



KAMALDNP

kamaldnp.com • IT Training, Varanasi

PDF created by Raj Kamal using AI

Short Notes — Database Management System

BTEUP Diploma in Computer Science & Engineering • 3rd Semester • Short Notes
Session 2026-27

Short Revision Notes — prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus. This is NOT a BTEUP board paper or publication.

These **short revision notes** cover all five units of Database Management System from the official BTEUP 2026-27 syllabus for CSE 3rd semester, in syllabus order. Each point gives the definition, key terms and small examples that a short or five-mark answer needs, including short SQL syntax where a topic is a command family.

Use them with the **official syllabus PDF** and the matching practice paper: revise one unit, recall its points and draw its diagrams from memory, then write the likely question at the end of that unit. These are revision notes only, not a replacement for your textbook, SQL practice and class notes.

Unit 1 — Introduction (6 Periods, 10%)

Topics: Database system concepts and architecture; components of DBMS; table structure; schema definition; three views of data (external view, conceptual view, internal view).

Database system concepts and architecture

A **database** is an organised collection of related data stored so that it can be accessed, managed and updated easily. A **DBMS (Database Management System)** is the software that creates and manages the database — examples are MySQL, Oracle and PostgreSQL. Together, database + DBMS + application programs + users form a **database system**.

Why not simple files? A file system suffers from **data redundancy** (same data repeated), **inconsistency** (copies disagreeing), difficult access, data isolation, integrity problems and weak security. A DBMS reduces all of these by storing data once, in a controlled way.

Architecture in brief: users and applications give queries through the DBMS; the query processor and DDL/DML compilers process them; the storage manager and transaction manager work with the data files, indexes and data dictionary stored on disk. Draw this as a block diagram with users at the top and the database on disk at the bottom.

Components of DBMS

The main **components of a DBMS** are grouped as hardware, software, data, procedures and people, and inside the software as functional components:

- **Query processor** — interprets DDL statements and compiles DML queries into low-level instructions; includes the DDL interpreter, DML compiler and embedded DML pre-compiler.
- **Storage manager** — interfaces between low-level stored data and the queries; it manages authorisation and integrity, transactions, files and buffers.
- **Disk storage** — holds the data files, the **data dictionary** (information about the structure of the database), indexes and statistical data.
- **Transaction manager** — ensures the database stays consistent even on failure or concurrent access.

People: the database administrator (DBA) controls the whole system; designers design the schema; application programmers and end users use the data. In an answer, name the component first and then write one line on its job.

Table structure

In the relational view, data is stored in **tables**. A table is a collection of related records arranged in rows and columns.

- **Row (tuple / record)** — one complete record, for example all details of one student.
- **Column (attribute / field)** — one property of the record, for example RollNo, Name or Course; each column has a fixed data type and a domain of allowed values.
- **Cell** — intersection of a row and a column; it holds a single value in a well-designed table.
- **Degree** — number of columns in the table; **cardinality** — number of rows in the table.

Example — STUDENT table:

RollNo	Name	Course
101	Aarav	CSE
102	Meera	CSE

Rules of a proper table: no two rows should be fully duplicate, column names should be unique within the table, and the order of rows and columns does not matter. One column or a set of columns identifies each row — this leads to the idea of keys in Unit 2.

Schema definition

A **schema** is the overall design or structure of a database — it describes what tables exist, what columns they have, their data types and the constraints on them. The schema is defined once, using the **Data Definition Language (DDL)**, and changes rarely. The actual data inside the tables, which changes daily, is called the **instance** of the database.

Easy distinction: schema is like the blank format of a marksheet (headings printed once); the instance is the filled marksheet of one student at one moment.

- **Physical schema** — how data is actually stored in files and indexes.
- **Logical schema** — what tables, columns and relationships exist; programmers and the DBA work at this level.
- **Sub-schema (view schema)** — the part of the database that a particular user group sees.

The description of the schema is stored in the **data dictionary / system catalog**, which the DBMS consults before accessing any data.

Three views of data (external view, conceptual view, internal view)

A DBMS gives **data abstraction** — it hides storage details and shows data at three levels, called the three views (ANSI/SPARC architecture):

- **External view (user view)** — the highest level. Each user or user group sees only the part of the database it needs, in its own form. For example, a student sees his marks; a clerk sees fees. Many external views can exist for one database, and this level also improves security.
- **Conceptual view (logical view)** — the middle level. It describes **all** data in the database and the relationships among them — all tables, columns, keys and constraints — without saying how they are stored. It is defined by the conceptual schema.
- **Internal view (physical view)** — the lowest level. It describes how data is actually stored: file structures, indexes, record formats and access paths on disk.

Benefit: **data independence** — change the internal storage without changing the conceptual schema (physical independence), or change the conceptual schema without changing most external views (logical independence).

Likely question: Explain the three views of data (external, conceptual and internal) with a diagram. How is a DBMS better than a traditional file system?

Unit 2 — Data Model and Keys (8 Periods, 25%)

Topics: Data model definition; network model, hierarchical model, E-R model; advantages and disadvantages of each data model; concept of keys (primary, candidate, super, foreign); constraints; strong entity set and weak entity set.

Data model — definition

A **data model** is a collection of concepts, rules and tools for describing data, the relationships among data, and the constraints on data in a database. In simple words, it is the plan that says how data will be organised before the database is actually built.

A data model has three parts:

- **Structure** — the form in which data is arranged (tables, trees, graphs or objects).
- **Operations** — the allowed actions on the data, such as insert, delete, update and retrieve.
- **Integrity constraints** — rules the data must always obey.

The syllabus covers three classical models: the **hierarchical model** (tree structure), the **network model** (graph structure) and the **E-R model** (a design model using entities and relationships), on which the relational model of Unit 3 is built. Choosing a data model is the first step of database design: it fixes what can be stored, how records connect, and how easily the design can grow or change later.

Network model

In the **network model**, data is represented as records connected by links in a **graph** structure. A record type is a collection of fields; records are grouped into **set types**, each having one owner record type and one or more member record types.

Its special feature: a member record can have **more than one owner**, so many-to-many relationships can be shown directly. Example: a STUDENT record can belong both to a COURSE set and to a HOSTEL set.

Advantages: handles many-to-many relationships naturally; data access can be fast because relationships are stored as pointers; a record can be reached through several paths.

Disadvantages: the structure becomes very complex as the database grows; the programmer must know the physical paths and navigate record by record; changing the structure is difficult and affects existing programs. Because of this complexity, the network model is mostly of historical and examination importance today; draw its diagram as boxes (records) joined by several crossing links, unlike the single-parent tree of the hierarchical model.

Hierarchical model

The **hierarchical model** organises records in a **tree** structure. Each record type has fields; a parent record can have many child records, but each child has **exactly one parent**. The topmost record, which has no parent, is the **root**. Relationships are therefore only one-to-many, from parent to child.

Example: College (root) → Department → Student. To reach a student, you must start at the root and follow the path downward.

Advantages: very simple and natural for genuinely one-to-many data; searching along a known path is fast; data integrity is easy to maintain because a child cannot exist without its parent.

Disadvantages: many-to-many relationships cannot be shown directly (a student taking many courses needs duplicated records, causing redundancy); inserting a child without a parent is not allowed; deleting a parent deletes all its children; the user must know the exact tree path. IBM's IMS is the classic example of this model.

E-R model

The **E-R (Entity-Relationship) model** is a high-level design model used to draw the blueprint of a database before creating tables. It describes the real world in three basic concepts:

- **Entity** — a real-world object that can be distinctly identified, such as Student, Course or Book. A collection of similar entities is an **entity set**.
- **Attribute** — a property of an entity, such as RollNo, Name or Price. Types: simple, composite (Address = City + PIN), derived (Age from Date of Birth), multivalued (many Phone numbers) and key attributes.
- **Relationship** — an association among entities, such as Student enrolls in Course. Its **degree** is the number of entity sets taking part; its mapping can be one-to-one, one-to-many or many-to-many.

Diagram symbols: rectangle = entity, ellipse = attribute (double ellipse = multivalued, dashed = derived), diamond = relationship, line = link; the key attribute is underlined. An E-R diagram is later converted into relational tables by the mapping rules of Unit 3.

Advantages and disadvantages of each data model

Write this as a comparison; examiners expect the three models side by side.

- **Hierarchical model — advantages:** simple structure, fast access along a fixed path, good for one-to-many data, parent-child integrity is automatic. **Disadvantages:** no direct many-to-many relationships, data duplication, child cannot exist without parent, rigid — hard to reorganise.
- **Network model — advantages:** many-to-many relationships shown directly through multiple owners, several access paths, less redundancy than the hierarchical model.

Disadvantages: very complex structure and navigation, programmer must know all paths, structural changes are difficult.

- **E-R model — advantages:** simple and clear picture of the real world, easy for designers and users to discuss, converts systematically into relational tables, independent of any particular DBMS. **Disadvantages:** it is a design tool only — it stores no data itself and gives no operations; large diagrams become crowded, and some constraints cannot be shown in the basic diagram.

Conclusion line for any answer: the relational model became popular because it keeps the clarity aimed at by E-R design while storing data in simple tables.

Concept of keys (primary, candidate, super, foreign)

A **key** is a column, or a set of columns, used to identify rows and link tables. Learn all four with one example — STUDENT(RollNo, AadhaarNo, Email, Name, CourseID):

- **Super key** — any set of columns whose values are unique for every row, for example {RollNo}, {RollNo, Name}. It may contain extra, unnecessary columns.
- **Candidate key** — a **minimal** super key: unique, with no column removable without losing uniqueness. Here RollNo, AadhaarNo and Email are all candidate keys. A table can have many candidate keys.
- **Primary key** — the one candidate key chosen by the designer as the main identifier, for example RollNo. Its values are always unique and **never NULL**.
- **Foreign key** — a column in one table that refers to the primary key of another table, for example CourseID in STUDENT referring to COURSE(CourseID). It creates the relationship between tables and enforces referential integrity.

Order to remember: every primary key is a candidate key, and every candidate key is a super key — but not the reverse.

Constraints

Constraints are rules applied to columns or tables so that only valid data enters the database. The DBMS checks them automatically on every insert and update, so wrong data is rejected at the source.

- **NOT NULL** — the column cannot be left empty.
- **UNIQUE** — all values in the column must be different.
- **PRIMARY KEY** — combination of NOT NULL + UNIQUE for the identifying column.
- **FOREIGN KEY** — the value must match a primary-key value in the referenced table, or be NULL where allowed.
- **CHECK** — the value must satisfy a condition, for example Marks BETWEEN 0 AND 100.
- **DEFAULT** — if no value is given, a fixed default value is stored.
- **Domain constraints** — the value must be of the declared type and within the allowed range/set.

- **Entity integrity** — no part of a primary key may be NULL; **referential integrity** — a foreign key must refer to an existing row.

In SQL, constraints are written inside CREATE TABLE (Unit 5), so connect this theory with that syntax in answers.

Strong entity set and weak entity set

An **entity set** is a collection of entities of the same type sharing the same attributes. Entity sets are of two kinds:

Strong entity set — an entity set that has its own primary key, so its entities can be identified independently. Example: STUDENT with key RollNo, COURSE with key CourseID. In an E-R diagram it is drawn as a simple **rectangle**.

Weak entity set — an entity set that has **no primary key of its own**; its entities can be identified only through a strong entity set, called its **owner (identifying) entity set**, to which it is linked by an **identifying relationship**. Example: DEPENDENT (Name, Age) of an EMPLOYEE — two employees can both have a dependent named Riya, so a dependent is identified only as “dependent of employee E101”. It has a **partial key (discriminator)**, such as DependentName, which together with the owner's key identifies it.

Diagram symbols: weak entity = **double rectangle**, identifying relationship = **double diamond**, partial key = dashed underline. Weak entities always have total participation in the identifying relationship.

Likely question: Differentiate between the hierarchical and network models. Define candidate key, primary key and foreign key with an example, and explain a weak entity set with its E-R diagram symbols.

Unit 3 — Relational Model (10 Periods, 25%)

Topics: The relational data model and relational database constraints; ER/EER to relational model mapping; relational algebra and relational calculus; basic operations (union, intersection, difference, Cartesian product); additional operations (projection, selection, division, rename, join).

The relational data model and relational database constraints

In the **relational data model** (proposed by E. F. Codd), the whole database is a collection of **relations (tables)**. A relation has a name and a set of **attributes (columns)**, each with a domain; its data is a set of **tuples (rows)**. Properties: no duplicate tuples, tuple and attribute order do not matter, and each cell holds a single atomic value.

Relational database constraints are the rules that keep relations valid:

- **Domain constraints** — every value must belong to its attribute's domain (type and allowed set/range).
- **Key constraints** — no two tuples can have the same value for a super key; the primary key is unique.
- **Entity integrity** — no primary-key attribute can be NULL.
- **Referential integrity** — a foreign-key value must equal an existing primary-key value in the referenced relation, or be NULL where permitted; this links tables correctly.
- **Semantic integrity constraints** — business rules, such as salary of a manager being greater than that of a clerk, often enforced by CHECK constraints or triggers.

ER/EER to relational model mapping

An E-R diagram is converted into relational tables by fixed **mapping rules**. EER means the extended E-R model (with specialisation/generalisation); its mapping extends the same rules.

- **Strong entity** → one table; its simple attributes become columns and one key attribute becomes the primary key.
- **Weak entity** → one table; include the owner's primary key as a foreign key; primary key = owner's key + partial key.
- **Composite attribute** → only its simple parts become columns. **Multivalued attribute** → a separate table with the entity's key as foreign key.
- **Relationship 1:1** → put the primary key of one side as a foreign key in the other (prefer the side with total participation). **1:N** → put the key of the “one” side as a foreign key in the “many” side table. **M:N** → create a **new table** containing the keys of both sides (together its primary key) plus any relationship attributes.
- **EER specialisation** → either one table for the superclass with a type attribute, or separate tables for superclass and each subclass sharing the key.

In an exam, apply the rules step by step to the given diagram and list each final table with its primary key underlined.

Relational algebra and relational calculus

Relational algebra and **relational calculus** are the two formal query languages on which SQL is based. Both have the same power; they differ in style.

Relational algebra is procedural: the user writes a sequence of operations (select, project, join and so on) that takes one or two relations and produces a new relation, step by step, until the answer is obtained. It tells **how** to get the result.

Relational calculus is non-procedural (declarative): the user only describes **what** result is wanted, as a condition on tuples, and the system decides how to obtain it. It has two forms: **tuple relational calculus**, where variables range over tuples, written like { t | STUDENT(t)

and $t.Course = 'CSE' \}$, and **domain relational calculus**, where variables range over attribute values.

Comparison for exams: algebra = procedural, operation sequence, basis of how queries execute; calculus = declarative, condition only, basis of what SQL statements express. A language that can express all algebra operations is called relationally complete.

Basic operations (union, intersection, difference, Cartesian product)

These set operations work on whole relations. Union, intersection and difference require the two relations to be **union-compatible** — same number of attributes and matching domains.

- **Union ($R \cup S$)** — tuples that are in R or in S (or both); duplicate tuples are removed. Example: students of CSE union students of IT gives all students of either branch.
- **Intersection ($R \cap S$)** — tuples that are in **both** R and S. Example: students who play both cricket and football.
- **Difference ($R - S$)** — tuples that are in R but **not** in S. Note $R - S$ is not the same as $S - R$. Example: all students minus hostel students gives day scholars.
- **Cartesian product ($R \times S$)** — every tuple of R combined with every tuple of S. If R has m tuples and k attributes and S has n tuples and j attributes, the result has $m \times n$ tuples and $k + j$ attributes. Alone it usually gives meaningless combinations; combined with a selection on matching columns it becomes a join.

Always state the number of rows in the result — examiners check it.

Additional operations (projection, selection, division, rename, join)

- **Selection (σ)** — chooses **rows** satisfying a condition. $\sigma Course='CSE' (STUDENT)$ gives all CSE students, with all columns.
- **Projection (π)** — chooses **columns** and removes duplicate rows. $\pi Name, Course (STUDENT)$ gives only names and courses.
- **Division ($R \div S$)** — gives values in R associated with **all** values in S. Classic question: find students enrolled in all courses listed in S. Used rarely, but important for theory.
- **Rename (ρ)** — renames a relation or its attributes, $\rho NewName (STUDENT)$; needed when the same table is used twice in one expression, such as finding an employee earning more than another.
- **Join ($\bowtie$)** — combines related tuples from two relations into single tuples, normally matching a foreign key with a primary key. Types: **natural join** (automatic match on common columns, duplicates removed), **equijoin** (join condition uses $=$) and **outer joins** (left/right/full — keep unmatched rows also, filling missing values with NULL).

Mnemonic: selection cuts rows horizontally, projection cuts columns vertically.

Likely question: Convert a given E-R diagram of a library (Book, Member, Issue) into relational tables, stating the primary and foreign keys. Also explain selection, projection and join operations with examples.

Unit 4 — Relational Database Design (10 Periods, 25%)

Topics: Functional dependencies and normalisation for relational databases; types of normalisation (1NF, 2NF, 3NF, multivalued dependencies and BCNF, fourth normal form, fifth normal form).

Functional dependencies and normalisation for relational databases

A **functional dependency (FD)**, written $X \rightarrow Y$, means: if two tuples agree on attribute set X, they must also agree on Y — that is, the value of X determines the value of Y. Example: RollNo $\rightarrow$ Name, because one roll number always has one name. X is the determinant. An FD is a property of the meaning of the data, not of one particular instance.

Normalisation is the process of organising relations to remove redundancy and the problems it causes:

- **Insertion anomaly** — cannot add some data without adding unrelated data (cannot add a new course until a student joins it).
- **Deletion anomaly** — deleting one fact deletes another fact too (removing the last student of a course loses the course itself).
- **Update anomaly** — the same fact stored in many rows must be changed in all of them, or data becomes inconsistent.

Normalisation proceeds step by step through normal forms — 1NF, 2NF, 3NF, BCNF, 4NF, 5NF — mainly by decomposing a table into smaller tables connected by keys, without losing information.

Types of normalisation — 1NF, 2NF, 3NF

First Normal Form (1NF): a relation is in 1NF if every cell holds a **single, atomic value** — no multivalued or repeating groups. Fix: if a student has three phone numbers in one cell, move phones to separate rows or a separate table.

Second Normal Form (2NF): the relation is in 1NF and every non-key attribute depends on the **whole** primary key, not on a part of it (no **partial dependency**). This problem arises only with a composite key. Example: in MARKS(RollNo, SubjectID, SubjectName, Marks) with key (RollNo, SubjectID), SubjectName depends only on SubjectID — move it to a SUBJECT table.

Third Normal Form (3NF): the relation is in 2NF and no non-key attribute depends on another non-key attribute (no **transitive dependency**). Example: in STUDENT(RollNo, DeptID, DeptName), DeptName depends on DeptID, not on RollNo — move department details to a DEPARTMENT table.

Memory line: 1NF removes groups, 2NF removes partial dependence, 3NF removes transitive dependence. Each step keeps keys to join the tables back without loss.

Multivalued dependencies and BCNF

Boyce-Codd Normal Form (BCNF) is a stronger version of 3NF: a relation is in BCNF if, for **every** functional dependency $X \rightarrow Y$, X is a **super key** of the relation. In 3NF a small exception is allowed (Y may be part of a key); BCNF removes that exception. If a table has a determinant that is not a key — for example, in TEACHING(Student, Subject, Teacher) where each teacher teaches only one subject, so $\text{Teacher} \rightarrow \text{Subject}$ but Teacher is not a key — split the table so the dependency becomes a key of its own table. BCNF tables have very little FD-based redundancy, though decomposition may occasionally not preserve all dependencies.

Multivalued dependency (MVD), written $X \twoheadrightarrow Y$, occurs when, for a given X value, the set of Y values is independent of the other attributes. Example: an employee can have many skills and many languages, independently — storing them together multiplies rows ($3 \text{ skills} \times 2 \text{ languages} = 6 \text{ rows}$) and causes redundancy. MVDs are the basis of 4NF in the next point.

Fourth normal form

A relation is in **Fourth Normal Form (4NF)** if it is in BCNF and it has **no non-trivial multivalued dependencies** other than those based on a super key. In practical words: one table should not store two independent many-valued facts about the same key.

Example: EMPLOYEE(EmpNo, Skill, Language) stores skills and languages of an employee in one table. Skills do not depend on languages, yet every skill must be combined with every language, creating rows that carry no extra information and causing insertion, deletion and update problems — adding a language means adding one row per skill.

Fix by decomposition: split into two tables — EMP_SKILL(EmpNo, Skill) and EMP_LANG(EmpNo, Language). Now each fact is stored once, and the original table can be recovered by joining them. Test before splitting: the two attribute sets must truly be independent; if skill and language were related (for example, a language needed for a skill), the table must not be split this way.

Fifth normal form

A relation is in **Fifth Normal Form (5NF)**, also called **Project-Join Normal Form (PJ/NF)**, if it cannot be decomposed further into smaller relations without losing information — every **join dependency** in it is implied by its candidate keys. In simple words, even after removing all FD and MVD problems, a table may still contain redundancy because it secretly combines three or more relationships that could be stored separately and rejoined exactly.

Classic example: SUPPLY(Supplier, Part, Project) under the rule “if a supplier supplies a part, the part is used in a project, and the supplier supplies to that project, then the supplier supplies that part for that project.” The table can be split into three two-column tables — (Supplier, Part), (Part, Project) and (Project, Supplier) — and joined back without any extra false rows. If that exact recovery is possible, the original table should be split to reach 5NF.

5NF cases are rare in practice; most well-designed databases aim for 3NF or BCNF, and 4NF/5NF are mainly examination and theory topics. State this honestly in an answer.

Likely question: Define functional dependency. A table ORDER(OrderID, ProductID, ProductName, Quantity) has key (OrderID, ProductID). Identify the normalisation problems in it and decompose it up to 3NF, giving reasons at each step.

Unit 5 — SQL / MySQL (8 Periods, 15%)

Topics: MySQL data types; data definition commands; data manipulation commands; data retrieval commands; types of operators — arithmetic, comparison and logical operators.

MySQL data types

A **data type** fixes what kind of value a column can hold, how much space it takes and what operations are valid on it. Main MySQL data types:

- **Numeric** — INT (whole numbers, e.g. RollNo, Marks), BIGINT (larger whole numbers), FLOAT and DOUBLE (decimal numbers with fraction), DECIMAL(p,s) (exact decimals — use for money, e.g. DECIMAL(8,2) for fees).
- **String** — CHAR(n) (fixed length, padded), VARCHAR(n) (variable length up to n — the most used, e.g. VARCHAR(50) for Name), TEXT (long text).
- **Date and time** — DATE (YYYY-MM-DD), TIME (HH:MM:SS), DATETIME (date + time together), YEAR.
- **Other** — BOOLEAN (true/false, stored as 1/0) and ENUM (one value from a fixed list, e.g. ENUM('CSE','EE','ME')).

Choosing the right type saves space and enforces validity: a phone number should be VARCHAR, not INT, because it is never added or subtracted and may start with 0 or +91. Syntax pattern: ColumnName DataType, as in Name VARCHAR(50).

Data definition commands

DDL (Data Definition Language) commands define and change the **structure** of database objects. They work on tables and databases themselves, not on rows. Main commands with syntax:

```
CREATE DATABASE school;
CREATE TABLE student (
  roll INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  marks INT CHECK (marks BETWEEN 0 AND 100)
);
ALTER TABLE student ADD COLUMN email VARCHAR(100);
```

```
ALTER TABLE student MODIFY COLUMN marks DECIMAL(5,2);
ALTER TABLE student DROP COLUMN email;
RENAME TABLE student TO students;
DROP TABLE students;
TRUNCATE TABLE students;
```

- **CREATE** — makes a new database or table.
- **ALTER** — changes a table's structure: add, modify or drop a column.
- **DROP** — deletes the table completely, structure and data both.
- **TRUNCATE** — deletes all rows but keeps the structure (compare with DELETE in the next point).

DDL statements are auto-committed in MySQL — they cannot be undone by ROLLBACK.

Data manipulation commands

DML (Data Manipulation Language) commands work on the **data (rows)** inside tables; the structure is not changed.

```
INSERT INTO student (roll, name, marks)
VALUES (101, 'Aarav', 88);
INSERT INTO student VALUES (102, 'Meera', 91), (103, 'Kabir', 76);
UPDATE student SET marks = 90 WHERE roll = 101;
DELETE FROM student WHERE roll = 103;
```

- **INSERT** — adds one or more new rows; column list and values must match in number and type.
- **UPDATE** — changes values in existing rows. The **WHERE** clause selects which rows; without WHERE, **all** rows are updated — the most common beginner mistake.
- **DELETE** — removes selected rows; the table and its structure remain, and the operation can be rolled back inside a transaction. Compare: DELETE removes chosen rows, TRUNCATE removes all rows faster, DROP removes the whole table.

Order of writing: verb first (INSERT / UPDATE / DELETE), then table name, then values or SET clause, then WHERE condition.

Data retrieval commands

Data is retrieved with **SELECT**, the most used SQL command. It does not change any data.

```
SELECT name, marks FROM student;
SELECT * FROM student WHERE course = 'CSE';
SELECT name, marks FROM student
WHERE marks >= 60 ORDER BY marks DESC;
SELECT DISTINCT course FROM student;
SELECT course, COUNT(*) AS total FROM student
GROUP BY course;
```

```
SELECT name FROM student WHERE name LIKE 'A%';  
SELECT * FROM student WHERE marks BETWEEN 50 AND 80  
AND roll IN (101, 102, 105) LIMIT 5;
```

- **SELECT ... FROM** — choose columns; * means all columns.
- **WHERE** — filter rows by a condition.
- **ORDER BY** — sort the result, ASC (default) or DESC.
- **DISTINCT** — remove duplicate values from the result.
- **GROUP BY / aggregate functions** — COUNT, SUM, AVG, MAX, MIN summarise rows group-wise.
- **LIKE, BETWEEN, IN, LIMIT** — pattern match (% = any characters), range test, list test, and limit on the number of rows returned.

Types of operators — arithmetic, comparison and logical operators

Operators are used mainly in WHERE clauses and expressions to calculate values and test conditions.

1. Arithmetic operators — work on numeric values: + (add), – (subtract), * (multiply), / (divide), % (modulo — remainder).

Example: SELECT name, marks + 5 AS grace_total FROM student;

2. Comparison operators — compare two values and give true or false: = (equal), != or <> (not equal), > (greater), < (smaller), >= , <= .

Example: SELECT * FROM student WHERE marks >= 60;

3. Logical operators — combine or reverse conditions:

AND — true only if both conditions are true;

OR — true if at least one is true;

NOT — reverses the result.

Example: SELECT * FROM student

WHERE course = 'CSE' AND marks > 70 AND NOT city = 'Hostel';

Precedence note: NOT is evaluated first, then AND, then OR — use brackets when a condition mixes AND and OR, so the meaning is exactly what you intend. Special comparison operators BETWEEN, IN and LIKE (Unit's retrieval commands) are built on these three families.

Likely question: Write SQL commands to: (a) create a table EMPLOYEE(EmpID INT primary key, Name VARCHAR(50), Salary DECIMAL(10,2), Dept VARCHAR(20)); (b) insert two rows; (c) increase the salary of all employees of the 'Sales' department by 10%; and (d) display the names of employees earning more than 50000, in descending order of salary.

Quick Revision — One Line Each

- A DBMS removes the redundancy and inconsistency of file systems by storing data once, centrally.

- Three views: external (user), conceptual (all data and relations), internal (physical storage).
- Schema is the design of the database; instance is its data at one moment.
- Hierarchical model = tree, one parent per child; network model = graph, many owners allowed.
- Every primary key is a candidate key; every candidate key is a super key — not the reverse.
- A foreign key in one table refers to the primary key of another and enforces referential integrity.
- Weak entity has no own key; it is drawn as a double rectangle and identified through its owner.
- Selection picks rows, projection picks columns — σ works horizontally, π vertically.
- M:N relationship always becomes a separate table holding both keys.
- 1NF: atomic values; 2NF: no partial dependency; 3NF: no transitive dependency; BCNF: every determinant is a super key.
- CREATE, ALTER, DROP are DDL; INSERT, UPDATE, DELETE are DML; SELECT retrieves data.
- UPDATE or DELETE without WHERE acts on all rows — always check the WHERE clause first.

KAMALDNP • kamaldnp.com • support@kamaldnp.com • +91 8318013442 • Varanasi, UP

Prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus topics. Short explanations are the author's own plain-English summaries, not syllabus text. Please confirm with your own polytechnic for any local changes.