# COMP9120: Review Session

Week 13: Review

Semester 1, 2025

Professor Athman Bouguettaya
School of Computer Science

THE UNIVERSITY OF SYDNEY

# 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.*

# Copyright warning

**COMMONWEALTH OF AUSTRALIA**

**Copyright Regulations 1969**

**WARNING**

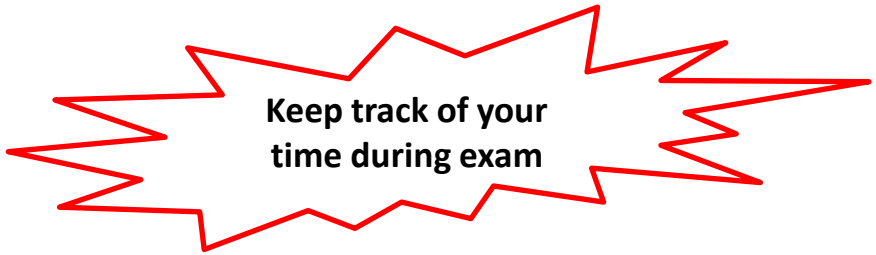
This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

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.**

# Final Exam

- Date: **Wednesday 18 June 2025**
- Time: **1:00 PM**
- Mode: **Pen-and-paper based exam.**
- Duration: **130 Minutes**
  - **Reading time: 10 Minutes**
  - **Writing time: 120 Minutes**



# Final Exam

- **EXAM CONDITIONS:**
  - This is a CLOSED book exam - no other material permitted.
  - **No electronic aids** are permitted e.g., laptops, phones, etc.
  - Answer all questions in the spaces provided on this paper. You may use pencil or ink. Marks may not be given where there is insufficient evidence of the working required to obtain the solution. If you need additional writing space, please use the extra pages provided at the end of this exam booklet. Only pages in this exam booklet will be marked.
  - MATERIALS TO BE SUPPLIED TO STUDENTS:
    - 1 x 16-page question/answer book
  - MATERIALS PERMITTED IN THE EXAM VENUE:
    - **Approved Calculator - non-programmable**

**From Canvas** (when you login to your canvas account)

***Approvable calculators and linguistic dictionaries***

*The Student Centre will be available to approve non-programmable calculators and linguistic dictionaries on **Mondays, Wednesdays and Fridays, between 9 am** and **12 pm.***

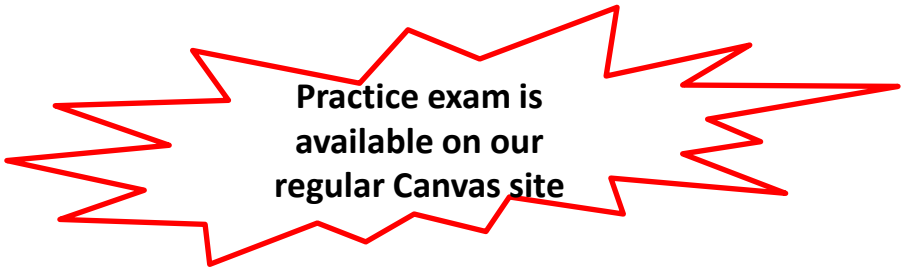
*If you visit the Student Centre outside of these designated periods you will be told to come back at the correct time.*

*The calculator you bring should be one that is listed on this page.*

# Final Exam

**Question Types**

- **4 MCQ questions** (8 marks)
- **8 Essay-type questions** (42 marks)
- Topics:
  - ERD
  - Relational Model
  - Relational Algebra
  - SQL
  - Integrity Constraints
  - Transactions
  - Normalization
  - Storage
  - Indexing
  - Query Processing
- For MCQs, **tick the correct option** in the **answer sheet**.
- Please use only **black/blue pen or pencil** to write your answers.




› Exam covers 50% of the final mark

› You must obtain at least **40% in the final exam**, as well as an **overall mark of at least 50%**, to pass the unit

# What is potentially covered in the exam?

| | Week | Topic |
|---|---|---|
| Foundations | Week 1 | Introduction |
| | Week 2 | Conceptual Database Design |
| | Week 3 | Relational Data Model / Logical Database Design |
| | Week 4 | Relational Algebra and SQL |
| | Week 5 | Database Integrity |
| | Week 6 | Advanced SQL |
| Applications | ~~Week 7~~ | ~~Database Application Development and Security~~ |
| Foundations | Week 8 | Schema Refinement and Normalisation |
| Internals | Week 9 | Transaction Management |
| | Week 11 | Storage and Indexing |
| | Week 12 | Query Processing and Evaluation |

# Some advice on how to approach the final exam

**Useful tips**:

- **Quickly** read thru **all the questions.**
- **Rank** questions from **easiest to most difficult.**
- Always **start** with the ones that are the **easiest, second easiest,** etc.
- Keep **track** on how **much time** you are **spending** on each **question vs** how much it is **worth in terms of marks.**
- If you feel you **misjudged** a **question level** of ease, **go to the next question**.
- **Suggest** that you **earmark** at least **10 minutes** at the **end** of the exam to **check** all your **answers**. Make sure you **read the questions again**.
- ***Read*** *the questions,* ***Read*** *the questions,* ***Read*** *the questions,* ***Read*** *the questions,* ***Read****….*
- Above all, do **not** panic! Panic is **much worse** than *not knowing* something in the exam!
- **Get some rest** before the exam!
- **All the best!**

# Extra resources to prepare for the exam

- Practice exam and solution – questions available from Saturday
- Practice SQL and algebraic questions (no solutions provided) – available from Saturday
- GenAI Tools to generate problems
  - Prompt design strategy: use provided sample exam questions, tutorials, menti, etc to generate new exercises.
  - Ask for a solution once you have tried.
  - However, a word of caution: many of the solutions provided by genAI tools are incorrect. Please check them. If in doubt, post them on ed and have either the teaching team or classmates check them!
- SQL lessons on ED!
- Review all ED questions and answers
- Weekly menti questions/answers (see lecture recordings)

# Which is the correct model?

"Each chef prepares **at most** one dish. Every dish must be prepared by **at least** one chef."

a.


b.


c.


d.

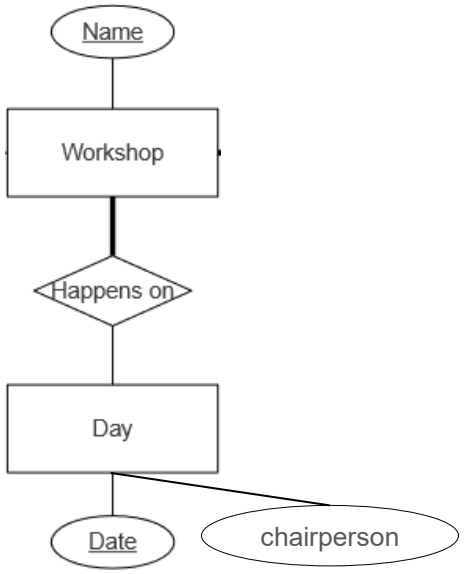
# ER diagram design

The organizers of an international conference need to keep track of a large collection of workshops associated with the event. For this, the following information needs to be recorded.

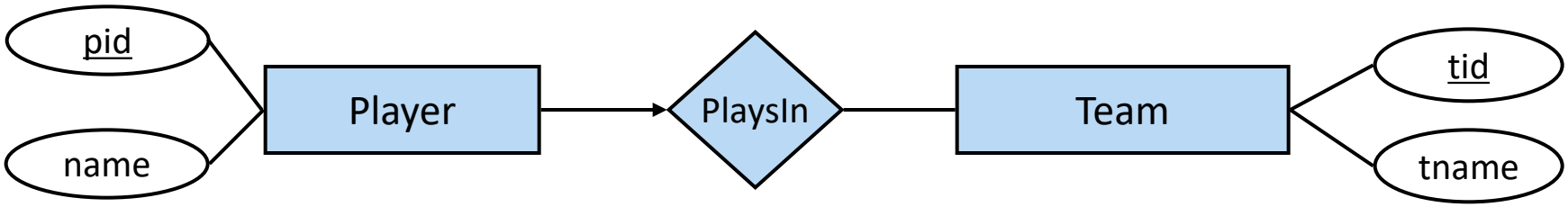
- Each workshop has a name and happens on a particular date or dates, as some workshops last more than one day. They also store the name of the person who chairs the workshop each day.
- There are several participants, each of whom must sign up for one or more workshops.
- For each participant, it is important to record their name, email address, and the workshops which they wish to attend.
- There are several meeting rooms at the conference venue, each of a fixed capacity. Meeting rooms are identified by a floor and room number.
- Every workshop needs an allocated meeting room; where a workshop lasts for more than one day, it will use the same room on all the days.

# ER diagram design

- Each workshop has a name and happens on a particular date or dates, as some workshops last more than one day. They also store the name of the person who chairs the workshop each day.
- There are several participants, each of whom must sign up for one or more workshops.
- For each participant, it is important to record their name, email address, and the workshops which they wish to attend.
- There are several meeting rooms at the conference venue, each of a fixed capacity. Meeting rooms are identified by a floor and room number.
- Every workshop needs an allocated meeting room; where a workshop lasts for more than one day, it will use the same room on all the days.

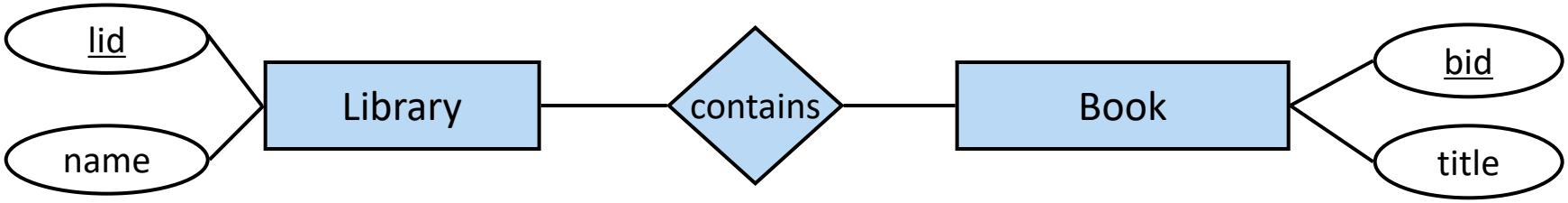


# Mapping Relationship Types with Key Constraints

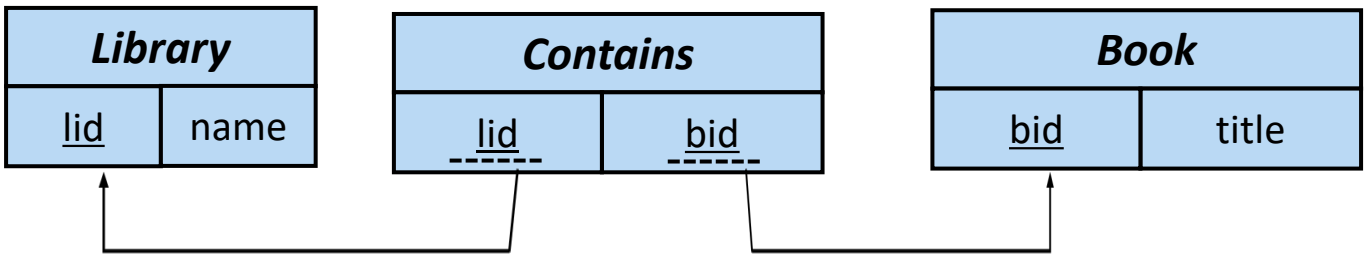



- Looking at each relationship side:
  - 1 player plays in at most 1 team
  - 1 team can have 0 to Many players

# Mapping Relationship Types with Key Constraints

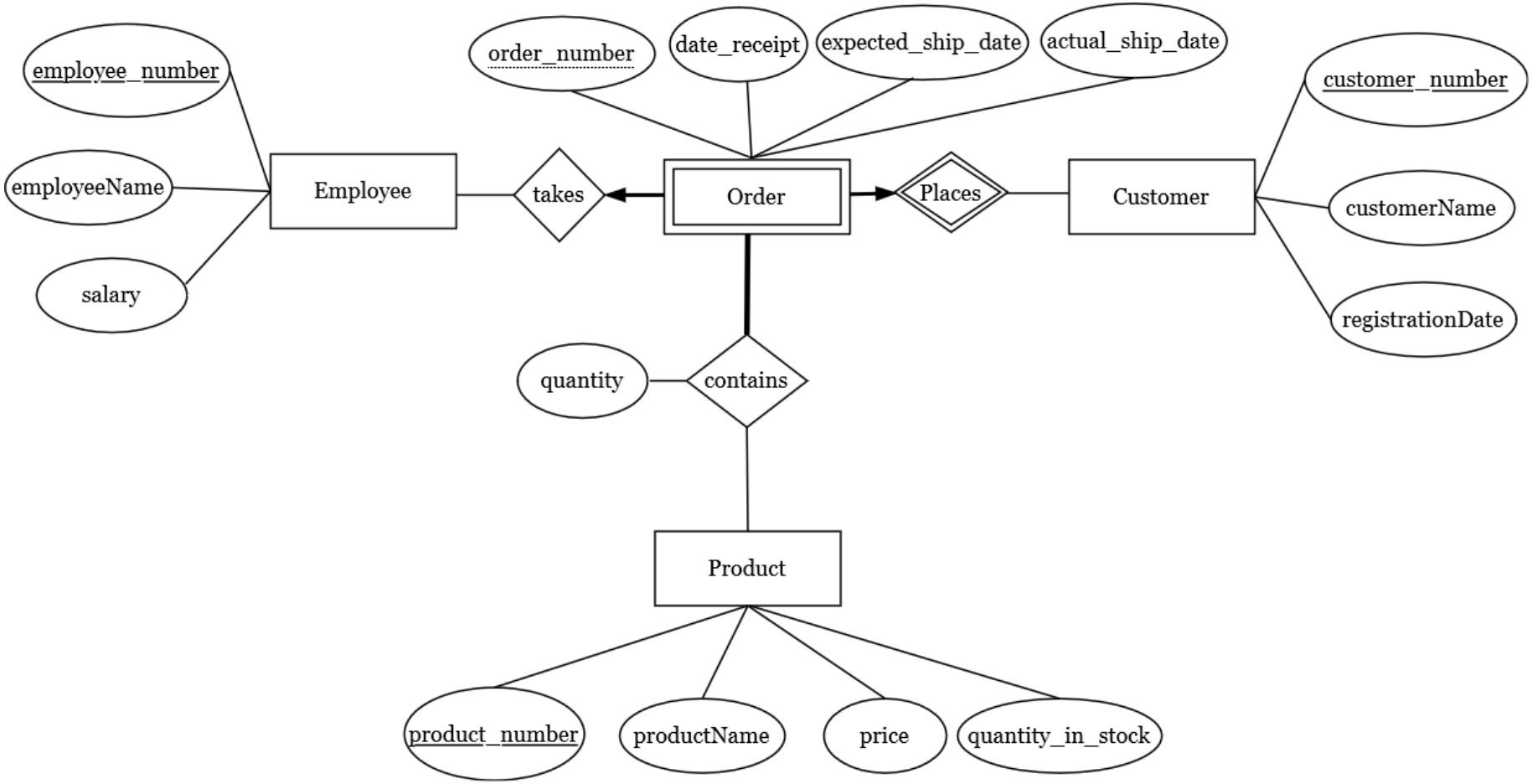



- Looking at each relationship side:
  - 1 library contains 0 to many books
  - 1 book can be found in 0 to many libraries



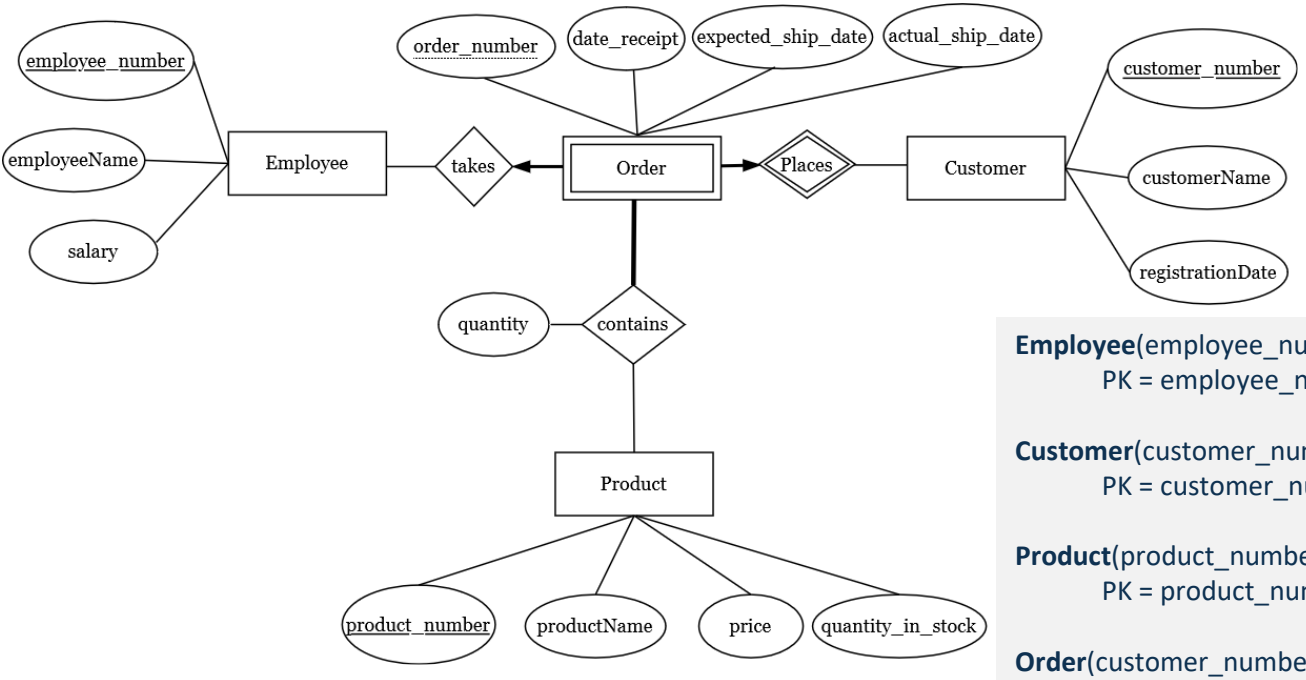
# Creating Schema Diagram

Create the schema diagram equivalent to the following ER diagram



# Creating Schema Diagram

Create the schema diagram equivalent to the following ER diagram




**Employee**(employee_number, employeeName, salary)
PK = employee_number

**Customer**(customer_number, customerName, registrationDate)
PK = customer_number

**Product**(product_number, productName, price, quantity_in_stock)
PK = product_number

**Order**(customer_number, order_number, date_receipt, expected_ship_date, actual_ship_date, employee_number)
PK = (customer_number, order_number)
FK = customer_number --> Customer
employee_number --> Employee

**Contains**(customer_number, order_number, product_number, quantity)
PK = (customer_number, order_number, product_number)
FK = (customer_number, order_number) --> Order
product_number --> Product

| *Student* | | | |
|---|---|---|---|
| **sid** | **name** | **gender** | **country** |
| 1001 | Ian | M | AUS |
| 1002 | Ha Tschi | F | ROK |
| 1003 | Grant | M | AUS |
| 1004 | Simon | M | GBR |
| 1005 | Jesse | F | CHN |
| 1006 | Franzisca | F | GER |

```
CREATE TABLE Student
(
    sid       INTEGER PRIMARY KEY,
    name      VARCHAR(20),
    gender    CHAR CHECK(gender IN ('F','M')),
    country   CHAR(3)
);
```

- Modify the table to set the *name* attribute as NOT NULL

  **ALTER TABLE** Student **ALTER COLUMN** name **SET NOT NULL**;

- Write a SQL statement to change the *country* of Simon from GBR to USA

  **UPDATE** Student **SET** country = 'USA' **WHERE** name = 'Simon'

- Write a SQL statement to delete all records from *Student* table

  **DELETE FROM** Student;

- Write a SQL statement to delete *Student* table

  **DROP TABLE** Student;

# Relational Algebra

- **List the names of all students enrolled in 'DBMS'**
  - $\pi_{name}$ ($\sigma_{title='DBMS'}$ ((Student $\bowtie$ Enrolled) $\bowtie$ UnitOfStudy))

- **List the id of students who are not enrolled in any unit of study**

  $\pi_{sid}$ (Student) - $\pi_{sid}$ (Enrolled)

Student(<u>sid</u>, name, gender, country)
Enrolled(<u>sid, uos_code</u>, semester)
UnitOfStudy(<u>uos_code</u>, title, points)

# Relational Algebra

**Suppliers(sid, sname, address)**

**Product(pid, pname, colour)**

**Catalog(sid, pid, price)**

› Find the names of all suppliers who supply a product that is red or green.

$$\pi_{sname}((\pi_{sid}((\sigma_{colour='red' \vee colour='green'}(Product)) \bowtie Catalog)) \bowtie Suppliers)$$

› Find the names of all suppliers who supply *both* red *and* green products

$$\pi_{sname}(Suppliers \bowtie (\pi_{sid}((\sigma_{colour='red'}(Product)) \bowtie Catalog) \cap \pi_{sid}((\sigma_{colour='green'}(Product)) \bowtie Catalog)))$$

› Find the id of all students who are enrolled in both ‘COMP5138’ and ‘COMP5318’.

```
SELECT sid FROM Enrolled WHERE uos_code=‘COMP5138’
INTERSECT
SELECT sid FROM Enrolled WHERE uos_code=‘COMP5318’
```

› Find the names of students who are enrolled in both ‘COMP5138’ and ‘COMP5318’

```
SELECT name
 FROM Student
WHERE sid IN ( SELECT sid FROM Enrolled WHERE uos_code=‘COMP5138’
               INTERSECT
               SELECT sid FROM Enrolled WHERE uos_code=‘COMP5318’
             )
```

Student(sid, name, gender, country)
Enrolled(sid, uos_code, semester)
UnitOfStudy(uos_code, title, points)

› Find the name of the students who are enrolled in the unit of study that has the highest credit points

Approach 1:

```
SELECT name
FROM Student
WHERE sid IN (SELECT sid
              FROM Enrolled
              WHERE uos_code IN (SELECT uos_code
                                 FROM UnitOfStudy
                                 WHERE points = (SELECT max(points) FROM UnitOfStudy )
                                )
             )
```

Approach 2:

```
SELECT name
FROM Student NATURAL JOIN Enrolled NATURAL JOIN UnitOfStudy
WHERE points = ( SELECT max(points) FROM UnitOfStudy)
```

Student(sid, name, gender, country)
Enrolled(sid, uos_code, semester)
UnitOfStudy(uos_code, title, points)

› Find the number of students in each country, except Australia ('AUS'). Only include countries that have equal or more students than Australia. Show the output in sorted order of country name.

```
SELECT country, COUNT(sid)
FROM Student
WHERE country <> 'AUS'
GROUP BY Country
HAVING COUNT(sid) >= (SELECT COUNT(sid) FROM Student WHERE country = 'AUS')
ORDER BY country
```

Student(sid, name, gender, country)
Enrolled(sid, uos_code, semester)
UnitOfStudy(uos_code, title, points)

Short 5mn break:

Please complete the Unit of Study Survey (USS)!

**Important note:** When you complete your Survey, this will give you an entry into the prize draw for a range of **JB HiFi Giftcards totalling $2500.**

THE UNIVERSITY OF SYDNEY

# Unit of Study Survey (USS) now open!

## How to make your USS feedback count

Your Unit of Study Survey (USS) feedback is **confidential**.

It's a way to share what you enjoyed and found most useful in your learning, and to provide constructive feedback. It's also a way to 'pay it forward' for the students coming behind you, so that their **learning experience** in this class is as good, or even better, than your own.

When you complete your USS survey (https://student-surveys.sydney.edu.au), please:

**Be specific.**
Which class tasks, assessments or other activities helped you to learn? *Why* were they helpful? Which one(s) *didn't* help you to learn? *Why* didn't they work for you?

**Be constructive.**
What practical changes can you suggest to class tasks, assessments or other activities, to help the next class learn better?

**Be relevant.**
Imagine you are the teacher. What sort of feedback would you find most useful to help make your teaching more effective?

# DDL

```
CREATE TABLE Film(
        filmID  int PRIMARY KEY,
        title   varchar(30),
        releaseYear int);

CREATE TABLE Actor(
        actorID int PRIMARY KEY,
        actorName   varchar(30),
        nationality varchar(20));

CREATE TABLE FilmActor(
        filmID  int,
        actorID int DEFAULT 0,
        PRIMARY KEY(filmID, actorID),
        FOREIGN KEY (filmID) REFERENCES Film ON DELETE CASCADE,
        FOREIGN KEY (actorID) REFERENCES Actor ON UPDATE SET DEFAULT);
```

| Film | | |
|---|---|---|
| **filmID** | **title** | **releaseYear** |
| 101 | No time to die | 2021 |
| 102 | Titanic | 1998 |
| 205 | Baby Day out | 2000 |

| Actor | | |
|---|---|---|
| **actorID** | **actorName** | **nationality** |
| 0 | No Actor | null |
| 981 | Daniel | UK |
| 965 | Kate | USA |
| 901 | Lily | Australia |

| FilmActor | |
|---|---|
| **filmID** | **actorID** |
| 101 | 981 |
| 102 | 965 |
| 205 | 901 |

What will be the records of Film and FilmActor table after executing the following query?

**DELETE FROM** Film **WHERE** filmID = 101;

| Film | | |
|---|---|---|
| **filmID** | **title** | **releaseYear** |
| 102 | Titanic | 1998 |
| 205 | Baby Day out | 2000 |

| FilmActor | |
|---|---|
| **filmID** | **actorID** |
| 102 | 965 |
| 205 | 901 |

# DDL

```
CREATE TABLE Film(
        filmID  int PRIMARY KEY,
        title   varchar(30),
        releaseYear int);

CREATE TABLE Actor(
        actorID int PRIMARY KEY,
        actorName   varchar(30),
        nationality varchar(20));

CREATE TABLE FilmActor(
        filmID  int,
        actorID int DEFAULT 0,
        PRIMARY KEY(filmID, actorID),
        FOREIGN KEY (filmID) REFERENCES Film ON DELETE CASCADE,
        FOREIGN KEY (actorID) REFERENCES Actor ON UPDATE SET DEFAULT);
```

| Film | | |
|---|---|---|
| **filmID** | **title** | **releaseYear** |
| 101 | No time to die | 2021 |
| 102 | Titanic | 1998 |
| 205 | Baby Day out | 2000 |

| Actor | | |
|---|---|---|
| **actorID** | **actorName** | **nationality** |
| 0 | No Actor | null |
| 981 | Daniel | UK |
| 965 | Kate | USA |
| 901 | Lily | Australia |

| FilmActor | |
|---|---|
| **filmID** | **actorID** |
| 101 | 981 |
| 102 | 965 |
| 205 | 901 |

What will be the records of Actor and FilmActor table after executing the following query?

**UPDATE** Actor **SET** actorID = 999 **WHERE** actorID = 981;

| Actor | | |
|---|---|---|
| **actorID** | **actorName** | **nationality** |
| 0 | No Actor | null |
| 999 | Daniel | UK |
| 965 | Kate | USA |
| 901 | Lily | Australia |

| FilmActor | |
|---|---|
| **filmID** | **actorID** |
| 101 | 0 |
| 102 | 965 |
| 205 | 901 |

# DDL

```
CREATE TABLE Film(
        filmID  int PRIMARY KEY,
        title   varchar(30),
        releaseYear int);

CREATE TABLE Actor(
        actorID int PRIMARY KEY,
        actorName   varchar(30),
        nationality varchar(20));

CREATE TABLE FilmActor(
        filmID  int,
        actorID int DEFAULT 0,
        PRIMARY KEY(filmID, actorID),
        FOREIGN KEY (filmID) REFERENCES Film ON DELETE CASCADE,
        FOREIGN KEY (actorID) REFERENCES Actor ON UPDATE SET DEFAULT);
```

| Film | | |
|---|---|---|
| **filmID** | **title** | **releaseYear** |
| 101 | No time to die | 2021 |
| 102 | Titanic | 1998 |
| 205 | Baby Day out | 2000 |

| Actor | | |
|---|---|---|
| **actorID** | **actorName** | **nationality** |
| 0 | No Actor | null |
| 981 | Daniel | UK |
| 965 | Kate | USA |
| 901 | Lily | Australia |

| FilmActor | |
|---|---|
| **filmID** | **actorID** |
| 101 | 981 |
| 102 | 965 |
| 205 | 901 |

- Create an assertion for the following:
  - An actor cannot take part in more than 3 movies in any given year

```
CREATE ASSERTION  ActingConstraint CHECK (
                NOT EXISTS (
                        SELECT 1
                        FROM Film NATURAL JOIN FilmActor
                        GROUP BY actorID, releaseYear
                        HAVING COUNT(filmID) > 3
                )
);
```

# Review

› Consider the following relational schema:

Emp(*eid*, *ename*, *exp*, *salary*)

Works(*eid*, *did*, *since*)

Dept(*did*, *budget*, *managerid*)

Define an assertion that will ensure that all managers have *experience* > 5 years.

› **CREATE ASSERTION** managerExp

**CHECK** ( **NOT EXISTS** (**SELECT** exp

**FROM** Emp, Dept

**WHERE** eid = managerid **AND** exp <= 5))

# Review

Student(*snum*, *sname*, *major*, *level*, *age*)

Class(*name*, *meets_at*, *room*, *fid*)

Enrolled(*snum*, *cname*)

Faculty(*fid*, *fname*, *deptid*)

Express each of the following integrity constraints in SQL

- Every class has a maximum enrolment of 30 students.
- Every faculty member must teach at least two courses.
- No department can have more than 10 faculty members.

```
CREATE TABLE Enrolled (snum INTEGER, cname CHAR(20),
        PRIMARY KEY (snum, cname),
        FOREIGN KEY (snum) REFERENCES Student,
        FOREIGN KEY (cname) REFERENCES Class,
        CHECK (( SELECT COUNT (snum)
                FROM Enrolled
                GROUP BY cname) <= 30))
```

```
CREATE ASSERTION TeachConstraint
CHECK ( ( SELECT COUNT (*)
        FROM Faculty F LEFT NATURAL JOIN Class C
        GROUP BY C.fid
        HAVING COUNT (*) < 2) = 0)
```

```
CREATE TABLE Faculty (fid INTEGER, fname CHAR(20), deptid INTEGER,
                PRIMARY KEY (fnum),
                CHECK ( ( SELECT COUNT (*)
                        FROM Faculty F
                        GROUP BY F.deptid
                        HAVING COUNT (*) > 10) = 0))
```

# Review

| uosCode | lecturerId |
|---|---|
| COMP5138 | 3456 |
| COMP5338 | 4567 |

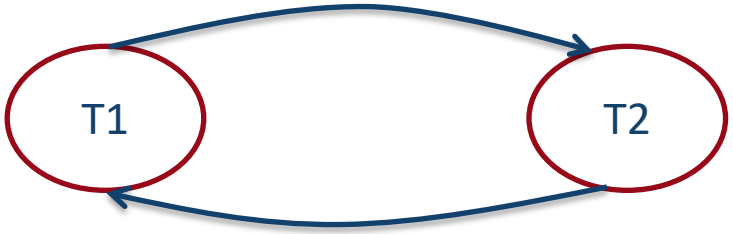
```
BEGIN;
UPDATE Course SET lecturerId=4567 WHERE uosCode='COMP5138';
COMMIT;
SELECT lecturerId FROM Course WHERE uosCode='COMP5138';
```

1. 1234
2. 3456 ✓
3. 4567

# Review

Determine whether the following schedule is conflict serializable; justify your answer. If it is conflict serializable, please give a conflict equivalent serial schedule.

- R1(x),W2(y),R1(z),R3(z),W2(x),R1(y)




There are two conflicts in this case.

First conflict is between R1(x) and W2(x).

Because of this conflict we need to put T1 before T2 i.e T1 --->T2.

Second conflict is between W2(y) and R1(y).

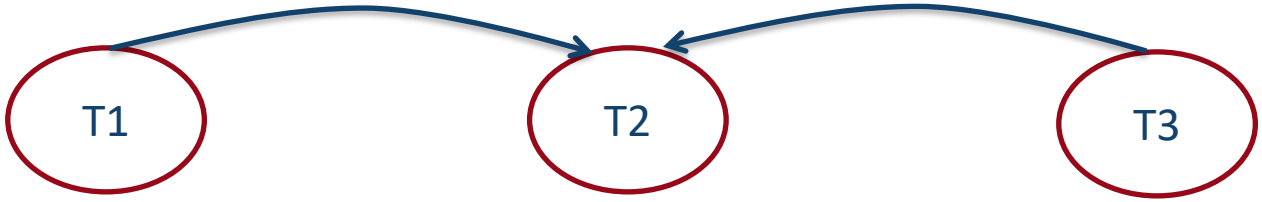
Because of this conflict we need to put T2 before T1 i.e T2 --->T1

Since we have **a cycle** between T1 and T2, that's why this schedule is **NOT** conflict serializable.

# Review

Determine whether the following schedule is conflict serializable; justify your answer. If it is conflict serializable, please give a conflict equivalent serial schedule.

- R1(x),W2(y),R1(z),R3(x),W2(x),R2(y)




There are two conflicts in this case.

First conflict is between R1(x) and W2(x).

Because of this conflict we need to put T1 before T2 i.e T1 --->T2.

Second conflict is between R3(x) and W2(x).

Because of this conflict we need to put T3 before T2 i.e T3 --->T2

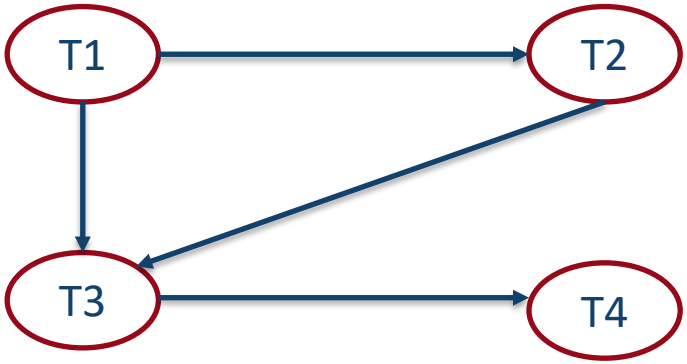
Since there is **no cycle**, so this schedule **is conflict serializable**.

According to the condition, T1 and T3 must be performed before T2. So, either of T1, T3, T2 OR T3, T1, T2 is a conflict equivalent serial schedule in this case.

# Review

› Given the following schedule S:
W1(x), R1(x), W3(z), R2(x), R2(y), R4(z), R3(x), W3(y)

Check whether this schedule is conflict serializable. If the schedule is conflict equivalent to a serial schedule, choose the equivalent serial order.




There are 4 conflicts in this case.

First conflict is between W1(x) and R2(x). Because of this conflict we need to put T1 before T2 i.e T1 --->T2.

Second conflict is between W1(x) and R3(x). Because of this conflict we need to put T1 before T3 i.e T1 --->T3

Third conflict is between W3(z) and R4(z). Because of this conflict we need to put T3 before T4 i.e T3 --->T4

Fourth conflict is between R2(y) and W3(y). Because of this conflict we need to put T2 before T3 i.e T2 --->T3

Since there is **no cycle**, so this schedule **is conflict serializable**. The equivalent serial order is T1, T2, T3, T4.

# Review

Consider a database with a table *Results(StudentId, UnitCode, Grade)* that contains the following tuples

| (Row) | StudentId | UnitCode | Grade |
|---|---|---|---|
| A | 3245 | INFO2120 | 57 |
| B | 3245 | COMP2129 | 82 |
| C | 4290 | INFO2120 | 68 |
| D | 4290 | MATH2002 | 56 |

The steps in Figure 1 can be treated for the purpose of transaction management as an execution schedule consisting of **read (r)** and **write (w)** operations by either transaction 1 or 2 upon different objects - in this case we shall assume that the granularity of these objects is row-level, with each row denoted by the letter given in the table above. Example: **r1(A)** is the same as **T1: read (B)**

Which one of the following execution schedule matches the steps in Figure 1?

1. r1(B),r2(A),r2(B),r2(C),r2(D),r1(A),r1(B),r1(A),r1(C),r2(A),r2(C)
2. r1(B),r2(A),r2(B),w2(C),w2(D),r1(A),r1(B),r1(A),r1(C),r2(A),r2(C)
3. r1(B),r2(A),r2(B),r2(C),w2(C),r2(D),w2(D),r1(A),r1(B),r1(A),r1(C),r2(A),r2(C)

| | |
|---|---|
| T1: | Begin Transaction |
| T1: | SELECT Grade FROM Results<br>WHERE StudentId = '3245' AND UnitCode = 'COMP2129' |
| T2: | Begin Transaction |
| T2: | SELECT UnitCode, Grade FROM Results<br>WHERE StudentId = '3245' |
| T2: | UPDATE Results SET Grade = Grade + 5<br>WHERE StudentId = '4290' |
| T1: | SELECT UnitCode, Grade FROM Results<br>WHERE StudentId = '3245' |
| T1: | SELECT Sum(Grade) FROM Results<br>WHERE UnitCode = 'INFO2120' |
| T1: | COMMIT |
| T2: | SELECT Sum(Grade) FROM Results<br>WHERE UnitCode = 'INFO2120' |
| T2: | COMMIT |

Figure 1

# Review

› Consider the following instance of a relation.

| A | B | C | D |
|---|---|---|---|
| 1 | x | a | 10 |
| 2 | y | b | 20 |
| 3 | z | b | 20 |
| 1 | z | c | 30 |
| 2 | y | b | 20 |

› Which of the following functional dependencies are valid according to the data in this relation?

1. AB -> C
2. C -> D
3. ~~A -> B~~
4. ABC -> D
5. ~~B -> C~~
6. D -> C

# Review

*R(A, B, C, G, H, I)*

*F = {*

*A → B*

*A → C*

*CG → H*

*CG → I*

*B → H*

*}*

Is *(AG) a candidate key?*

› Compute the attribute closure $(AG)^+$

1. *result = AG*
2. *result = ABG (A → B)*
3. *result = ABCG (A → C)*
4. *result = ABCGH (CG → H)*
5. *result = ABCGHI (CG → I)*

› First, Is *(AG) a superkey?*

1. **YES**! The closure $(AG)^+ = R$ *(i.e., **all** attributes in R)*
2. Is (AG) minimal (i.e., is any subset of (AG) a superkey?)? **It is minimal**!

Proof:

- $(A)^+ = ABCH \neq R$
- $(G)^+ = G \neq R$

*Therefore AG is a **candidate key** because it is minimal.*

# Review

- Consider a relation R(A, B, C, D, E, F) with the following functional dependencies:

  A -> BDF

  BC -> DEF

  D -> CE.

- Is AB a candidate key ?
- Compute the closure $(AB)^+$

  1. *result = AB*
  2. *result = ABDF (A → BDF)*

     *= ABCDEF (D → CE)*

- *AB is a **superkey** because its closure is R*
  - Compute the closure of A, i.e., $A^+$

    1. *result = A*
    2. *result = ABDF (A → BDF)*

       *= ABCDEF (D → CE)*

- *AB is **not** a candidate key because a subset of (AB), i.e. A, is a key.*

# Review

Work on a project is tracked with the following relation:

**WorksOn(EmpID, ProjID, start, finish, manager)**

The following FDs hold over this relation:

| | |
|---|---|
| EmpID -> manager | (Each employee can only have one manager) |
| EmpID, ProjID -> start, finish | (Each employee can only work on a project once) |
| ProjID, start -> EmpID | (Only one employee can start work on the project at a time) |

Which of the following are candidate keys for the relation?

1. **ProjID, start**
2. **EmpID, ProjID**
3. **start, finish, manager**
4. **EmpID, ProjID, start**

# Review

Based on the data in the relation, is this a valid MVD?

*UoS* ↠ *Tutor*

Note that according to the values in the relation, the relationship between the UoS and Tutor is ***independent*** from the relationship between UoS and Textbook. *This means that the above MVD is valid*. This also implies that the Tutor of a UoS is selected independently by the school.

Assume a new Tutor, Lijun C is added for the UoS COMP9120. What must happen to maintain this independence (i.e., MVD)?

- Add one row for each different textbook with Tutor *Lijun C* of that UoS.

| UoS | Textbook | Tutor |
|---|---|---|
| COMP9120 | Silberschatz | Ying Z |
| COMP9120 | Widom | Ying Z |
| COMP9120 | Silberschatz | Mohammad P |
| COMP9120 | Widom | Mohammad P |
| COMP9120 | Silberschatz | Alan F |
| COMP9120 | Widom | Alan F |
| COMP5110 | Silberschatz | Ying Z |
| COMP5110 | Silberschatz | Mohammad P |
| **COMP9120** | **Widom** | **Lijun C** |
| **COMP9120** | **Silberschatz** | **Lijun C** |

# Review

Assume the only key to the following relation is the set (UoS, Textbook, Tutor).

Is this relation in 4NF?

| UoS | Textbook | Tutor |
|---|---|---|
| COMP9120 | Silberschatz | Ying Z |
| COMP9120 | Widom | Ying Z |
| COMP9120 | Silberschatz | Mohammad P |
| COMP9120 | Widom | Mohammad P |
| COMP9120 | Silberschatz | Alan F |
| COMP9120 | Widom | Alan F |
| COMP5110 | Silberschatz | Ying Z |
| COMP5110 | Silberschatz | Mohammad P |

No: There are *two non-trivial multivalued dependencies*

UoS ↠ Textbook *and* UoS ↠ Tutor

and UoS is *not* a superkey.

Solution: Split the above relation into two relations:

Now both relations are

In 4NF! Why?

MVDs are trivial now!

# Review

› Give an example of a set of FDs for the relation schema *R(A,B,C,D)* with candidate key *AB,* so that *R* is **not in 2NF**.

2NF: *There cannot be a functional dependency between a subset of a key to non-key attributes*. This is applicable only when the key consists of more than one attribute.

An example of such an FD is:

$B \rightarrow C$

The FD $B \rightarrow C$ violates 2NF since:

*B is* a proper subset of the key *AB*

C is not part of a key

# Review

› Give an example of a set of FDs for the relation schema *R(A,B,C,D)* with candidate key *AB* under which ***R* is in 2NF but not in 3NF**.

**Third Normal Form (3NF)**

Formal Definition: a relation R is in 3NF if for each dependency X→ Y in $F^+$, *at least one of the following holds*:

- X → Y is a trivial FD (Y ⊆ X)
- X is a superkey for R
- Y ⊂ (is a proper subset of) a candidate key for R

An example of such an FD is:

$C \rightarrow D$

The FD: $C \rightarrow D$ violates 3NF but complies with 2NF since:

- $C \rightarrow D$ *is not* a trivial FD
- *C is not* a superkey
- *D is not* part of some key for *R*

# Review

› Consider a relation *R* with attributes *ABCDE* and the following FDs:

*A* → *BC*, *BC* → *E*, and *E* → *DA*.

Is *R* in BCNF? If not, give a lossless join decomposition of R.

The schema R has keys *A*, *E* and *BC*. Why? Use attribute closure to find keys.

It follows that R is in BCNF.

› Let *S* be a relation with attributes *ABCDE* and the following FDs are given:

*A* → *CE*, D → *B*, and *E* → *DA*.

Is S in BCNF? If not, give a lossless join decomposition of R.

The schema S has keys *A* and *E*. We use attribute closure to find keys.

It follows that S is *not* in BCNF because of the FD D → *B*.

Therefore, decompose S into S1 = (D,B) and S2 = (A, C, D, E). It is lossless join decomposition because the intersection of S1 and S2 is a key to S1. S1 and S2 are both in BCNF.

# Review

Is the following relation in BCNF? If not, give a lossless-join decomposition.

***contracts* (*contractID, supplierID, projectID, deptID, itemID, quantity, value*)**

Functional dependencies:

*contractID → supplierID, projectID, deptID, itemID, quantity, value*
*supplierID, deptID → itemID*
*projectID → supplierID*

**Solution:**

Note that **contractID is the key**. The above relation is **not in BCNF**. Because the LHS is not a key in the 2nd and 3rd FDs.

Therefore, looking at **supplierID, deptID → itemID**, we can divide *contracts* relation into

R1 = (supplierID, deptID, itemID) and
R2 = (supplierID, deptID, contractID, projectID, quantity, value)

R1 is in BCNF but not R2. Then looking at **projectID → supplierID**, we can divide R2 into

R3 = (projectID , supplierID) and
R4 = (projectID, deptID, contractID, quantity, value)

Therefore, R1, R3 and R4 is the lossless-join decomposition of *contracts* relation. Note the intersection of the above relations is a key to one of the relations, i.e., first, R1, and then R3.

# Review

Assume that there are 1,000,000 records in a relation R and each record is 200 bytes long. Each page is 4K bytes, of which 250 bytes are reserved for header and array of record pointers. Assume pages are 90% full on average.

- How many records can we store in each page?
- How many pages are required to store R?

**Solution:**

Empty space in each page is (4*1024 – 250) = 3846 bytes

Number of records per page = floor(3846 / 200) = 19 records

On average, each page contains floor(19 * 90%) = 17 records

Number of pages required to store the table = ceil(1,000,000/17) = 58,824 pages

# Join

Consider two tables *employee(eid, ename, did)* and *department(did, dname).* There are 10,000 tuples in the *employee* table and 1,000 tuples in the *department t*able. It requires 500 pages and 100 pages to store the records of *employee* and *department* tables, respectively.

› Calculate the cost of nested loop join and block-nested loop join of these two tables.

The estimated cost of nested loop join

*employee* as outer table: 500+ 10000 * 100 = **1,000,500 disk I/Os**

*department* as outer table: 100 + 1000 ∗ 500 = **500,100 disk I/Os**

The estimated cost of block nested loop join

*employee* as outer table: 500 + 500 * 100 = **50,500 disk I/Os**

*department* as outer table: 100 + 100 ∗ 500 = **50,100 disk I/Os**

# Query Processing

› True/False?

- Parser takes a query-evaluation plan, executes that plan, and returns the answers. False
- SQL systems remove duplicates even if the keyword **DISTINCT** is not specified in a query. False
- Pipelined evaluation is cheaper than materialization True
- External merge-sort works even if the entire table does not fit in the main memory True
- Natural join is a special case of Equi-join True

# Student Engagement Statistics in COMP9120

| Number of questions/posts | Number of answers/comments | Number of views | Student Participation |
|---|---|---|---|
| 724 | 1279 | 182k | 683 |

**Per week, on Ed**

~61 questions

~107 answers/comments

**Top 3 Student Contributors**

- Lihang Shen
- Priyansh Goyal
- Jiang Ziyang Jiang

Thank you!

**Top 3 Staff Contributors**

- Athman Bouguettaya
- Abbey Lin
- Dipankar Chaki

**Resources**

- Lecture: slides, demos, recordings
- Tutorial: sheets, solutions
- Menti
- Practice SQL resources, practice exam
- PostgreSQL/Python/Java/pgdmin

**Average marks**

- Assignment1 → 69%
- Quiz → 52.24%
- Assignment 2 → being marked!
- Final Exam → Best of luck!

# Menti Q/A

That’s all for me..

All the best in your final exam!

Great pleasure to have had you in my class!

THE UNIVERSITY OF SYDNEY