                   WHERE semester = '2023-S2')
```

Views

- A view is a *virtual* table
  - Defined through a SQL query and *used as a table* in other queries
  - Normally evaluated on *each use*
    - Provides an *abstraction*
    - Provides *extra security*
  - Convenient way of encapsulating *repetitive* and *complex 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 the name of male students who are enrolled in units that have the lowest credit point

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

```
(SELECT points
 FROM UnitOfStudy)));
```

Exercise

› Find male students' names who are enrolled in units that have 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 students who have 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 names of the students who have 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	2023-S2	72
1002	COMP5702	2023-S2	85
1003	COMP5138	2023-S2	67
1006	COMP5318	2023-S2	94
1003	ISYS3207	2023-S1	78
1006	ISYS3207	2023-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	Thesis	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 values 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 the *grouping-list*.
  - Example: 'IBM' for the company attribute in the Sales table.
- 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 Example

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

Filtering Groups with HAVING Clause

- List courses and their average marks

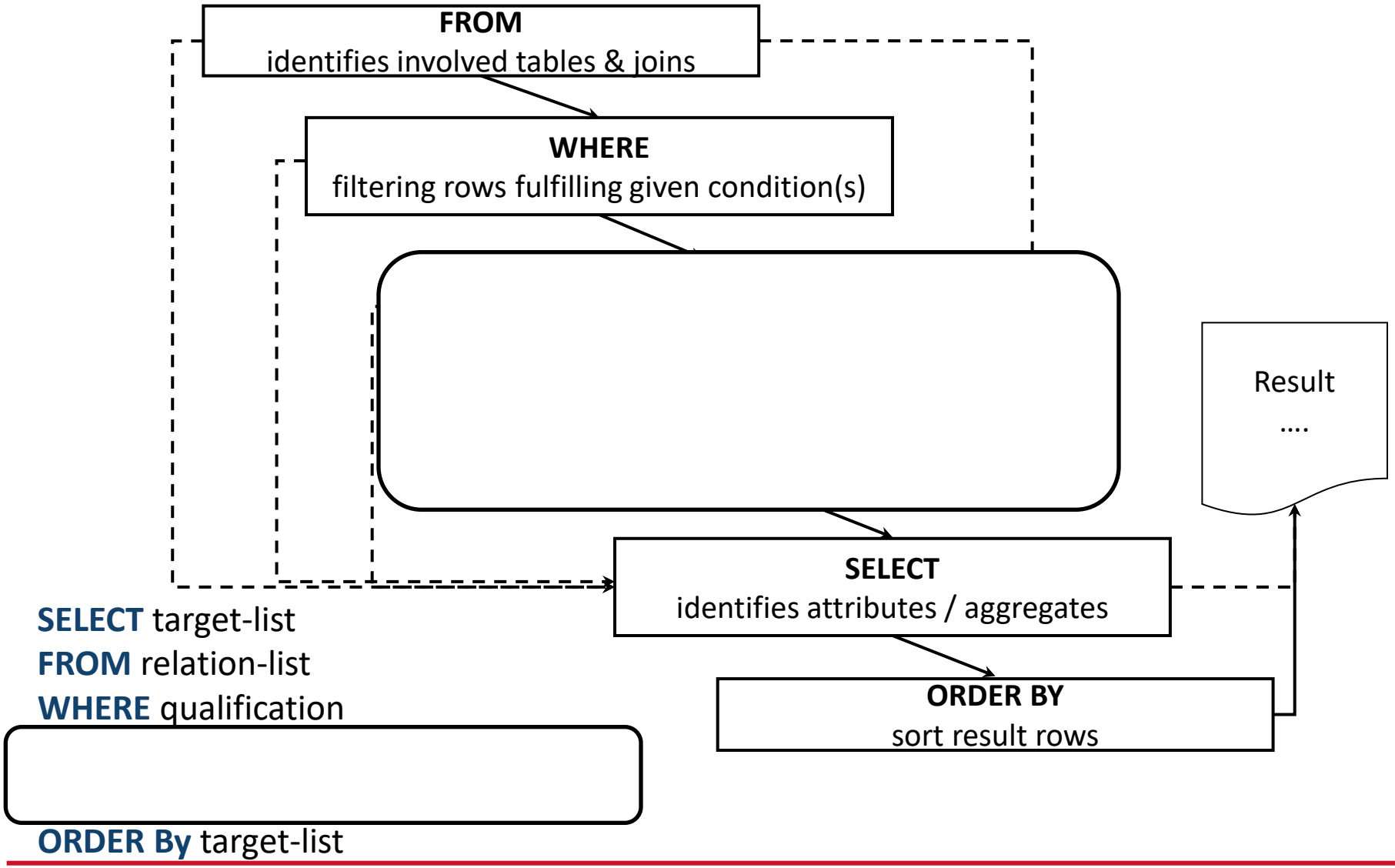
```
SELECT uos_code, AVG(marks)
FROM enrolled
GROUP BY uos_code
```

- **HAVING** clause can further *filter groups to fulfill a predicate*
  - Example: Find course code and average mark of courses that *have average marks > 60*

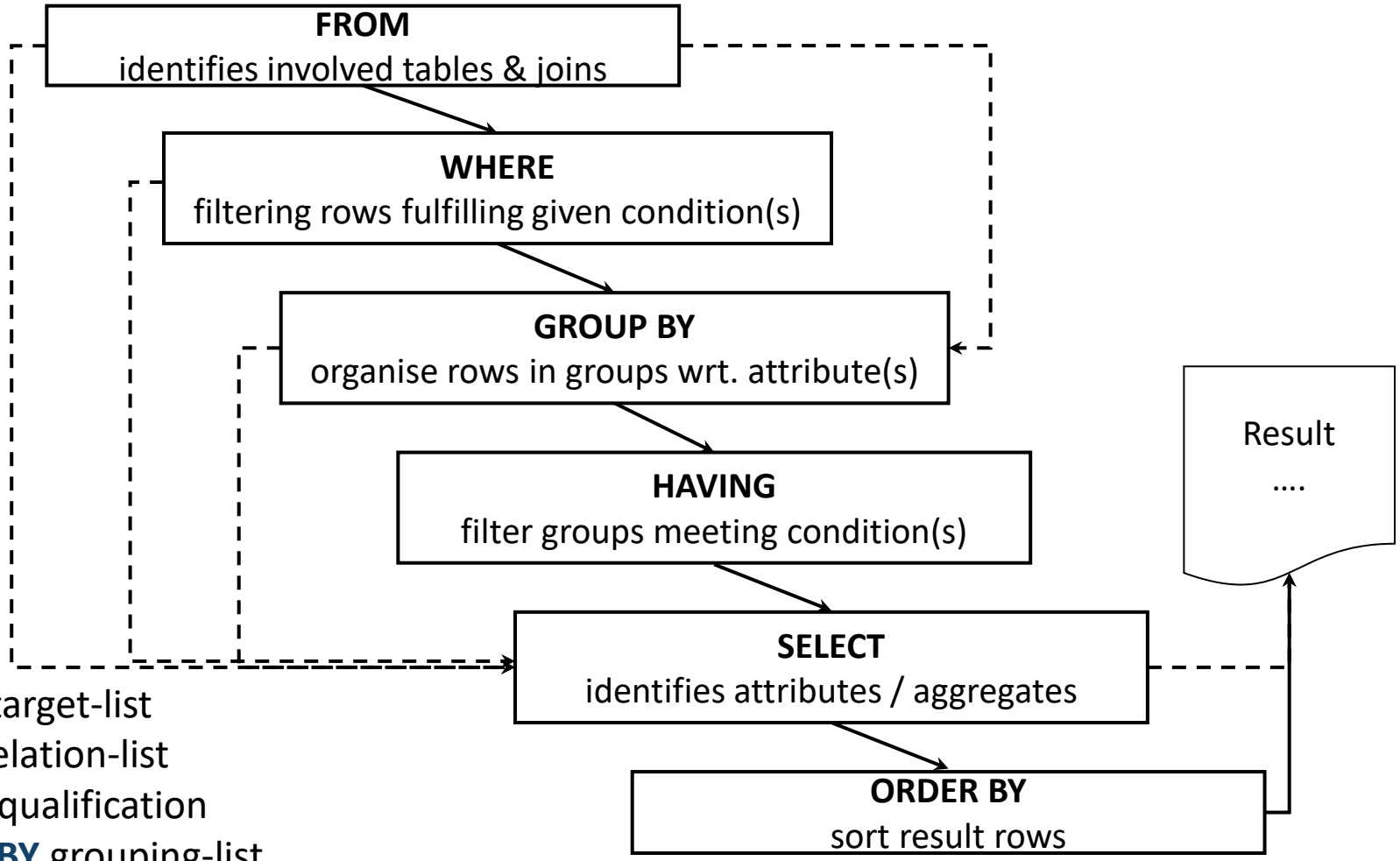
```
SELECT uos_code, AVG(mark)
FROM enrolled
GROUP BY uos_code
HAVING AVG(mark) > 60
```

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

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 Units of Study 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	2023-S2	72	Relational DBMS	6
2	COMP5702	1002	2023-S2	85	Thesis	18
3	COMP5138	1003	2023-S2	67	Relational DBMS	6
4	COMP5318	1006	2023-S2	94	Data Mining	6
5	ISYS3207	1003	2023-S2	78	IS Project	4
6	ISYS3207	1006	2023-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	2023-S2	72	Relational DBMS	6
COMP5138	1003	2023-S2	67	Relational DBMS	6
~~COMP5318~~	~~1006~~	2023-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 the lowest average mark

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

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

Short break

Let us have some fun again…

THE UNIVERSITY OF SYDNEY

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 *is 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

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
unknown = 0.5
false = 0

B1 AND B2 = min(B1, B2)
B1 OR B2 = max(B1, B2)
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 *except* **COUNT(*)** ignore NULL values on the aggregated attributes
- Result of an aggregate function is NULL if all rows have null values.

- 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
```

THE UNIVERSITY OF SYDNEY

You should now be able to…

- **…formulate even complex SQL Queries**
  - Nested queries
  - Aggregate functions
  - Grouping and Having conditions
  - Handling NULL values

Let us menti again!

Q/A session

THE UNIVERSITY OF SYDNEY

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

THE UNIVERSITY OF SYDNEY

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