

# COMP9311

## 2022T1

### Revision

# Details of Final Exam:

---

1. Time: 2PM to 5PM, Sydney time. 7<sup>th</sup> May 2022.
2. Exam paper: will be released on our course website just before 2PM 7<sup>th</sup> May 2022.
3. How to submit: via Moodle
  1. Same procedure as what you did for the two assignments.
  2. In case near 5PM and you cannot submit via Moodle, send your solution to [comp9311unsw@gmail.com](mailto:comp9311unsw@gmail.com) with you zID and name.
  3. You can submit multiple times and we will mark the last one.
  4. We accept any format: directly answer using word or handwriting and convert to word/pdf. As long as the file is in .doc or .pdf format and clear.
4. Any question during the exam, send email to [comp9311unsw@gmail.com](mailto:comp9311unsw@gmail.com) and cc [mryu@cse.unsw.edu.au](mailto:mryu@cse.unsw.edu.au)

# Consultation:

---

We will also run **daily consultation** next week:

3PM to 4PM via Moodle

- Monday 25th Aug
- Tuesday 26th Aug
- Wednesday 27th Aug
- Thursday 28th Aug
- Friday 29th Aug

# Overview: Database Design

---

Data models:

- ER, Relational Data Model and their mapping

Relational Algebra:

- Be able to use relational algebra to answer question.

Database Languages:

- SQL (final exam: may need to write some SQL)

Relational Database Design:

- Functional Dependency
- Normal Forms
- Design Algorithms for 3NF and BCNF

# Data Models

---

ER, Relational Data Model and their mapping

Application  E-R  Relational DB

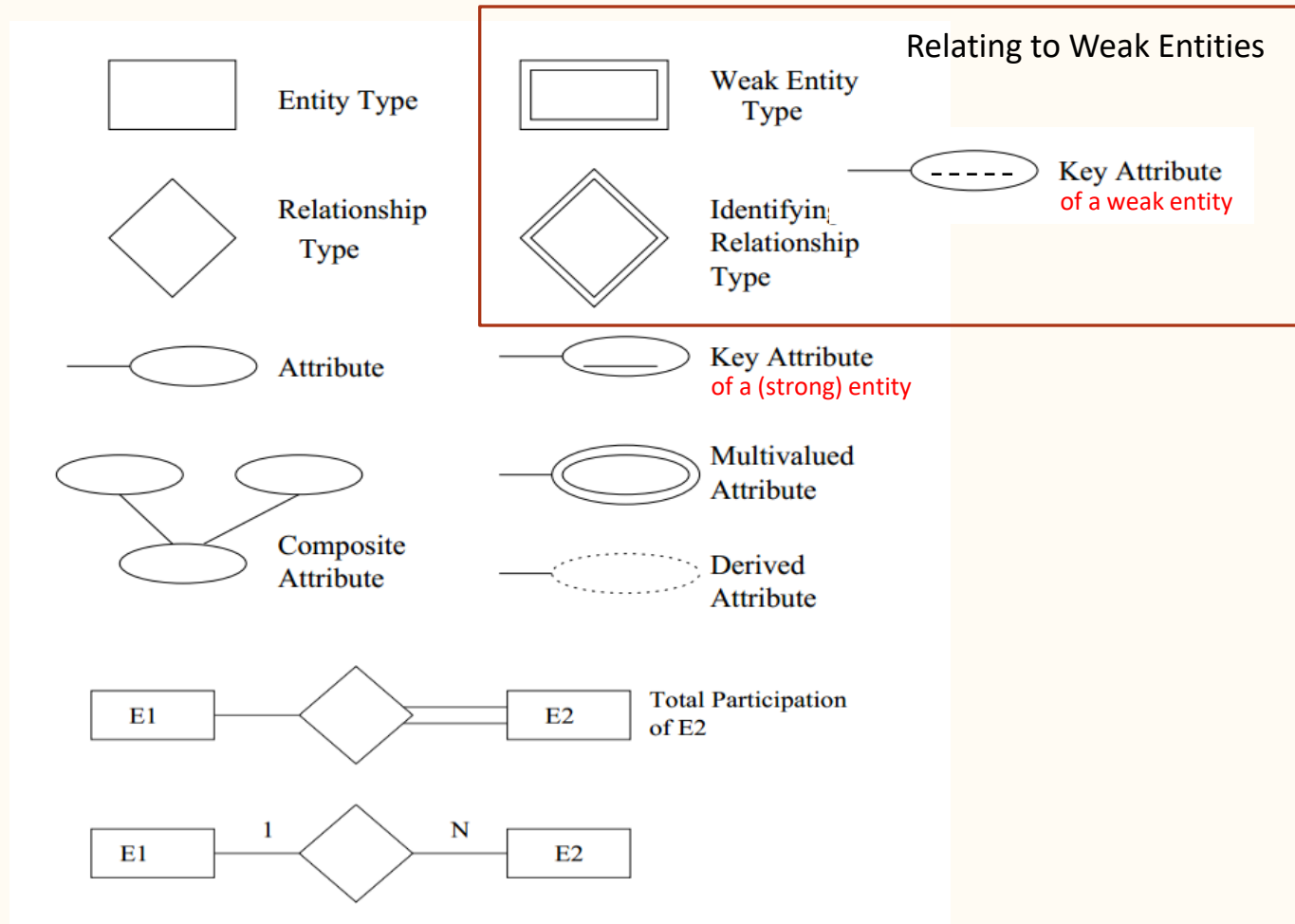
# Entity-Relationship Model(cont)

---

1. Entity type: Group of object with the same properties
2. Entity: member of an entity type - analogous to an object.
3. Attribute: a property of object
4. Relationship: among objects

# Notations (cont)

The notation used for ERDs is summarised in Elmasre/Navathe Figure 3.15.



# Relational Data Model

---

In the relational model, everything is described using relations.

A relation can be thought of as a named table.

Each column of the table corresponds to a named attribute.

The set of allowed values for an attribute is called its domain.

Each row of the table is called a tuple of the relation.

N.B. There is no ordering of column or rows.



# Keys

---

Keys are used to identify tuples in a relation.

A *superkey* is a set of attributes that uniquely determines a tuple.

Note that this is a property of the relation that does not depend on the current relation instance.

A *candidate key* is a superkey, none of whose *proper* subsets is a superkey.

Keys are determined by the applications.

# Integrity Constraints

---

There are several kinds of integrity constraints that are an integral part of the relational model:

1. *Key constraint*: candidate key values must be unique for every relation instance.
2. *Entity integrity*: an attribute that is part of a primary key cannot be NULL.
3. *Referential integrity*: The third kind has to do with “foreign keys”.

# Foreign Keys

---

Foreign keys are used to refer to a tuple in another relation.

A set, FK, of attributes from a relation schema R1 may be a foreign key if

- the attributes have the same domains as the attributes in the primary key of another relation schema R2, and
- a value of FK in a tuple t1 of R1 either occurs as a value of PK for some tuple t2 in R2 or is null.

Referential integrity: The value of FK must occur in the other relation or be entirely NULL.

# ER to Relational Model Mapping

---

One technique for database design is to first design a conceptual schema using a high-level data model, and then map it to a conceptual schema in the DBMS data model for the chosen DBMS.

Here we looked at a way to do this mapping from the ER to the relational data model. (see details in the lecture notes of Relational Data Model).

Composite and multivalued attributes are allowed in ER model, but not allowed in relational data model.

# Relational Algebra

---

*Relational Algebra* is a procedural data manipulation language (**DML**).

It specifies operations on relations to define new relations:

**Unary Relational Operations:** Select, Project

**Operations from Set Theory:** Union, Intersection, Difference,  
Cartesian Product

**Binary Relational Operations:** Join, Divide.

**Relational Algebra:** be able to use relational algebra to answer question.

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R	$\sigma_{\langle \text{selection condition} \rangle} (R)$
PROJECT	Produces a new relation with only some of the attributes of R and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle} (R)$
THETA-JOIN	Produces all combinations of tuples from R and S that satisfy the join condition.	$R \bowtie_{\langle \text{join condition} \rangle} S$
EQUI-JOIN	Produces all the combinations of tuples from R and S that satisfy a join condition with only equality comparisons.	$R \bowtie_{\langle \text{join condition} \rangle} S$
NATURAL-JOIN	Same as EQUIJOIN except that the join attributes of S are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R \bowtie_{\langle \text{join condition} \rangle} S$
UNION	Produces a relation that includes all the tuples in R or S or both R and S; R and S must be union compatible.	$R \cup S$
INTERAECTION	Produces a relation that includes all the tuples in both R and S; R and S must be union compatible.	$R \cap S$
DIFFERENCE	Produces a relation that includes all the tuples in R that are not in S; R and S must be union compatible.	$R - S$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R and S and includes as tuples all possible combinations of tuples from R and S.	$R \times S$
DIVISION	Produces a relation T(X) that includes all tuples t[X] in R(Z) that appear in R in combination with every tuple from S(Y), where $Z = X \cup Y$ .	$R(Z) \div S(Y)$

# Relational Algebra Notation:

---

- Condition should be in round brackets

- E.g., Select:

$$\sigma_{(Supervisor=1)}(ENROLMENT)$$

- E.g., Join:

$$ENROLMENT \bowtie_{(Supervisor=Person\#)} RESEARCHER$$

- For the attributes in projection operation, you may use { } or omit it

- E.g., the following are both accepted

- $\pi_{\{degree\}}(Course)$

- $\pi_{degree}(Course)$

# Database Languages

---

- ❖ Database Languages: SQL
- ❖ **final exam: may need to write some straightforward SQL**



# SQL Queries<sub>(cont)</sub>

---

Query syntax is:

SELECT attributes

FROM relations

WHERE condition

The result of this statement is a table, which is typically displayed on output.

The SELECT statement contains the functionality of *select*, *project* and *join* from the relational algebra.

# SQL Comparisons

---

Comparison operators are defined on all types.

<                      >                      <=                      >=                      =                      !=

Boolean operators AND, OR, NOT are available within WHERE expressions to combine results of comparisons.

Comparison against NULL yields FALSE.

Can explicitly test for NULL using:

- ***attr IS NULL***                      ***attr IS NOT NULL***

Most data types also have type-specific operations available (e.g., arithmetic for numbers).

# Relational Database Design

---

- ❖ Relational Database Design: Functional Dependency, Normal Forms, Design Algorithms for 3NF and BCNF

# Functional Dependencies

---

A function  $f$  from  $S_1$  to  $S_2$  has the property

$$\text{if } x, y \in S_1 \text{ and } x = y, \text{ then } f(x) = f(y) .$$

A generalization of keys to avoid design flaws violating the above rule.

Let  $X$  and  $Y$  be sets of attributes in  $R$ .

$X$  (*functionally*) determines  $Y$  ,  $X \rightarrow Y$  , iff  $t_1[X] = t_2[X]$  implies  $t_1[Y] = t_2[Y]$ .

$$\text{i.e., } f(t(X)) = t[Y]$$

We also say  $X \rightarrow Y$  is a *functional* dependency, and that  $Y$  is *functionally* dependent on  $X$ .

$X$  is called the *left side*,  $Y$  the *right side* of the dependency.

# Armstrong's Axioms (1974)

---

*Notation:* If  $X$  and  $Y$  are sets of attributes, we write  $XY$  for their union.

e.g.,  $X = \{A, B\}$ ,  $Y = \{B, C\}$ ,  $XY = \{A, B, C\}$

1. F1 (Reflexivity) If  $X \supseteq Y$  then  $X \rightarrow Y$ .
2. F2 (Augmentation)  $\{X \rightarrow Y\} \models XZ \rightarrow YZ$ .
3. F3 (Transitivity)  $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$ .
4. F4 (Additivity)  $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$ .
5. F5 (Projectivity)  $\{X \rightarrow YZ\} \models X \rightarrow Y$ .
6. F6 (Pseudo-transitivity)  $\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$ .

# Algorithm to Compute $X^+$

---

```
 $X^+ := X;$   
change := true;  
while change do  
  begin  
    change := false;  
    for each FD  $W \rightarrow Z$  in  $F$  do  
      begin  
        if  $(W \subseteq X^+) \text{ and } (Z \not\subseteq X^+)$  then do  
          begin  
             $X^+ := X^+ \cup Z;$   
            change := true;  
          end  
        end  
      end  
    end  
  end  
end
```

# Compute a Candidate Key

---

Given a relational schema  $R$  and a set  $F$  of functional dependencies on  $R$ .

A key  $X$  of  $R$  must have the property that  $X^+ = R$ .

## Algorithm

- Step 1: Assign  $X$  a superkey in  $F$ .
- Step 2: Iteratively remove attributes from  $X$  while retaining the property  $X^+ = R$ .
- The remaining  $X$  is a key.

# Compute all Candidate Keys

---

Given a relational schema  $R$  and a set  $F$  of functional dependencies on  $R$ , the algorithm to compute all the candidate keys is as follows:

$T := \emptyset$

*Main:*

$X := S$  where  $S$  is a super key which does not contain any candidate key in  $T$

remove := true

While remove do

    For each attribute  $A \in X$

        Compute  $\{X-A\}^+$  with respect to  $F$

        If  $\{X-A\}^+$  contains all attributes of  $R$  then

$X := X - \{A\}$

        Else

            remove := false

$T := T \cup X$

Repeat *Main* until no available  $S$  can be found. Finally,  $T$  contains all the candidate keys.



# Normal Forms

---

Normal Forms for relational databases:

- 1NF, 2NF, 3NF (Codd 1972)
- Boyce-Codd NF (1974)

# First Normal Form (1NF)

---

This simply means that attribute values are *atomic* and is part of the definition of the relational model.

Atomic: multivalued attributes, composite attributes, and their combinations are disallowed.

There is currently a lot of interests in non-first normal form databases, particularly those where an attribute value can be a table (nested relations).

# Second Normal Form (2NF)

---

A *prime* attribute is one that is part of a candidate key. Other attributes are *non-prime*.

**Definition:** In an FD  $X \rightarrow Y$ ,  $Y$  is *fully functionally dependent* on  $X$  if there is no  $Z \subset X$  such that  $Z \rightarrow Y$ . Otherwise,  $Y$  is *partially* dependent on  $X$ .

**Definition** (*Second Normal Form*): A relation scheme is in second normal form (2NF) if all non-prime attributes are fully functionally dependent on the relation keys.

A database scheme is in 2NF if all its relations are in 2NF.

# Third Normal Form (3NF)

---

**Definition** (*Third Normal Form*): A relation scheme is in *third normal form (3NF)* if for all non-trivial FD's of the form  $X \rightarrow A$  that hold, either  $X$  is a superkey or  $A$  is a prime attribute.

Note: a FD  $X \rightarrow Y$  is trivial iff  $Y$  is a subset of  $X$ .

*Alternative definition*: A relation scheme is in third normal form if every non-prime attribute is fully functionally dependent on the keys and not transitively dependent on any key.

A database scheme is in 3NF if all its relations are in 3NF.

# Boyce-codd Normal Form (BCNF)

---

**Definition** (*Boyce-codd Normal Form*):

A relation scheme is in *Boyce-codd* Normal Form (BCNF) if whenever  $X \rightarrow A$  holds (and  $X \rightarrow A$  is non-trivial),  $X$  is a superkey.

A database scheme is in BCNF if all its relations are in BCNF.

# On Relational Database Design

---

1. Anomalies can be removed from relation designs by decomposing them until they are in a normal form.
2. A decomposition of a relation scheme,  $R$ , is a set of relation schemes  $\{R_1, \dots, R_n\}$  such that  $R_i \subseteq R$  for each  $i$ , and  $\bigcup_{i=1}^n R_i = R$
3. In a decomposition  $\{R_1, \dots, R_n\}$ , the intersect of each pair of  $R_i$  and  $R_j$  does not have to be empty.
  - Example:  $R = \{A, B, C, D, E\}$ ,  $R_1 = \{A, B\}$ ,  $R_2 = \{A, C\}$ ,  $R_3 = \{C, D, E\}$
4. A naive decomposition: each relation has only attribute.

# Dependency Preserving

---

**A good decomposition should have the following property.**

Recall : Two sets  $F$  and  $G$  of FD's are equivalent if  $F^+ = G^+$ .

Given a decomposition  $\{R_1, \dots, R_n\}$  of  $R$ :

$$F_i = \{X \rightarrow Y : X \rightarrow Y \in F \text{ \& } X \in R_i, Y \in R_i\}.$$

The decomposition  $\{R_1, \dots, R_n\}$  of  $R$  is dependency preserving with respect to  $F$  if

$$F^+ = \left( \bigcup_{i=1}^n F_i \right)^+.$$

**Note: We will not ask intensive questions that ask you whether all the dependencies are preserved.**

# Lossless-join Decomposition

---

**A good decomposition should have the following property.**

A decomposition  $\{R_1, \dots, R_n\}$  of  $R$  is a *lossless join* decomposition with respect to a set  $F$  of FD's if for every relation instance  $r$  that satisfies  $F$ :

$$r = \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_n}(r).$$

If  $r \subset \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_n}(r)$ , the decomposition is *lossy*.



# Lossless Decomposition into BCNF

---

## Algorithm TO\_BCNF

$D := \{R_1, R_2, \dots, R_n\}$

**While**  $\exists$  a  $R_i \in D$  and  $R_i$  is not in BCNF **Do**

- { find a  $X \rightarrow Y$  in  $R_i$  that violates BCNF; replace  $R_i$  in  $D$  by  $(R_i - Y)$  and  $(X \cup Y)$ ; }

# Computing a Minimum cover

---

A set  $F$  of FD's is **minimal** if

1. Every FD  $X \rightarrow Y$  in  $F$  is simple:  $Y$  consists of a single attribute,
2. Every FD  $X \rightarrow A$  in  $F$  is *left-reduced*: there is no proper subset  $Y \subset X$  such that  $X \rightarrow A$  can be replaced with  $Y \rightarrow A$ .

that is, there is no  $Y \subset X$  such that

$$((F - \{X \rightarrow A\}) \cup \{Y \rightarrow A\})^+ = F^+$$

→ i.e., iff  $F \models Y \rightarrow A$

3. No FD in  $F$  can be removed; that is, there is no FD  $X \rightarrow A$  in  $F$  such that

$$(F - \{X \rightarrow A\})^+ = F^+$$

→ i.e., iff  $X \rightarrow A$  is inferred from  $F - \{X \rightarrow A\}$

# Computing a Minimum cover

---

A *minimal cover* (or *canonical cover*) for F:

- a minimal set of FD's  $F_{min}$  such that  $F^+ = F_{min}^+$ .

## Algorithm Min\_Cover

Input: a set F of functional dependencies.

Output: a minimum cover of F.

Step 1: *Reduce right side.*

Apply Algorithm Reduce\_right to F.

Step 2: *Reduce left side.*

Apply Algorithm Reduce\_left to the output of Step 2.

Step 3: *Remove redundant FDs.*

Apply Algorithm Remove\_redundancy to the output of Step 2.

The output is a minimum cover.

Next we detail the three algorithms (Reduce\_right, Reduce\_left, Reduce\_redundancy) .

# Computing a Minimum cover (cont)

---

## 1. Algorithm Reduce\_right

*INPUT:  $F$ .*

*OUTPUT: right side reduced  $F'$ .*

1. For each FD  $X \rightarrow Y \in F$  where  $Y = \{A_1, A_2, \dots, A_k\}$ , we use all  $X \rightarrow \{A_i\}$  (for  $1 \leq i \leq k$ ) to replace  $X \rightarrow Y$ .

## 2. Algorithm Reduce\_left

*INPUT: right side reduced  $F$ .*

*OUTPUT: right and left side reduced  $F'$ .*

1. For each  $X \rightarrow \{A\} \in F$  where  $X = \{A_i : 1 \leq i \leq k\}$ , do the following.  
For  $i = 1$  to  $k$ , replace  $X$  with  $X - \{A_i\}$  if  $A \in (X - \{A_i\})^+$ .

## 3. Algorithm Reduce\_redundancy

*INPUT: right and left side reduced  $F$ .*

*OUTPUT: a minimum cover  $F'$  of  $F$ .*

1. For each FD  $X \rightarrow \{A\} \in F$ , remove it from  $F$  if:  $A \in X^+$  with respect to  $F - \{X \rightarrow \{A\}\}$ .

# Decomposition into 3NF

---

A **lossless and dependency-preserving decomposition** into 3NF is always possible.

Algorithm 3NF decomposition (Lossless and Dependency-preserving)

1. Find a minimal cover  $G$  for  $F$ .
2. For each left-hand-side  $X$  of a functional dependency that appears in  $G$ , create a relation schema in  $D$  with attributes  $\{X \cup \{A_1\} \cup \{A_2\} \dots \cup \{A_k\}\}$ , where  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_k$  are the only dependencies in  $G$  with  $X$  as left-hand-side ( $X$  is the key to this relation).
3. If none of the relation schemas in  $D$  contains a key of  $R$ , then create one more relation schema in  $D$  that contains attributes that form a key of  $R$ .
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation  $R$  is considered redundant if  $R$  is a projection of another relation  $S$  in the schema; alternately,  $R$  is subsumed by  $S$ .

# Overview: DBMS

---

- ❖ Disk, Buffer Replacement Policy
- ❖ Transaction Management
  - ACID properties
  - Various schedules: Serializable, Conflict-Serializable, Schedule Graph, Wait for Graph, ...
  - Concurrency control (locking, locking with time-stamps).

# Buffer Replacement Policies

---

## 1. Least Recently Used (LRU)

- release the frame that has not been used for the longest period.
- intuitively appealing idea but can perform badly

## 2. First in First Out (FIFO)

## 3. Most Recently Used (MRU):

- release the frame used most recently

No one is guaranteed to be better than the others. Quite dependent on applications.

# Desirable Properties of Transaction Processing

---

## **ACID Properties**

### **Atomicity:**

A transaction is either performed in its entirety or not performed at all.

### **Consistency:**

A correct execution of the transaction must take the database from one consistent state to another.

### **Isolation:**

A transaction should not make its updates visible to other transactions until it is committed.

### **Durability/ Permanency:**

Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.



# Check Conflict Serializability

---

- **Algorithm**

Step 1: Construct a *schedule* (or *precedence*) graph – a *directed graph*.

Step 2: Check if the graph is *cyclic*:

- Cyclic: non-serializable.
- Acyclic: serializable.

# Construct a Schedule Graph

---

Schedule Graph  $GS = (V, A)$  for a schedule  $S$

A vertex in  $V$  represents a transaction.

For two vertices  $T_i$  and  $T_j$ , an arc  $T_i \rightarrow T_j$  is added to  $A$  if

- there are two *conflicting* operations  $O_1 \in T_i$  and  $O_2 \in T_j$ ,
- in  $S$ ,  $O_1$  is before  $O_2$ .

Recall: two operations  $O_1$  and  $O_2$  are *conflicting* if

- they are in different transactions but on the same data item,
- one of them must be a write.

# Locking Rules

---

In this schema, every transaction  $T$  must obey the following rules.

1) If  $T$  has **only one** operation (read/write) manipulating an item  $X$ :

- obtain a read lock on  $X$  before reading it,
- obtain a write lock on  $X$  before writing it,
- unlock  $X$  when done with it.

2) If  $T$  has **several** operations manipulating  $X$ :

- obtain one proper lock only on  $X$ :
  - a read lock if all operations on  $X$  are reads;
  - a write lock if one of these operations on  $X$  is a write.
- unlock  $X$  after the last operation on  $X$  in  $T$  has been executed.

# Advanced Topic – Graph Data Analytics

---

- K-core: definition and computation
- Given a graph  $G$ , the  $k$ -core of  $G$  is a subgraph where **each node** has at least  $k$  neighbors (i.e.,  $k$  adjacent nodes, or a degree of  $k$ ).
- **Computation** of  $k$ -core: Given a graph  $G$ , the  $k$ -core of  $G$  can be computed by recursively deleting every node and its adjacent edges if its degree is less than  $k$ . Until the edges are

# Final Exam

---

- ❖ 3 hrs
- ❖ **If you do not feel well on the exam day, please not attend the exam. By sitting or submitting an exam or assessment on the scheduled assessment date, you are declaring that you are fit to do so and cannot later apply for Special Consideration. (I.e., no sup-exam will be given.)**

# Question Types:

---

- ❖ Sample exam paper. Please note that sample questions just reflect some of the question types to expect but not the scope nor the similarity.
- ❖ Final exam will not have Multiple Choice Questions
  - ❖ Instead, Question 1 will be replaced with a small number of short questions that test your general understanding of concepts.
    - ❖ e.g., give an example of... explain the difference between...
  - ❖ Remaining Question Types are similar

# Note 1:

---

## Computer Updates

You must ensure that auto-updates are **disabled** on your computer prior to the online assessment.

Special consideration will not be awarded on the grounds that your computer performed an update during an online assessment.

# Note 2:

---

**If you accidentally upload the wrong document or wrong version of your exam**

Students are responsible for uploading the correct version of the correct document. Once uploaded, there will be no opportunity to replace or re-upload your exam papers AFTER the end of the exam.

The documents submitted will be the documents that are marked. There is no provision for students who upload incorrect or incomplete documents.

Therefore, you must check the work before you submit.



# Note 3:

---

## **Communication during the exam**

Students are not permitted to communicate with other people during the exam (including the reading and submission periods).

Attempts to communicate with other students will be considered to be serious academic misconduct.

This includes communication in person, by email, text, message, telephone, or internet.

I.e., do the work yourself

# Note 4:

---

## **Sharing answers with others or posting them online**

Any attempts to collaborate or share your answers with others will be considered a very serious case of academic misconduct

# Note 5 (Checklist):

---

1. Be logged in at your computer and ready to go 20 minutes before the exam commences.
2. Ensure your device has power, and the charger is plugged in.
3. If applicable remind your roommates or family that you'll be taking an exam to avoid interruptions.

# Note 6:

---

If you experience other issues, take a lot of timestamped screenshots  
All screenshots must include the date and time the issue occurred.

Contact the Course Coordinator or Tutor via email; Moodle or chat to  
advise you are experiencing a technical issue, as soon as possible.

---

Thank you and all the best!

# My Experience Survey

---

The UNSW My Experience survey still open for 22T1, participation is highly encouraged.

*“Please participate in the my Experience Survey and take the opportunity to share your constructive thoughts on your 2022 learning experience.*

*Your contributions help your teachers and shape the future of education at UNSW.”*

More information: <https://www.student.unsw.edu.au/myexperience>