CS Exit Ethiopia

Revision Outlines

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 3

Advanced Database Systems

Revision Outlines

Objects, identity, classes and persistence | Exam focus

- Object identity, state, behavior
  - Class → attributes/operations
  - Object → class instance
  - OID independent of attribute values
  - Lifetime-stable OID
  - Equal values/different identities
  - Name change ≠ identity change
- Persistence and database approaches
  - Object survives creating program
  - OODBMS durable storage/transactions
  - Querying/controlled sharing
  - Objects not primarily rows/tables
  - Object-relational → richer relational types
  - ORM → objects mapped to tables
  - Three distinct approaches
  - Complex CAD/multimedia suitability
  - M:N alone needs no relational abandonment

Encapsulation, abstraction and object associations | Exam focus

- Encapsulation versus abstraction
  - Encapsulation → state plus operations
  - Controlled implementation access
  - Account.withdraw(amount) → balance rule
  - Hidden storage details
  - Abstraction → relevant operation contract
  - What versus implementation how
  - Overlapping information hiding
  - Combining state/methods → encapsulation

- Object associations and collaboration
  - References link objects
  - Department.employees → Employee objects
  - Association ≠ inheritance
  - Inheritance → is-a types
  - Methods invoke other-object operations
  - Class → structure/behavior
  - Attribute → state
  - Method → operation implementation
  - Part names ≠ whole object

Inheritance, polymorphism and safe substitutability | Exam focus

- Inheritance versus association
  - Subtype from superclass
  - Reuse applicable attributes/behavior
  - GraduateStudent is-a Student
  - Student belongs-to Department
  - EER inherited attributes/relationships
  - Programming object model adds behavior
- Subtype polymorphism
  - Superclass/interface reference → compatible subclasses
  - Common operation/different implementations
  - Shape.area() dispatch
  - Actual Circle.area()/Rectangle.area()
  - No arbitrary unrelated-class conversion
  - No dynamic class mutation
  - Inheritance → structural subtype prerequisite
  - Polymorphism → runtime common-interface use
  - Ambiguous MCQ needs requested distinction

Object type constructors | Exam focus

- Object type constructors
  - (OID,type constructor,state)

- Atom → basic value
- Integer/real/Boolean/string
- Tuple → named component values
- (Street:'A',City:'B')
- Structured record ≠ record collection

• Collection constructors and nesting

- Set → unordered/no duplicates
- Bag → duplicates allowed
- List → ordered
- Array → indexed positions
- Nested constructors → complex objects
- Tuple with phone-number set
- One variable ≠ atomic constructor
- Type constructor → value shape
- Executable constructor → object creation

ODL, OQL and inverse relationships | Exam focus

• ODMG object languages

- ODL → Object Definition Language
- Object database schema definition
- Interfaces/classes/attributes/operations/relationships
- OQL → Object Query Language
- Host-language bindings → Java/C++
- Definitions/operations connected to host languages
- Logical schema ≠ storage implementation

• Inverse relationship declaration

- Department.employees ↔ Employee.department
- Opposite directions/same relationship
- Employee in department D
- Employee appears in D collection
- Relationship consistency
- Not inheritance keyword/scalar attribute
- ODL DDL/logical-schema descriptions overlap

- Exact MCQ wording matters

Transactions, ACID and state transitions

- Transactions and ACID
  - Logical database-work unit
  - Read-only transactions included
  - Atomicity → all effects or none
  - Consistency → correct integrity-preserving work
  - Isolation → concurrent interference control
  - Durability → committed effects survive failure
  - DBMS cannot repair incorrect business logic
- Transaction states and responsibilities
  - Active → executing operations
  - Partially committed → last operation finished
  - Durability not yet assured
  - Committed → commitment succeeded
  - Failure → Failed
  - Rollback → Aborted
  - Terminate or restart new attempt
  - Concurrency control → isolation
  - Recovery → atomicity/durability
  - Transaction management broader
  - Broad responsibility answers can overlap
- Worked example Requested transfer versus invariant
  - Initial A=5,000/B=8,000
  - Transfer 750 A→B
  - Final A=4,250/B=8,750
  - Total=13,000
  - Unchanged/reverse balances also preserve total
  - Correct debit/credit required

Read/write operations, buffers and log records

- Item operations and buffer policy

- read_item(X) → locate page
- Fetch absent page into buffer
- Copy X into program variable
- write_item(X) → variable into buffered page
- Mark page dirty
- Disk flush may occur later
- write_item ≠ immediate disk flush
- Insert/delete modify data
- Sorting → query processing
- Sorting ≠ primitive item update

• Log records and images
- START/UPDATE(T,X,old,new)
- COMMIT/ABORT/END
- Before-image → undo
- After-image → redo
- Physical/logical record alternatives
- Not every read needs logging
- COMMIT → successful completion
- ABORT may only start rollback
- ABORT ≠ completed undo proof

Diagnose concurrency anomalies from a trace

• Lost update versus dirty read
- Both read 500
- T2 +200 → writes 700
- T1 stale +100 → writes 600
- Correct serial result 800
- Dirty read → other's uncommitted write
- Writer abort → reconsider dependent work
- Own changes ≠ dirty read

• Read and summary anomalies
- Nonrepeatable → same-row committed change
- Phantom → changed predicate result set

- Matching insert/delete/update
- Incorrect summary → inconsistent versions
- Transfer 10: A=100/B=100
- Old A=100 plus new B=110
- Observed 210 versus invariant 200
- Single committed B=20→50 insufficient
- Invalid-summary 90 needs more evidence

Serial schedules, conflicts, and precedence graphs

• Serial versus serializable
- Serial → consecutive transaction operations
- Serializable → equivalent serial order
- Interleaving permitted
- Same final value insufficient
- Accidental coincidence possible
- Conflict → different transactions
- Same item/at least one write
- Read/read nonconflicting

• Precedence graph test
- Vertex per transaction
- Earlier/later conflict → Ti→Tj
- Acyclic → conflict-serializable
- Topological order → valid serial order
- Cycle → not conflict-serializable
- View serializability weaker test
- Read-from relationships/final writers
- Some blind-write schedules permitted

• Worked example Worked conflict graphs
- r1(X),w2(X),r2(Y),w1(Y)
- X → T1→T2
- Y → T2→T1
- Cycle → not conflict-serializable
- r1(X),w1(X),r2(X),w2(X)

- Only T1→T2
- Acyclic → conflict-serializable

Recoverable, cascadeless and strict schedules

- Commit-dependency schedule hierarchy
  - Recoverable → reader commits after writer
  - Avoid committed dependency on aborted writer
  - Cascadeless → read committed writes only
  - No cascading aborts
  - Strict → no other's uncommitted-item reads
  - No other's uncommitted-item overwrites
  - Strict ⇒ cascadeless ⇒ recoverable
- Recoverability worked trace
  - w1(X),r2(X),c2,a1
  - Dirty dependency committed → unrecoverable
  - c2 after c1 → recoverable dependency
  - Earlier dirty read remains
  - Not cascadeless
  - Serializability → equivalent order
  - Recoverability → safe commit/rollback dependencies
  - Separate tests

Locks, compatibility, conversion and managers

- Shared/exclusive lock compatibility
  - S → read permission
  - Multiple S locks coexist
  - X → holder reads/writes
  - X conflicts with other's S/X
  - Different-item locks need not conflict
  - Basic lock-based rules
  - Multiversion snapshot readers may differ
- Lock ownership, conversion, management
  - Appropriate lock before access

- Release only own locks
- S→X upgrade
- Conflicting holders must leave
- X→S downgrade
- Lock manager → request decisions
- Lock table → granted/waiting requests
- Already locked request allowed
- Compatible → granted
- Incompatible → wait/abort by policy
- No-double-locking ≠ shared-reader prohibition

• Held S compatibility
- Requested S → compatible
- Requested X → conflict

• Held X compatibility
- Requested S → conflict
- Requested X → conflict

Two-phase locking variants

• Basic two-phase locking
- Growing → acquire/upgrade
- Shrinking → release/downgrade
- Any release → no new acquisition
- Lock point → final lock obtained
- Well-formed 2PL → conflict serializability
- Deadlock possible
- Basic 2PL may allow cascading aborts

• 2PL variants versus 2PC
- Strict → X until commit/abort
- Rigorous → S/X until commit/abort
- Conservative/static → all locks beforehand
- No execution with partial lock set
- Prevents hold-and-wait deadlock
- Requires known lock set

- May reduce concurrency
- 2PL → data-access ordering
- 2PC → distributed commit decision

Deadlock detection, prevention and starvation

• Deadlock detection and repair
- T1 holds X/wants Y
- T2 holds Y/wants X
- Wait-for Ti→Tj → held-lock wait
- Cycle → ordinary lock-wait deadlock
- Select victim/abort/undo/release locks
- Repeat for remaining cycles
- Aborting all also works
- Aborting all wastes more work

• Deadlock prevention versus starvation
- Fixed resource ordering
- Conservative 2PL prevents cycles
- No-wait → abort incompatible request
- No age timestamp needed
- Timeout → abort long waits
- Slow progress mistaken for deadlock
- Starvation → indefinite postponement
- Other work continues/no required cycle
- Fair queues/sensible victim selection
- Retain original retry timestamps
- Deadlock freedom ≠ fairness

Wait-die and wound-wait requester/holder rules

• Wait-die versus wound-wait
- Smaller timestamp → older
- Identify requester/holder first
- Wait-die older requester → wait
- Wait-die younger requester → self-abort

- Wound-wait older requester → abort holder
- Wound-wait younger requester → wait

• Timestamp priority and correction
- Restart with original timestamp
- Relative priority grows with new arrivals
- Starvation avoidance needs finishing holders
- Fair scheduling also required
- Guide reverses wait-die incorrectly
- Younger waits/older dies incorrect
- Prevention priorities ≠ timestamp-ordering validity tests

• Older requester/younger holder
- Wait-die → requester waits
- Wound-wait → abort younger holder

• Younger requester/older holder
- Wait-die → abort requester
- Wound-wait → requester waits

Basic timestamp ordering tests

• Basic timestamp-ordering tests
- Assign TS(T)
- read_TS(X) → largest successful reader timestamp
- write_TS(X) → latest successful writer timestamp
- Read abort: TS(T)<write_TS(X)
- Otherwise read/update read_TS
- Write abort: TS(T)<read_TS(X)
- Or TS(T)<write_TS(X)
- Otherwise write/set write_TS
- Preserve timestamp order

• Timestamp example and protocol limits
- TS(T)=10/write_TS(X)=20
- Read aborts → logically later write
- Older transaction can abort
- Younger-always-aborts claim false

- Thomas rule → obsolete write ignored
- Different protocol must be stated
- No lock-wait deadlocks
- Repeated restarts possible

Optimistic control, versions, and lock granularity

- Optimistic and multiversion control
  - Private reads/computation
  - Validate read/write sets
  - Check overlapping transactions
  - Validation success → publish
  - Failure → retry
  - Uncommon conflicts favorable
  - High contention → wasted aborted work
  - MVCC → multiple versions
  - Reader uses appropriate snapshot
  - Snapshot isolation ≠ automatic serializability
  - Different-row writes can violate shared invariant
- Lock granularity tradeoffs
  - Fine row locks → independent operations
  - Many-lock tracking overhead
  - Coarse table/database → simpler management
  - Unrelated rows blocked/lower concurrency
  - Intention → planned descendant locks
  - Hierarchy
  - Escalation → many small to coarse
  - Small granularity ≠ more logical conflicts
  - Choose contention versus management overhead

Query processing before execution | Exam focus

- Query-processing phases
  - Parsing → SQL grammar
  - Semantic analysis → names/types/permissions

- Translation → logical representation
- Relational algebra/logical query tree
- Optimization → equivalent operations
- Join order/physical access methods
- Evaluation → execute plan/return results
- Phase boundaries implementation-dependent

• Query representation and scope
- Tree leaves → base relations
- Internal nodes → operators
- Graph → relations/predicates
- Physical plan → scans/indexes/algorithms
- Query processing → whole activity
- Optimization → efficient strategy selection
- DDL unnecessary before each SELECT
- Authentication→DDL→DML permutations incorrect

Heuristic and semantic query rewrites | Exam focus

• Heuristic rewrites
- Push selective filters toward bases
- Product plus join predicate → join
- Project unneeded columns away
- Retain join/filter columns
- Reorder equivalent inner joins
- Reuse common expressions when worthwhile
- Split conjunctive selections
- Apply each predicate to own input

• Semantic versus cost-based optimization
- Trusted constraints drive semantic rewrites
- Key → join cannot multiply matches
- Cost-based → estimate candidate plans
- Outer joins/NULLs/duplicates/aggregation restrict equivalence
- LEFT JOIN right filter ON→WHERE
- Preserved unmatched rows may disappear

- Fewer intermediates ≠ minimum-runtime proof

Costs, cardinalities, and selectivity | Exam focus

- Cost and cardinality estimation
  - Page I/O/CPU/memory
  - Temporary storage/network traffic
  - Catalog row/page counts
  - Distinct values/distributions
  - Selection rows ≈ s×N
  - Equijoin ≈ |R|×|S|/max(V(R,k),V(S,k))
  - Non-NULL join-key counts
  - Uniform-value assumption
  - Smaller distinct domain contained in larger
  - Adjust for NULLs
  - Empty non-NULL input → zero
  - Skew/correlation/incomplete overlap invalidate estimate
  - Known disjoint domains → zero
- Disk-time model and objective
  - Cost = seeks×seek_time + blocks×block_transfer_time
  - 5×5ms + 10×1ms = 35ms
  - Textbook local disk → I/O often dominates
  - In-memory → CPU may dominate
  - Distributed → network may dominate
  - State objective before ranking plans

Join algorithms and access paths | Exam focus

- Nested-loop page costs
  - Tuple loops → inner scan/outer tuple
  - Page loops ≈ B(R)+B(R)×B(S)
  - M buffers
  - Block loops ≈ B(R)+ceil(B(R)/(M−2))×B(S)
  - Reserve inner/output buffers
  - Try both outer relations

- Indexed, merge, and hash joins
  - Indexed loops → probe/qualifying outer tuple
  - Small outer/useful inner index
  - Sort-merge → sorted join keys
  - Include sorting cost
  - Hash → partition/build/probe
  - Equality suited/ranges generally unsuited
  - Build side fits → one-pass hash
  - Each input read once
  - Include required output/materialization costs

Compare actual query strategies | Exam focus

- Fair strategy comparison
  - Same result/same assumptions
  - Add scans/probes/sorts
  - Intermediate writes/communication
  - Consistent cost inclusion
  - Record memory limits
  - Already-ordered inputs?
  - Lowest estimated specified-objective cost
  - Compare estimates with actual execution
- Worked join-cost comparison
  - B(R)=100/B(S)=1,000/M=12
  - M−2=10 block pages
  - R outer: 100+ceil(100/10)×1,000=10,100
  - S outer: 1,000+ceil(1,000/10)×100=11,000
  - Selective R index=3 reads
  - 20 qualifying outer tuples
  - S probe including matches=4 reads
  - Indexed loops: 3+20×4=83
  - Indexed wins supplied assumptions
  - No universal join ranking

Pipelining, parallelism and caching | Exam focus

- Materialization versus pipelining
  - Materialize → store intermediate before consumption
  - Pipeline → tuples as available
  - Producer/consumer overlap
  - Reduced temporary I/O
  - Sort normally blocking
  - Hash build before probe output
  - Later probe/output can stream
  - Separate CPUs unnecessary
- Parallelism and cache distinctions
  - Multiple workers → parallel execution
  - Partitioned scans/joins
  - Independent subplans/operator pipelines
  - One-CPU interleaving → concurrency
  - Simultaneous cores → parallelism
  - Result cache → answers/invalidation
  - Plan cache → strategy, not rows
  - Buffer cache → data pages
  - Materialization → intermediate rows
  - Query-caching question needs cache kind

Indexes, statistics and re-optimization | Exam focus

- Index and storage choices
  - B/B+ trees → ordered/range
  - Hash → equality
  - Bitmap → some analytic selections
  - Clustered storage → nearby related rows
  - Index read savings
  - Space/update overhead
  - Most rows qualify → scan may win
  - Common usefulness ≠ unique index property
- Re-optimization and measurement

- Statistics/parameters/observed cardinalities trigger reconsideration
- Choose suitable plan again
- Statistics update ≠ query execution
- EXPLAIN plus measured execution
- Estimates/spills/scans/join choices
- Joins not universally faster than subqueries
- Functions can use indexes
- More indexes not always better

Security, integrity and inference control

- Security versus integrity mechanisms
  - Authentication → identity
  - Authorization/access control → permitted actions
  - Encryption → stored/transmitted content
  - Integrity constraints → valid data
  - Auditing → actions
  - View-only/no-change → restricted privileges
  - Views → visible row/column restriction
  - Privilege/execution rules must support restriction
- Inference control
  - Protected facts from allowed aggregates
  - SUM(group)−SUM(group excluding person)
  - Individual value disclosed
  - Row suppression alone insufficient
  - Query restrictions/minimum aggregate sizes
  - Appropriately designed noise
  - Aggregates-visible/details-hidden also fits views
  - Indirect-disclosure prevention uniquely identifies inference control

GRANT, REVOKE, and privilege scope

- GRANT versus REVOKE scope
  - GRANT SELECT ON Catalogue TO Mohammed
  - Read permission

- REVOKE DELETE, UPDATE (Price,Year)
- Catalogue FROM Mohammed
- Dialect must support form
- DELETE → rows/table, not columns
- UPDATE can be column-scoped
- REVOKE removes/GRANT gives permission

• Roles and effective privileges
- Roles → privilege groups
- WITH GRANT OPTION → onward delegation
- Upstream removal → CASCADE/RESTRICT possibilities
- Chosen command/DBMS determines behavior
- Other roles/grant paths may retain access
- Direct-grant removal ≠ no effective access
- View → stored query, not encryption
- Least privilege → only needed access

Failures and recovery starting resources | Exam focus

• Failure classes and starting resources
- Transaction → invalid operations/constraint violations
- Concurrency-control aborts
- Concurrency enforcement cannot prevent every failure
- System crash → volatile memory lost
- Disks may survive
- Media failure → persistent storage damage
- Restore backup/replay available archived logs
- Human error → point-in-time restoration
- Restore before bad committed operation

• Cooperating recovery mechanisms
- Restart → valid post-crash database
- Preserve durable commits/remove incomplete effects
- Rollback → failed/uncommitted work
- Redo → missing committed page effects
- Checkpoints → less log searching

- Backups → another stored copy
- Not mutually exclusive complete alternatives
- Buffer management ≠ durable backup
- Generic manager ≠ exact MySQL component
- Full backup → separate-storage database copy
- Scheduled backups plus archived logs
- Chosen recoverable-point restoration

WAL, undo and redo | Exam focus

- Write-ahead logging ordering
  - Recovery update log durable first
  - Then changed data page durable
  - Durable commit acknowledgment → required logs forced
  - Commit record included
  - Buffer change may precede log flush
  - WAL orders durable writes
  - Not every CPU assignment
- Undo versus redo models
  - Undo → incomplete-effect before-images
  - Usually backward transaction traversal
  - Redo → logged after-images as needed
  - Usually forward traversal
  - Set X to after-value → idempotent
  - Repeat X=X+10 twice → non-idempotent
  - Simple model → undo losers/redo winners
  - ARIES → repeat history then undo losers

Steal/force recovery requirements | Exam focus

- Steal and force independence
  - Steal → uncommitted page reaches disk
  - May need undo
  - No-steal → uncommitted effects stay off disk
  - Force → updated pages at commit

- No-force → committed dirty pages buffered
- May need redo

• Recovery-policy matrix assumptions
- Conventional page/log model
- Steal/no-force → undo+redo
- Steal/force → undo only
- No-steal/no-force → redo only
- No-steal/force → neither data-page operation
- After system crash
- Logging/atomic commit may remain necessary
- Independent of overwrite/separate-location updates
- Immediate update alone insufficient

• Steal recovery requirements
- Force → UNDO
- No-force → UNDO+REDO

• No-steal recovery requirements
- Force → neither data-page UNDO/REDO
- No-force → REDO

Deferred, immediate, in-place, and shadow updates | Exam focus

• Deferred versus immediate updates
- Deferred → uncommitted outside persistent database
- Usual no-force → NO-UNDO/REDO
- Redo-only = NO-UNDO/REDO
- Synonyms cannot be unique alternatives
- Immediate → persistent changes before commit
- May require undo
- write_item need not flush immediately

• In-place versus shadow updates
- In-place → overwrite original location
- Shadow/copy-on-write → new location
- Committed version preserved
- Unchanged committed page map

- Modified page map
- Commit → durable atomic root/map-pointer installation
- Required pages durable first
- Precommit crash → old map
- Alternative to conventional log recovery
- Allocation/fragmentation/concurrency costs
- Not solely recovery-time action

Checkpoints and recovery boundaries | Exam focus

- Checkpoint boundaries
  - Durable state avoids all-history scan
  - Simple quiescent/checkpoint assumptions
  - Specified flushing may remove earlier redo
  - Active transactions can span checkpoint
  - Earlier active updates may need undo
  - Before checkpoint ≠ no processing
- Fuzzy checkpoints
  - Transactions continue
  - Active-transaction/dirty-page information recorded
  - No simultaneous all-page forcing
  - Analysis may start at checkpoint
  - Redo may start earlier
  - Checkpoint ≠ full backup
  - Checkpoint ≠ active-transaction commit
  - No checkpoint/truncation → expensive log scans
  - Deferred update does not avoid scans

ARIES and repeated crashes during recovery | Exam focus

- ARIES phases and structures
  - WAL/pageLSN
  - Transaction table/dirty-page table
  - Analysis → active/loser transactions
  - Reconstruct dirty-page information

- Redo → repeat history
- Earliest relevant dirty-page update start
- pageLSN skips installed actions
- Undo → reverse losers

• CLRs and repeated-crash recovery

- Compensation log records → undo actions
- Continuation information
- CLRs redone, never undone
- Restartable recovery
- Durable undo progress retained
- No-checkpoint alone ≠ identical undo/redo lists
- Specify algorithm/durable logs/page state
- Specify recovery progress

Execute a small recovery problem | Exam focus

• Worked undo/redo recovery

- Steal/no-force/durable WAL
- Simple undo/redo algorithm
- Initially X=100/Y=50
- T1 X:100→120, durable COMMIT
- T2 Y:50→70, no commit
- Crash → T1 winner/T2 loser
- Redo X=120 if needed
- Undo Y=50 if reached disk
- No-steal → Y never persisted
- No Y undo under no-steal

• Recovery-solving procedure

- List durable transaction commit status
- Identify checkpoint type/buffer policy
- Select recovery algorithm
- Apply required redo/undo order
- Recompute final values
- Active-at-failure ≠ automatically restart/commit

- Business retry → controlled new attempt
- Uncertain commit → duplicate-effects risk
- Idempotency prevents duplicate effects

Distributed, homogeneous, and heterogeneous systems | Exam focus

- Distributed database distinctions
  - Logically related data/networked sites
  - Coordinated management
  - Homogeneous → compatible DBMS technology
  - Heterogeneous → differing DBMS/model/schema/interface
  - Operating systems do not define distinction
  - Same DBMS ≠ identical local schemas
  - No automatic replication/immediate synchronization
- Heterogeneous reconciliation and distribution costs
  - Naming/types/units/identifiers
  - Query capabilities/transaction semantics
  - ODBC/JDBC → connectivity interfaces
  - No automatic semantic reconciliation
  - No automatic distributed transaction
  - Potential locality/expansion/replicated availability
  - Communication/coordination/security costs
  - Outage without usable replica → unavailable data

Architectures and local autonomy | Exam focus

- Distributed architecture responsibilities
  - Client → requests
  - Server → shared-data management/operations
  - Supported integrity/transaction enforcement
  - Collaborating servers → cooperative subqueries/transactions
  - Middleware → request translation/data location
  - Operation coordination/result combination
  - Independent servers

- Responsibilities ≠ fragmentation types

• Integration versus local autonomy

- Tight integration → global schema/shared control
- Federation → autonomous databases/mappings
- Loose multidatabase → more independence
- May lack complete global schema
- Specify schema/transaction/administrative autonomy
- Heterogeneous ≠ surrender all autonomy
- No requirement to copy every row

Distribution transparency | Exam focus

• Distribution transparency types

- Location → site hidden
- Fragmentation → fragments/reconstruction hidden
- Query whole logical relation
- Replication → chosen copy/maintenance hidden
- Single logical database may combine all

• Visible fragments versus hidden locations

- Explicit fragment name → fragmentation visible
- Unnamed site → location possibly transparent
- Specific-fragment stem needs clarification
- Column subsets → vertical fragmentation
- Split itself ≠ transparency type
- Hidden column split → fragmentation transparency

Fragmentation: correctness and reconstruction | Exam focus

• Fragmentation correctness

- Horizontal → predicate row subsets
- Union reconstructs relation
- Completeness → every original row represented
- Disjointness if overlap forbidden
- Explicit NULL/other-value fragment when needed
- Vertical → projected column subsets

- Repeat key/row identifier
- Key join reconstructs original rows
- No spurious matches

• Hybrid reconstruction and availability
- Combine row/column splits
- Either suitable split order
- Reconstruct by reversing operations
- Vertical pieces → key joins
- Horizontal pieces → unions
- FULL OUTER JOIN not universally required
- Partitioning → locality/parallelism
- Partitioning alone ≠ spare copy
- Only-fragment-site loss → fragment unavailable
- Other sites may continue

Replication policies and their costs | Exam focus

• Replication extent and backup distinction
- Full → whole database/relevant sites
- Partial → selected fragments/selected sites
- Nonredundant → one copy/fragment
- Read locality/possible availability gains
- Storage/update-propagation costs
- Mistakes can propagate
- Replication ≠ independent backups

• Replication timing and transfer policies
- Eager/synchronous → propagation before commit/ack point
- Higher update latency
- Lazy/asynchronous → later propagation
- Stale reads/conflict resolution
- Snapshot → periodic copy
- Incremental → changes only
- Policy dimensions can coexist
- Commonly-used stem not unique

- Full replication ≠ simultaneous updates
- State synchronization rule

Distributed plans and semijoin arithmetic | Exam focus

- Distributed query planning
  - Logical query decomposition
  - Localize relations → fragments/replicas
  - Global placement/order
  - Optimize local subplans
  - Bytes/messages/latency/bandwidth/local work
  - Equivalent filters/projections before shipping
  - Semijoin → matching rows from one input
  - Only that input's attributes returned
- Worked semijoin communication
  - R at A=100MB
  - S at B=20MB
  - Result required at A
  - Ship all S → 20MB
  - Send distinct R keys → 0.2MB
  - Return matching S → 1MB
  - Semijoin bytes=0.2+1=1.2MB
  - Other costs excluded
  - Extra messages/high latency may dominate
  - Nearly all S matches → little benefit

Distributed atomic commit and Oracle roles | Exam focus

- Two-phase commit protocol
  - Prepare → durable commit promise
  - All yes → coordinator durably records commit
  - Then inform participants
  - Otherwise decide abort
  - Prepared participant retains decision-following state
  - Coordinator/communication failure → in-doubt blocking

- Wait until decision learned
- Post-prepare timeout ≠ invented commit/abort

• Atomic commit versus isolation
- 2PC → atomic outcome
- Locking/other concurrency protocols → isolation
- Oracle COMMIT_POINT_STRENGTH
- Higher eligible strength → commit point site
- Oracle-specific 2PC role selection
- Not lock phase/checkpoint setting
- Coordinator → decision coordination
- Not every site's physical row storage

Construct homogeneous and heterogeneous designs | Exam focus

• Homogeneous three-branch bank design
- Compatible DBMS
- Horizontal accounts by BranchID
- Home branch owns account
- Replicate selected read-mostly reference data
- Cross-branch → global identifiers
- Coordinated commit/stated isolation
- Replica placement meets specified failure service
- Verify reconstruction/capacity/latency/recovery

• Heterogeneous merger design
- Different branch DBMSs retained
- AccountID/currency/time-zone/precision mappings
- Middleware/gateways for supported operations
- All engines support atomic distributed commit?
- Otherwise explicit compensating workflow
- State intermediate-state limits
- Autonomy versus coordination/availability
- Fact ownership/copy placement
- Failed-link permitted behavior
- Conflicting-write resolution

Warehouses, OLTP, OLAP, and columnar storage

- OLTP, OLAP, and warehouses
  - OLTP → frequent operational transactions
  - Account updates
  - OLAP → large-data scans/summaries
  - Warehouse → integrated historical analysis
  - Extraction/transformation/loading
  - Workload tendencies, not absolute restrictions
  - OLTP may retain history
  - Warehouses may be updated
- Warehouse schemas and storage layouts
  - Star → declared fact grain
  - Fact measurements linked to dimensions
  - Time/product/store dimensions
  - Snowflake → normalized dimensions
  - Columnar → values grouped by column
  - Fewer unnecessary column reads/compression
  - Row-oriented → record fields grouped
  - Physical layout ≠ relational-model determination
  - Indexes/denormalization depend on workload

NoSQL families, wide columns and CAP

- NoSQL families
  - Key-value → retrieve by key
  - Document → structured documents
  - Wide-column/column-family → sparse rows
  - Related column families
  - Graph → nodes/edges/relationship traversal
  - Cassandra wide-column example
  - Not interchangeable with analytic relational columnar engines
  - Flexible schema ≠ absent schema rules
  - NoSQL ≠ no transactions
- CAP under network partitions

- Cannot guarantee both linearizable consistency/availability
- Every request at nonfailed nodes
- Partition condition essential
- Not casually pick-any-two forever
- Failure behavior choices
- May vary by operation
- CAP consistency → single-copy visibility/order
- ACID consistency → application invariant preservation

Data sources and association rules

• Structured versus unstructured data
- Structured → explicit fields
- Registrar relational tables
- Unstructured → free text/images/audio
- Web/search collections may mix both
- DBMS can store text/BLOBs
- Storage product ≠ all-structured contents

• Association rules and mining contrasts
- X⇒Y → co-occurrence
- Support=count(X and Y)/N
- Confidence=count(X and Y)/count(X)=P(Y|X)
- 75/100 shoe purchases include socks
- Confidence(shoes⇒socks)=0.75
- No causation proof
- Support needs total transaction count
- Clustering → similarity/no target labels
- Classification → categories
- Regression → numeric outcomes
- Purchasing co-occurrence → association mining
- Store-layout use does not change task

Data-mining workflow and evaluation

• CRISP-DM workflow

- Business understanding first
- Decision/success criteria
- Data understanding → preparation
- Modeling → evaluation → deployment
- Iterative process
- Preparation → cleaning/integration/feature selection
- Business problem precedes preparation

• Model evaluation and leakage
- Appropriate data split
- Train on training data
- Tune on validation
- Evaluate once on held-out test
- Before deployment
- Testing/evaluation labels can overlap
- Training→testing→evaluation→deployment needs convention
- Model-selection evaluation may precede final testing
- Deployment monitoring
- Return to preparation/modeling when needed
- No future/test outcome leakage

Repair wording before trusting answer keys

• Overlapping answer mechanisms
- ARIES/deferred update both log-based
- Restart → checkpoints/rollback/redo cooperation
- NO-UNDO/REDO = redo-only
- Best/important/common/faster → criterion/workload
- Literal scope over familiar-term uniqueness

• Minimum missing assumptions
- Vendor behavior → exact DBMS/version
- Constructors → OID/type convention
- Waiting → requester/holder ages
- Recovery → durable log/page state/policy
- Distribution → fragments/network cost

- Multiple true descriptions → genuine ambiguity
- Index → question-specific clarifications/corrections
- No whole-question reproduction needed