COMP9120 Database Management Systems

UNIVERSITY OF SYDNEY · POSTGRADUATE COURSEWORK

Full guide + ask Sia about COMP9120

SIDE 1/2 CLOSED-BOOK MEMORY SHEET · definitions → rules → formulae → decisions **FINAL EXAM · 50%** Sem 2 2026 · **what must be in your head**

0 · Exam Map

COURSE.JSON + BRIEF

Assignm | Quiz 24% | Assignm | Final Exam 50%

ASSESSMENT	WEIGHT	FORMAT / FOCUS
Assignment 1 — Conceptual design & schema	13%	group · ER → SQL schema
Quiz	24%	in-class secure · 90 min · closed book
Assignment 2 — Database application development	13%	group · client + back end
Final Exam	**50%**	paper-based · 2 h · formal period

"This is a CLOSED book exam - no other material permitted." *COMP9120 Lecture 13 UoS review slides, 2026 (PDF_L13_UoS_Review)*

★ **Hurdle:** ≥40% in the final *and* ≥50% overall; otherwise the final mark is capped at 45. Use this page for **active recall**: cover each box, recite it, then reproduce the method from a blank page.

1 · Orient the Problem

CH 1

IF THE PROMPT ASKS…	THINK…	PRODUCE…
domain picture	conceptual	ER/EER diagram
tables + keys	logical	relations, PKs, FKs
remove anomalies	refinement	closures + decomposition
fast access	physical	files/indexes + I/Os

Reqs → ER → Rel. schema → Norm → Phys. → App + security

conceptual → logical → refine → physical

Design pipeline: identify the stage before choosing the method. *Ch 1, Fig design pipeline*

OLTP = many short operational reads/writes on a normalised schema. *OLAP* = long, read-heavy aggregates over history, often denormalised. **Physical independence**: change storage/indexes without changing the conceptual schema. **Logical independence**: change the conceptual schema while external views/apps survive.

THREE LEVELS + DBMS SERVICES

External = user views; **conceptual** = one community-wide logical schema; **internal** = physical files and indexes. A DBMS bundles persistence and sharing with a query language, integrity, concurrency control, recovery and data independence. Ask which level changed before naming the independence.

2 · ER/EER: Read Every Mark

CH 2–3

Entity = object set; *attribute* = property; *relationship degree* = participating entity sets. A relationship attribute belongs to the *pairing*, not either entity.

Arrow/key constraint = at most one relationship instance on that edge. **Double/thick line** = total participation (at least one).

WEAK ENTITY

```
no own key · identifying owner · total participation
PK = owner PK + discriminator
```

ISA

```
F ⊆ E and attrs(F) ⊇ attrs(E)
subclass inherits superclass attributes + relationships
```

SUPERKEYS · any unique set

CANDIDATE KEYS · minimal

PRIMARY KEY one chosen candidate

Key sets nest; a candidate key is minimal, not merely unique. *Ch 3, Fig 3.1*

ER CONSTRUCT	RELATIONAL MAPPING	KEY RULE
strong entity	own relation	entity key → PK
weak entity	relation + owner PK	owner PK + discriminator
1:N	post 1-side PK on N side	FK on N; NOT NULL if total
N:M / n-ary	new relation + all FKs	combined FKs usually PK
relationship attrs	with FK/new relationship row	follow its relationship
IsA subclass	own relation	PK also FK → superclass

2b · Mapping Algorithm

EXAM METHOD

1. Map **strong entities first**; underline each PK.
2. Map weak entities: copy owner PK, add discriminator; mark the copied column as both PK-part and FK.
3. Read each cardinality. For 1:N add one FK to N; for N:M create a relation.
4. Place relationship attributes where the relationship is represented.
5. Map IsA; then draw every FK arrow and state nullability from participation.

RECALL *"Many receives one; many-to-many becomes one." Never invent a weak-entity key, omit a composite key, or leave PK/FK status implicit.*

3 · Relational Algebra

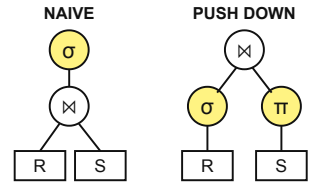
CH 4

OPERATOR	EFFECT / CONDITION	EXAM BITE
$\sigma_c(R)$	rows satisfying c	schema unchanged
$\pi_A(R)$	keep A columns	**duplicates removed**
R × S	all tuple pairs	|R|·|S| rows
R ∪ S	either set	union-compatible: same arity + matching domains
R − S	in R, not S; directional	
R ∩ S	in both; derived	
ρ	rename relation/attrs	resolve ambiguity
R ⋈ S	join; derived from × + σ	natural uses all same names

SQL ↔ ALGEBRA

```
SELECT ↔ π · FROM ↔ × / ⋈ · WHERE ↔ σ
```

"Never / none" pattern: begin with all wanted X, subtract X that did Y. A not-equal selection is wrong when an X has both a Y and a non-Y row. Project both operands to the same schema before difference.



Equivalent trees, different cost: push legal σ and π below joins. *Ch 4/12, Fig 4.1/12.1*

3b · Rewrite Safely

CH 4 + 12

- **Selection cascade:** $\sigma_{c\wedge d}(R) = \sigma_c(\sigma_d(R))$.
- **Join:** commutative and associative; reorder to keep intermediates small.
- Push σ below a join only if its predicate uses attributes of that input.
- Push π early but retain join keys and every attribute needed above.

ANSWER FORM *Write the expression, label intermediate schemas, then show the cheaper equivalent. The same result is not enough: name why fewer/narrower tuples reach the join.*

3c · Join + Set Checks

CH 4–5

Theta join = $\sigma_{condition}(R\times S)$. An **equi-join** uses equality; a **natural join** equates *all* same-named attributes and keeps one copy of each. With no common names, natural join degenerates to a product. Before writing ∪, ∩ or −, state union compatibility: same number of attributes and matching domains in corresponding positions.

4 · SQL Execution Order

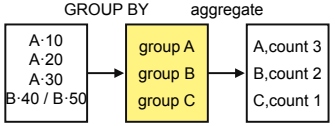
CH 5–6

LOGICAL EVALUATION

```
FROM / JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY
```

NEED	USE	TRAP
matching rows only	INNER JOIN … ON	missing ON → product
unmatched rows too	LEFT/RIGHT/FULL OUTER	NULL padding matters
remove result repeats	SELECT DISTINCT	SQL is a bag by default
combine results	UNION / INTERSECT / EXCEPT	union-compatible; duplicates removed
keep multiplicities	… ALL	m+n / min(m,n) / max(0,m−n)
pattern	LIKE; % many, _ one	strings single-quoted

GROUP rule: every SELECT/HAVING column must be aggregated or named in GROUP BY. WHERE filters *rows* before grouping; HAVING filters *groups* after aggregation. ORDER BY is last and may use a SELECT alias.



Rows → groups → one aggregate row per group; HAVING then filters groups. *Ch 6, Fig 6.1*

4b · NULL + Subqueries

CH 6

NULL is unknown/not applicable, not zero or blank. Arithmetic with NULL → NULL; comparison with NULL → **unknown**; WHERE keeps only true, so unknown is dropped. Test with `IS NULL`, never `= NULL`.

```
COUNT(*) counts rows · COUNT(col), SUM, AVG, MIN, MAX ignore NULL
all-NULL input: COUNT = 0; other aggregates = NULL
```

PREDICATE	MEANING	RECALL
EXISTS(Q)	Q returns ≥1 row	tests emptiness
v op ANY(Q)	true for at least one	SOME = ANY
v op ALL(Q)	true for every value	≥ALL highest; ≤ALL cheapest; ties stay
NOT EXISTS	no matching row	NULL-safe anti-match
NOT IN	not equal to every item	NULL in Q can yield no rows

5 · Integrity: Pick the Enforcer

CH 7

FAMILY	USE	KEY DISTINCTION
domain	type, DEFAULT, NOT NULL, CHECK	CHECK alone still lets NULL through
key	PRIMARY KEY / UNIQUE	PK: one, NOT NULL; UNIQUE: alternatives
referential	FOREIGN KEY	child value matches parent key or is NULL
semantic	CHECK / assertion / trigger	business rule
dynamic	trigger	Event–Condition–Action

Parent delete/update actions: `NO ACTION` reject (default); `CASCADE` propagate; `SET NULL`; `SET DEFAULT`. They are declared on the child FK but fire when the parent changes.

Timing: NOT DEFERRABLE checks immediately; DEFERRABLE checks at COMMIT. An *assertion* spans rows/tables; PostgreSQL does not support CREATE ASSERTION, so use a trigger. Rewrite "for all" as `NOT EXISTS` a violating row: ∀x P(x) ≡ ¬∃x ¬P(x).

EVENT INS/UPD → WHEN? test → ACTION reject/fix

BEFORE check · AFTER propagate

Trigger ECA: row-level fires once per row; statement-level once per statement. *Ch 7, trigger figure*

6 · Closures + Normal Forms

CH 9

FD X→Y: tuples agreeing on X must agree on Y. Semantics decides it; one instance can disprove, never prove. Armstrong: **reflexivity** Y⊆X ⇒ X→Y; **augmentation** X→Y ⇒ XZ→YZ; **transitivity** X→Y, Y→Z ⇒ X→Z; derived union/decomposition.

CLOSURE ALGORITHM

```
X⁺ ← X; repeatedly add Y for every U→Y with U⊆X⁺; stop at no change.
X superkey ⇔ X⁺ = all attrs · candidate key = minimal superkey
```

Any attribute never on an FD right-hand side belongs to every candidate key. Seed with those attributes; close; add until all attributes appear; then test minimality by removing each member.

5b · DDL Decision Routine

CH 7

1. For every column: domain/type, DEFAULT, NULL status and local range CHECK.
2. Choose exactly one PRIMARY KEY; declare alternate candidate keys UNIQUE.
3. Add composite FOREIGN KEY clauses with matching parent key order.
4. Choose parent-change action from semantics; default is rejection, not cascade.

6b · Normalise + Prove Quality

CH 9

1NF · atomic values

2NF · no partial FD to non-prime

3NF

X key OR RHS prime

BCNF: X key

Normal forms nest; 4NF is stronger still for multivalued dependencies. *Ch 9, Fig 9.1*

FORM	TEST FOR EVERY RELEVANT DEPENDENCY	FAILURE CUE
1NF	atomic domains	repeating/multivalued field
2NF	no proper subset of candidate key → non-prime	partial dependency
3NF	non-trivial X→A: X superkey OR A prime	transitive non-key determinant
BCNF	non-trivial X→Y: X superkey	every arrow must leave a key
4NF	non-trivial X↠Y: X superkey	independent multivalued facts

BCNF SPLIT ON VIOLATING X→Y

```
R₁ = X ∪ Y
R₂ = R - (Y - X)
```

1. Find a non-trivial FD whose determinant is not a superkey.
2. Split using the formula; project relevant FDs into each component.
3. Repeat until every determinant is a key of its component.

Lossless: for binary R_1,R_2, shared attrs $R_1\cap R_2$ must determine all of R_1 or all of R_2. **Dependency preserving:** dependencies projected into components together imply the original set, so each can be checked without a join. BCNF can sacrifice preservation; 3NF can preserve.

EXAM SEQUENCE *forced key attrs → closure → minimality → highest form → one violating FD per split → lossless check → preservation check. Name the FD at each step.*

7 · Transactions + Cost Belt

CH 10–12

A all/none | C valid→valid | I serial-like | D crash-safe

recovery A,D · concurrency I · developer C

ACID guarantees and primary responsibility. *Ch 10, Fig 10.1*

Conflict iff same item + different transactions + at least one write. Add Ti→Tj when Ti's conflicting op occurs first. Acyclic precedence graph ⇔ conflict-serializable; topological sort gives equivalent serial order.

Semester 2 formal examination period, from 16 November 2026 (the Exams Office publishes the exact date).

Your exam is closed book: nothing printed goes in. Use this sheet to revise and self-test, not to carry.

COMP9120 Self-Test Drills · Database Management Systems

UNIVERSITY OF SYDNEY · ANSWER PATH → FINAL ANSWER → TRAP

Full guide + ask Sia about COMP9120

SIDE 2/2 · SELF-TEST · 5 worked drills · judgement cards · classic traps · COVER → SOLVE → CHECK · Write the method even when arithmetic stalls

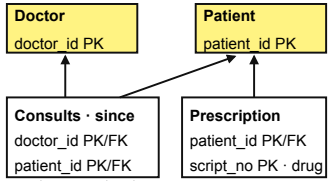
W1 · Clinic ER → Relations

CH 3 WORKED EXAMPLE

Prompt. Doctor(doctor_id,name,speciality), Patient(patient_id,name,dob); N:M Consults has *since*; weak Prescription is owned by Patient, discriminator *script_no*, attribute *drug*.

ANSWER PATH

1. Strong entities first: one relation each, own key.
2. Weak Prescription copies Patient key; combine it with script_no.
3. N:M Consults becomes a relation; carry both FKs and the relationship attribute.
4. Underline PKs; mark every FK arrow.



Four relations; weak and N:M structures create composite keys. *Ch 3 worked clinic scenario*

FINAL

```
Doctor(doctor_id, name, speciality)
Patient(patient_id, name, dob)
Prescription(patient_id, script_no,
drug), patient_id→Patient
Consults(doctor_id, patient_id,
since), FKs→Doctor/Patient
```

TRAP *A 1:N would add only one FK on N. N:M needs a new relation. A weak entity never receives a made-up key.*

Judge Fast: ER

IF X → DO Y

IF YOU SEE…	DO…
"each A has at most one B"	arrow/key constraint toward B
"every A participates"	total/double line on A
relationship has attribute	store with representation of relationship
owner-dependent identity	weak entity; owner key + discriminator
subclass	shared PK that is also superclass FK

Final check: each relation has exactly one PK; every copied owner/participant key is marked FK; relationship attributes stay with the relationship row. A FK references a candidate key, and each non-NULL value must match a parent row. For 1:N total participation on N, make its FK NOT NULL. Draw every reference arrow.

W2 · "Never Indie" + Grouped NULL

CH 4 + 6 WORKED

A · RELATIONAL ALGEBRA

Data. Artist = {(1,Mia),(2,Leo),(3,Ada),(4,Sam)}. Indie Track aids are {1,1,4}. Find artist names that never released on Indie.

1. All names: $\pi_{aname}(\text{Artist}) = \{\text{Mia, Leo, Ada, Sam}\}$.
2. Indie rows: $\sigma_{label='Indie'}(\text{Track})$.
3. Names that did: $\pi_{aname}(\text{Artist} \bowtie \sigma_{label='Indie'}(\text{Track})) = \{\text{Mia, Sam}\}$.
4. Subtract did from all → {Leo, Ada}.

$$\pi_{aname}(\text{Artist}) - \pi_{aname}(\text{Artist} \bowtie \sigma_{label='Indie'}(\text{Track}))$$

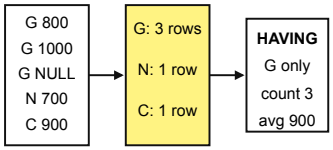
Trap: $\sigma_{label\neq'Indie'}(\text{Track})$ wrongly includes Mia because she also has a Major track. "Never Y" is set difference or NOT EXISTS, not merely ≠Y.

B · GROUPED SQL WITH HIDDEN NULL

After `WHERE bedrooms ⩾ 2`, rows 1,2,3,5,6 remain. Groups: Glebe {1,2,3}; Newtown {5}; Camperdown {6}. `HAVING COUNT(*) ⩾ 2` keeps only Glebe. COUNT(*) = 3; AVG(price) uses {800,1000}, ignores NULL, so 1800/2 = 900.

FINAL ROW

```
(Glebe, 3, 900)
```



WHERE keeps rows; GROUP BY partitions; HAVING keeps groups; AVG skips NULL. *Ch 6 worked example*

SQL Trap Deck

FAST RECALL

IF YOU SEE…	DO / SAY…
unmatched parents required	LEFT OUTER JOIN from parent
"highest", ties retained	value ≥ ALL(subquery)
anti-match + possible NULL	NOT EXISTS, not NOT IN
group condition	HAVING, never WHERE
joined fan-out	ask whether DISTINCT is required

Write the SQL Skeleton

FROM THE WORKED DATA

```
SELECT suburb, COUNT(*) AS n,
AVG(price) AS avg_price
FROM Listing
WHERE bedrooms ⩾ 2
GROUP BY suburb
HAVING COUNT(*) ⩾ 2;
```

W3 · Key, Highest NF, BCNF Split

CH 9 WORKED

Enrol(StudentID, UnitCode, UnitName, Tutor, TutorRoom). FDs: (StudentID,UnitCode)→Tutor; UnitCode→UnitName; Tutor→TutorRoom.

1 · KEY BY CLOSURE

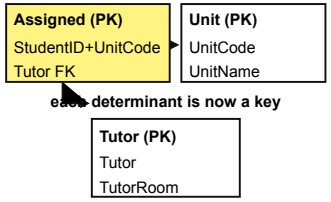
StudentID and UnitCode never appear on an RHS, so both are forced. Starting {StudentID,UnitCode}: add Tutor; from UnitCode add UnitName; from Tutor add TutorRoom. Closure = all five. StudentID⁺={StudentID}; UnitCode⁺={UnitCode,UnitName}; neither alone is a superkey. Therefore candidate key = (StudentID, UnitCode).

2 · HIGHEST NORMAL FORM

UnitCode is a proper part of the composite key and determines non-prime UnitName: a **partial dependency**. Therefore relation is 1NF only, not 2NF.

3 · DECOMPOSE

1. UnitCode→UnitName gives Unit(UnitCode,UnitName).
2. Tutor→TutorRoom gives Tutor(Tutor,TutorRoom).
3. Remainder Assigned(StudentID,UnitCode,Tutor), key (StudentID,UnitCode).



Lossless BCNF decomposition; each original FD remains inside one table. *Ch 9 worked example / Fig 9.2*

Quality. Shared UnitCode keys Unit; shared Tutor keys Tutor ⇒ lossless. Every original FD is enforceable inside a component ⇒ dependency preserving.

TRAP *A superkey with extra attributes is not a candidate key. Always write the removal/minimality check.*

Normalisation Judgement Cards

IF X → DO Y

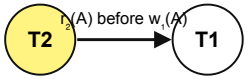
EVIDENCE	CONCLUSION
proper key subset → non-prime	fails 2NF
non-key determinant → non-prime	fails 3NF and BCNF
non-key determinant → prime	may pass 3NF; fails BCNF
intersection keys one component	binary split is lossless
an FD needs a join to check	not dependency preserving

W4 · Conflict Graph + Locking

CH 10 WORKED

Schedule: $r_1(A)\ r_2(A)\ w_1(A)\ r_2(B)\ w_2(B)$.

1. On A, $r_2(A)$ precedes $w_1(A)$: same item, different transactions, one write ⇒ conflict ⇒ edge T2→T1.
2. $r_1(A)$ with $r_2(A)$ is read/read ⇒ no conflict.
3. B is touched only by T2 ⇒ no cross-transaction conflict.
4. Graph has one edge and no cycle ⇒ conflict-serializable.
5. Topological order: T2, then T1.

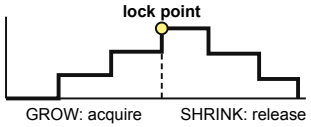


acyclic ⇒ serial order T2 → T1

List conflict and direction before drawing the graph. *Ch 10 worked example*

LOCKS

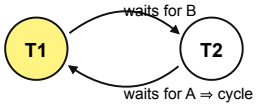
S/S compatible; every pair involving X is incompatible. 2PL: growing phase acquires only, shrinking phase releases only; after first release, never acquire again. Strict 2PL holds every lock until COMMIT/ROLLBACK, preventing dirty reads and cascading aborts, while deadlock remains possible.



One peak: no re-acquire after release; strict 2PL delays the drop to commit. *Ch 10, Fig 10.3*

Deadlock + Anomalies

CH 10



Waits-for cycle = deadlock; prevent it or detect and abort a victim. *Ch 10, Fig 10.4*

PATTERN	NAME / RESPONSE
two writers overwrite	lost update
read uncommitted value	dirty read / temporary update
aggregate while rows change	incorrect summary
circular lock wait	deadlock; prevention or detect + abort

Serial has no interleaving. **Serializable** may interleave but equals a serial order. 2PL guarantees conflict-serializability; strict 2PL stops dirty reads and cascading aborts.

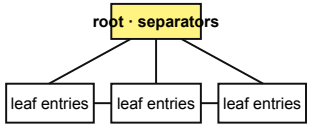
W5 · Storage + Join Cost

CH 11–12 WORKED

A · DATA + DENSE B+ TREE

Given: 48,000 Booking records; record 160 B; page 4096 B with 96 B header ⇒ 4000 B usable; 75% fill; key 8 B + rowid 4 B.

1. Capacity = ⌊4000/160⌋ = 25 records/page.
2. Effective capacity = 25×0.75 = 18.75; data pages = ⌈48,000/18.75⌉ = **2,560**.
3. Entry = 8+4 = 12 B; raw entries/page = ⌊4000/12⌋ = 333; fan-out = 333×0.75 = 249.75.
4. Dense leaves = ⌈48,000/249.75⌉ = **193**; parent = ⌈193/249.75⌉ = **1 root**.
5. Index = 194 pages; total = 2,754 pages. Equality lookup = 2 index levels + 1 record page = **3 I/Os**.



linked leaves make range scans cheap

B+ tree: separators route; all data entries live in linked leaves. *Ch 11, Fig 11.4*

B · COST THE JOIN BOTH WAYS

Order: |R|=5,000, bR=500. Item: |S|=40,000, bS=1,000.

PLAN	SUBSTITUTION	I/OS
tuple, R outer	500+5,000·1,000	5,000,500
tuple, S outer	1,000+40,000·500	20,001,000
block, R outer	500+500·1,000	**500,500**
block, S outer	1,000+1,000·500	501,000

Winner: block NLJ, Order outer, 500,500 I/Os. State the chosen outer, not just the number.

Index Decision Table

CH 11

NEED	PREFER	WHY
equality only	hash or B+ tree	both locate key
range / ordering	B+ tree	sorted linked leaves
many range matches	clustered	consecutive data pages
unclustered range	compare with scan	up to one I/O/match
sparse index	sorted/clustered data	one entry per page

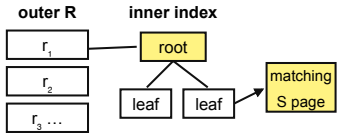
Physical-Layer Checks

CH 11

Heap: cheap insertion; unique equality averages about half-file scan; range scans all pages. **Sorted file:** binary search helps lookups, but insertion pays to preserve order. **Clustered index:** index order equals data order, so matching ranges occupy consecutive pages; at most one per table. **Unclustered:** matches scatter and may cost one record-page I/O each.

Cost & Method Chooser

CH 11–12

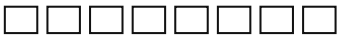


per outer row: index probe → fetch

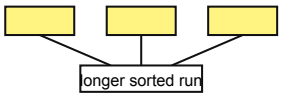
Index NLJ needs an equi/natural join and an index on the inner join attribute. *Ch 12, Fig 12.2*

QUESTION CUE	RESPONSE
"cheapest join"	write formulas; try both outers; include index NLJ only for equi/natural with inner index
"optimise tree"	push legal σ and π; keep join/needed attrs; explain smaller intermediates
"external sort"	$runs_0 = \lceil N/B \rceil$; merge fan-in B−1; count pass 0; each pass 2N I/O
"clustered vs unclustered"	clustered range ≈ consecutive pages; unclustered up to one I/O per match
"capacity/pages"	floor capacity; apply fill; ceil pages/levels
"sorted input/output"	consider sort-merge join; attractive when an input is already sorted or output must be sorted
"operator hand-off"	pipeline to avoid temporary I/O when possible; materialise when the full intermediate must exist

Pass 0: sort B-page chunks → runs



Merge pass: up to B−1 runs → one



Pass 0 plus merge passes; each reads and writes N pages = 2N I/O. *Ch 12, Fig 12.3*

Exam-Morning Plan

CLOSED-BOOK RETRIEVAL

1. From blank paper, rebuild: six algebra operators; SQL clause order; five ER mapping rules.
2. Write closure, BCNF split, lossless test; then conflict rule, graph test, 2PL phases.
3. Write floor/ceil page formulae; three NLJ costs; external-sort passes/cost.
4. Do one 10-minute mixed drill. Mark only the first wrong decision, then redo without looking.
5. In the paper: scan, budget by marks, state formula/rule, substitute, show intermediate values, box conclusion, move on.

FINAL RECALL *Shape first, arithmetic second: identify relationship/cardinality, operator order, determinant/key status, conflict direction, access path and outer relation before calculating.*