CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

1 selected course | 18 November 2026

Selected courses

01. Advanced Database Systems

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Advanced Database Systems 6

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

Mark the exact thing that stopped you: an unknown term, a rule, a calculation, a trace or a choice between alternatives.

Use the question lookup to open the relevant section.

Read one section, close it, and explain the rule in your own words.

Solve the question again and say why each other choice fails.

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

A useful 60-minute session: 10 minutes of cold questions, 20 minutes of focused reading, 20 minutes of closed-book solving and 10 minutes updating an error log.

Adjust the timing to your needs.

Revisit mistakes after a gap, for example the next day, several days later and the following week.

These intervals are a practical starting plan, not a fixed learning law.

Record the section ID, your mistaken rule, the corrected rule and one boundary case.

Keep the log short enough to review.

Mix related courses during revision: programming with data structures; databases with security; networking with administration.

Explain how the same word changes meaning across contexts.

START02 What to emphasize

Emphasize definitions, key differences, algorithm costs, SDLC models, databases, networking and security.

Use short oral questions to check understanding and correct common misconceptions.

Combine those habits with the tracing, calculation, design and diagnosis demanded by the Ministry learning outcomes.

Allocate revision by both exam share and your error log.

A course with fewer items can still deserve more time if your foundations are weak.

The blueprint is a planning baseline, not a prediction of exact questions.

Definitions: distinguish nearby concepts, not just their names.

Calculations: write units, formulas, assumptions and intermediate steps.

Code and SQL: check validity before tracing the text; state any changes before analyzing a corrected version.

Algorithms and protocols: know the input, decision rule, state change, termination and failure cases.

Design choices: state the criterion.

A choice can change when cost, reliability, consistency or latency matters more.

Course group	Items in the 100-item blueprint	Revision implication
Web Programming	9	Reserve extra practice for HTML, CSS, JavaScript, PHP, client/server behavior and safe data handling.
Data Structures and Algorithms	7	Trace operations and compare their assumptions and costs.
Each of the other fourteen courses	6 each	Study all sixteen; do not abandon a whole course based on a short note or an alleged repeated-question list.

START03 How to handle exam traps

Underline NOT, FALSE, BEST and EXCEPT before comparing choices.

Check whether the stem supplies enough facts to select one answer.

A long scenario is not automatically an analysis question, and a familiar answer letter is not evidence.

When code is invalid, a diagram is absent, or two choices express the same rule, identify the clarification needed before answering.

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

Distinguish worst case from average case, and an upper bound from a tight bound.

Check language version, SQL dialect, addressing assumptions, graph weights and tie-breaking rules.

For subnetting and scheduling, calculate rather than guess from a familiar number.

Use timed mixed papers only after you can explain your errors.

Revisit the reading sections behind recurring mistakes.

Advanced Database Systems

Complete Study Notes

ADB01 | Objects, identity, classes and persistence

An object combines identity, state and behavior.

A class specifies the attributes and operations of its instances; an object is one instance.

The object identifier (OID) distinguishes objects independently of their attribute values and remains stable over that object's lifetime.

Two objects can have equal values and different identities; changing a name does not create a new identity.

Persistence lets an object survive beyond the program execution that created it.

An OODBMS adds durable storage, transactions, querying and controlled sharing to object concepts.

It does not primarily present objects as rows and tables.

An object-relational DBMS extends a relational system with richer types; ORM maps programming objects into relational tables.

These three approaches differ.

Complex CAD or multimedia structures can suit an object model, but many-to-many relationships alone do not require abandoning relational design.

University of Pittsburgh: Object-oriented databases

ADB02 | Encapsulation, abstraction and object associations

Encapsulation groups state with operations and controls access to implementation details.

Account.withdraw(amount) can enforce a balance rule while hiding its storage.

Abstraction presents the relevant contract-what withdrawal does-without exposing how it is implemented.

These ideas overlap in information hiding, but combining state and methods specifically describes encapsulation.

An association links objects, often through references; Department.employees can reference Employee objects.

It is different from inheritance, which means an is-a relationship between types.

Methods can invoke operations on other objects, supporting collaboration.

A class defines structure and behavior; an attribute describes state; a method implements an operation.

Do not treat each of these as an interchangeable name for the whole object.

University of Pittsburgh: Object-oriented databases

ADB03 | Inheritance, polymorphism and safe substitutability

Inheritance derives a subtype from a superclass, reusing applicable attributes and behavior.

GraduateStudent is a Student; association instead says Student belongs to Department.

EER teaches inherited attributes and relationships, while a programming object model also specifies behavior.

Subtype polymorphism lets a superclass/interface reference denote objects of different compatible subclasses.

The same operation can dispatch to different implementations: Shape.area() calls Circle.area() or Rectangle.area() according to the actual object.

This is not arbitrary conversion of an object into an unrelated class, nor dynamic class mutation.

Inheritance establishes a subtype relationship; polymorphism uses that relationship through a common interface.

Distinguish inheritance as a structural relationship from polymorphism as a use of that relationship at runtime.

Oracle Java Tutorials: Polymorphism

ADB04 | Object type constructors

In a common object model, an object is described by (OID,type constructor,state).

An atom is one basic value, such as an integer, real, Boolean or string.

A tuple constructor groups named component values into one structured record, such as (Street:'A',City:'B').

It is not a collection of records.

A set is an unordered collection without duplicates; a bag allows duplicates; a list has order; an array provides indexed positions.

Constructors can be nested to represent complex objects, such as a tuple with a set of phone numbers.

An array is not an atomic constructor merely because stored as one programming variable.

A type constructor defines the shape of a value; it is not the same as an executable constructor operation that creates a new object.

University of Pittsburgh: Object type constructors

ADB05 | ODL, OQL and inverse relationships

Object Definition Language (ODL) is the ODMG schema-definition language for object databases; it describes interfaces/classes, attributes, operations and relationships.

Object Query Language (OQL) expresses queries.

Host-language bindings connect these definitions and operations to languages such as Java or C++.

Defining a logical schema is different from implementing its storage.

An inverse declaration pairs opposite directions of the same relationship.

Conceptually, Department.employees is inverse to Employee.department: when an employee belongs to department D, the corresponding employee appears in D's collection.

The declaration expresses relationship consistency; it is not an inheritance keyword or an ordinary scalar attribute.

An MCQ calling ODL both a DDL and a language for developing logical schemas offers overlapping true descriptions, so its exact wording matters.

Juniata College: Object Databases

ADB06 | Transactions, ACID and state transitions

A transaction is a logical unit of database work, including read-only work.

Atomicity means all effects or none; consistency means correct work preserves integrity rules; isolation controls concurrent interference; durability preserves committed effects despite failure.

A DBMS cannot automatically make incorrect business logic consistent.

Active means operations are executing.

Partially committed means the last operation finished but durable commitment is not yet assured.

Committed means commitment succeeded.

A failure leads to Failed, then Aborted after rollback; a transaction may terminate or be restarted as a new attempt.

Concurrency control primarily enforces isolation; recovery supports atomicity and durability.

Transaction management is the broader subsystem, so both it and recovery management can be legitimate answers to broad responsibility questions.

Example

Worked example

- A=5,000 and B=8,000 become 4,250 and 8,750 after transferring 750 from A to B. Both balances total 13,000. Unchanged or reverse-transfer balances can also preserve that sum; only the requested debit/credit implements the specified successful transfer.

Fundamental of Database Systems module

Advanced Database Systems module

ADB07 | Read/write operations, buffers and log records

read_item(X) locates X's page, brings it into a buffer if absent, and copies X into a program variable. write_item(X) copies the new program value into the buffered page, marking it dirty.

Writing that page to disk can occur later according to the buffer policy; write_item is not inherently an immediate disk flush.

Insert and delete also modify data; sorting is a query-processing operation rather than a primitive item update.

A conceptual log can contain START, UPDATE(T,X,old,new), COMMIT, ABORT and END records.

Before-images support undo; after-images support redo.

Real systems may use physical or logical records, and need not log every read.

COMMIT records successful completion; an ABORT record can mark the start of rollback rather than prove every undo step is complete.

Fundamental of Database Systems module

Advanced Database Systems module

ADB08 | Diagnose concurrency anomalies from a trace

Lost update: T1 and T2 both read 500; T2 adds 200 and writes 700; T1 adds 100 to its stale value and writes 600.

Correct serial execution yields 800.

Dirty read: T2 reads a value written by uncommitted T1; if T1 aborts, dependent work must be reconsidered.

Reading one's own changes is not a dirty read.

Nonrepeatable read means a repeated read of one row sees a committed change.

A phantom means repeating a predicate query finds a changed qualifying set after another transaction inserts, deletes or changes matching rows.

Incorrect summary mixes mutually inconsistent versions: a transfer moves 10 from A=100 to B=100; a sum reading old A=100 and new B=110 gets 210 instead of the invariant 200.

A single committed update from B=20 to B=50 does not alone prove an invalid summary of 90.

Fundamental of Database Systems module

Advanced Database Systems module

ADB09 | Serial schedules, conflicts and precedence graphs

A serial schedule executes each transaction's operations consecutively.

A serializable schedule is equivalent to some serial order; it may interleave transactions.

Equality of one final value is insufficient evidence because different executions can accidentally coincide.

Two operations conflict when they belong to different transactions, access the same item and at least one writes.

Read/read pairs do not conflict.

For conflict serializability, make one graph vertex per transaction.

For every earlier conflicting operation of Ti and later operation of Tj, add Ti→Tj.

Acyclic means conflict-serializable; a topological ordering gives a valid serial order.

Cyclic means not conflict-serializable.

This test is stronger than view serializability, which also compares read-from relationships and final writers and may permit some schedules with blind writes.

Example

Worked example

- r1(X), w2(X), r2(Y), w1(Y) gives T1→T2 on X and T2→T1 on Y, a cycle. r1(X), w1(X), r2(X), w2(X) gives only T1→T2 and is conflict-serializable.

Fundamental of Database Systems module

Advanced Database Systems module

ADB10 | Recoverable, cascadeless and strict schedules

Recoverable means if Tj reads Ti's value, Tj cannot commit before Ti commits.

Otherwise Ti might abort after Tj already committed dependent work.

Cascadeless means a transaction reads only values written by committed transactions, avoiding cascading aborts.

Strict means no other transaction reads or overwrites an item written by an uncommitted transaction.

Strict ⇒ cascadeless ⇒ recoverable.

For w1(X), r2(X), c2, a1, T2 committed a dirty dependency, so the schedule is unrecoverable.

Moving c2 after c1 makes that dependency recoverable, but the earlier dirty read still makes it not cascadeless.

Serializability and recoverability are separate tests: one concerns equivalent transaction order and the other safe commit/rollback dependencies.

Fundamental of Database Systems module

Advanced Database Systems module

ADB11 | Locks, compatibility, conversion and managers

A shared (S) lock permits reading; multiple S locks on the same item can coexist.

An exclusive (X) lock permits its holder to read and write and conflicts with another S or X on that item.

Locks on different items do not conflict merely because one is exclusive.

These are the basic lock-based rules; multiversion systems can permit additional snapshot-reader behavior.

A transaction must own the appropriate lock before access and can release only its own locks.

S→X is an upgrade, possible only after conflicting other holders leave.

X→S is a downgrade.

The lock manager decides requests; the lock table stores granted locks and waiting requests.

One transaction may request a lock on an already locked item: it is granted if compatible, otherwise waits or aborts under the policy.

No double-locking is not a prohibition on shared readers.

Held / requested	S	X
S	Compatible	Conflict
X	Conflict	Conflict

Fundamental of Database Systems module

Advanced Database Systems module

ADB12 | Two-phase locking and its variants

Basic 2PL has a growing phase that acquires/upgrades locks and a shrinking phase that releases/downgrades them.

Once any lock is released, no new lock may be acquired.

The lock point is the instant the final lock is obtained. 2PL guarantees conflict serializability for well-formed transactions, but can deadlock and basic 2PL alone need not prevent cascading aborts.

Strict 2PL retains exclusive locks until commit/abort.

Rigorous 2PL retains both shared and exclusive locks until that point.

Conservative/static 2PL obtains all needed locks before execution and does not begin while only a subset is available, preventing hold-and-wait deadlock.

It requires knowing the lock set and can reduce concurrency.

Two-phase locking orders access to data; two-phase commit coordinates a distributed commit decision.

Fundamental of Database Systems module

Advanced Database Systems module

ADB13 | Deadlock detection, prevention and starvation

Deadlock requires a waiting cycle: T1 holds X and wants Y; T2 holds Y and wants X. In a wait-for graph, an edge Ti→Tj means Ti waits for a lock held by Tj.

A cycle indicates deadlock under the ordinary lock-wait model.

Abort a suitable victim, undo its effects and release its locks; repeat if other cycles remain.

Aborting every deadlocked transaction also breaks cycles but wastes more work.

Fixed resource ordering and conservative 2PL prevent cycles.

A no-wait policy aborts immediately on an incompatible lock request and needs no age timestamp.

A timeout aborts long waits but may mistake slow progress for deadlock.

Starvation is indefinite postponement while other work proceeds, without a required cycle.

Fair queues, sensible victim selection and retaining original transaction timestamps on retry reduce starvation; deadlock freedom alone does not prove fairness.

Fundamental of Database Systems module

Advanced Database Systems module

ADB14 | Wait-die and wound-wait: compare requester and holder

A smaller timestamp means an older transaction.

In wait-die, an older requester may wait for a younger holder; a younger requester needing an older holder aborts itself.

In wound-wait, an older requester aborts the younger holder; a younger requester waits for the older holder.

Always identify requester versus holder before applying the names.

Retain the original timestamp when restarting an aborted transaction so it gains relative priority as new transactions arrive.

Starvation avoidance additionally assumes holders eventually finish and scheduling is fair.

In wait-die, an older transaction may wait for a younger one; a younger transaction requesting a resource held by an older one aborts.

Deadlock-prevention priority rules are also different from basic timestamp ordering's read/write validity tests.

Requester relative to holder	Wait-die	Wound-wait
Older	Wait	Abort younger holder
Younger	Abort requester	Wait

Fundamental of Database Systems module

Advanced Database Systems module

San Jose State University: Deadlocks and distributed databases

ADB15 | Basic timestamp ordering: execute the tests

Assign each transaction TS(T).

For item X track read_TS(X), the largest timestamp of a successful reader, and write_TS(X), that of its latest successful writer.

A read by T aborts if TS(T)<write_TS(X); otherwise it succeeds and updates read_TS.

A write aborts if TS(T)<read_TS(X) or TS(T)<write_TS(X); otherwise it succeeds and sets write_TS.

These tests preserve timestamp order.

For TS(T)=10 and write_TS(X)=20, T cannot read X: it would observe a logically later write.

Thus the transaction aborted can be older, contrary to the guides' blanket younger-always-aborts claim.

The Thomas write rule can ignore an obsolete write instead of aborting in one case, but that is a different protocol and must be stated.

Basic timestamp ordering avoids lock-wait deadlocks but may cause repeated restarts.

Fundamental of Database Systems module

Advanced Database Systems module

Virginia Tech: Timestamp protocols

ADB16 | Optimistic control, versions and lock granularity

Optimistic/validation control reads and computes privately, validates its read/write sets against overlapping transactions, then publishes if validation succeeds; otherwise it retries.

It works well when conflicts are uncommon, but high contention wastes work on aborts.

Multiversion control keeps versions so a reader can use an appropriate snapshot.

Snapshot isolation is not automatically serializable: concurrent writes to different rows can violate a shared invariant.

Fine-grained row locks permit more independent operations but require tracking many locks.

Coarse table/database locks simplify management but block unrelated rows and reduce concurrency.

Intention locks advertise planned descendant locks in a hierarchy; escalation replaces many small locks with a coarser one.

Small granularity does not inherently cause more logical conflicts.

Choose granularity from contention and management overhead, not a universal rule.

Fundamental of Database Systems module

Advanced Database Systems module

ADB17 | Query processing: representation before execution

Parsing checks SQL grammar; semantic analysis resolves names, types and access permissions.

Translation constructs a logical representation, typically relational algebra or a logical query tree.

Optimization chooses equivalent operations, join order and physical access methods.

Evaluation executes that plan and returns results.

Not every implementation exposes exactly the same phase boundaries.

A query tree has base relations at leaves and operators at internal nodes; a query graph emphasizes relations and predicates.

A physical execution plan adds scan/index choices and algorithms.

Query processing is the whole activity; selecting a suitable efficient strategy is optimization.

DDL is not a mandatory stage before each SELECT, so authentication→DDL→DML permutations do not describe the processing sequence.

Fundamental of Database Systems module

Advanced Database Systems module

ADB18 | Heuristic and semantic query rewrites

Heuristics reduce intermediate work: push selective row filters toward base tables; replace product followed by a join predicate with a join; project unneeded columns away while retaining join/filter columns; reorder inner joins when equivalence permits; reuse common expressions when worthwhile.

Split conjunctive selections when each predicate can be applied earlier to its own input.

Semantic optimization additionally uses trusted constraints, such as a key proving a join cannot multiply matching rows.

Cost-based optimization estimates candidate plans instead of relying on rules alone.

Outer joins, NULLs, duplicates and aggregation restrict equivalences: moving a right-table filter from ON to WHERE after a left join may delete its preserved unmatched rows.

Fewer intermediate rows is a useful heuristic, not proof of minimum runtime.

Fundamental of Database Systems module

Advanced Database Systems module

ADB19 | Estimate costs, cardinalities and selectivity

A cost model estimates resources for a candidate plan: page I/O, CPU, memory, temporary storage and network traffic.

Catalog statistics include row/page counts, distinct values and distributions.

Selection with fraction s predicts s×N rows.

For non-NULL equijoin keys, a common estimate is $|R|\times|S|/\max(V(R,k),V(S,k))$, assuming uniform values and containment of the smaller distinct-value domain within the larger.

Adjust counts for NULLs; if either non-NULL input is empty, the result is zero.

Skew, correlations or incomplete domain overlap can invalidate this estimate; known disjoint domains give zero matches.

For a simple rotating-disk model, cost = seeks×seek_time + transferred_blocks×block_transfer_time.

Five seeks at 5 ms plus ten blocks at 1 ms cost 35 ms.

Disk I/O often dominates textbook local-disk examples; CPU can dominate in-memory work and network can dominate distributed plans.

State the assumed cost objective before ranking strategies.

Fundamental of Database Systems module

Advanced Database Systems module

CMU 15-445/645 Spring 2024 Query Optimization slides

Leis et al., How Good Are Query Optimizers, Really?

ADB20 | Join algorithms and access-path tradeoffs

Tuple nested loops scan the inner relation for each outer tuple.

Page nested loops approximately cost B(R)+B(R)×B(S) page reads.

With M buffers, block nested loops can cost B(R)+ceil(B(R)/(M−2))×B(S), reserving an inner and output buffer.

Try both outer choices.

An indexed nested-loop join probes an inner index for each qualifying outer tuple; it suits a small outer result and a useful index.

Sort-merge joins sorted join keys and must include any sorting cost.

Hash join partitions/builds a hash structure and probes it; equality joins suit it, ranges generally do not.

A one-pass in-memory hash join can read each input once when the build side fits.

Include output/materialization cost if the exercise requires it.

Fundamental of Database Systems module

Advanced Database Systems module

ADB21 | Compare actual query strategies

Compare only plans returning the same result under the same assumptions.

Add scans, probes, sorting, intermediate writes and communication as applicable; do not count an omitted cost in only one plan.

Record memory limits and whether inputs are already ordered.

Choose the lowest estimated cost for the specified objective, then test estimates against actual execution.

Example: R has 100 pages, S has 1,000 pages and M=12 buffers.

Block nested loops with R outer cost 100+ceil(100/10)×1,000=10,100 reads; S outer costs 1,000+ceil(1,000/10)×100=11,000.

If a selective R index costs 3 reads and returns 20 outer tuples, with each S index probe costing 4 reads including matches, indexed nested loops cost 3+20×4=83 reads.

That wins under these stated assumptions; it is not a universal join ranking.

Fundamental of Database Systems module

Advanced Database Systems module

ADB22 | Pipelining, parallelism and caching

Materialization stores an intermediate result before the next operator consumes it.

Pipelining passes tuples as they become available, overlapping producer and consumer work and reducing temporary I/O. A sort is normally blocking; a hash join must build its hash table before probing can output, though later probe/output can stream.

Pipelining does not require separate CPUs.

Parallel execution uses multiple workers, for example partitioned scans/joins, independent subplans or operator pipelines.

Interleaving on one CPU is concurrency; simultaneous cores provide parallelism.

Result caching reuses answers and needs invalidation when underlying data changes.

Plan caching reuses an execution strategy, not the rows returned.

Buffer caching keeps data pages; materialization retains intermediate rows.

Specify the kind of query caching being discussed.

Fundamental of Database Systems module

Advanced Database Systems module

ADB23 | Indexes, statistics and re-optimization

B/B+ tree indexes support ordered and range predicates; hash indexes target equality; bitmap indexes can help some analytic selections.

Clustered storage keeps related rows close.

An index may reduce reads but adds space and update work; scanning can be faster when most rows qualify.

Commonly useful is not a unique property of one index type.

Re-optimization chooses a plan again when statistics, parameters or observed cardinalities suggest the old plan is unsuitable.

Updating statistics does not itself execute the query.

Use EXPLAIN and measured execution to compare estimates, spills, scans and join choices.

Avoid universal rules that joins always beat subqueries, functions can never use indexes or more indexes always improve performance.

Fundamental of Database Systems module

Advanced Database Systems module

ADB24 | Security, integrity and inference control

Authentication establishes identity; authorization/access control determines permitted actions.

Encryption protects stored or transmitted content; integrity constraints restrict valid data; auditing records actions.

A user who can view but cannot change a record has restricted access privileges.

A view can restrict visible rows/columns, but its privilege and execution rules must support that restriction.

Inference control concerns learning protected individual facts indirectly from allowed aggregate answers.

Releasing SUM for a group and SUM for the same group excluding one known person may disclose that person's value by subtraction.

Suppressing individual rows alone is insufficient; protections may restrict queries, aggregate sizes or add appropriately designed noise.

Access-restricted views can expose aggregates while hiding details; inference control additionally concerns preventing indirect disclosure.

Fundamental of Database Systems module

Advanced Database Systems module

Jagannathan and Wright: Private Inference Control for Aggregate Database Queries

ADB25 | GRANT, REVOKE and privilege scope

GRANT SELECT ON Catalogue TO Mohammed; permits reads.

REVOKE DELETE, UPDATE (Price,Year) ON Catalogue FROM Mohammed; removes the specified table deletion and column update grants in SQL dialects supporting that form.

DELETE applies to rows/table access rather than particular columns; UPDATE can be column-scoped.

REVOKE removes permission, whereas GRANT gives it.

Roles group privileges.

WITH GRANT OPTION permits a recipient to grant a privilege onward; removing an upstream grant can cascade or be restricted according to the chosen command and DBMS.

Removing one direct grant may leave effective access through another role or grant path.

A view is a stored query definition, not encryption.

Least privilege assigns only the access needed for the task.

Fundamental of Database Systems module

Advanced Database Systems module

ADB26 | Classify failures and choose the starting resource

Transaction failures include invalid operations, constraint violations and concurrency-control aborts.

Enforcing concurrency can deliberately abort work; it does not prevent every transaction failure.

A system crash loses volatile memory while disks may survive.

Media failure damages persistent storage and normally requires restoring a backup, then replaying available archived logs.

Human mistakes may require point-in-time restoration before the bad committed operation.

Restart recovery restores a valid database after a crash while preserving durable commits and removing incomplete effects.

Rollback handles failed/uncommitted work; redo preserves committed work missing from pages; checkpoints reduce log-search work; backups provide another stored copy.

These are cooperating mechanisms, not four mutually exclusive complete alternatives.

Buffer management is not durable backup.

Generic recovery-manager terminology should not be mistaken for an exact MySQL component specification.

A full backup copies the database to separate storage; scheduled backups plus archived logs support restoration to a chosen recoverable point.

Fundamental of Database Systems module

Advanced Database Systems module

ADB27 | WAL, undo and redo

Write-ahead logging requires the update log needed for recovery to reach stable storage before the corresponding changed data page reaches disk.

To acknowledge a durable commit, force the transaction's required log records, including its commit record, to stable storage.

A buffer can be changed before that log flush; WAL orders durable writes, not every CPU assignment.

Undo restores before-images of incomplete effects, usually traversing that transaction's updates backward.

Redo installs logged after-images when necessary, usually forward.

Setting X to its logged after-value is idempotent; blindly replaying X=X+10 twice is not.

In a simple undo/redo teaching model, undo losers and redo winners.

ARIES has a different redo phase that repeats history before undoing losers.

Fundamental of Database Systems module

Advanced Database Systems module

ADB28 | Steal and force determine recovery work

Steal permits a page with uncommitted changes to be written to disk; hence recovery may need undo.

No-steal prevents those uncommitted effects reaching the persistent database.

Force writes a transaction's updated data pages at commit; no-force permits committed dirty pages to remain buffered, hence recovery may need redo.

For the conventional page/log model: steal/no-force needs undo and redo; steal/force needs undo but no redo; no-steal/no-force needs redo but no undo; no-steal/force needs neither data-page undo nor redo after a system crash.

Logging/atomic commit machinery may still be necessary.

These policies are independent of whether an update overwrites an old page or uses a separate location.

Immediate update alone does not uniquely specify every recovery requirement.

Buffer policy	Force	No-force
Steal	UNDO	UNDO + REDO
No-steal	Neither data-page UNDO nor REDO	REDO

Fundamental of Database Systems module

Advanced Database Systems module

ADB29 | Deferred, immediate, in-place and shadow updates

Deferred update keeps uncommitted changes outside the persistent database; in the usual no-force setting it is NO-UNDO/REDO.

Redo-only and NO-UNDO/REDO mean the same thing, so those cannot be separate unique answers.

Immediate update allows persistent changes before commit and therefore may require undo; it need not flush every write_item immediately.

In-place update overwrites the original location.

Shadow/copy-on-write update writes a new location while preserving the committed version.

Shadow paging keeps an unchanged committed page map and a modified map; commit durably installs a new root/map pointer atomically after required pages are durable.

Before commit, a crash uses the old map.

It is an alternative to conventional log-based recovery, with allocation, fragmentation and concurrency costs; it is not primarily a special action done only during recovery.

Fundamental of Database Systems module

Advanced Database Systems module

ADB30 | Checkpoints and recovery boundaries

A checkpoint records enough durable recovery state to avoid scanning all old history.

In a simple quiescent/checkpoint exercise, specified flushing may make earlier committed work unnecessary to redo.

Active transactions can span the checkpoint, so their relevant earlier updates can still need undo.

Do not assume every transaction before a checkpoint needs no processing.

A fuzzy checkpoint permits ongoing transactions and records active-transaction and dirty-page information without forcing all pages simultaneously.

Recovery may start analysis at the checkpoint but redo from an earlier required log position.

A checkpoint is neither a full backup nor an automatic commit of active transactions.

Without checkpoints or another valid truncation boundary, long-running systems may need expensive log scans; deferred update does not avoid that problem.

Fundamental of Database Systems module

Advanced Database Systems module

ADB31 | ARIES and repeated crashes during recovery

ARIES uses WAL, page log sequence numbers (pageLSN), a transaction table and a dirty-page table.

Analysis reconstructs active/loser transactions and dirty-page information.

Redo repeats history, starting from the relevant earliest dirty-page update; pageLSNs help skip already-installed actions.

Undo reverses loser transactions.

Compensation log records (CLRs) describe undo actions and their continuation information.

They can be redone but are not themselves undone.

This makes recovery restartable: after a second crash, durable undo progress need not be undone again as if no progress occurred.

Absence of checkpoints alone does not imply identical undo/redo lists.

State the recovery algorithm, durable logs/page state and progress before predicting repeated-crash behavior.

Fundamental of Database Systems module

Advanced Database Systems module

Silberschatz, Korth and Sudarshan: Recovery System slides

ADB32 | Execute a small recovery problem

Assume steal/no-force, durable WAL and a simple undo/redo method.

Initially X=100,Y=50.

The durable log contains START T1; UPDATE T1 X 100→120; COMMIT T1; START T2; UPDATE T2 Y 50→70; crash.

T1 is a winner and T2 a loser.

Redo X=120 if needed; undo Y=50 if T2's change reached disk.

Under no-steal, Y never reached the persistent database, so no undo of it is needed.

To solve larger cases, list transactions and durable commit status, identify the checkpoint type and buffer policy, select the algorithm, apply redo/undo in its required order and recompute final values.

Transactions active at failure are not automatically restarted and committed.

Business-level retries require a new controlled attempt; idempotency prevents duplicate effects when the client is unsure whether a commit succeeded.

Fundamental of Database Systems module

Advanced Database Systems module

ADB33 | Distributed, homogeneous and heterogeneous systems

A distributed database contains logically related data across networked sites, managed as a coordinated system.

A homogeneous system uses compatible DBMS technology across sites; a heterogeneous system integrates differing DBMSs, data models, schemas or interfaces.

Different operating systems are possible but do not define the distinction.

Same DBMS does not imply identical local schemas, automatic replication or immediate synchronization.

Heterogeneous design must reconcile naming, types, units, identifiers, query capabilities and transaction semantics.

ODBC and JDBC provide connectivity interfaces; they do not automatically solve semantic differences or create one distributed transaction.

Locality, expansion and replicated availability are potential benefits.

Distribution adds communication, coordination and security costs; a site outage can still make needed data unavailable if no usable replica exists.

Fundamental of Database Systems module

Advanced Database Systems module

ADB34 | Architectures and local autonomy

In client/server architecture, clients submit requests while servers manage shared data, execute operations and enforce supported integrity/transaction rules.

In collaborating-server designs, servers cooperate on subqueries and transactions.

Middleware can translate requests, locate data, coordinate operations and combine results across independent servers.

These describe communication responsibilities, not fragmentation types.

A tightly integrated distributed system can present one global schema and shared control.

A federation integrates autonomous databases through mappings; a loosely coupled multidatabase system may preserve more independence without one complete global schema.

State how much schema, transaction and administrative autonomy local sites retain.

Calling a system heterogeneous does not require all sites to surrender autonomy or to copy every row everywhere.

Fundamental of Database Systems module

Advanced Database Systems module

ADB35 | Transparency: what exactly is hidden?

Location transparency hides which site holds a named object.

Fragmentation transparency hides the existence and reconstruction of fragments: users query the whole logical relation.

Replication transparency hides which copy is used and how copies are maintained.

The broad single-logical-database experience may involve all three.

If users explicitly name a fragment but need not specify its site, fragmentation is visible while location can remain transparent.

Distinguish whether fragment names, storage sites or both are hidden from the user.

Column subsets define vertical fragmentation, not a type of transparency.

Hiding this column split from queries is fragmentation transparency; the split itself is a data-design operation.

Fundamental of Database Systems module

Advanced Database Systems module

ADB36 | Fragmentation: correctness and reconstruction

Horizontal fragmentation selects row subsets by predicates; their union reconstructs the relation.

Check completeness so every original row is represented and disjointness if no overlap is intended.

Define an explicit fragment for NULL/other values when needed.

Vertical fragmentation projects column subsets and repeats a key or row identifier so joining on it reconstructs the original rows without spurious matches.

Hybrid fragmentation combines row and column splits in either suitable order.

Reconstruct by reversing the operations: key joins for corresponding vertical pieces and unions for horizontal pieces.

A full outer join is not universally required.

Partitioning improves locality and parallelism but alone creates no spare copy.

Losing the only site containing a fragment makes that fragment unavailable, even while other sites continue.

Fundamental of Database Systems module

Advanced Database Systems module

ADB37 | Replication policies and their costs

Full replication places the whole database at every relevant site; partial replication copies selected fragments to selected sites; nonredundant allocation stores each fragment once.

Copies improve read locality and can improve availability but require storage and update propagation.

They can also spread mistakes, so replication does not replace independent backups.

Eager/synchronous replication coordinates propagation before the specified commit/acknowledgment point, increasing update latency.

Lazy/asynchronous replication propagates later and may expose stale reads or need conflict resolution.

Snapshot replication transfers a copy at intervals; incremental replication transfers changes.

These dimensions can coexist, so commonly used does not identify one unique policy.

Full replication does not inherently mean every copy is updated simultaneously; its synchronization rule must be stated.

Fundamental of Database Systems module

Advanced Database Systems module

ADB38 | Distributed query plans and semijoin arithmetic

Distributed processing decomposes a logical query, localizes referenced relations to fragments/replicas, chooses global placement/order and optimizes local subplans.

Include bytes moved, message latency, bandwidth and local work.

Apply filters and projections before shipping when equivalent.

A semijoin retains only rows of one input that have a match in the other, returning that input's attributes.

Suppose R at A occupies 100 MB and S at B 20 MB; compute the result at A. Shipping all S costs 20 MB.

A semijoin can send 0.2 MB of distinct R join keys to B, then return 1 MB of matching S rows: 1.2 MB before other costs.

It wins on transferred bytes, but extra messages may dominate on high-latency links.

If nearly all S matches, key shipping may not help.

Fundamental of Database Systems module

Advanced Database Systems module

ADB39 | Distributed atomic commit and Oracle commit points

Two-phase commit (2PC) first asks participants to prepare and durably promise they can commit.

If all vote yes, the coordinator records commit durably and tells participants; otherwise it decides abort.

A prepared participant must retain enough state to obey the final decision.

Coordinator/communication failure can leave it in doubt and block progress until the decision is learned.

Timeout after preparation is not permission to invent a commit or abort.

2PC coordinates atomic outcome; locking/other concurrency protocols provide isolation.

Oracle's COMMIT_POINT_STRENGTH helps choose the commit point site in its distributed transaction mechanism, with higher eligible strength preferred.

This is an Oracle-specific 2PC role rule, not a lock phase or checkpoint setting.

The transaction coordinator coordinates decisions rather than physically storing every site's rows.

Fundamental of Database Systems module

Advanced Database Systems module

Oracle: COMMIT_POINT_STRENGTH

ADB40 | Construct homogeneous and heterogeneous designs

For a bank using one compatible DBMS at three branches, horizontally fragment branch accounts by BranchID; keep each account at its home branch and replicate selected read-mostly reference data.

For cross-branch transfers, use global identifiers, coordinated commit and a stated isolation policy.

Choose replica placement so the required service remains available after the specified failure.

Verify reconstruction, capacity, latency and recovery procedures.

For a merger retaining different branch DBMSs, add mappings for AccountID, currency, time zones and numeric precision, plus middleware/gateways for supported operations.

Decide whether all engines support atomic distributed commits; if not, define an explicit compensating workflow and its intermediate-state limits.

Compare retaining autonomy against coordination and availability requirements.

A good design answers who owns each fact, where copies live, what a failed link permits and how conflicting writes are resolved.

Fundamental of Database Systems module

Advanced Database Systems module

ADB41 | Warehouses, OLTP, OLAP and columnar storage

OLTP handles frequent operational transactions, such as account updates; OLAP scans and summarizes large datasets for analysis.

A warehouse integrates data for historical analysis, often through extraction, transformation and loading.

These are workload tendencies: OLTP can retain history and warehouses can be updated.

A star schema places measurements at a declared grain in a fact table linked to descriptive dimensions, such as time, product and store.

A snowflake normalizes dimension structure further.

Columnar storage groups values by column, reducing unnecessary column reads and aiding compression in scans; row-oriented storage groups each record's fields.

Neither physical layout alone determines whether the logical model is relational.

Indexes and denormalization choices depend on workload.

Fundamental of Database Systems module

Advanced Database Systems module

ADB42 | NoSQL families, wide columns and CAP

Key-value stores retrieve values by key; document stores hold structured documents; wide-column/column-family systems organize sparse rows and related column families; graph systems model nodes and edges for relationship traversal.

Wide-column systems such as Cassandra are not simply interchangeable with analytic columnar relational engines.

Flexible schema does not mean no schema rules, and NoSQL does not imply no transactions.

CAP states that under a network partition, a distributed service cannot guarantee both linearizable consistency and availability for every request at nonfailed nodes.

It is not a universal menu saying casually pick any two forever.

Systems choose failure behavior and may vary it by operation.

CAP consistency is a single-copy visibility/order property, distinct from ACID consistency's preservation of application invariants.

Gilbert and Lynch: Perspectives on the CAP Theorem

ADB43 | Data sources and association rules

Structured data follows explicit fields, such as a registrar's relational tables.

Unstructured data includes free text, images and audio; web/search collections can include these alongside structured content.

A DBMS can also store text or BLOBs, so a storage product does not prove all its contents are structured.

An association rule $X \Rightarrow Y$ describes co-occurrence.

Support=count(X and Y)/N; confidence=count(X and Y)/count(X)=P(Y|X).

If 75 of 100 shoe purchases include socks, confidence(shoes⇒socks)=0.75.

This does not prove causation or specify support without total transaction count.

Clustering groups similar cases without target labels; classification predicts categories and regression predicts numeric outcomes.

Purchasing co-occurrence is association mining even when its result informs store layout.

University of Tennessee: Association rules

ADB44 | Data-mining workflow and evaluation

CRISP-DM starts with business understanding: define the decision and success criteria.

Then understand data, prepare it, model, evaluate and deploy; the process is iterative.

Cleaning, integration and feature selection occur during preparation, not before the business problem exists.

Within modeling, split data appropriately, train on training data, tune using validation and evaluate once on a held-out test set before deployment.

Testing and evaluation are sometimes overlapping labels rather than universally distinct sequential phases.

If an MCQ asks training→testing→evaluation→deployment, state that convention; another valid workflow can evaluate during model selection before final testing.

Deployment includes monitoring and may trigger another preparation/modeling cycle.

Do not leak future or test outcomes into training.

IBM SPSS Modeler: CRISP-DM View

ADB45 | Compare database mechanisms under explicit assumptions

Database mechanisms can overlap: ARIES and deferred update can both be log-based; restart recovery can use checkpoints, rollback and redo; NO-UNDO/REDO equals redo-only.

Questions using best, important, common or faster need a workload and comparison criterion.

Read the literal scope rather than treating one familiar term as automatically unique.

State the minimum missing information: exact DBMS/version for vendor behavior; OID/type convention for object constructors; requester/holder ages for waiting; durable log, page state and policy for recovery; fragment definitions and network cost for distribution.

Evaluate each description separately when several database mechanisms apply.

Use the stated assumptions to explain which database mechanisms apply to each question.

Fundamental of Database Systems module

Advanced Database Systems module

ADB-S01 | Query representations and relational operators

A relation is represented by a table of tuples. Selection filters its rows by a predicate; projection retains chosen columns. A Cartesian product constructs every pair of input tuples. A matching join combines tuples satisfying its predicate, and can be represented as a selection over a product. Sorting orders results, while grouping collects rows sharing grouping values; these are different operations.

A query tree corresponds to a relational-algebra expression: base relations appear at leaves and operators at internal nodes. A query graph represents the predicate relationships of a relational-calculus expression without itself prescribing an operator execution order. An optimizer can transform the representation and choose access paths; the representation is not the query result.

Advanced Database Systems module · page 24

Advanced Database Systems module · page 34

PostgreSQL documentation - Table Expressions

PostgreSQL documentation - Sorting Rows

ADB-S02 | Transaction boundaries and savepoints

In SQL transaction-block notation, BEGIN starts a block, COMMIT completes it successfully, and ROLLBACK cancels its uncommitted changes. Committing requests a durable transaction outcome; it is distinct from merely finishing the last read or write operation.

SAVEPOINT marks an intermediate position inside the current transaction. ROLLBACK TO discards changes made after that position while retaining earlier work in the transaction. A savepoint is not an independent durable commit, and rolling back to it does not itself commit the retained changes.

PostgreSQL documentation - Transactions

ADB-S03 | Availability and evidence against false denial

Availability concerns keeping services and resources accessible to their authorized users. Non-repudiation concerns evidence that an event or action occurred and who originated it, so a participant cannot falsely deny it. Neither property specifies which particular user has permission to change a database row; that is the separate authorization decision.

Computer Security module · page 4

ADB-S04 | Elementary binary locks and shared-lock compatibility

An elementary binary lock has two states: free and held. A request on a free item sets the lock to held; a request on an already-held item waits until release. That simple model does not distinguish shared readers from a writer. Shared/exclusive locking adds modes so multiple shared readers can coexist while incompatible exclusive access is excluded.

Advanced Database Systems module · page 83

ADB-S05 | Classify a failure by its stated cause

A software failure originates in erroneous program behavior, such as a logical programming defect, invalid operation or operating-system fault. A hardware failure originates in malfunctioning physical equipment; loss of disk contents is media failure. An accidental user deletion is human error even if the software executes the requested command correctly. Physical catastrophes such as fire can also cause failure. The observed symptom of a crash does not alone identify which cause occurred.

Advanced Database Systems module · page 98