DATABASE ENGINEERING

Unit 1

Introduction to DBMS and the ER Data Model

Topic-wise study notes with importance ratings

32 topics — from file systems and DBMS architecture through the complete ER and EER model

Diagrams are drawn wherever a concept needs one

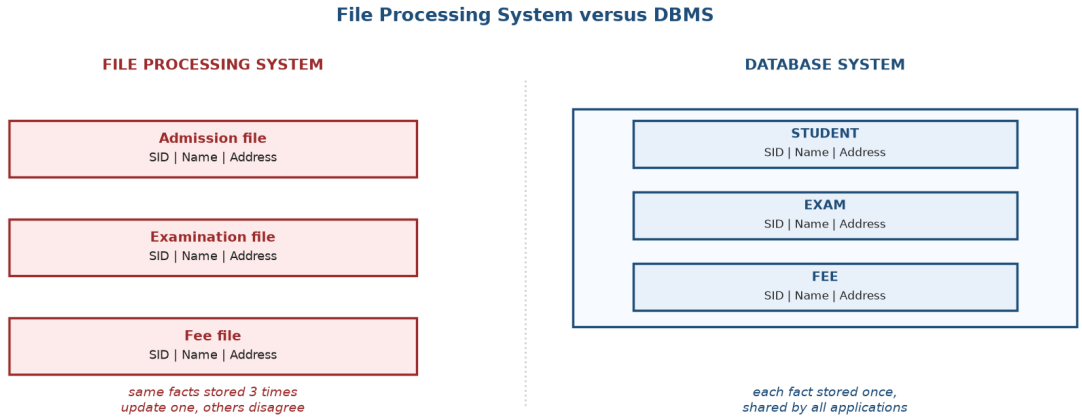
Topic-wise study notes. Every topic carries an importance rating from one to three stars: three stars means the topic is asked almost every year and must be known perfectly, two stars means it is regularly asked and should be prepared, one star means it is supporting material to read once. The notes explain the concept only; worked questions and answers are kept in a separate file.

Index

Topic, importance rating and page number. ★★★ must know, ★★ should know, ★ read once.

Topic	Importance	Page
1.1 File Processing System	★★★	3
1.2 Data, Information and Metadata	★★	3
1.3 Database, DBMS and Database System	★★★	4
1.4 Need for a DBMS	★★★	4
1.5 Functions of a DBMS	★	5
1.6 Data Dictionary	★★	5
1.7 Users of a DBMS	★★	5
1.8 Database Administrator	★★★	6
1.9 Data Abstraction	★★	6
1.10 Three-Schema Architecture	★★★	6
1.11 Data Independence	★★★	7
1.12 Database Languages	★★	8
1.13 Data Models — Overview	★★	8
1.14 Hierarchical Model	★★	9
1.15 Network Model	★★	10
1.16 Relational Model	★★	10
1.17 Object-Oriented and Object-Relational Models	★	11
1.18 Entity and Entity Set	★★★	11
1.19 Types of Attributes	★★★	12
1.20 Relationship and Relationship Set	★★★	12
1.21 Degree of Relationship	★★	13
1.22 Role and Recursive Relationship	★★	13
1.23 Cardinality Ratio	★★★	14
1.24 Participation Constraint	★★★	14
1.25 Weak Entity Set	★★★	15
1.26 Generalization and Specialization	★★★	16
1.27 Aggregation	★★	17
1.28 Category, Union Type and Inheritance	★	17
1.29 Constraints on Generalization	★★	18
1.30 EER Model	★★	18
1.31 Drawing an ER Diagram	★★★	19
1.32 Worked ER Diagrams	★★★	19

1.1 File Processing System ★★★



A file processing system is the traditional way of storing organisational data: each department or application keeps its **own private files**, and writes its **own programs** to read and update them. There is no central control over the data, and no shared description of what the data means.

How it works

Each application — admissions, examinations, fees, library — maintains a separate file. A student's record therefore exists in four places, each with its own layout, its own program and its own copy of the same facts.

Its limitations

Limitation	What goes wrong
Data redundancy	The same fact is stored in several files, wasting storage
Data inconsistency	When one copy is updated and another is not, the files disagree
Program–data dependence	The file layout is written into every program, so any change forces rewrites
Difficulty of access	Only pre-written reports exist; a new question needs a new program
No concurrency control	Simultaneous updates can overwrite one another
Weak atomicity	A half-completed update leaves the data inconsistent
Poor security	Access control works per file, not per record or field
Limited sharing	Data stays isolated inside departments

Why this matters

Every one of these limitations is a problem a DBMS is designed to solve, so this topic is the foundation of the whole subject. When a question asks why a DBMS is needed, the answer is simply this list turned inside out.

1.2 Data, Information and Metadata ★★

These three terms are often confused, and a short definition question on them is common.

Term	Meaning	Example
Data	Raw, unprocessed facts with no meaning on their own	72, 85, 90
Information	Processed data that has meaning and supports a decision	Average marks = 82.3
Metadata	Data about data — it describes the structure and meaning of stored data	SID is CHAR(5) and is the primary key

Metadata in practice

Metadata records the name of every table, its columns, data types, sizes, keys, constraints, indexes, users and privileges. It is stored in the **data dictionary**, and the DBMS reads it constantly — to parse a query, to check a constraint, to decide whether a user may see a column.

Data and information together

Data becomes information only after processing. The number 82.3 means nothing by itself; "the average marks of the CSE third semester is 82.3" is information, because a decision can be taken on it.

1.3 Database, DBMS and Database System ★★★

These three are used loosely in conversation but must be distinguished precisely.

Term	What it is	Example
Database	An organised collection of related data stored so that it can be accessed and managed	The university's student, course and marks data
DBMS	The software that creates, stores, retrieves and controls access to a database	MySQL, Oracle, SQL Server, PostgreSQL
Database system	The complete environment — database + DBMS + application programs + users + procedures	The whole examination system of a university

The relationship

The DBMS is one component of the database system. The database is the data itself. A database system includes all of it plus the people who use it and the procedures they follow.

1.4 Need for a DBMS ★★★

A DBMS is needed because it removes each limitation of the file system in a specific, identifiable way.

Problem in a file system	How a DBMS solves it
Redundancy	One integrated database, each fact stored once
Inconsistency	Constraints enforced centrally by the DBMS
Program–data dependence	Schema stored separately in the data dictionary; three-schema architecture gives data independence
Difficulty of access	SQL allows ad hoc queries without writing programs
No concurrency control	Transactions with isolation and locking
Weak atomicity and recovery	Logs, checkpoints, rollback and redo
Poor security	Views plus GRANT and REVOKE
Limited sharing	All authorised applications read the same data

The single-sentence answer

A DBMS stores the data once, describes it in one place, controls who may do what, and guarantees that a transaction either completes fully or not at all.

1.5 Functions of a DBMS ★

Function	What it does
Data definition	Creates and alters the schema using DDL
Data manipulation	Inserts, updates, deletes and retrieves data using DML and DQL
Data storage management	Manages files, pages and indexes on disk
Data security	Authenticates users and enforces privileges
Integrity enforcement	Checks domain, key, entity and referential constraints
Concurrency control	Keeps interleaved transactions correct
Recovery	Restores a consistent state after failure
Data dictionary management	Maintains metadata
Utilities	Backup, restore, import, export, statistics, reorganisation

1.6 Data Dictionary ★★

A data dictionary, also called the **system catalog**, is the centralised repository in which the DBMS stores all metadata about the database.

What it contains

Relation names and their attributes; data types and sizes; primary, candidate and foreign keys; integrity constraints; indexes and access paths; views; users, privileges and ownership; storage details and statistics.

Why it is needed

- The DBMS reads it to parse and validate every query.
- It is how constraints are enforced without reading the actual data.
- It is how access control is checked.
- It gives the DBA a single place to see and document the whole schema.

One-line definition for the exam

The data dictionary is the DBMS's own database of definitions — it stores descriptions of the data, never the data itself.

1.7 Users of a DBMS ★★

User	What they do
Naive end users	Invoke pre-written application programs through forms and reports; unaware of the database
Application programmers	Write programs in a host language with embedded DML calls
Sophisticated users	Analysts and scientists who write their own queries directly
DBA	Controls the schema, storage, authorisation, performance and recovery

User	What they do
Database designers	Identify the data to be stored and choose the structures and views

Why the distinction matters

Each class of user needs a different level of the three-schema architecture. A naive user needs only an external view; a programmer needs the conceptual schema; the DBA alone works with the internal schema.

1.8 Database Administrator ★★★

The DBA is the person responsible for the whole database environment.

Area	Duties
Schema definition	Define the conceptual, external and internal schemas and their mappings
Storage and access	Choose file organisations, indexes and access paths; reorganise as data grows
Authorisation	Grant and revoke privileges; create views to restrict what each user sees
Integrity	Declare and monitor constraints; keep the database consistent
Performance	Monitor usage, tune queries, plan capacity, resolve contention
Backup and recovery	Plan backups, maintain logs, restore after failure
User liaison	Identify requirements, choose the DBMS, train and support users
Documentation	Maintain the data dictionary and all system documentation

The four-word summary

Design, secure, tune, recover.

1.9 Data Abstraction ★★

Data abstraction is the process of hiding the low-level detail of how data is stored and exposing only the part a user needs.

Why it is necessary

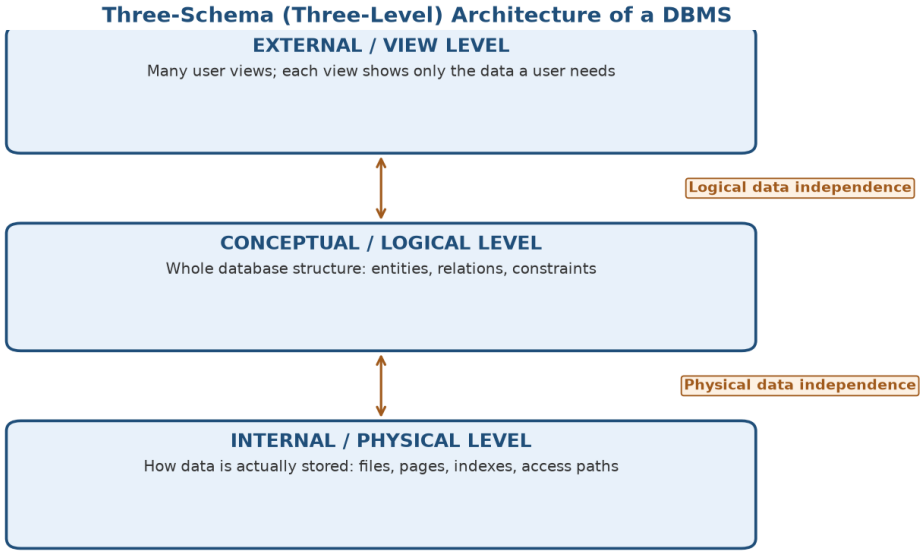
A user does not need to know the block size, the index structure or the file organisation. Presenting only a simplified view makes the database usable by non-specialists, and it protects their programs from changes made deep in the storage layer.

How it is implemented

Through the three levels of the three-schema architecture — external, conceptual and internal — with a mapping between each pair of levels. Each level hides the one below it.

1.10 Three-Schema Architecture ★★★

Also called the three-level architecture. It separates the description of the database into three levels so that each kind of user sees only what concerns them.



External/Conceptual mapping and Conceptual/Internal mapping isolate each level from changes in the level below.

Level	What it describes	Who uses it
External (view)	Each user's own view of the part of the database he needs	End users and applications
Conceptual (logical)	The whole database: entity sets, attributes, relationships, constraints	DBA and designers
Internal (physical)	How data is actually stored: files, pages, indexes, access paths	DBA and system

The two mappings

The **external/conceptual mapping** defines how each view is derived from the conceptual schema. The **conceptual/internal mapping** defines how the conceptual schema is realised on disk.

What the architecture achieves

Because of the mappings, the conceptual schema can change without touching the views, and the internal schema can change without touching the conceptual schema. This is exactly what gives **logical** and **physical** data independence, and it is the reason application programs survive a change of storage device.

1.11 Data Independence ★★★

Data independence is the ability to change the schema at one level without changing the schema at the level above it.

Physical data independence	Logical data independence
Change the internal schema without changing the conceptual schema	Change the conceptual schema without changing external schemas or programs
Easier to achieve	Harder to achieve
Adding an index, changing file organisation, moving to new storage	Adding an attribute or relation, splitting a relation, changing constraints
Sits between conceptual and internal levels	Sits between external and conceptual levels

Why logical is harder

A change to the conceptual schema can remove or rename something a view depends on. Physical changes, by contrast, can almost always be absorbed by the conceptual/internal mapping.

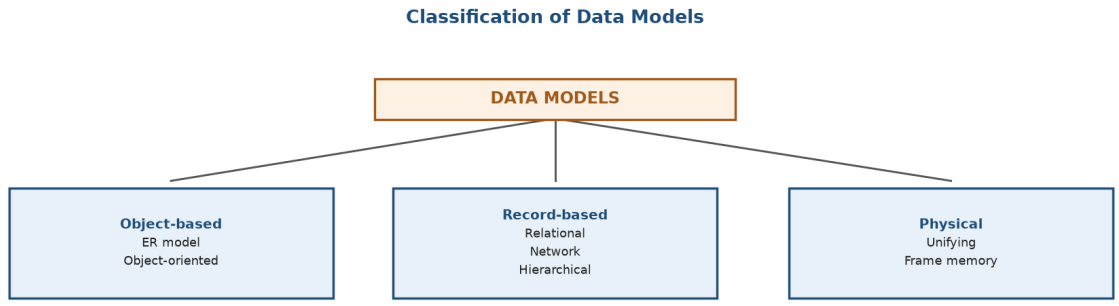
1.12 Database Languages ★★

Language	Purpose	Statements
DDL (Data Definition Language)	Defines and changes the structure	CREATE, ALTER, DROP, TRUNCATE
DML (Data Manipulation Language)	Changes the data	INSERT, UPDATE, DELETE
DQL (Data Query Language)	Retrieves data	SELECT
DCL (Data Control Language)	Controls access	GRANT, REVOKE
TCL (Transaction Control Language)	Controls transactions	COMMIT, ROLLBACK, SAVEPOINT

The distinction to remember

DDL changes the **shape**, DML changes the **content**, DQL only reads, DCL changes **who may do what**. DDL statements update the data dictionary and are usually auto-committed; DML statements are transactional and can be rolled back.

1.13 Data Models — Overview ★★

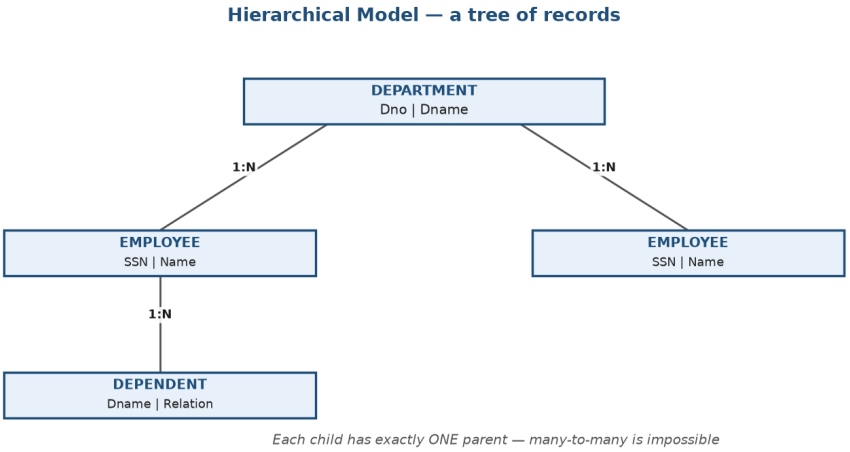


Object-relational model combines the object-based and record-based ideas

A data model is a collection of concepts used to describe the structure of a database — the data types, the relationships among them, the constraints, and the operations allowed.

Category	Models	Idea
Object-based logical	ER model, object-oriented model	Describe data as entities and objects
Record-based logical	Relational, network, hierarchical	Describe data as fixed-format records
Object-relational	Extended relational	Add objects and inheritance to tables
Physical	Unifying, frame-memory	Describe how data is stored at the lowest level

1.14 Hierarchical Model ★★



Data is organised as a **tree** of records: each parent may have many children, but each child has exactly one parent.

Structure

A record type is a node; parent and child record types are linked by a parent–child relationship. Example: DEPARTMENT at the root, EMPLOYEE as children, DEPENDENT as grandchildren.

Advantages

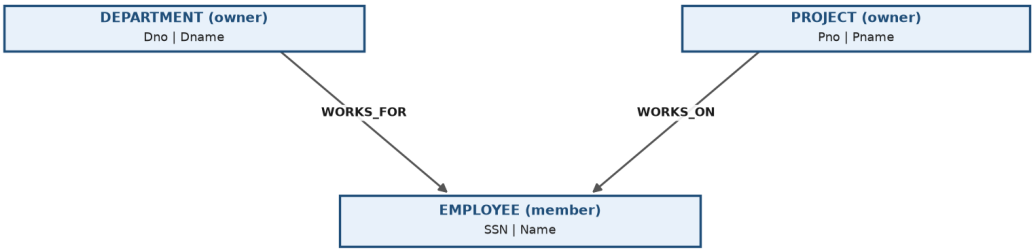
Simple and fast for one-to-many navigation along the tree; storage is efficient because the structure is fixed; access along a known path is quick.

Disadvantages and anomalies

Problem	Consequence
Insertion anomaly	A child cannot be inserted without its parent
Deletion anomaly	Deleting a parent deletes all its children
Update anomaly	A repeated fact must be updated at every occurrence
Structural rigidity	M:N relationships cannot be represented directly
Navigational access	The programmer must know the tree structure

1.15 Network Model ★★

Network Model — a graph of records linked by sets



A member record may have SEVERAL owners — many-to-many is possible

Data is organised as a **graph** of records connected by sets, so a record may have several parents.

Structure

Each set has one **owner record** and one or more **member records**, linked by pointers. This removes the single-parent restriction of the hierarchical model and allows many-to-many relationships to be represented directly.

Advantages

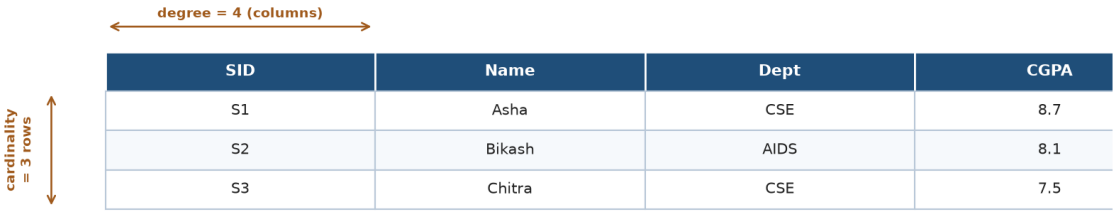
Represents M:N relationships; less redundancy than the hierarchical model; efficient navigation once the pointers are known.

Disadvantages

The programmer must understand the physical pointer structure; any change to the set structure forces programs to be rewritten; design is complex; deletion of an owner affects its members.

1.16 Relational Model ★★

Relational Model — data as a table of rows and columns



SID	Name	Dept	CGPA
S1	Asha	CSE	8.7
S2	Bikash	AIDS	8.1
S3	Chitra	CSE	7.5

A relation is a set of tuples; each tuple is one entity

Data is represented as **tables** of rows and columns, and relationships are represented by **common values** rather than pointers.

Characteristics

- Data is visible as relations, which is easy to understand.
- No pointers exist, so there is nothing to maintain when data moves.
- Any M:N relationship is expressed by an extra table.
- It has a sound theoretical base in set theory and functional dependencies.
- It is queried declaratively with SQL.

Why it displaced the earlier models

Because relationships are values, not addresses, the logical view is independent of storage. Normalisation theory then guarantees a design free of the anomalies that plagued the hierarchical and network models.

1.17 Object-Oriented and Object-Relational Models ★

Object-oriented model	Object-relational model
Data stored as objects with attributes and methods	Relational tables extended with objects
Supports encapsulation, inheritance and polymorphism	Supports user-defined types and inheritance inside tables
Suits CAD, multimedia and complex data	Suits applications needing both complex types and SQL

Where they are used

The object-oriented model suits data whose structure is complex and nested. The object-relational model is the practical compromise, and most modern DBMSs — Oracle, PostgreSQL — implement it.

1.18 Entity and Entity Set ★★★

An **entity** is a real-world object or thing that has an independent existence — the student Asha, the department CSE, the loan L-101. An **entity set** is a collection of entities of the same type, written STUDENT, and drawn as a rectangle.

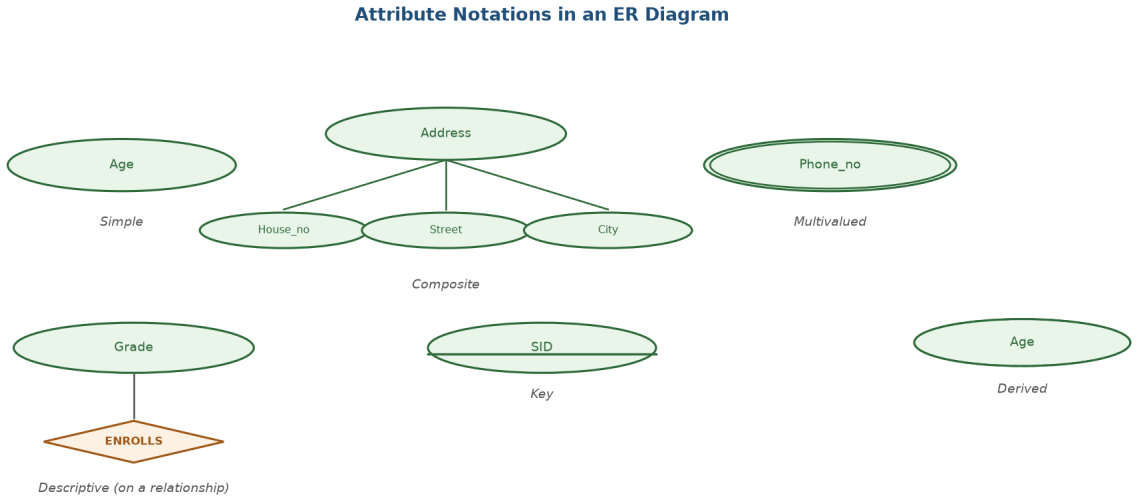
Strong and weak entity sets

A **strong** entity set has a key attribute of its own and exists independently. A **weak** entity set has no key of its own and depends on an owner; it is covered in topic 1.25.

Why entity sets rather than entities

A database stores the structure of a set, not individual objects. The ER diagram shows the set; the table later holds the instances.

1.19 Types of Attributes ★★★



Type	Definition	Example
Simple	Atomic, cannot be divided further	Age, Salary, Roll_no
Composite	Can be divided into sub-parts with independent meaning	Address (House_no, Street, City, PIN)
Single-valued	One value per entity	Date_of_birth, Gender
Multivalued	A set of values for one entity	Phone_no, Hobbies, Qualifications
Stored	Physically stored in the database	Date_of_birth, Basic_pay
Derived	Computed from other stored attributes	Age from DOB, Total from Qty and Rate
Key	Uniquely identifies the entity; shown underlined	Student_id, SSN
Partial key	Distinguishes entities of a weak set within one owner	Dependent_name
Descriptive	Belongs to a relationship, not to an entity	Grade in ENROLLS, Hours in WORKS_ON
NULL	Value unknown, not applicable or not yet available	Middle_name not given

Composite attributes on the diagram

A composite attribute is drawn as an ellipse with its components attached beneath it. When the diagram is mapped to tables, only the simple components are stored — the composite itself disappears.

Why descriptive attributes matter

A descriptive attribute describes one particular association. Grade does not belong to the student alone or to the course alone; it belongs to the pair. This is why an M:N relationship must become its own table when mapped.

1.20 Relationship and Relationship Set ★★★

A **relationship** is an association among two or more entities — Asha is enrolled in DBMS. A **relationship set** is the set of all such associations of one type, drawn as a diamond, and labelled with its name.

Reading a relationship on the diagram

The diamond sits between the entity rectangles; the lines to the rectangles carry the cardinality labels; a descriptive attribute, if any, hangs off the diamond.

Relationships versus foreign keys

In the ER model a relationship is a first-class concept. In the relational model it is represented either by a foreign key inside one of the tables or by a separate table, depending on its cardinality.

1.21 Degree of Relationship ★★

The degree is the number of entity sets that participate.

Degree	Name	Example
1	Unary (recursive)	An employee supervises another employee
2	Binary	A student enrolls in a course
3	Ternary	A supplier supplies a part to a project
n	n-ary	A general association among n entity sets

Note

Most practical relationships are binary. A ternary relationship cannot always be replaced by three binary relationships without losing information, so the degree must be taken from the problem statement.

1.22 Role and Recursive Relationship ★★

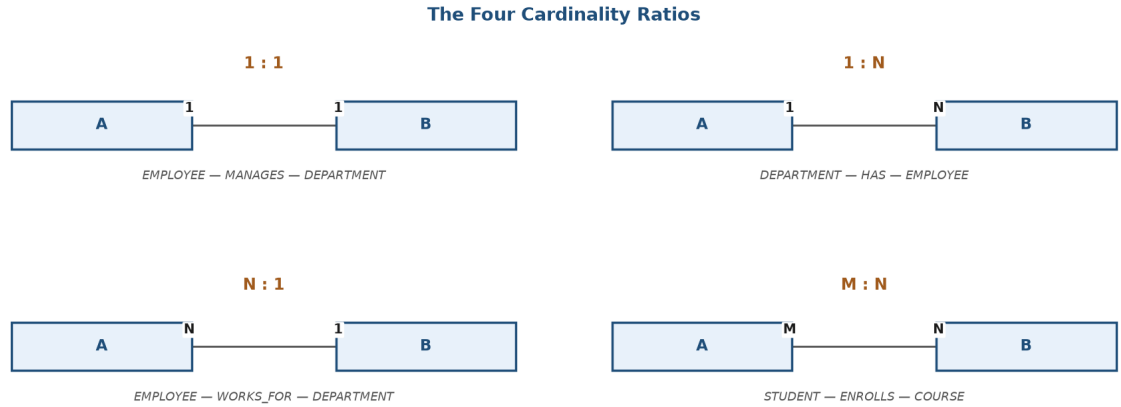
The **role** is the function an entity plays in a relationship. In WORKS_FOR the employee plays the role of worker and the department the role of employer. Role names become essential when the same entity set participates twice.

A **recursive relationship** is one in which the same entity set participates more than once, in different roles — EMPLOYEE supervises EMPLOYEE, where one instance is the supervisor and another the subordinate.

Mapping a recursive relationship

A 1:N recursive relationship is mapped by adding a foreign key inside the same table that references its own primary key, for example ManagerID in EMPLOYEE referencing EMPLOYEE.SSN. A self join is then needed to query it.

1.23 Cardinality Ratio ★★★



The cardinality ratio gives the **maximum** number of relationship instances in which an entity may participate.

Ratio	Meaning	Example
1:1	Each entity relates to at most one of the other side	EMPLOYEE manages DEPARTMENT
1:N	One A relates to many B, but each B relates to one A	DEPARTMENT has many EMPLOYEEs
N:1	Many A relate to one B	Many EMPLOYEEs work for one DEPARTMENT
M:N	Many A relate to many B	STUDENT enrolls in many COURSEs

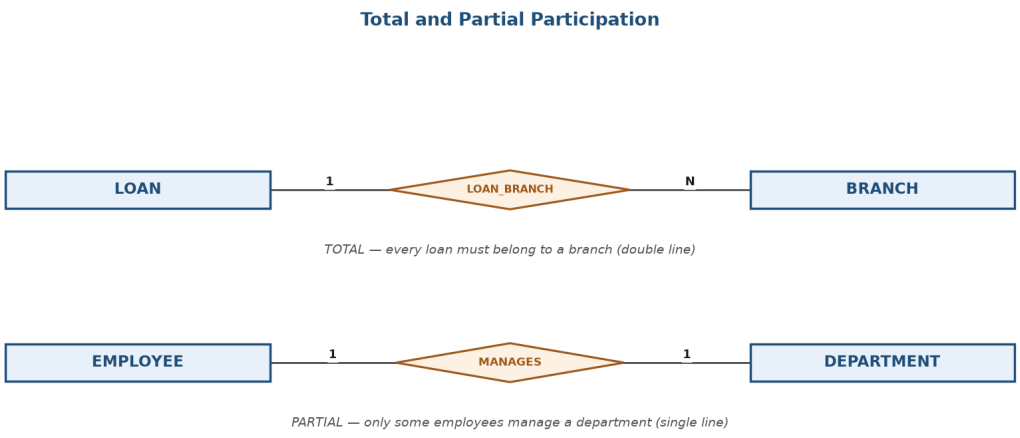
Reading the labels

The label nearest an entity gives the number of instances of that entity that may relate to one instance of the other. In DEPARTMENT 1 : N EMPLOYEE, one department relates to many employees.

Effect on mapping

The cardinality ratio decides how many tables the relationship produces: M:N needs its own table, while 1:N and N:1 are merged into the many side.

1.24 Participation Constraint ★★★



The participation constraint gives the **minimum** — it states whether every entity of a set must take part in the relationship.

Total participation	Partial participation
Every entity must participate	Only some entities participate
Drawn with a double line	Drawn with a single line
Existence depends on the relationship	Entity exists independently
Every LOAN must belong to a BRANCH	Only some EMPLOYEEs manage a department

Cardinality and participation together

Cardinality gives the maximum, participation gives the minimum. Together they completely describe the structural constraint of a relationship, and both are needed before the diagram can be mapped to tables.

Effect on mapping

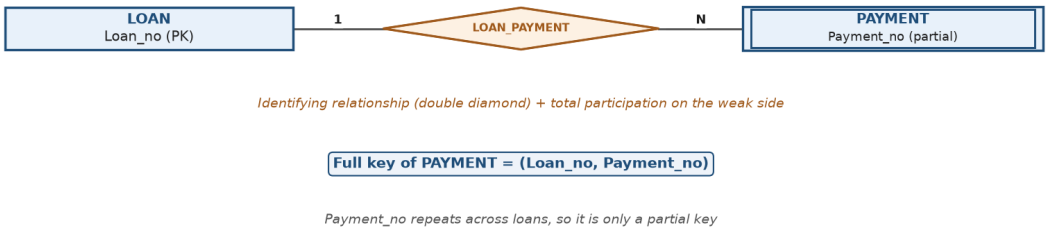
When a relationship is merged into the many side and the participation there is total, the merged foreign key acquires a **NOT NULL** constraint, because it can never be empty.

1.25 Weak Entity Set ★★★

A weak entity set has **no key attribute of its own**. It is identified through its owner entity set by an **identifying relationship**, and it uses a **partial key** together with the owner's primary key.

Strong entity set	Weak entity set
Own key attribute	Only a partial key
Independent existence	Exists only with an owner
Single rectangle	Double rectangle
Participation may be partial	Total participation in the identifying relationship
Maps to a table with its own primary key	Maps to a table keyed by owner key + partial key
Example: EMPLOYEE, LOAN	Example: DEPENDENT of EMPLOYEE, PAYMENT of LOAN

Weak Entity Set and Identifying Relationship



Identifying relationship

The relationship that links the weak set to its owner, drawn with a double diamond. It always has total participation on the weak side, because a weak entity cannot exist without its owner.

Partial key

The attribute that distinguishes entities belonging to the same owner. In LOAN–PAYMENT, Payment_no alone repeats across loans, but the combination (Loan_no, Payment_no) is unique, so Payment_no is

the partial key.

1.26 Generalization and Specialization ★★★

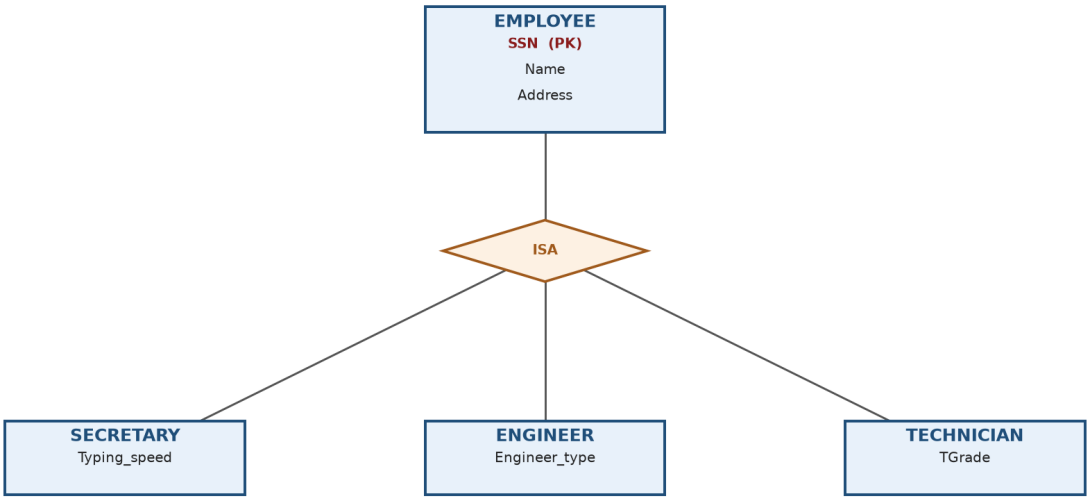
Generalization

A **bottom-up** process: entity sets that share common attributes are combined into a single superclass, and the common attributes move up to it. Example: CAR and TRUCK generalise into VEHICLE; SECRETARY, ENGINEER and TECHNICIAN generalise into EMPLOYEE.

Specialization

A **top-down** process: a superclass is divided into subclasses on the basis of a distinguishing characteristic, and attributes specific to a subclass move down to it. Example: EMPLOYEE specialises into SECRETARY (Typing_speed), ENGINEER (Engineer_type) and TECHNICIAN (TGrade).

Generalization / Specialization — EER Notation



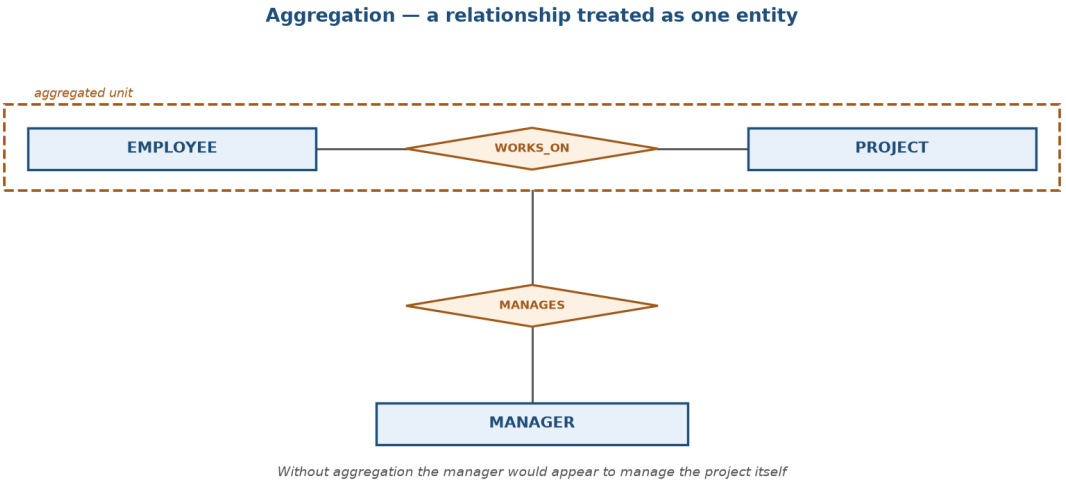
ISA is the specialization/generalization relationship. Attributes are inherited downward; the constraint may be disjoint or overlapping, total or partial.

Generalization	Specialization
Bottom-up	Top-down
Combines entity sets into a superclass	Divides a superclass into subclasses
Removes duplication of common attributes	Adds attributes specific to each subclass
Starts from specific entity sets	Starts from one general entity set

Attribute inheritance

A subclass inherits every attribute of its superclass and adds its own, and it also inherits the superclass's participation in relationships.

1.27 Aggregation ★★



Aggregation treats a **relationship set together with its participating entity sets** as a single higher-level entity, so that this unit can itself take part in another relationship.

When it is needed

When a relationship must be related to another entity. For example, an employee manages another employee's work on a project. Without aggregation the diagram is ambiguous — the manager would appear to manage the project or the employee, rather than the participation itself.

How it is drawn

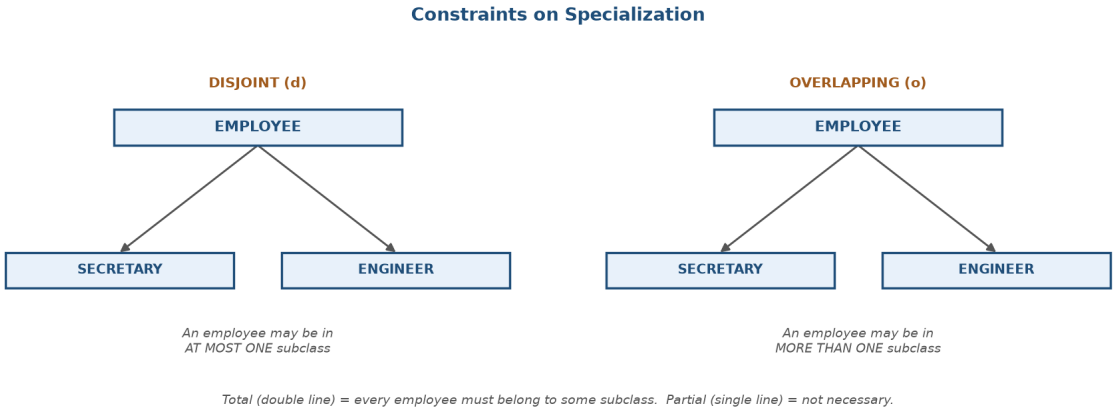
The relationship and its entity sets are enclosed in a box, and the new relationship connects to that box.

1.28 Category, Union Type and Inheritance ★

A **category** (union type) is a subclass that is a subset of the union of several superclasses. Example: OWNER is a category whose members may be either a PERSON or a COMPANY, but not both.

Inheritance is the automatic transmission of attributes and relationships from a superclass to its subclasses, and it is what makes specialization useful — common properties are declared once.

1.29 Constraints on Generalization ★★



Constraint	Meaning
Disjoint	An entity belongs to at most one subclass (exclusive)
Overlapping	An entity may belong to more than one subclass
Total	Every superclass entity must belong to some subclass (exhaustive)
Partial	An entity need not belong to any subclass
Attribute-defined	Membership is decided by a specific attribute value
User-defined	Membership is decided by the user at insertion time

Why the constraint matters

It decides which mapping strategy is valid later. A total and disjoint specialization can be merged into a single table with a type attribute; an overlapping one cannot be enforced that way without extra checks.

1.30 EER Model ★★

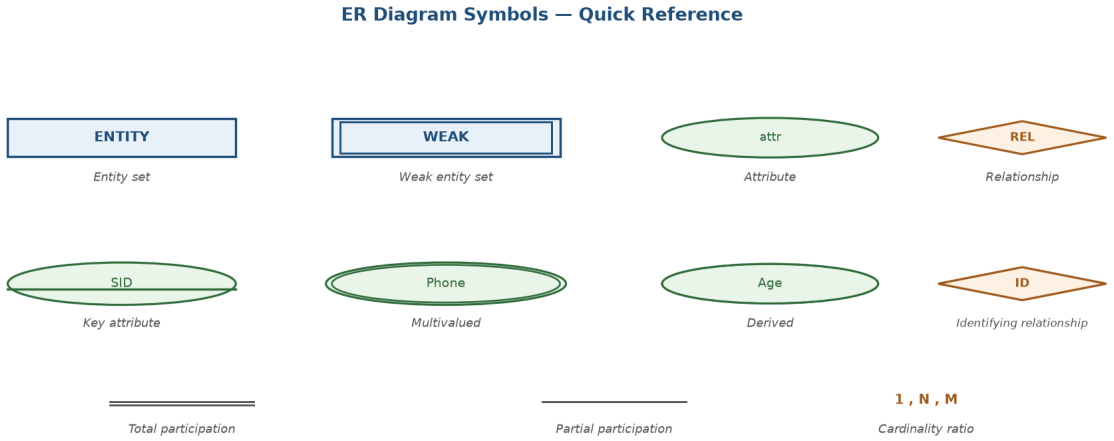
The Enhanced (Extended) ER model adds to the basic ER model the concepts needed for accurate design of complex schemas.

Feature	Purpose
Subclass and superclass	Represent a class hierarchy
Generalization and specialization	Build the hierarchy bottom-up or top-down
Attribute inheritance	Avoid repeating common attributes
Category / union type	Model an entity that may belong to one of several sets
Aggregation	Treat a relationship as an entity
Constraints on specialization	Disjoint or overlapping, total or partial

ER versus EER

The ER model describes entity sets, attributes and relationships. The EER model adds the hierarchy, the union type and the constraints on specialization. Use EER when the application genuinely has a class hierarchy, such as different kinds of employees or different kinds of accounts.

1.31 Drawing an ER Diagram ★★★



The symbols

Symbol	Meaning
Rectangle	Entity set
Double rectangle	Weak entity set
Ellipse	Attribute
Underlined attribute	Key attribute
Double ellipse	Multivalued attribute
Dashed ellipse	Derived attribute
Diamond	Relationship set
Double diamond	Identifying relationship
Double line	Total participation
Single line	Partial participation
Label 1, N, M on the line	Cardinality ratio

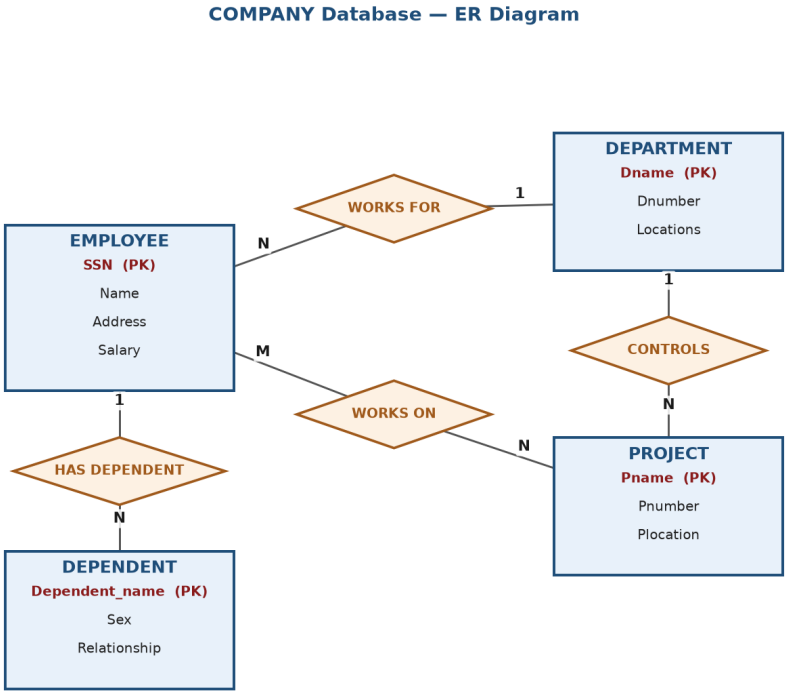
A method that works every time

1. Read the problem and underline every **noun** — these are the candidate entity sets. 2. Give each entity set its **key** attribute. 3. Read the verbs — these are the relationships, with their cardinality. 4. Attach the remaining attributes to the right place: to an entity, or to a relationship if they describe the association. 5. Mark weak entities, multivalued attributes and participation constraints. 6. Write one line of **assumptions** — examiners award marks for stated assumptions.

1.32 Worked ER Diagrams ★★★

Company database

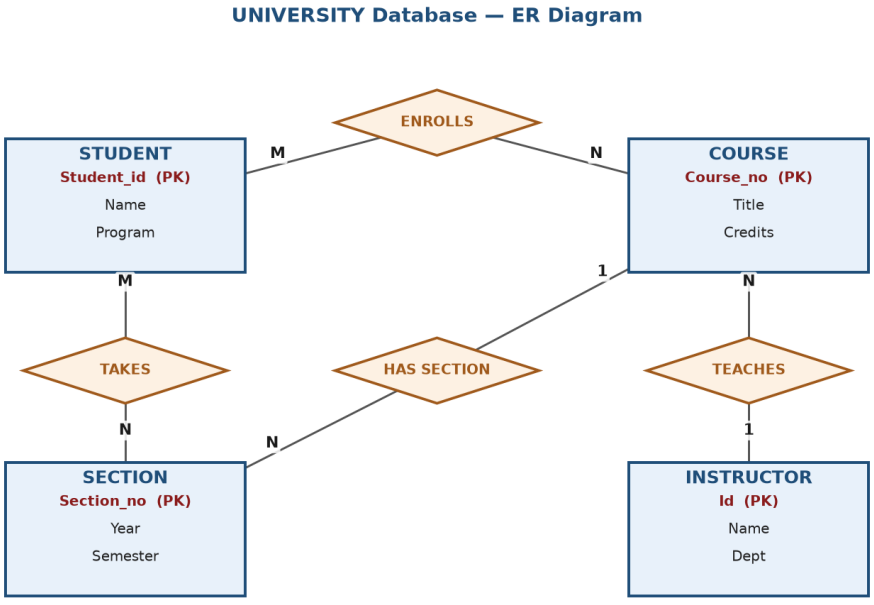
Departments with a name, number, manager and locations; projects controlled by departments; employees who work for one department and on many projects; dependents of employees.



WORKS_ON carries the descriptive attribute Hours. DEPENDENT is a weak entity (partial key Dependent_name).

University database

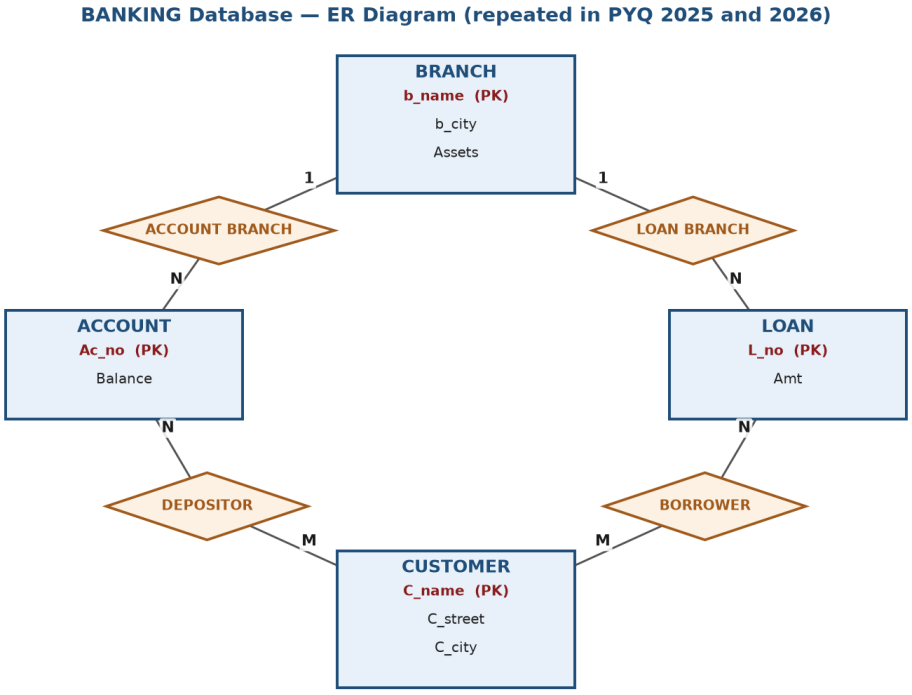
Students, courses and instructors, with sections as a weak entity of a course and grade as a descriptive attribute of the enrolment.



SECTION is a weak entity of COURSE; its partial key is Section_no. Grade is a descriptive attribute of TAKES.

Banking database

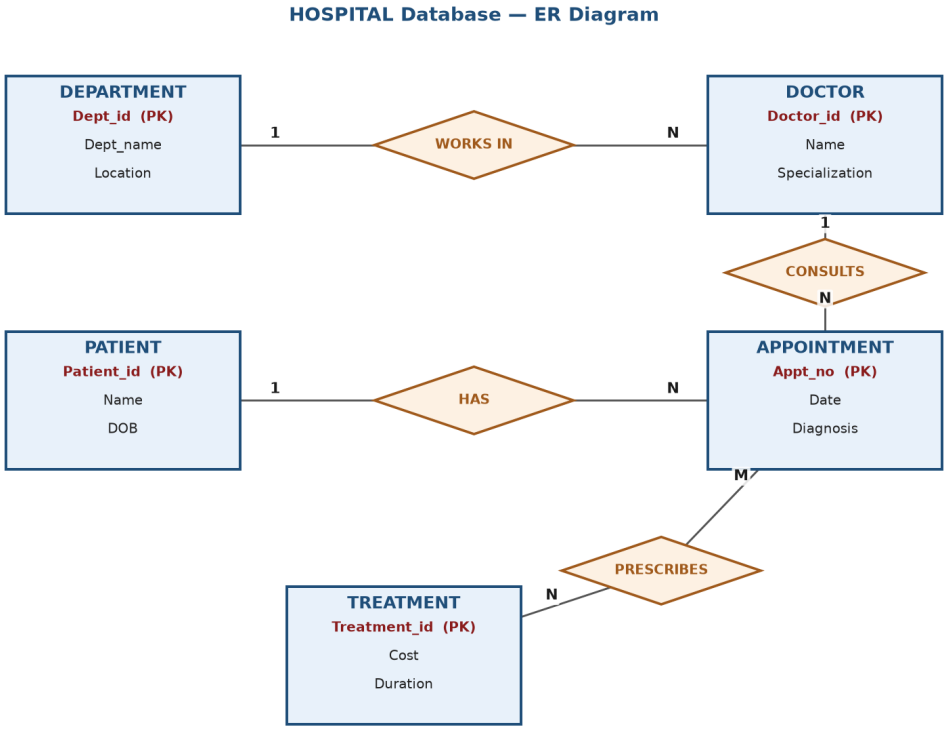
Branches owning accounts and loans, with customers related to both through depositor and borrower.



Maps to 6 tables: Account, Branch, Loan, Borrower, Customer, Depositor.

Hospital database

Departments and doctors, patients and appointments, with treatments prescribed during an appointment.



Appointment is modelled as an associative entity because it carries its own attributes.

What every one of these has in common

Each diagram is built from the same four decisions: which are the entity sets and their keys, which relationships exist and with what cardinality, which attributes belong to a relationship rather than an entity, and which entity sets are weak. Once those four are settled, the diagram follows, and so does the mapping to tables.