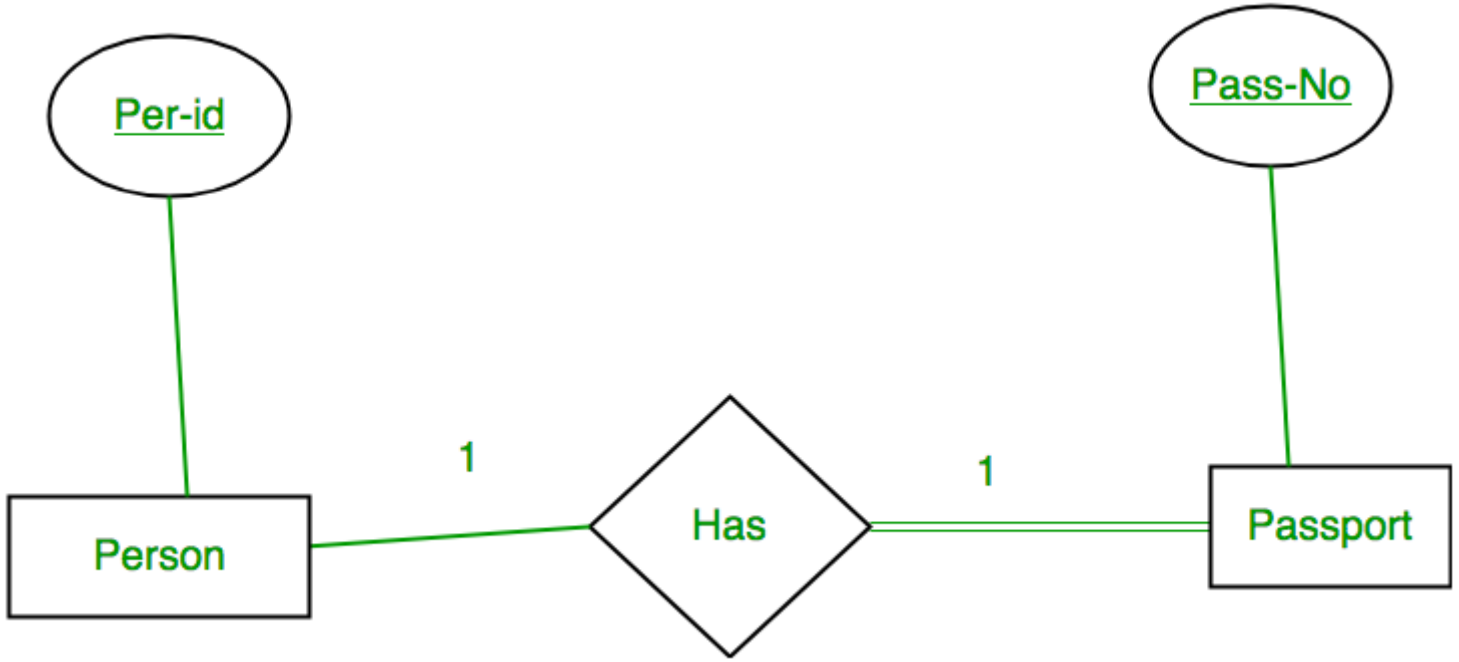
Mapping from ER Model to Relational Model

After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL etc. In this article we will discuss how to convert ER diagram to Relational Model for different scenarios.

Mapping from ER Model to Relational Model

Case 1: Binary Relationship with 1:1 cardinality with total participation of an entity



Mapping from ER Model to Relational Model

A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.

First Convert each entity and relationship to tables. Person table corresponds to Person Entity with key as Per-Id. Similarly Passport table corresponds to Passport Entity with key as Pass-No. HashTable represents relationship between Person and Passport (Which person has which passport). So it will take attribute Per-Id from Person and Pass-No from Passport.

Mapping from ER Model to Relational Model

Person			**Has**			**Passport**	
Per-Id	**Other Person Attribute**		**Per-Id**	**Pass-No**		**Pass-No**	**Other PassportAttribute**
PR1	–		PR1	PS1		PS1	–
PR2	–		PR2	PS2		PS2	–
PR3	–						

Table 1

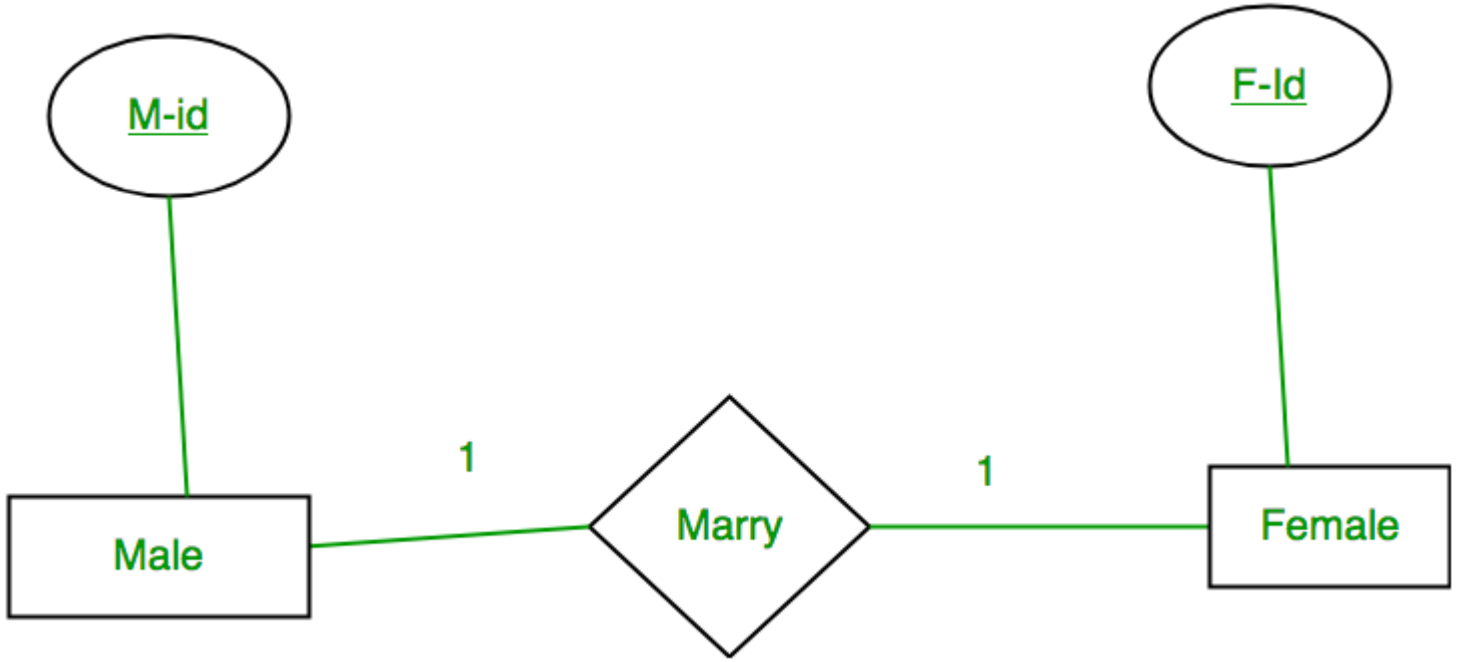
As we can see from Table 1, each Per-Id and Pass-No has only one entry in **Hash**table. So we can merge all three tables into 1 with attributes shown in Table 2. Each Per-Id will be unique and not null. So it will be the key. Pass-No can't be key because for some person, it can be NULL.

Per-Id	Other Person Attribute	Pass-No	Other PassportAttribute

Table 2

Mapping from ER Model to Relational Model

Case 2: Binary Relationship with 1:1 cardinality and partial participation of both entities



Mapping from ER Model to Relational Model

A male marries 0 or 1 female and vice versa as well. So it is 1:1 cardinality with partial participation constraint from both. First Convert each entity and relationship to tables. Male table corresponds to Male Entity with key as M-Id. Similarly Female table corresponds to Female Entity with key as F-Id. Marry Table represents relationship between Male and Female (Which Male marries which female). So it will take attribute M-Id from Male and F-Id from Female.

Table 3

Male			**Marry**			**Female**	
M-Id	Other Male Attribute		M-Id	F-Id		F-Id	Other FemaleAttribute
M1	–		M1	F2		F1	–
M2	–		M2	F1		F2	–
M3	–					F3	–

As we can see from Table 3, some males and some females do not marry. If we merge 3 tables into 1, for some M-Id, F-Id will be NULL. So there is no attribute which is always not NULL. So we can't merge all three tables into 1. We can convert into 2 tables. In table 4, M-Id who are married will have F-Id associated. For others, it will be NULL. Table 5 will have information of all females. Primary Keys have been underlined.

Table 4

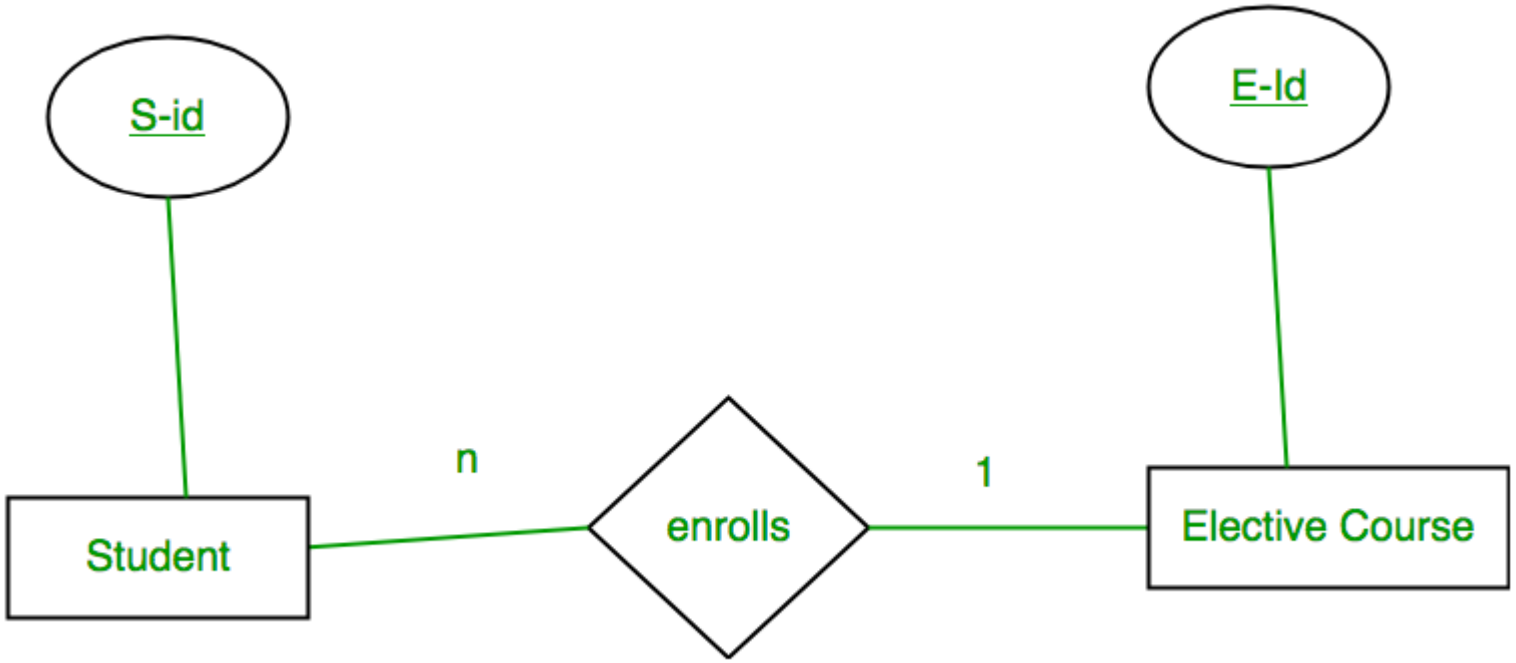
M-Id	Other Male Attribute	F-Id

Table 5

F-Id	Other FemaleAttribute

Note: Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship. If atleast 1 entity has total participation, number of tables required will be 1.

Case 3: Binary Relationship with n: 1 cardinality



In this scenario, every student can enroll only in one elective course but for an elective course there can be more than one student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with key as S-Id. Similarly Elective_Course table corresponds to Elective_Course Entity with key as E-Id. Enrolls Table represents relationship between Student and Elective_Course (Which student enrolls in which course). So it will take attribute S-Id from and Student E-Id from Elective_Course.

Table 6

Student			**Enrolls**			**Elective_Course**	
S-Id	Other Student Attribute		S-Id	E-Id		E-Id	Other Elective CourseAttribute
S1	–		S1	E1		E1	–
S2	–		S2	E2		E2	–
S3	–		S3	E1		E3	–
S4	–		S4	E1			

As we can see from Table 6, S-Id is not repeating in Enrolls Table. So it can be considered as a key of Enrolls table. Both Student and Enrolls Table's key is same; we can merge it as a single table. The resultant tables are shown in Table 7 and Table 8. Primary Keys have been underlined.

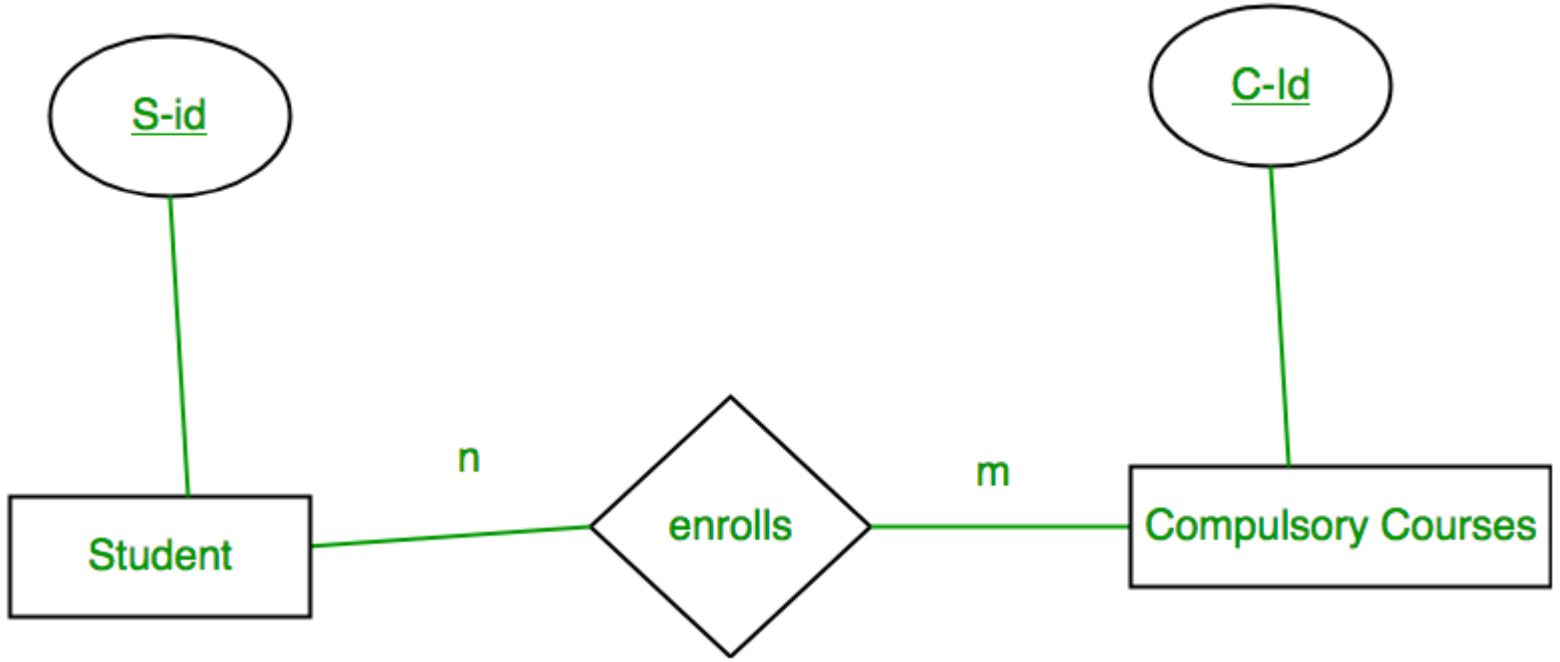
Table 7

S-Id	Other Student Attribute	E-Id

Table 8

E-Id	Other Elective CourseAttribute

Case 4: Binary Relationship with m: n cardinality



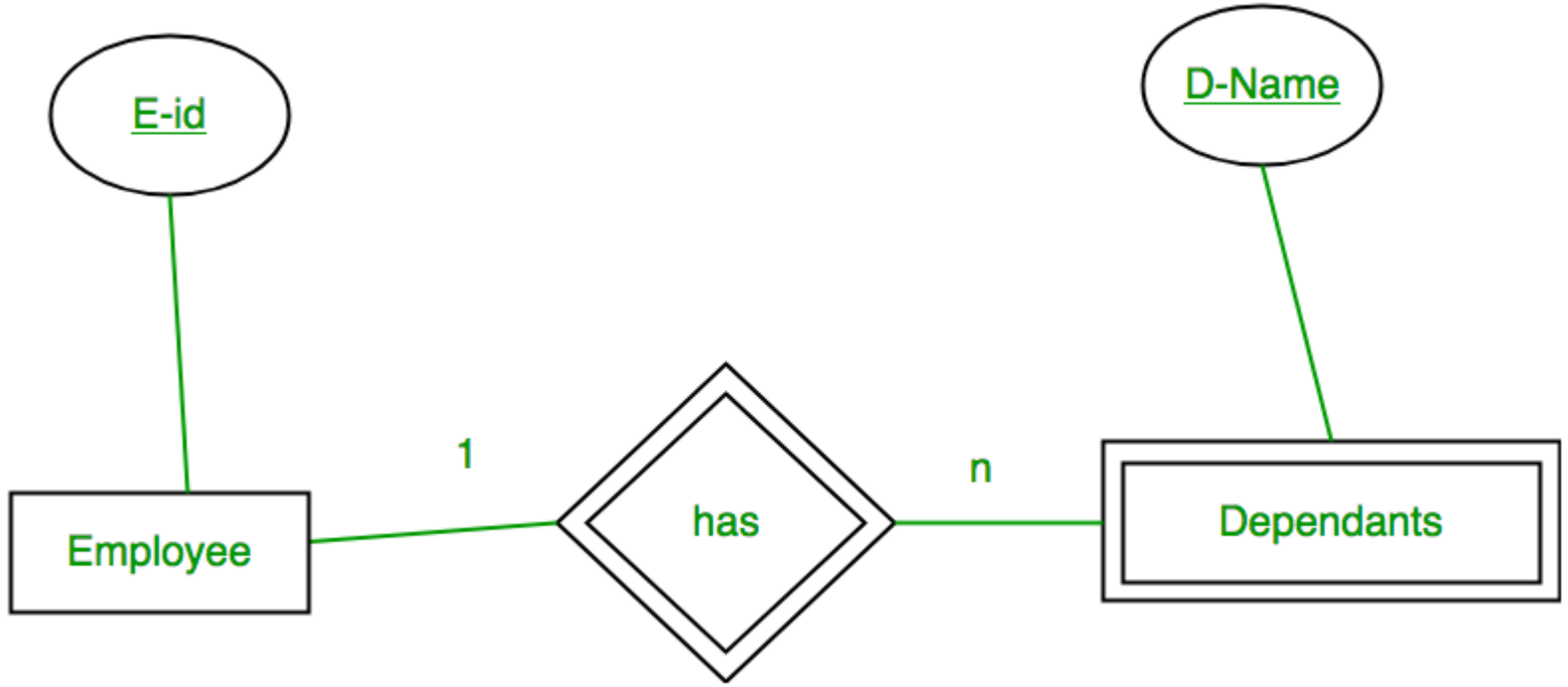
In this scenario, every student can enroll in more than 1 compulsory course and for a compulsory course there can be more than 1 student. First Convert each entity and relationship to tables. Student table corresponds to Student Entity with key as S-Id. Similarly Compulsory_Courses table corresponds to Compulsory Courses Entity with key as C-Id. Enrolls Table represents relationship between Student and Compulsory_Courses (Which student enrolls in which course). So it will take attribute S-Id from Person and C-Id from Compulsory_Courses.

Table 9

Student			**Enrolls**			**Compulsory_Courses**	
S-Id	Other Student Attribute		S-Id	C-Id		C-Id	Other Compulsory CourseAttribute
S1	–		S1	C1		C1	–
S2	–		S1	C2		C2	–
S3	–		S3	C1		C3	–
S4	–		S4	C3		C4	–
			S4	C2			
			S3	C3			

As we can see from Table 9, S-Id and C-Id both are repeating in Enrolls Table. But its combination is unique; so it can be considered as a key of Enrolls table. All tables' keys are different, these can't be merged. Primary Keys of all tables have been underlined.

Case 5: Binary Relationship with weak entity



In this scenario, an employee can have many dependents and one dependent can depend on one employee. A dependent does not have any existence without an employee (e.g; you as a child can be dependent of your father in his company). So it will be a weak entity and its participation will always be total. Weak Entity does not have key of its own. So its key will be combination of key of its identifying entity (E-Id of Employee in this case) and its partial key (D-Name).

First Convert each entity and relationship to tables. Employee table corresponds to Employee Entity with key as E-Id. Similarly Dependents table corresponds to Dependent Entity with key as D-Name and E-Id. HashTable represents relationship between Employee and Dependents (Which employee has which dependents). So it will take attribute E-Id from Employee and D-Name from Dependents.

Table 10

Employee		**Has**		**Dependents**		
E-Id	Other Employee Attribute	E-Id	D-Name	D-Name	E-Id	Other DependentsAttribute
E1	–	E1	RAM	RAM	E1	–
E2	–	E1	SRINI	SRINI	E1	–
E3	–	E2	RAM	RAM	E2	–
		E3	ASHISH	ASHISH	E3	–

As we can see from Table 10, E-Id, D-Name is key for **Has** as well as Dependents Table. So we can merge these two into 1. So the resultant tables are shown in Tables 11 and 12. Primary Keys of all tables have been underlined.

Table 11

E-Id	Other Employee Attribute

Table 12

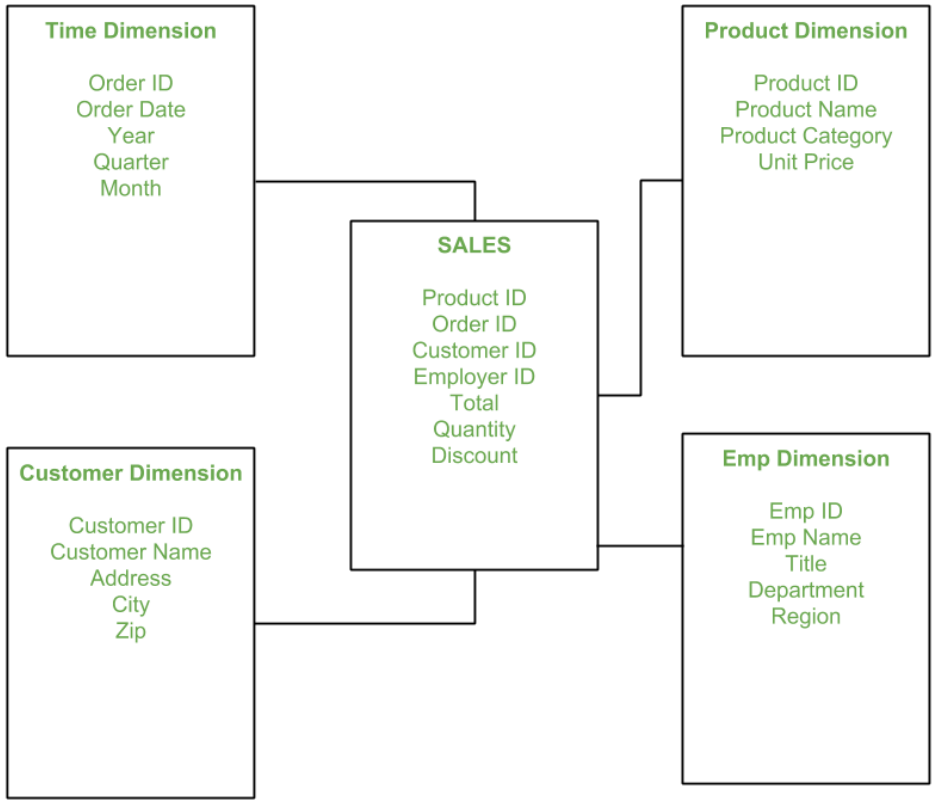
D-Name	E-Id	Other DependentsAttribute

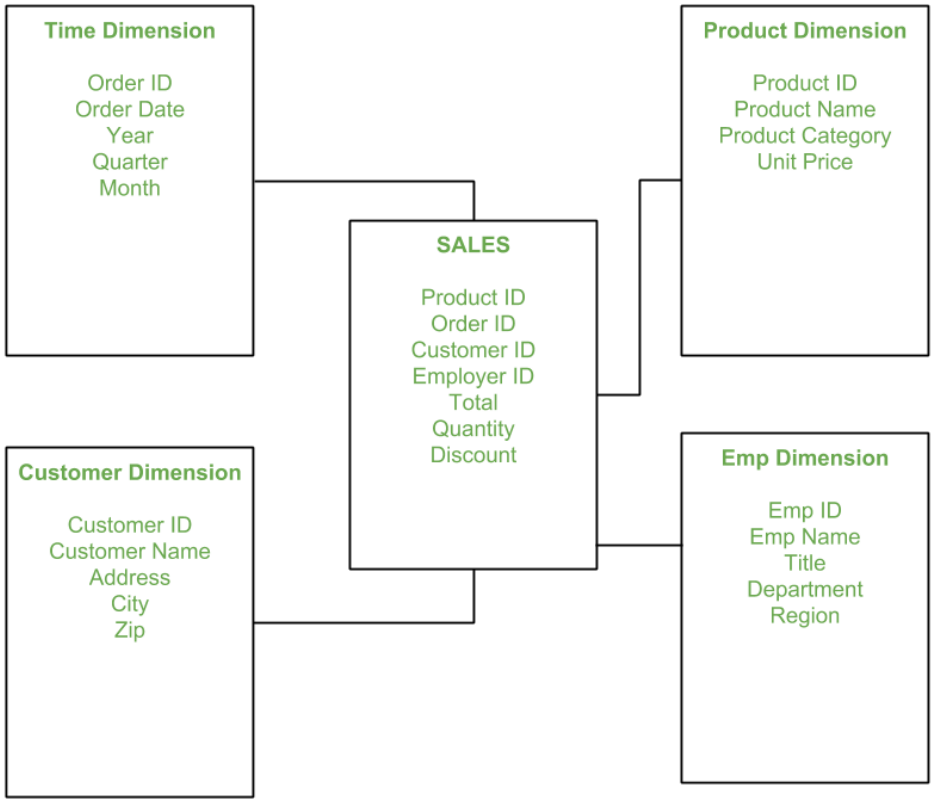
Star Schema in Data Warehouse modeling

Star schema is the fundamental schema among the data mart schema and it is simplest. This schema is widely used to develop or build a data warehouse and dimensional data marts. It includes one or more fact tables indexing any number of dimensional tables. The star schema is a necessary cause of the snowflake schema. It is also efficient for handling basic queries.

Star Schema in Data Warehouse modeling

It is said to be star as its physical model resembles to the star shape having a fact table at its center and the dimension tables at its peripheral representing the star's points. Below is an example to demonstrate the Star Schema:





In the above demonstration, SALES is a fact table having attributes i.e. (Product ID, Order ID, Customer ID, Employer ID, Total, Quantity, Discount) which references to the dimension tables. **Employee dimension table** contains the attributes: Emp ID, Emp Name, Title, Department and Region. *Product dimension table* contains the attributes: Product ID, Product Name, Product Category, Unit Price. *Customer dimension table* contains the attributes: Customer ID, Customer Name, Address, City, Zip. *Time dimension table* contains the attributes: Order ID, Order Date, Year, Quarter, Month.

Advantages and disadvantages of star schema

Advantages of Star Schema :

Simpler Queries –
Join logic of star schema is quite cinch in comparison to other join logic which are needed to fetch data from a transactional schema that is highly normalized.

Simplified Business Reporting Logic –
In comparison to a transactional schema that is highly normalized, the star schema makes simpler common business reporting logic, such as as-of reporting and period-over-period.

Feeding Cubes –
Star schema is widely used by all OLAP systems to design OLAP cubes efficiently. In fact, major OLAP systems deliver a ROLAP mode of operation which can use a star schema as a source without designing a cube structure.

Disadvantages of Star Schema –

Data integrity is not enforced well since in a highly de-normalized schema state.

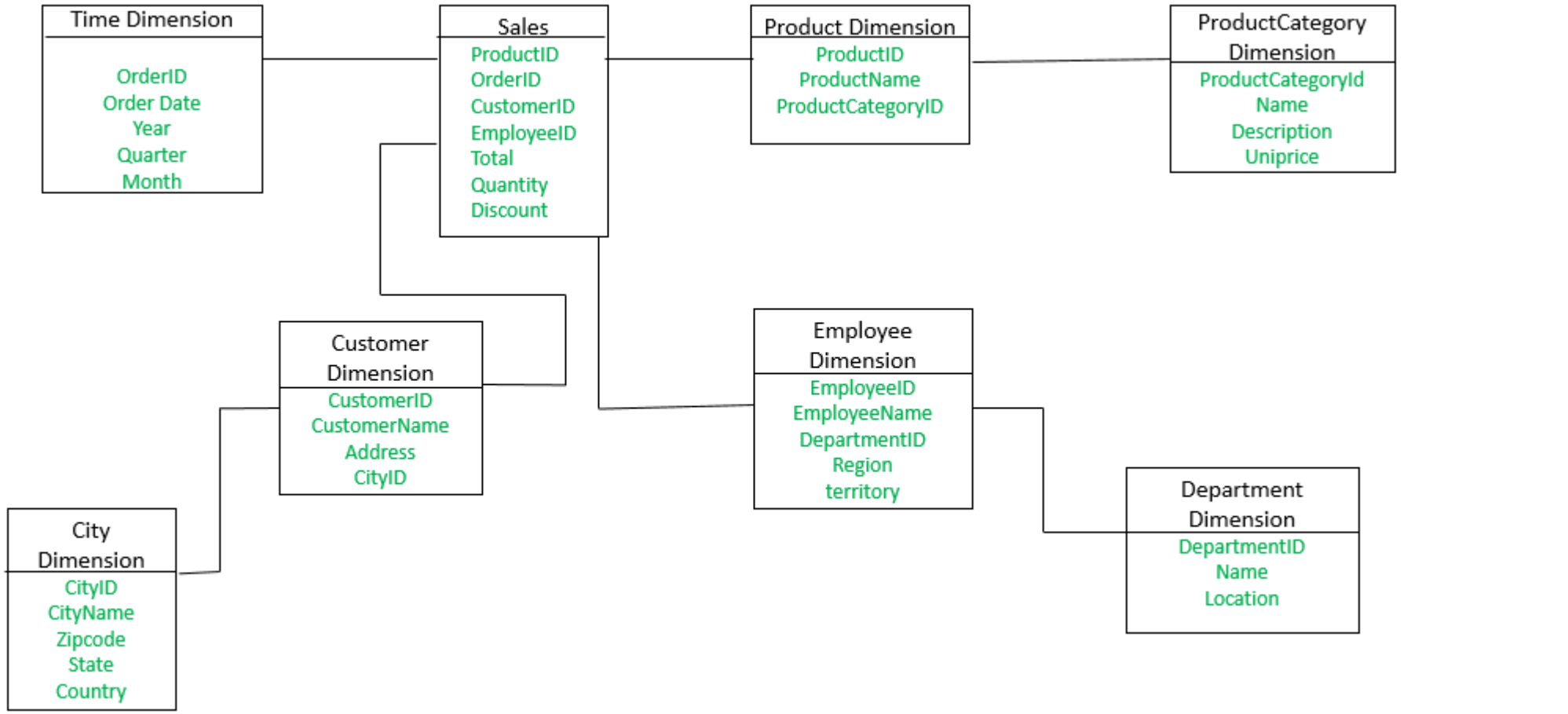
Not flexible in terms if analytical needs as a normalized data model.

Star schemas don't reinforce many-to-many relationships within business entities – at least not frequently.

Snowflake Schema in Data Warehouse Model

Introduction: The snowflake schema is a variant of the star schema. Here, the centralized fact table is connected to multiple dimensions. In the snowflake schema, dimensions are present in a normalized form in multiple related tables. The snowflake structure materialized when the dimensions of a star schema are detailed and highly structured, having several levels of relationship, and the child tables have multiple parent tables. The snowflake effect affects only the dimension tables and does not affect the fact tables.

Snowflake Schema in Data Warehouse Model



Snowflake Schema in Data Warehouse Model

The **Employee** dimension table now contains the attributes: EmployeeID, EmployeeName, DepartmentID, Region, Territory. The DepartmentID attribute links with the **Employee** table with the **Department** dimension table. The **Department** dimension is used to provide detail about each department, such as the Name and Location of the department. The **Customer** dimension table now contains the attributes: CustomerID, CustomerName, Address, CityID. The CityID attributes link the **Customer** dimension table with the **City** dimension table. The **City** dimension table has details about each city such as CityName, Zipcode, State, and Country.

The main difference between star schema and snowflake schema is that the dimension table of the snowflake schema is maintained in the normalized form to reduce redundancy. The advantage here is that such tables (normalized) are easy to maintain and save storage space. However, it also means that more joins will be needed to execute the query. This will adversely impact system performance.

Advantages: There are two main advantages of snowflake schema given below:

It provides structured data which reduces the problem of data integrity.

It uses small disk space because data are highly structured.

Disadvantages:

Snowflaking reduces space consumed by dimension tables but compared with the entire data warehouse the saving is usually insignificant.

Avoid snowflaking or normalization of a dimension table, unless required and appropriate.

Do not snowflake hierarchies of one dimension table into separate tables. Hierarchies should belong to the dimension table only and should never be snowflakes.

Multiple hierarchies that can belong to the same dimension have been designed at the lowest possible detail.