UCLA COMPUTER SCIENCE FIELD NOTES

CS 143

Database Systems

Relational design, SQL, storage, indexes, query processing, transactions, recovery, and security

How these notes are written. I wrote each chapter the way I wish the material had been explained the first time: the real idea, the point where I usually get stuck, the simplest mental model that still stays honest, and a worked decision instead of a polished answer appearing from nowhere. Every chapter closes with practice and a fully written solution. If a sentence sounds impressive but does not help me solve or explain something, it does not belong here.

Daniel Mastick

Curriculum map, working notes, practice, and solutions

Course map

Week	Core thread	What should be solid by the end
1	**Relational model and algebra**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
2	**SQL and three-valued logic**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
3	**Conceptual design and normalization**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
4	**Storage and buffer management**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
5	**Indexes and B+ trees**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
6	**Query processing and optimization**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
7	**Transactions and concurrency control**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
8	**Logging, recovery, integrity, and security**	Explain the model, apply it to a concrete case, and defend the relevant correctness or design decision.
9	**Integration workshop**	Connect at least three chapters in one end-to-end problem and explain where their contracts meet.
10	**Synthesis and project review**	Audit assumptions and explain a complete solution from interface to failure handling.

Scope anchor: https://catalog.registrar.ucla.edu/course/2025/COMSCI143. The sequence is aligned to the official catalog description and expanded into a practical ten-week study plan.

How I would use this packet

Treat each numbered section as one chapter. Read the formal explanation, pause at the student interpretation, and see whether the easy version still says the same thing. Rework the example without looking, then cover the solution in the chapter practice box. The cumulative set at the end is deliberately mixed: knowledge becomes durable when I have to choose the tool instead of being told which chapter I am in.

Contents

CHAPTER 1 Relational model and algebra

The idea

A relation is a set of tuples under a schema. Keys identify, foreign keys connect, and algebraic operators select, project, join, union, difference, and rename. Set and bag semantics must not be confused.

Rebuild the chapter from first principles

The claim I need to own is this: A relation is a set of tuples under a schema. Keys identify, foreign keys connect, and algebraic operators select, project, join, union, difference, and rename. Set and bag semantics must not be confused. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of relational model and algebra. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach relational model and algebra on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from the course foundations to sql and three-valued logic. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** A relation is a set of tuples under a schema.
- **Mechanism.** Keys identify, foreign keys connect, and algebraic operators select, project, join, union, difference, and rename.
- **Boundary.** Set and bag semantics must not be confused.
- **Decision test.** I should be able to say when relational model and algebra is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I write the result schema beside every relational-algebra step.

The easy version

A relation is a set of tuples under a schema.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why can projection remove rows? The conclusion is Under relational set semantics, duplicate projected tuples collapse. SQL normally uses bag semantics unless DISTINCT is requested.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why can projection remove rows?
Solution. Under relational set semantics, duplicate projected tuples collapse. SQL normally uses bag semantics unless DISTINCT is requested.

Practice 2. Give a short oral explanation of relational model and algebra that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct relational model and algebra from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a relational model and algebra problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 2 SQL and three-valued logic

The idea

SQL combines declarative queries, aggregation, subqueries, windows, and updates. NULL introduces unknown, so predicates evaluate true, false, or unknown; WHERE keeps only true.

Rebuild the chapter from first principles

The claim I need to own is this: SQL combines declarative queries, aggregation, subqueries, windows, and updates. NULL introduces unknown, so predicates evaluate true, false, or unknown; WHERE keeps only true. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of sql and three-valued logic. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach sql and three-valued logic on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from relational model and algebra to conceptual design and normalization. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** SQL combines declarative queries, aggregation, subqueries, windows, and updates.
- **Mechanism.** NULL introduces unknown, so predicates evaluate true, false, or unknown; WHERE keeps only true.
- **Boundary.** The useful test is whether I can apply sql and three-valued logic to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when sql and three-valued logic is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I test NULL behavior explicitly instead of treating it as an ordinary value.

The easy version

SQL combines declarative queries, aggregation, subqueries, windows, and updates.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why does col=NULL fail? The conclusion is Equality with NULL is unknown. Use IS NULL or IS NOT NULL; also reason carefully about NOT IN when its subquery can contain NULL.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why does col=NULL fail?
Solution. Equality with NULL is unknown. Use IS NULL or IS NOT NULL; also reason carefully about NOT IN when its subquery can contain NULL.

Practice 2. Give a short oral explanation of sql and three-valued logic that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct sql and three-valued logic from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and

choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a sql and three-valued logic problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 3 Conceptual design and normalization

The idea

Entity-relationship design models identity, relationships, cardinality, participation, and weak entities. Functional dependencies drive normalization. BCNF decomposition removes certain redundancy while requiring lossless joins and considering dependency preservation.

Rebuild the chapter from first principles

The claim I need to own is this: Entity-relationship design models identity, relationships, cardinality, participation, and weak entities. Functional dependencies drive normalization. BCNF decomposition removes certain redundancy while requiring lossless joins and considering dependency preservation. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of conceptual design and normalization. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach conceptual design and normalization on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from sql and three-valued logic to storage and buffer management. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Entity-relationship design models identity, relationships, cardinality, participation, and weak entities.
- **Mechanism.** Functional dependencies drive normalization.
- **Boundary.** BCNF decomposition removes certain redundancy while requiring lossless joins and considering dependency preservation.
- **Decision test.** I should be able to say when conceptual design and normalization is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I normalize from stated business dependencies, not from column names that merely look related.

The easy version

Entity-relationship design models identity, relationships, cardinality, participation, and weak entities.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Decompose R(A,B,C) with A->B and B->C into BCNF. The conclusion is B->C violates BCNF if B is not a superkey. Decompose to R1(B,C) and R2(A,B); the decomposition is lossless through shared determinant B.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Decompose R(A,B,C) with A->B and B->C into BCNF.
Solution. B->C violates BCNF if B is not a superkey. Decompose to R1(B,C) and R2(A,B); the decomposition is lossless through shared determinant B.

Practice 2. Give a short oral explanation of conceptual design and normalization that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct conceptual design and normalization from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a conceptual design and normalization problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 4 Storage and buffer management

The idea

Pages are the database's I/O unit. Heap files, sorted files, slotted pages, records, buffer pools, replacement, and write policies connect logical rows to durable blocks. I/O count usually dominates CPU cost.

Rebuild the chapter from first principles

The claim I need to own is this: Pages are the database's I/O unit. Heap files, sorted files, slotted pages, records, buffer pools, replacement, and write policies connect logical rows to durable blocks. I/O count usually dominates CPU cost. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of storage and buffer management. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach storage and buffer management on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from conceptual design and normalization to indexes and b+ trees. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Pages are the database's I/O unit.
- **Mechanism.** Heap files, sorted files, slotted pages, records, buffer pools, replacement, and write policies connect logical rows to durable blocks.
- **Boundary.** I/O count usually dominates CPU cost.
- **Decision test.** I should be able to say when storage and buffer management is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I estimate page reads before comparing plans.

The easy version

Pages are the database's I/O unit.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why use slotted pages for variable records? The conclusion is A slot directory keeps stable record identifiers while records move during compaction; offsets and lengths can change without rewriting every external reference.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why use slotted pages for variable records?
Solution. A slot directory keeps stable record identifiers while records move during compaction; offsets and lengths can change without rewriting every external reference.

Practice 2. Give a short oral explanation of storage and buffer management that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct storage and buffer management from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a storage and buffer management problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 5 Indexes and B+ trees

The idea

Clustered and unclustered indexes trade lookup and update costs. B+ trees stay shallow through high fanout and keep data entries or pointers in linked leaves. Hash indexes excel at equality but do not preserve range order.

Rebuild the chapter from first principles

The claim I need to own is this: Clustered and unclustered indexes trade lookup and update costs. B+ trees stay shallow through high fanout and keep data entries or pointers in linked leaves. Hash indexes excel at equality but do not preserve range order. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover

the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of indexes and b+ trees. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach indexes and b+ trees on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from storage and buffer management to query processing and optimization. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method

valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Clustered and unclustered indexes trade lookup and update costs.
- **Mechanism.** B+ trees stay shallow through high fanout and keep data entries or pointers in linked leaves.
- **Boundary.** Hash indexes excel at equality but do not preserve range order.
- **Decision test.** I should be able to say when indexes and b+ trees is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

An index is useful only when selectivity and access pattern repay its maintenance and random I/O.

The easy version

Clustered and unclustered indexes trade lookup and update costs.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why do B+ tree records live at leaves? The conclusion is Uniform leaf placement increases internal fanout and keeps all searches the same structural depth; linked leaves support ordered range scans.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why do B+ tree records live at leaves?
Solution. Uniform leaf placement increases internal fanout and keeps all searches the same structural depth; linked leaves support ordered range scans.

Practice 2. Give a short oral explanation of indexes and b+ trees that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct indexes and b+ trees from a blank page. Include the central

definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a indexes and b+ trees problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 6 Query processing and optimization

The idea

Operators have physical implementations: scans, nested loops, sort-merge, and hash joins. Selection pushdown, join order, indexes, statistics, and cardinality estimates shape cost. Optimization is search under imperfect estimates.

Rebuild the chapter from first principles

The claim I need to own is this: Operators have physical implementations: scans, nested loops, sort-merge, and hash joins. Selection pushdown, join order, indexes, statistics, and cardinality estimates shape cost. Optimization is search under imperfect estimates. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of query processing and optimization. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach query processing and optimization on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from indexes and b+ trees to transactions and concurrency control. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Operators have physical implementations: scans, nested loops, sort-merge, and hash joins.
- **Mechanism.** Selection pushdown, join order, indexes, statistics, and cardinality estimates shape cost.
- **Boundary.** Optimization is search under imperfect estimates.
- **Decision test.** I should be able to say when query processing and optimization is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I shrink intermediate results early and identify the largest join before trusting a plan.

The easy version

Operators have physical implementations: scans, nested loops, sort-merge, and hash joins.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: When is hash join attractive? The conclusion is For an equijoin with enough memory to build or partition the smaller input. It does not directly support arbitrary inequality joins or ordered output.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. When is hash join attractive?
Solution. For an equijoin with enough memory to build or partition the smaller input. It does not directly support arbitrary inequality joins or ordered output.

Practice 2. Give a short oral explanation of query processing and optimization that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct query processing and optimization from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a query processing and optimization problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 7 Transactions and concurrency control

The idea

ACID describes atomicity, consistency, isolation, and durability. Schedules may be conflict-serializable; two-phase locking, timestamp ordering, MVCC, and deadlock handling enforce different tradeoffs.

Rebuild the chapter from first principles

The claim I need to own is this: ACID describes atomicity, consistency, isolation, and durability. Schedules may be conflict-serializable; two-phase locking, timestamp ordering, MVCC, and deadlock handling enforce different tradeoffs. I do not count that as learned just because the sentence looks familiar. I should be able to name the objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of transactions and concurrency control. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach transactions and concurrency control on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from query processing and optimization to logging, recovery, integrity, and security. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** ACID describes atomicity, consistency, isolation, and durability.
- **Mechanism.** Schedules may be conflict-serializable; two-phase locking, timestamp ordering, MVCC, and deadlock handling enforce different tradeoffs.
- **Boundary.** The useful test is whether I can apply transactions and concurrency control to a new case and explain the assumption that makes the answer valid.
- **Decision test.** I should be able to say when transactions and concurrency control is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

I draw the interleaving and dependency edges; concurrency anomalies are histories, not vague race feelings.

The easy version

ACID describes atomicity, consistency, isolation, and durability.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Detect a lost update. The conclusion is Two transactions read the same old value, compute separately, and both write; the later write erases the first. Atomic update, locking, or serializable control prevents it.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Detect a lost update.
Solution. Two transactions read the same old value, compute separately, and both write; the later write erases the first. Atomic update, locking, or serializable control prevents it.

Practice 2. Give a short oral explanation of transactions and concurrency control that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct transactions and concurrency control from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a transactions and concurrency control problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

CHAPTER 8 Logging, recovery, integrity, and security

The idea

Write-ahead logging records changes before dirty data reaches disk. Recovery repeats committed effects and removes incomplete ones according to the protocol. Constraints, least privilege, parameterized queries, auditing, encryption, and backups protect correctness and access.

Rebuild the chapter from first principles

The claim I need to own is this: Write-ahead logging records changes before dirty data reaches disk. Recovery repeats committed effects and removes incomplete ones according to the protocol. Constraints, least privilege, parameterized queries, auditing, encryption, and backups protect correctness and access. I do not count that as learned just because the sentence looks familiar. I should be able to name the

objects, state what changes and what stays fixed, recover the rule from the definitions, and explain why the result is allowed.

The move

My working sequence

1. **Translate.** Rewrite the prompt in the language of logging, recovery, integrity, and security. List the known quantities, the unknown, the units or types, and any hidden constraint.
2. **Choose.** State the theorem, model, algorithm, accounting rule, or physical law before substituting values. This is where I make the assumption visible.
3. **Execute.** Work one justified step at a time and keep intermediate state readable enough that I can audit it later.
4. **Verify.** Check a boundary case, sign, unit, invariant, order of magnitude, or alternative derivation. Then translate the result back into an ordinary sentence.

Details I would refuse to skip

- **Definitions:** I should be able to define every technical noun in the chapter without using that same noun in the definition.
- **Assumptions:** I write the condition that makes the method legal beside the method itself. A correct procedure under the wrong assumptions is still a wrong answer.
- **Representation:** I move freely among words, equations or code, a diagram, and a small numerical example. If one representation disagrees with another, that disagreement is useful evidence.
- **Failure modes:** I keep the nearest counterexample in mind: the boundary where the rule changes, the invariant breaks, or the model stops matching reality.
- **Communication:** My final answer names what the result means and what it does not establish. I do not let notation do the explanatory work for me.

Sanity check

Closed-note check. Can I teach logging, recovery, integrity, and security on a blank page, derive or reconstruct its main rule, work a fresh example, and diagnose a deliberately broken solution? If I can only recognize the finished work, I am not done.

How this chapter connects

I read this chapter as the bridge from transactions and concurrency control to the cumulative course model. The earlier material supplies the language and constraints; this chapter adds a new reasoning move; the later material asks me to use that move without being told its name. That connection matters because exams and real projects mix chapters on purpose.

What I need to be able to do

For this chapter, I should be able to define the objects and state involved, state the governing invariant or contract, trace a small example without hand-waving, and explain which assumption makes the method valid. I should also be able to compare this model with the nearest alternative and say what failure would make me switch approaches.

Deep notes

- **Model.** Write-ahead logging records changes before dirty data reaches disk.
- **Mechanism.** Recovery repeats committed effects and removes incomplete ones according to the protocol.
- **Boundary.** Constraints, least privilege, parameterized queries, auditing, encryption, and backups protect correctness and access.
- **Decision test.** I should be able to say when logging, recovery, integrity, and security is the right model, when its assumptions fail, and what evidence would change my choice.

How I actually think about it

My shorthand for this chapter is: my shorthand for this chapter is: a backup is not real until restore has been tested.

The easy version

Write-ahead logging records changes before dirty data reaches disk.

Worked example

A useful way to work this chapter is to make the smallest concrete instance, write the state or contract explicitly, and then scale the reasoning only after that instance behaves correctly. The worked question below makes that reasoning concrete: Why does WAL require log-before-data? The conclusion is If a dirty page reaches disk first and the system crashes, recovery lacks a durable description needed to redo or undo that change.

Common miss

The common mistake is to memorize the visible procedure while dropping its precondition. I write the assumption beside the method and test one boundary case before trusting the result.

Solved chapter practice

Practice. Why does WAL require log-before-data?
Solution. If a dirty page reaches disk first and the system crashes, recovery lacks a durable description needed to redo or undo that change.

Practice 2. Give a short oral explanation of logging, recovery, integrity, and security that includes its model, one assumption, one failure mode, and one comparison with a neighboring method.
Solution. A strong answer begins with the model in the blue box, names the assumption that

licenses the method, uses the common miss above as the failure mode, and chooses the comparison according to the decision test in the deep notes. The goal is not one memorized sentence; it is a four-part explanation I can reproduce on a new example.

Mastery practice A. Reconstruct logging, recovery, integrity, and security from a blank page. Include the central definition, the governing rule, one required assumption, and one boundary case.
Solution. Start from the blue idea box, but rewrite it in my own words. Identify the objects and state, name the rule before using it, copy the relevant assumption from the details audit, and choose a boundary where that assumption becomes equality or fails. A complete solution explains why each part belongs; a list of vocabulary is not enough.

Mastery practice B. A classmate gets a polished-looking answer to a logging, recovery, integrity, and security problem but never states a model or checks the result. Write the shortest useful review of that solution.
Solution. I would ask four concrete questions: What exactly are the inputs and desired output? Which rule licenses the main step? Which assumption could invalidate it? Which independent check supports the conclusion? If the answer cannot survive those four questions, I treat the numerical or verbal result as unverified.

Final study check

I should be able to close this packet and reconstruct the core models, not merely recognize their names. My final check is to explain one complete problem aloud: what the inputs mean, which invariant or contract controls the work, why the method is legal, what it costs, how it can fail, and what evidence would convince another engineer.