Transform ER Model to Relational Model || ER-Diagram to Tables

1. ER Model and Relational Model

Both ER-Model and Relational Model describe real-world data in a structured way.
Because both follow similar principles, we can convert an ER design into a Relational design.

2. ER Diagram to Relational Design

When we take an ER diagram and change it into tables, we get the Relational Database Design.
In simple words — **ER diagram is the blueprint, and converting it into tables creates the actual database.**

Very simple:

ER Model = Diagram
Relational Model = Tables

Converting the diagram into tables = **Relational DB Design**

3. ER Diagram to Relational Tables

1. Strong Entity

- It becomes a **separate table**.
- Its attributes become **columns** of the table.
- Its **Primary Key in ER model becomes Primary Key of the table**.
- If it has relations with other tables, **foreign keys (FK) are added**.

Example:
Entity → *Student(id, name, age)*
Table → **Student**
Columns → *id (PK), name, age*

2. Weak Entity

- It also becomes a **table**.
- It cannot exist alone, so we add **the Primary Key of the Strong Entity as a Foreign Key (FK)**.
- Its Primary Key is **combined (composite)** = *(FK + its own partial key)*

Example:
Weak Entity → *OrderItem(itemNo)* depends on **Order(orderId)**
Table PK → *{orderId (FK) + itemNo}*

3. Single-Valued Attribute

- Directly added as **normal columns** in the table.

Example:
Attribute: *name, rollNo* → added directly.

4. Composite Attributes

- These attributes are made up of smaller attributes.
- We split them into **separate columns**, and we do not store the main attribute name.

Example:
Address = {street, houseNo}
In table → **street**, **houseNo**

(Not storing "Address" as a single attribute)

5. Multivalued Attribute

- A **new table is created** for each multivalued attribute.
- The **PK of main table becomes FK** in this new table.
- PK = *(FK + multivalue attribute column)*

Example:
Entity: Employee(emp-id, **dependent-name** (multiple))
New Table → **dependent-name**

emp-id (FK)	**dname**

PK → {emp-id + dname}
FK → emp-id

Short

ER Concept	In Table
Strong Entity	**Becomes table, attributes → columns, PK same**
Weak Entity	**Table + FK from strong entity + composite PK**
Single Attribute	**Simple column**
Composite Attribute	**Break into parts → separate columns**
Multivalued Attribute	**New table, FK used, composite PK**

6. Derived Attributes

- These are values that can be calculated using other attributes.
- Because we can derive them anytime, **they are not stored as separate columns in tables**.

Example:
Age = Current Year – Birth Year
We don't store *Age* in the table because it keeps changing.
We only store *Birth Year*.

7. Generalisation → Converting to Tables

Generalisation means **one parent entity and multiple child entities**.

Example:
Account (Parent)
→ Savings Account
→ Current Account

There are **two ways to convert them into tables**:

Method 1 (Separate table for parent + separate tables for children)

- Create **one table for higher-level (parent)** entity.
- Create **separate tables for each lower-level (child)** entity.
- Child tables will also include the **primary key of parent table** as foreign key.

Tables

Table	Columns
account	account-number, balance
savings-account	account-number, interest-rate, daily-withdrawal-limit
current-account	account-number, overdraft-amount, per-transaction-charges

Here account-number connects child tables with main account table.

Method 2 (Only child tables, no parent table)

This method is used only when:

✓ Every account is either **savings** or **current**
✓ One account cannot belong to both (disjoint)

Here we **do not create parent table**, instead:

- Create only two separate tables for savings and current.
- Add **parents' attributes** inside each child table.

Tables

Table	Columns
savings-account	account-number, balance, interest-rate, daily-withdrawal-limit
current-account	account-number, balance, overdraft-amount, per-transaction-charges

Now both child tables also contain "**balance**" (parent attribute)

Drawbacks of Method 2

- If an account could be both savings and current → **Same data will be repeated** (ex: balance will be copied in both tables).
- If some accounts are neither savings nor current → **We cannot store them anywhere**.

- So Method 2 works only when classification is **clean, fixed and non-overlapping**.

Summary

Feature	Method 1	Method 2
Parent table exists?	Yes	No
Child table has parent's attributes?	Only PK	All parent columns
When suitable?	Any generalisation	Only if complete + disjoint
Data duplication?	No	Yes, possible

8. Aggregation in ER → Conversion to Relations

Aggregation means **treating a whole relationship like an entity**
when another entity depends on it.

When converting Aggregation to relational tables:

1. We create a **separate table** for the aggregated relationship.
2. The table contains **Primary Keys of all entities** involved in the aggregation.
3. If the relationship has any **extra attributes**, we also add them as columns.

Simple Example

Suppose we have:

Entities:

- Student
- Project
- Company

Relationship:
A company funds a *student + project* pair.
Here, *Student–Project* is treated as one unit (aggregation).

Tables Created

Table	Columns
Student	student_id, name, ...
Project	project_id, title, ...
Company	company_id, company_name
Funds (aggregation table)	student_id (FK), project_id (FK), company_id (FK), amount

Why Aggregation Table is Needed?

Because **Company funds combined work of Student + Project**,
not student alone, not project alone.

So we treat *(Student + Project)* as one object → Aggregation.
And create a table to store that relation.

Simple

Aggregation converts complex relationships into a separate table containing the keys of all participating entities + relationship attributes.