CS Exit Ethiopia

Complete Study Notes

Computer Science Exit Examination

1 selected course | 18 November 2026

Selected courses

01. Fundamental of 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.

Fundamental of 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.

Fundamental of Database Systems

Complete Study Notes

FDB01 | Database approach, benefits and costs

A database is a coherent collection of related facts with meaning for an application.

A DBMS is software that defines, stores, retrieves, updates, protects and shares that data.

The database is the content; the DBMS manages it.

A database system also includes applications, people and infrastructure.

Traditional file applications define their own formats and access procedures.

Separate files can repeat facts, disagree after updates, isolate useful data and make shared integrity or concurrent updates difficult.

A DBMS centralizes metadata and supports multiple views, authorization, transactions, backup and recovery.

It controls redundancy rather than guaranteeing zero duplication.

Complexity, administration, storage and licensing can outweigh benefits for a tiny fixed application; lightweight embedded DBMSs also exist.

Concurrent access itself is useful; uncontrolled concurrent access causes anomalies.

The DBA manages access authorization, backup/recovery, availability, monitoring, tuning and routine maintenance, coordinating database use; designers define the schema and application developers build user-facing programs.

Fundamental of Database Systems module

Advanced Database Systems module

FDB02 | Data models and product names

A data model supplies concepts for structure, relationships, constraints and operations.

Conceptual ER models describe entities and relationships.

Relational models use tables; hierarchical models organize records in a parent-child tree; classic network models connect records through pointer-based owner/member links, allowing multiple parents.

Object models represent objects with identity, state and behavior.

A physical model specifies record layouts and access paths.

These are different representation choices, not stages of one query.

PostgreSQL, MySQL, Oracle Database, Microsoft SQL Server and IBM Db2 are relational DBMS products.

MongoDB is a document DBMS.

Java is a programming language.

IBM alone names a vendor, not a database product.

Distinguish a vendor such as IBM from a specific database product.

Fundamental of Database Systems module

Advanced Database Systems module

FDB03 | Relations, schemas, instances and metadata

A relation is a set of tuples, represented as a table.

A tuple/row/record contains related attribute values; an attribute/column/field describes a property.

Degree counts columns; cardinality counts rows.

Classical relations have distinct tuples, atomic cells and domains for attributes; row order has no meaning.

Column order has no logical meaning when attributes are identified by name.

SQL tables can contain duplicate rows unless constrained.

A schema describes names, structures and constraints; an instance, state or snapshot is the data at a particular time.

Inserting a student changes the state, while adding a column changes the schema.

Metadata is data describing data.

The catalog or data dictionary records table definitions, types, constraints and often access privileges and optimizer statistics.

A schema diagram draws tables and key references; labels such as database diagram can overlap.

Fundamental of Database Systems module

Advanced Database Systems module

FDB04 | Three-schema architecture and data independence

The external level gives each user group a selected view.

The conceptual level describes the whole logical database: entity types, relationships, data types and constraints.

The internal/physical level describes files, record placement and access paths.

These are abstraction levels, distinct from client/server deployment tiers.

Physical data independence permits changing internal storage without changing the conceptual schema, for example adding an index.

Logical data independence permits conceptual changes while preserving affected external views and application interfaces, for example splitting a table while maintaining a compatible view.

It is harder because applications depend on logical meaning.

Data abstraction hides irrelevant details; program-data insulation separates data definitions from application code.

Fundamental of Database Systems module

Advanced Database Systems module

FDB05 | Requirements through implementation

First collect business facts, required transactions, reports, volumes, privacy rules and quality targets.

Conceptual design turns them into an ER/EER model.

Logical design maps the conceptual model to relations, keys and constraints and checks normalization.

Physical design chooses file organization, indexes and storage.

Implementation creates and loads the actual objects; testing checks requirements.

Logical relational design is tied to a data model but can remain independent of a particular DBMS.

Vendor-specific implementation details may be introduced during later refinement.

Whether logical design is DBMS-dependent depends on the stated design methodology.

Conceptual design usually proceeds from a whole enterprise view downward; normalization can refine structures from dependencies upward.

These are useful approaches rather than mandatory directions for every modeling activity.

Fundamental of Database Systems module

Advanced Database Systems module

FDB06 | Entities, attributes and ER notation

An entity is a distinguishable real or conceptual thing, such as one student or one course.

An entity type describes common properties; its current collection of entities is an entity set.

A relationship associates entities.

In Chen notation rectangles represent entity types, diamonds relationships and ellipses attributes; double ellipses denote multivalued attributes and dashed ellipses derived attributes.

Always inspect the diagram legend.

Simple versus composite asks whether a value has meaningful components: Address may split into City and Street.

Single-valued versus multivalued asks how many values one entity may have: a student may have several phone numbers.

These dimensions differ: one address may be composite and single-valued.

Age derived from birth date is derived, even if also single-valued.

Attribute names alone do not establish multiplicity.

Fundamental of Database Systems module

Advanced Database Systems module

FDB07 | Superkeys, candidate keys and primary keys

A superkey uniquely identifies a tuple.

A candidate key is a minimal superkey: removing any component loses uniqueness.

Minimal means no unnecessary attribute, not necessarily the fewest columns among all keys.

A primary key is one selected candidate key; remaining candidates are alternate keys.

A composite key contains multiple attributes and need not be the chosen primary key.

If ID is unique, both {ID} and {ID,City} are superkeys, but only {ID} is minimal.

A table can have several candidates and foreign keys but at most one declared primary-key constraint, possibly containing several columns.

String columns can be keys.

Prefer stable, compact identifiers when practical, but Email, SSN or EmployeeID is a candidate only if business rules guarantee uniqueness.

A composite-key example must include multiple columns; a single-column key is not composite.

Example

Worked example

- Given candidate keys CustID, Email and ContactNo, all are eligible primary keys. CustID is preferable only with an explicit stability/size policy; uniqueness alone does not choose it.

Fundamental of Database Systems module

Advanced Database Systems module

FDB08 | Weak entities and identifying relationships

A weak entity lacks a complete independent key.

Its owner is normally a strong entity, so owner entity set and strong entity set may both describe the associated set.

Its partial key, also called discriminator, distinguishes weak entities of the same owner.

Identification combines the owner key and partial key.

For Dependent(EmployeeID,DependentName,BirthDate), suppose names are unique only within one employee's dependents.

Then (EmployeeID,DependentName) is the key, and EmployeeID is a foreign key.

The weak entity participates totally in its identifying relationship.

Chen notation uses a double rectangle, double identifying diamond and dashed underline for a partial key.

A weak entity is not merely any table with a foreign key.

Fundamental of Database Systems module

Advanced Database Systems module

FDB09 | Cardinality, participation and direction

Cardinality specifies maximum associated occurrences: 1:1, 1:N or M:N. Minimum cardinality specifies optional/partial participation (zero permitted) or mandatory/total participation (at least one).

A maximum of one does not mean participation is compulsory.

Relationship degree counts participating entity roles: unary recursive, binary or ternary.

Read business rules in both directions.

If each employee belongs to one department and each department has many employees, Employee→Department is N:1; Department→Employee is 1:N. Building→Location could also be N:1, so names alone cannot select a unique example.

Under the common arrow convention, arrows from advisor to both Instructor and Student impose maximum-one at both ends, giving 1:1.

Without a notation convention, arrows are unsafe evidence.

Fundamental of Database Systems module

Advanced Database Systems module

FDB10 | Map entities, composite and multivalued attributes

Create one relation per strong entity type, preserve its simple attributes and select a candidate key.

Expand composite attributes into their useful simple components; do not copy a composite ER structure unchanged into a flat atomic design.

Derived values can usually be calculated, avoiding stale stored copies.

Map each multivalued attribute to a separate relation containing the owner key and one value.

StudentPhone(StudentID,Phone) has primary key (StudentID,Phone) and a foreign key to Student.

Store separate rows for two phone numbers, not a comma-separated list or repeated Phone1/Phone2 columns.

Include the relational mapping step when translating an ER/EER design into tables.

Table names may follow entity names; naming does not invalidate a mapping.

Fundamental of Database Systems module

Advanced Database Systems module

University of Texas: ER-to-relational mapping

FDB11 | Map 1:1 and 1:N relationships

For 1:N, place the one-side key as a foreign key in the many-side table; store relationship attributes there.

Department(DeptID) and Employee(EmpID,DeptID,HireDate) implement many employees per department.

If participation is total, make the reference NOT NULL.

If partial, a nullable reference can represent no assignment.

For 1:1, one side may hold the other key with a UNIQUE constraint.

Prefer the totally participating side to reduce NULLs; NOT NULL enforces that side's mandatory reference.

Merging entities is reasonable only when their lifecycle and participation rules permit it.

For A→B many-to-one, A can absorb the relationship reference under either total or partial participation; requiring total participation as the only possible case overstates the rule.

Fundamental of Database Systems module

Advanced Database Systems module

FDB12 | Map M:N, recursive and n-ary relationships

A binary M:N relationship needs an association table in addition to the two entity tables.

Enrollment(StudentID,CourseID,Grade) references Student and Course; (StudentID,CourseID) is a key if one enrollment per pair is allowed.

Repeated attempts require Term or AttemptNo in the key.

This ordinarily produces three tables.

A recursive M:N friendship uses UserLink(UserID,OtherUserID), two foreign keys to User; decide whether links are directed and whether self-links are allowed.

A recursive 1:N supervision uses a nullable SupervisorID reference in Employee.

For a ternary Supply(SupplierID,PartID,ProjectID,Quantity), preserve all participating references; binary replacements can lose which triple occurred.

Choose the association key from actual cardinalities rather than automatically assuming every reference is necessary.

Fundamental of Database Systems module

Advanced Database Systems module

University of Texas: ER-to-relational mapping

FDB13 | EER inheritance and category constraints

EER extends ER with superclass/subclass structure.

A subclass inherits the superclass attributes and relationships.

Specialization divides Student into Undergraduate and Graduate; generalization combines common properties into Student.

Disjoint subclasses cannot overlap; overlapping subclasses can.

Total specialization requires every superclass member in some subclass; partial specialization does not.

Map Person(PersonID,Name) and Student(PersonID,Program) with Student.PersonID both primary key and foreign key to Person.

This preserves inherited identity.

A category/union subclass draws members from a union of different superclasses, such as Owner drawn from Person or Company; it differs from an intersection subclass requiring membership in both.

Object models also support inheritance; distinguish basic ER notation from EER when comparing models.

Structural EER inheritance alone does not define executable object methods.

Fundamental of Database Systems module

Advanced Database Systems module

FDB14 | Fan traps and chasm traps

A fan trap occurs when two 1:N relationships fan from a common entity and a path suggests an association it cannot determine.

Department→Employee and Department→Project do not reveal which employee works on which project; joining through Department pairs every employee with every project in that department.

Record WorksOn explicitly when that is the fact needed.

A chasm trap occurs when an apparently useful path fails for some occurrences because an optional relationship breaks it.

For example, a branch's property may be lost from a path that requires an assigned staff member when some properties have none.

Add the required direct relationship or revise the model.

Multiple 1:N relationships are legitimate; eliminating all fans is not a universal repair.

A missing path describes a chasm, not the defining ambiguity of a fan.

Fundamental of Database Systems module

Advanced Database Systems module

Dr G. Russell, Napier University: Database Systems notes

FDB15 | Integrity and constraint taxonomy

Domain integrity restricts type and allowed values.

Key integrity enforces uniqueness.

Entity integrity requires every primary-key component to be non-NULL.

Referential integrity requires a non-NULL foreign-key value to match an eligible referenced unique/candidate key; self-references are allowed.

A parent need not have a child.

Inherent constraints come from the data model; schema-based constraints are declared through the DDL; semantic/application rules may need triggers or application logic when DDL cannot express them.

Integrity improves data validity; authorization controls who may access it.

Uniqueness and domain validation are both integrity mechanisms.

If a Department primary key is NULL and Manager=107 lacks a referenced employee, entity integrity certainly fails and referential integrity also fails if that foreign key is declared.

One INSERT can violate multiple constraints.

Fundamental of Database Systems module

Advanced Database Systems module

FDB16 | Write and enforce SQL constraints

Example: CREATE TABLE Enrollment (StudentID INTEGER NOT NULL REFERENCES Student(StudentID), CourseID INTEGER NOT NULL REFERENCES Course(CourseID), Grade INTEGER CHECK (Grade BETWEEN 0 AND 100), PRIMARY KEY (StudentID,CourseID)); Each column declaration gives its name then type, with commas between declarations.

CHECK is a constraint clause, not a standalone GRANT-like statement.

POSITIVE is not a general SQL constraint keyword; CHECK (Amount > 0) expresses positivity.

PRIMARY KEY implies uniqueness and non-NULL.

UNIQUE does not generally imply NOT NULL; NULL treatment differs by DBMS.

A CHECK can pass when its result is unknown, so add NOT NULL if missing values must be forbidden.

Parent updates/deletes may use NO ACTION/RESTRICT, CASCADE or SET NULL.

A referenced key is not inherently unchangeable: legality depends on matching children and the declared action.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Constraints

FDB17 | Functional dependencies and attribute closure

X→Y means every legal pair of tuples agreeing on X also agrees on Y. It is a business constraint, not a correlation observed in one sample.

Full dependency requires all determinant attributes; partial dependency holds through a proper subset.

A prime attribute belongs to some candidate key; a nonprime attribute belongs to none.

To compute X^+, start with X and repeatedly add the right-hand side of any dependency whose left side is already present.

Stop when nothing changes.

X is a superkey exactly when X^+ contains the whole relation.

Remove attributes and recompute to test minimality.

Reflexivity gives X→Y when Y⊆X; augmentation permits XZ→YZ from X→Y; transitivity derives X→Z from X→Y and Y→Z.

Fundamental of Database Systems module

Advanced Database Systems module

FDB18 | Find all keys before deciding normal form

For R(A,B,C,D), let C→D, C→B and B→C. A never appears on a right-hand side, so every key needs A. (AB)+ gives ABCD through B→C→D; (AC)+ also gives ABCD.

AB and AC are minimal keys.

BC cannot derive A and is not a key.

A, B and C are prime; D is nonprime.

D depends on C alone and also on B through transitivity, proper subsets of those keys.

Thus 2NF fails; assuming atomic values, the highest form is 1NF.

Use the stated functional dependencies to justify each key and normal-form conclusion.

Confirm the actual dependency set before calculating closures; do not infer dependencies from unclear symbols.

Example

Worked example

- For F={A→B,B→C} on R(A,B,C), A+=ABC makes A the key. The chain through nonkey B causes a 3NF violation. If instead B were a candidate key, a dependency chain would not by itself prove such a violation.

Fundamental of Database Systems module

Advanced Database Systems module

FDB19 | Insertion, deletion and update anomalies

Suppose Enrollment repeats StudentName for every course.

An update anomaly requires changing every copy of the student's name; missed copies disagree.

An insertion anomaly prevents recording a new student until some enrollment exists.

A deletion anomaly loses the only record of a student's details when their last enrollment is deleted.

Revoke anomaly is not a normalization anomaly.

Normalization separates facts according to their determinants, reducing avoidable repetition and associated anomalies.

It does not require seeing bad current data: declared dependencies suffice.

Adding a generated row ID may make insertion technically possible, but does not remove repeated facts or fix the dependency design.

Reduced redundancy can improve integrity and storage use, but extra keys and joins mean neither storage savings nor faster retrieval is guaranteed.

Fundamental of Database Systems module

Advanced Database Systems module

FDB20 | First and second normal forms

1NF requires atomic values in the chosen relational domains and no repeating groups in cells.

It does not mean merely declaring a primary key.

A mapped ER design that correctly splits composite and multivalued structures gives ordinary 1NF tables; ER drawing alone guarantees no higher form.

2NF requires 1NF and no nonprime attribute partially dependent on any candidate key.

For Enrollment(SID,CID,SName,CTitle,Grade), assume key (SID,CID), SID→SName and CID→CTitle.

Split into Student(SID,SName), Course(CID,CTitle) and Enrollment(SID,CID,Grade).

Grade still depends on the whole pair.

A relation whose every candidate key is single-attribute has no nonprime partial-key dependency.

Higher forms inherit 2NF, so no partial dependencies establishes the minimum guaranteed stage only when 1NF is also assumed.

Fundamental of Database Systems module

Advanced Database Systems module

FDB21 | Third normal form, BCNF and higher forms

For every nontrivial FD X→A, 3NF permits it when X is a superkey or A is prime.

BCNF requires X to be a superkey.

The mnemonic whole key and nothing but the key describes the common single-key case; it cannot replace the full tests when keys overlap.

Remove dependencies of nonprime facts on nonkey determinants.

In Sale(Date,Customer,Product,Vendor,VendorCity,SalesRep), assume key (Date,Customer,Product), Product→Vendor and Vendor→VendorCity.

Product is a proper subset of the key, so Vendor and VendorCity violate 2NF.

With atomic values the highest form is 1NF, even though a transitive dependency also exists. 4NF addresses nontrivial multivalued dependencies with nonsuperkey determinants; 5NF addresses join dependencies.

Do not confuse a multivalued dependency with a list stored inside a cell.

Example

Worked example

- Employee(EID,DeptID,DeptName), EID→DeptID and DeptID→DeptName, splits into Employee(EID,DeptID) and Department(DeptID,DeptName). Each department name is stored once.

Fundamental of Database Systems module

Advanced Database Systems module

FDB22 | Decomposition: lossless and dependency preserving

A lossless decomposition reconstructs exactly the original legal tuples by joining its parts; a lossy one creates spurious combinations.

For a binary decomposition into R1 and R2 under FDs, it is lossless when their common attributes determine all of R1 or all of R2.

Sharing a column without this condition is insufficient.

Dependency preservation means the original constraints can be enforced on the individual relations without joining them.

Losslessness and dependency preservation are separate properties.

A practical 3NF synthesis uses a minimal FD cover, creates a relation for each determinant and its dependent attributes, removes contained relations, and adds a relation containing a candidate key if none already does.

BCNF decomposition removes an offending X→Y by separating XY and the remaining attributes with X; it can sacrifice dependency preservation.

Fundamental of Database Systems module

Advanced Database Systems module

Jan Chomicki, University at Buffalo: Relational database design

FDB23 | Choose data types and storage levels

CHAR(n) is fixed length; VARCHAR(n) has variable length up to a limit.

DATE represents a date and BLOB binary content such as images.

INTEGER represents whole numbers; DECIMAL(p,s) represents exact decimal values, making it suitable for money when scale is specified.

FLOAT/REAL/DOUBLE are approximate; names, widths and precision depend on the DBMS.

Numeric type widths depend on the DBMS and type; do not assume one universal width.

Primary storage includes RAM and processor cache and is usually volatile.

Secondary storage includes persistent SSDs/HDDs used online; database pages are copied into buffers before CPU processing.

Tertiary storage includes tape or removable archive media.

Choose types from meaning, range and accuracy requirements, not a blanket rule that integers are always faster than strings.

Fundamental of Database Systems module

Advanced Database Systems module

FDB24 | Records, pages and primary file organization

A page/block is a unit of database I/O and holds several records.

For fixed unspanned records of size r in usable page space B, the blocking factor is floor(B/r), and n records occupy ceil(n/blocking factor) pages.

Page headers and slots reduce usable space.

A heap has no search-key order and supports simple inserts, but searches may scan all pages.

A sorted file supports efficient range scans and binary search on the ordering field, with more costly insertions or overflow handling.

A hashed file chooses a bucket by hash(key), supports equality lookup well and requires collision/overflow handling; hash order does not support ordinary ranges.

Choose organization from actual reads, writes and predicates.

Partitioning splits a large dataset; replication copies it; migration moves it; archiving removes older data from active service.

Fundamental of Database Systems module

Advanced Database Systems module

FDB25 | Ordered, dense, sparse and multilevel indexes

An index stores search keys with pointers or row identifiers.

It often speeds selective retrieval but consumes space and must be maintained on writes.

A full scan can win for a tiny table or a query returning most rows.

Indexes do not themselves establish referential integrity.

In ordered-file indexing terminology, a primary ordered index is on the file's ordering key and can have one entry per page, hence sparse.

A clustering index is on a nonunique ordering field, often with an entry per distinct value.

Secondary indexes provide other access paths and must represent all searchable values with record pointers or pointer lists.

A dense index represents each search value; sparse indexes rely on ordered data between entries.

Multilevel indexes index lower index levels.

B/B+ trees support ordered and range search; hashing targets equality; bitmap indexes suit some analytic workloads.

No one type is universally best.

Fundamental of Database Systems module

Advanced Database Systems module

FDB26 | SQL command categories and exact scope

DDL changes structure: CREATE, ALTER, DROP, commonly RENAME and TRUNCATE in textbook classifications.

DML retrieves or changes data: SELECT, INSERT, UPDATE, DELETE.

Some texts separate SELECT as DQL.

DCL manages privileges through GRANT and REVOKE.

TCL includes COMMIT, ROLLBACK and SAVEPOINT.

Classify a statement by its effect, not its length.

DELETE removes rows and keeps the table; DROP TABLE removes the table object; ALTER changes its definition.

TRUNCATE empties a table using implementation-specific mechanisms.

PostgreSQL allows transactional rollback of TRUNCATE, so no rollback information is retained is not a universal SQL property.

SQL Server permits DROP DATABASE Sales, suppliers; that comma-list syntax is vendor-specific.

SQLite's .databases lists attached names/files in its command-line shell; .database is not the documented command.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: TRUNCATE

Microsoft: DROP DATABASE (Transact-SQL)

SQLite: Command-line shell

FDB27 | SELECT and its result columns

SELECT Name,Address FROM Customer; produces only those mailing-label fields.

SELECT * selects all columns; DISTINCT removes duplicate result rows; AS supplies an alias; ALL permits duplicates and does not mean all columns.

SQL normally preserves duplicates without DISTINCT.

A query reading table columns needs an appropriate FROM source.

A literal-only query such as SELECT 1 may legally omit FROM.

Typical written clause order is SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY ….

Conceptually form FROM/JOIN rows, filter WHERE, group, filter HAVING, select expressions and then order the result.

SELECT alone normally reads without changing the database state.

WHERE alone is a clause rather than a complete read-only statement, so it cannot be compared as an equivalent executable command.

Fundamental of Database Systems module

Advanced Database Systems module

FDB28 | Predicates, strings and NULL

Use single quotes for SQL string literals: WHERE ID = 'B001'.

Standard double quotes delimit identifiers, although some vendor modes accept them for strings.

Qualify ambiguous columns with aliases such as E.Department.

Conditions use =, <>, <, >, AND, OR and NOT; use parentheses to make precedence clear.

LIKE 'Ab%' matches a prefix, and BETWEEN includes its endpoints.

NULL represents a missing/unknown value.

Comparisons such as Salary = NULL are unknown, not true; use IS NULL or IS NOT NULL.

WHERE retains only true conditions.

An index-friendly year predicate for timestamps is BirthDate >= DATE '2000-01-01' AND BirthDate < DATE '2001-01-01', avoiding omitted times on December 31.

Expression indexes can support functions, so functions do not universally prevent indexing.

Fundamental of Database Systems module

Advanced Database Systems module

FDB29 | INSERT, UPDATE and DELETE

INSERT INTO Hotel(HotelID,HotelName,SubCity) VALUES (1,'Hilton','Yeka'); pairs each named column with a compatible value.

Counts and types must match; keys and other constraints still apply.

Without a column list, values must follow the table's declared column order.

UPDATE Book SET Price = 600 WHERE ID = 'B001'; changes matching rows.

UPDATE without WHERE targets every row.

DELETE FROM Publisher WHERE Name = 'XWZ'; deletes matching publisher rows, subject to reference policies.

DELETE FROM r; removes every row but leaves r defined.

Both clear relation entries and delete rows describe that effect.

Several UPDATE statements can all be valid while performing different tasks; most appropriate requires a requested change and a specified quoting dialect.

Fundamental of Database Systems module

Advanced Database Systems module

FDB30 | CREATE and ALTER tables

CREATE TABLE Employee (EmpID INTEGER PRIMARY KEY, Name VARCHAR(80)); declares two columns.

The name and its type stay together; the comma separates whole column declarations.

To add a primary key later: ALTER TABLE Employee ADD PRIMARY KEY (EmpID); existing data must be unique and non-NULL.

A table cannot gain a second primary key without replacing the existing constraint.

ALTER TABLE Employee ADD COLUMN Email VARCHAR(120); adds a column in PostgreSQL-style syntax.

ALTER TABLE Employee DROP COLUMN Email; removes it.

ALTER TABLE Employee ADD CONSTRAINT uq_email UNIQUE (Email); adds a named constraint.

Use ALTER rather than DALTER, and do not separate column_name and column_type with a comma; validate the resulting SQL before execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB31 | Join behavior and preservation

An inner join returns matching pairs satisfying ON.

An equijoin uses equality; a theta join permits another comparison.

A natural join is ordinarily an inner equality join on every same-named column and merges those duplicate columns.

JOIN ... USING (DeptID) specifies shared join columns explicitly.

CROSS JOIN forms every pair without a matching condition.

LEFT OUTER JOIN also preserves unmatched left rows, padding right columns with NULL; RIGHT preserves right rows and FULL preserves both.

Therefore inner and natural inner joins both discard unmatched rows.

A self-join uses aliases for two occurrences of one base table.

Each ordinary join operator combines two inputs, but its inputs can be earlier joins, so one query can join many tables.

Distinguish base-table count, table occurrences and binary operator arity.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Table expressions

FDB32 | Aggregation, GROUP BY and HAVING

COUNT(*) counts rows, COUNT(x) counts non-NULL x values; SUM and AVG summarize non-NULL numeric values.

SQL's usual average function is AVG, not AVERAGE.

GROUP BY creates one group per distinct grouping value; HAVING tests aggregate groups after WHERE has filtered individual rows.

SELECT Department, AVG(Salary) FROM Employee WHERE Department <> 'Sales' GROUP BY Department HAVING AVG(Salary) > 35000; lists qualifying department averages.

An ordinary AVG(Salary) predicate cannot be placed directly in WHERE.

Selecting EmpID, FirstName and LastName while grouping only by Department is also invalid unless the DBMS can establish the required functional dependence.

The grouping column need not itself appear in SELECT; check the functional dependence of other selected columns.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: GROUP BY and HAVING

FDB33 | Subqueries and views

A subquery is a SELECT nested inside another statement.

It may appear in WHERE, HAVING or FROM, not only the first two.

IN tests membership; EXISTS tests whether at least one qualifying row exists.

Correlated subqueries refer to the outer row.

A scalar subquery must produce an appropriate single value.

Beware NOT IN when its subquery can return NULL; NOT EXISTS often expresses the intended absence more safely.

CREATE VIEW CSStudent AS SELECT StudentID,Name FROM Student WHERE Major = 'CS'; stores a query definition and presents a virtual table.

Views can simplify repeated queries, expose derived values and restrict access when configured correctly.

An ordinary view does not necessarily store its result; a materialized view does.

Rewriting a subquery as a join can change duplicates, and does not guarantee faster execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB34 | Set operations and type conversion in SQL

SELECT ID FROM A UNION SELECT ID FROM B; combines compatible result columns and removes duplicate rows by default.

UNION ALL retains duplicates.

Explicit UNION DISTINCT is accepted in some dialects, including PostgreSQL; it is not a separate algebra rule.

INTERSECT returns common rows; EXCEPT returns first-result rows absent from the second.

Inputs need the same column count and compatible corresponding types.

CAST('42' AS INTEGER) converts an expression to a requested type if the value permits it.

CONVERT is vendor-specific and can also perform type conversion; specify the SQL dialect when comparing it with CAST.

In SELECT xyz FROM table1 UNION xyz FROM table2, the second SELECT is missing.

In the corrected query, xyz is a selected column/expression, not a table name.

Fundamental of Database Systems module

Advanced Database Systems module

PostgreSQL: Combining queries

PostgreSQL: Value expressions

FDB35 | Relational algebra: calculate results

Selection σ filters tuples; projection π retains attributes and removes duplicate tuples under set semantics.

Rename ρ changes relation or attribute names.

Union, difference and intersection require union-compatible schemas.

Difference R−S is not commutative; union and intersection are.

Inner joins are commutative up to attribute order when predicates are adjusted; left outer joins are not generally commutative.

Identify the join variant before predicting its behavior.

A Cartesian product has $|R|\times|S|$ tuples and degree degree(R)+degree(S), renaming conflicting attributes if necessary.

Thus $200\times100=20{,}000$ and $5\times7=35$.

Join equals selection over a product; intersection can be derived from difference.

A common minimal basis contains selection, projection, union, difference, product and rename.

Assignment is a useful notation, not an indispensable primitive; lists of basic operations vary across textbooks.

Fundamental of Database Systems module

Advanced Database Systems module

FDB36 | Relational calculus: state the desired facts

Relational algebra specifies a sequence of operations and is procedural at the logical level.

Relational calculus specifies a predicate describing desired results and is declarative/nonprocedural.

A tuple variable ranges over tuples; a domain variable ranges over individual attribute values.

Safe/range-restricted expressions produce results bounded by the database's active domain.

Tuple form {s | Student(s) AND s.Major='CS'} selects student tuples with that major.

A domain form can request names n satisfying $\exists$i,m (Student(i,n,m) AND m='CS').

Existential quantification means some matching tuple exists; universal quantification means every required case meets a condition.

SQL is declarative even though an optimizer later chooses a procedural execution plan.

Fundamental of Database Systems module

Advanced Database Systems module

FDB37 | Transactions and the ACID properties

A transaction groups operations into one logical unit.

Atomicity applies all its effects or none.

Consistency means a correct transaction preserves declared invariants.

Isolation controls interference between concurrent transactions; serializable isolation provides behavior equivalent to some serial execution.

Durability preserves committed effects after a crash, typically through durable logging rather than forcing every data page immediately.

For a transfer, subtracting 100 from one balance and adding 100 to another must commit together; losing either half violates atomicity and may violate the total-balance invariant.

BEGIN starts work, COMMIT completes it, ROLLBACK undoes uncommitted work and SAVEPOINT marks an intermediate rollback point.

A failed transaction may roll back while other users' committed transactions remain.

SQL transaction syntax and error-handling constructs differ by DBMS; @@ERROR/IF is SQL Server-specific.

Fundamental of Database Systems module

Advanced Database Systems module

FDB38 | Choose a design technique from the requirement

Normalization reduces dependency-driven anomalies.

Indexing supplies search paths.

Denormalization deliberately repeats or precomputes facts to reduce selected read costs, increasing maintenance work.

Partitioning divides rows or columns; replication creates copies to improve locality or availability while complicating updates.

These techniques can coexist.

An e-commerce or gaming label alone does not select one.

Privacy requires a threat model.

Authentication verifies identity; role-based authorization limits allowed records/actions; encryption protects data at rest or in transit; auditing records activity.

Both encryption and access control can protect patient data.

Scalability, availability, fast retrieval, redundancy control and security are all reasonable design considerations.

Functional requirements describe behavior; nonfunctional requirements quantify qualities such as 99.9% uptime or response time.

Authentication and encryption mechanisms alone do not uniquely establish that distinction.

In a CMS, tags/categories describe meaning, while foreign keys preserve valid references between content and tags.

A ContentTag(ContentID,TagID) association models multiple tags per item; these functions complement each other.

Fundamental of Database Systems module

Advanced Database Systems module

FDB39 | Integrated design: student registration and dormitory allocation

State the rules first: each student belongs to one department, may enroll in many courses, and has at most one current dorm assignment.

Each course accepts many students; each dorm holds at most six students.

Identify Department, Student, Course and Dorm entities; Enrollment is the M:N relationship with Grade.

Use Department(DeptID,Name), Student(SID,Name,DeptID,DormID), Course(CID,Title), Enrollment(SID,CID,Grade) and Dorm(DormID,FloorNumber).

Declare all entity IDs primary keys; Student.DeptID and Student.DormID reference their parents; Enrollment has composite key (SID,CID) and both references.

Make DormID nullable until assignment.

Separate StudentPhone(SID,Phone) for multiple contacts.

Normalize repeated department/course facts into their owning tables.

A simple foreign key does not enforce the six-student capacity; use a concurrency-safe trigger, allocation transaction or explicit slot design with six unique slots per dorm.

Example

Worked example

- Complete the reference as FOREIGN KEY (DormID) REFERENCES Dorm(DormID). STUDENT is the referencing child; DORM is referenced. Consistent names and an added Student.DormID column are required before execution.

Fundamental of Database Systems module

Advanced Database Systems module

FDB40 | Check SQL syntax and question assumptions

Justify each answer using the schema, SQL rules and stated assumptions.

Owner and strong entity can both describe the entity supporting a weak entity; row and record are synonyms in a relational table; difference and set-difference are the same operator.

DELETE, UPDATE and the invalid command REMOVE are not DROP; a question asking which is not DROP can therefore admit several choices.

Compare all choices with the relevant rule before selecting an answer.

For invalid SQL, identify the syntax error and validate the corrected statement before analyzing its result.

For unsupported cardinalities, request business rules and direction.

For best/efficient/important questions, specify workload, quality goal or comparison baseline.

A diagram-dependent conclusion needs the missing diagram or an explicit description of its relevant symbols.

Identify the smallest clarification needed when a question does not support a unique answer.

Fundamental of Database Systems module

Advanced Database Systems module

FDB-S01 | Algebra operator lists and derivable operations

A list of basic relational-algebra operators is a teaching convention, distinct from a claim that every listed operation is logically indispensable. The course module uses selection, projection, Cartesian product, set difference, union and intersection as its basic-operation list. Natural join and assignment are not members of that particular list.

Under set semantics, union includes tuples from either compatible input, intersection retains tuples common to both, and difference keeps first-input tuples absent from the second. Intersection can be expressed using difference, while a matching join can be expressed as selection over a Cartesian product. A smaller primitive basis may therefore omit intersection even when it is introduced as a basic operation.

Fundamental of Database Systems module · page 24

FDB-S02 | Relational systems with richer types

An object-relational DBMS retains relational data management while supporting richer extensions such as user-defined data types, functions and operators. PostgreSQL is an example. Richer relational types do not by themselves require all data to form a parent-child tree; that representation is the defining concern of hierarchical modeling.

PostgreSQL documentation - What Is PostgreSQL?