DATABASE MANAGEMENT SYSTEM

Chapter 6
Logical Design
Chapter 7
Normalization

Bikash Khadka Shah

MCSE, MSDA, OCP, RHCE, RHCVA, CCNA, CEH

9841766620 | 9801076620

Objectives of logical design...

- Translate the conceptual design into a logical database design that can be implemented on a chosen DBMS
  - Input: conceptual model (ERD)
  - Output: relational schema, normalized relations
- Resulting database must meet user needs for:
  - Data sharing
  - Ease of access
  - Flexibility

Relational database components

- Data structure
  - Data organized into tables
- Data manipulation
  - Add, delete, modify, and retrieve using SQL
- Data integrity
  - Maintained using business rules

Why do I need to know this?

- Mapping conceptual models to relational schema is straight-forward
- Computer Aided Software Engineering (CASE) tools can perform many of the steps, but..
  - Often CASE cannot model complexity of data and relationship (e.g., Ternary relationships, supertype/subtypes)
  - There are times when legitimate alternates must be evaluated
  - You must be able to perform a quality check on CASE tool results

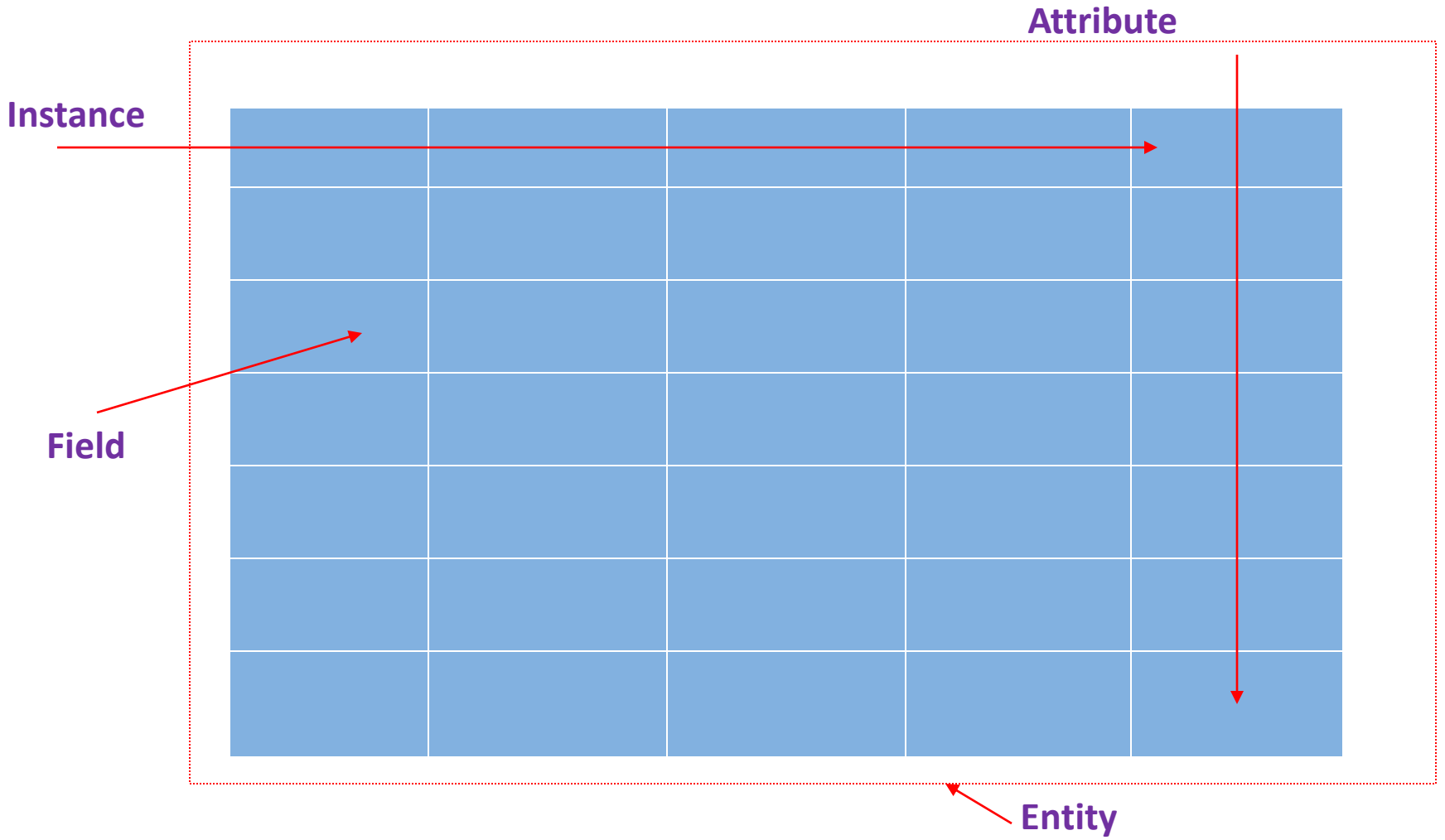
Some rules...

- Every table has a unique name.
- Attributes in tables have unique names.
- Every attribute value is atomic.
  - Multi-valued and composite attributes?
- Every row is unique.
- The order of the columns is irrelevant.
- The order of the rows is irrelevant.

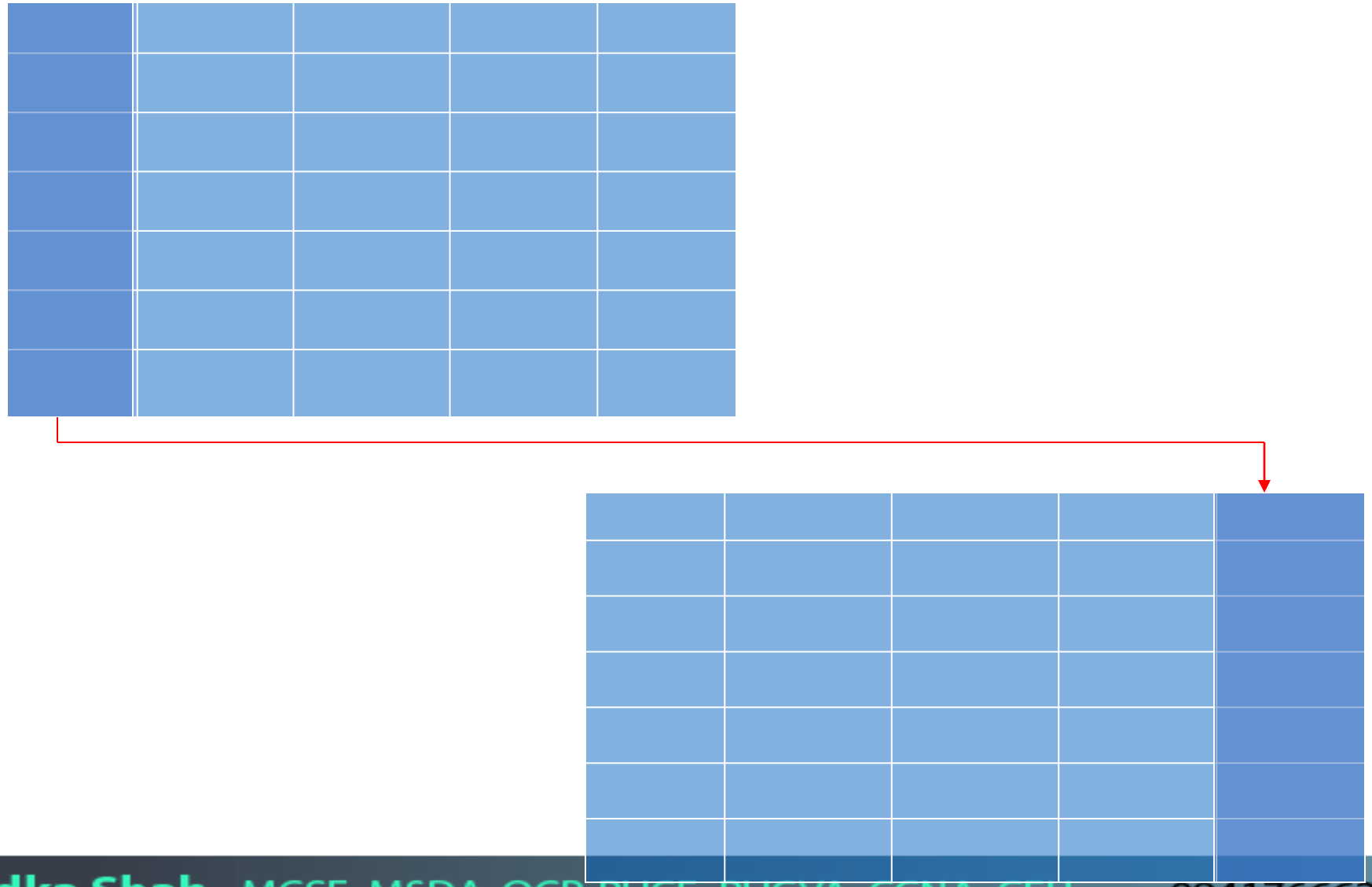
The key...

- Relational modeling uses primary keys and foreign keys to maintain relationships
- Primary keys are typically the unique identifier noted on the conceptual model
- Foreign keys are the primary key of another entity to which an entity has a relationship
- Composite keys are primary keys that are made of more than one attribute
  - Weak entities
  - Associative entities

Implementing it



What about relationships?



Constraints

- Domain constraints
  - Allowable values for an attribute as defined in the domain
- Entity integrity constraints
  - No primary key attribute may be null
- Operational constraints
  - Business rules
- Referential integrity constraints

Referential integrity constraint

- Maintains consistency among rows of two entities
  - matching of primary and foreign keys
- Enforcement options for deleting instances
  - Restrict
  - Cascade
  - Set-to-Null

Transforming the EER diagram into relations

The steps:

- Map regular entities
- Map weak entities
- Map binary relationships
- Map associative entities
- Map unary relationships
- Map ternary relationships
- Map supertype/subtype relationships

Transforming E-R diagrams into relations

Mapping regular entities to relations

- Composite attributes: use only their simple, component attributes
- Multi-valued attributes: become a separate relation with a foreign key taken from the superior entity

Looks like this using relational schema notation

CUSTOMER

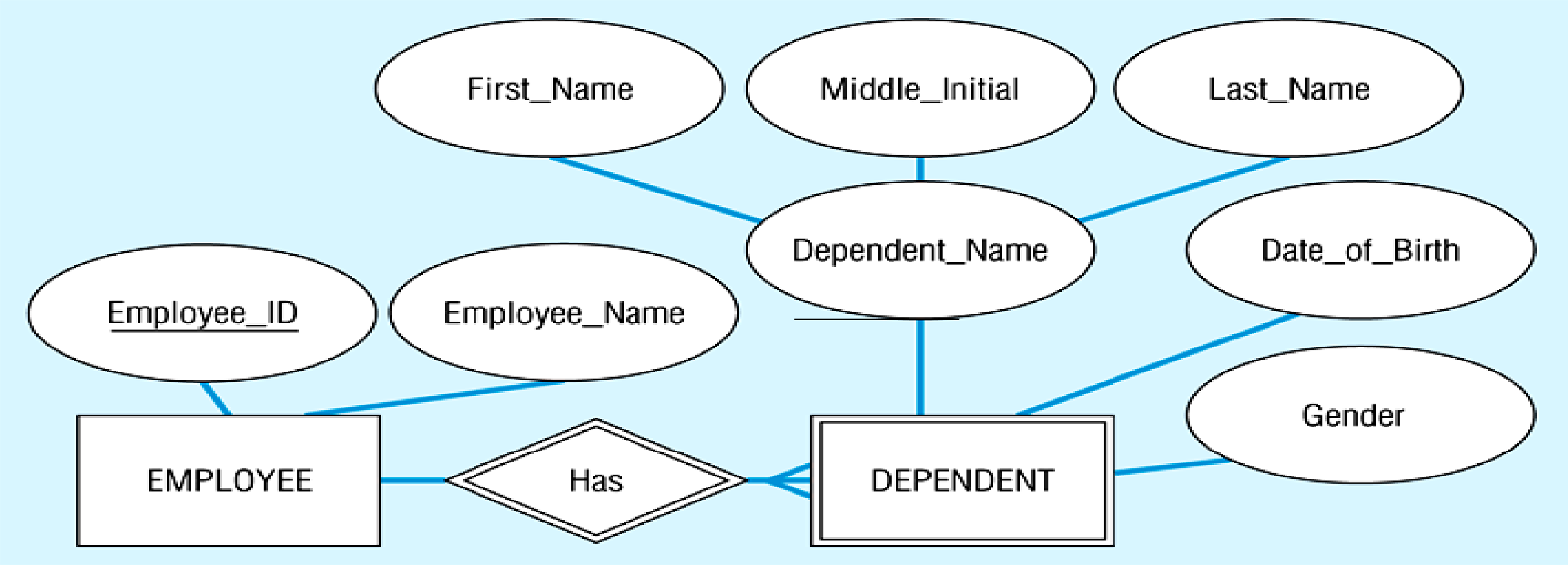
Customer_ID	Customer_Name	Street	City	State	Zip

Transforming E-R diagrams into relations

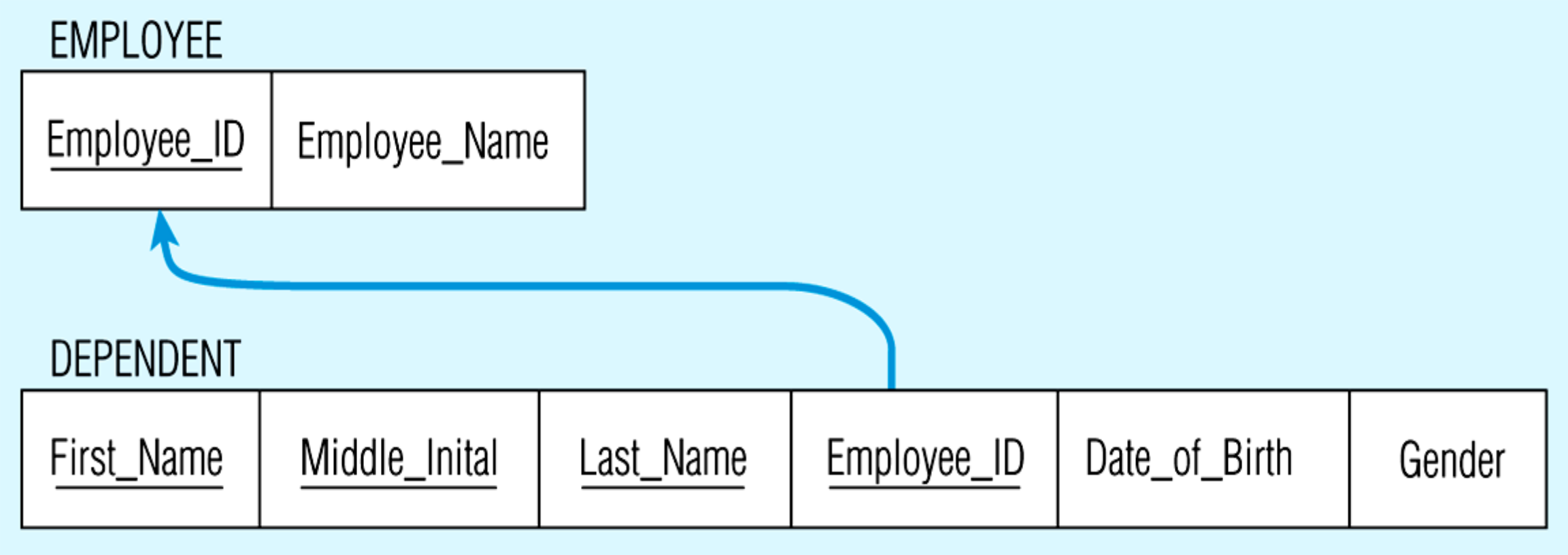
Mapping weak entities

- Becomes a separate relation with a foreign key taken from the superior entity

Example of mapping a weak entity



Looks like this using relational schema notation

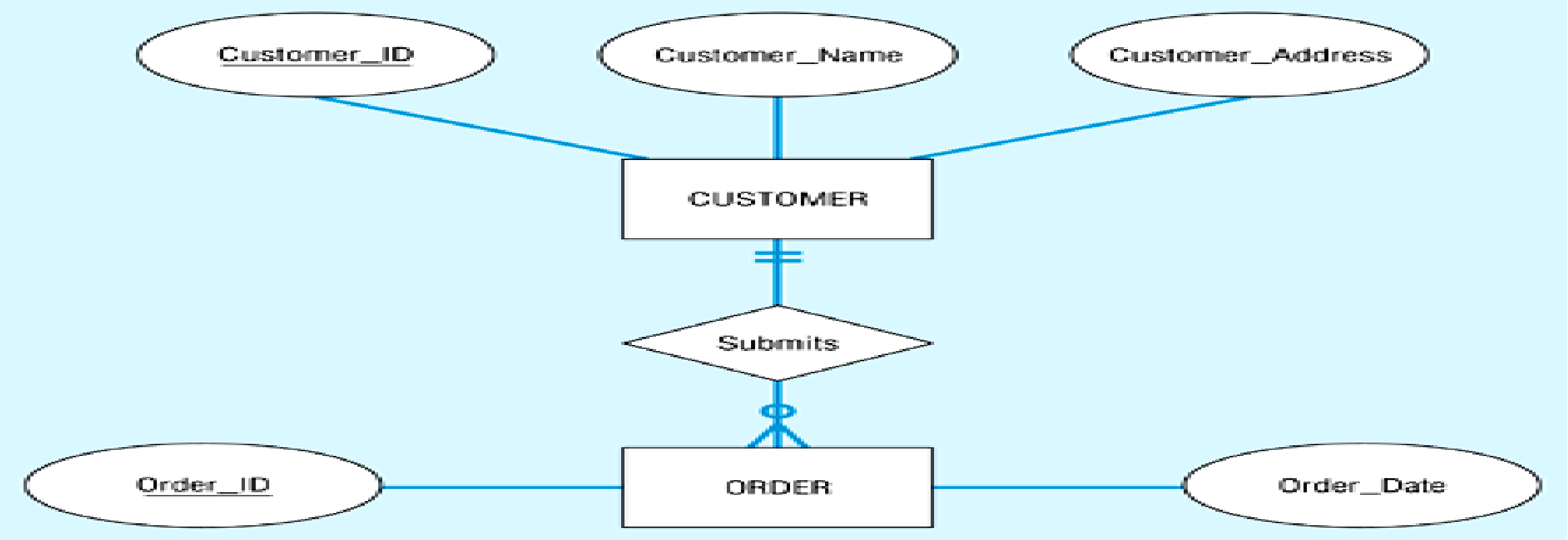


Transforming E-R diagrams into relations

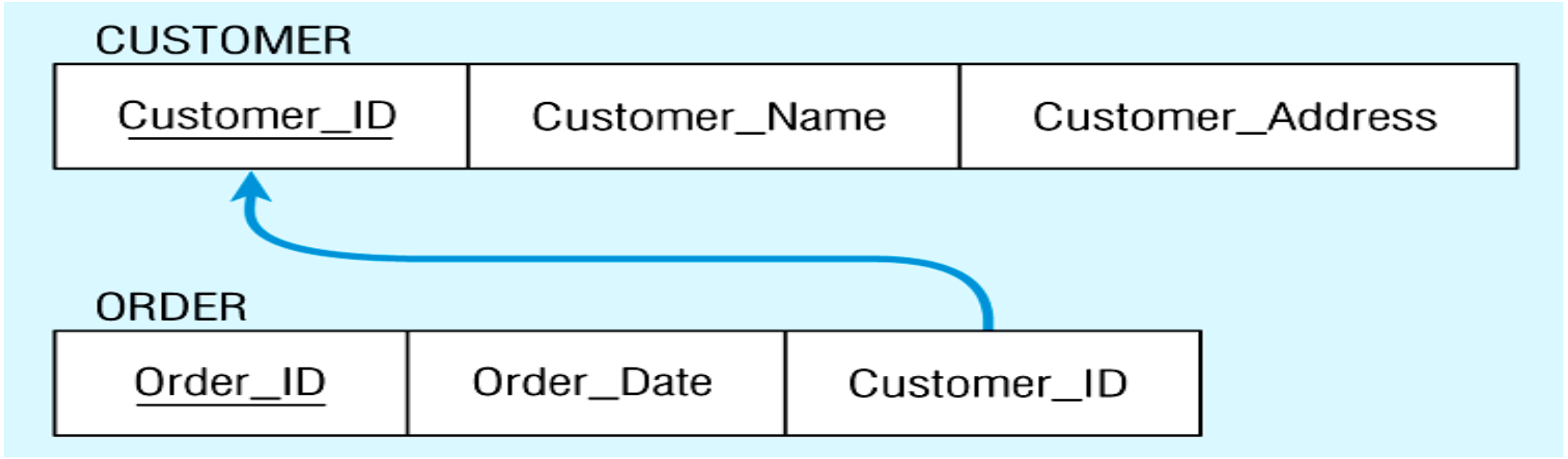
Mapping binary relationships

- One-to-many - primary key on the one side becomes a foreign key on the many side
- Many-to-many - create a new relation (associative entity) with the primary keys of the two entities as its primary key
  - I like to call these **intersection entities** to distinguish them from associative entities created at the conceptual level
- One-to-one - primary key on the mandatory side becomes a foreign key on the optional side

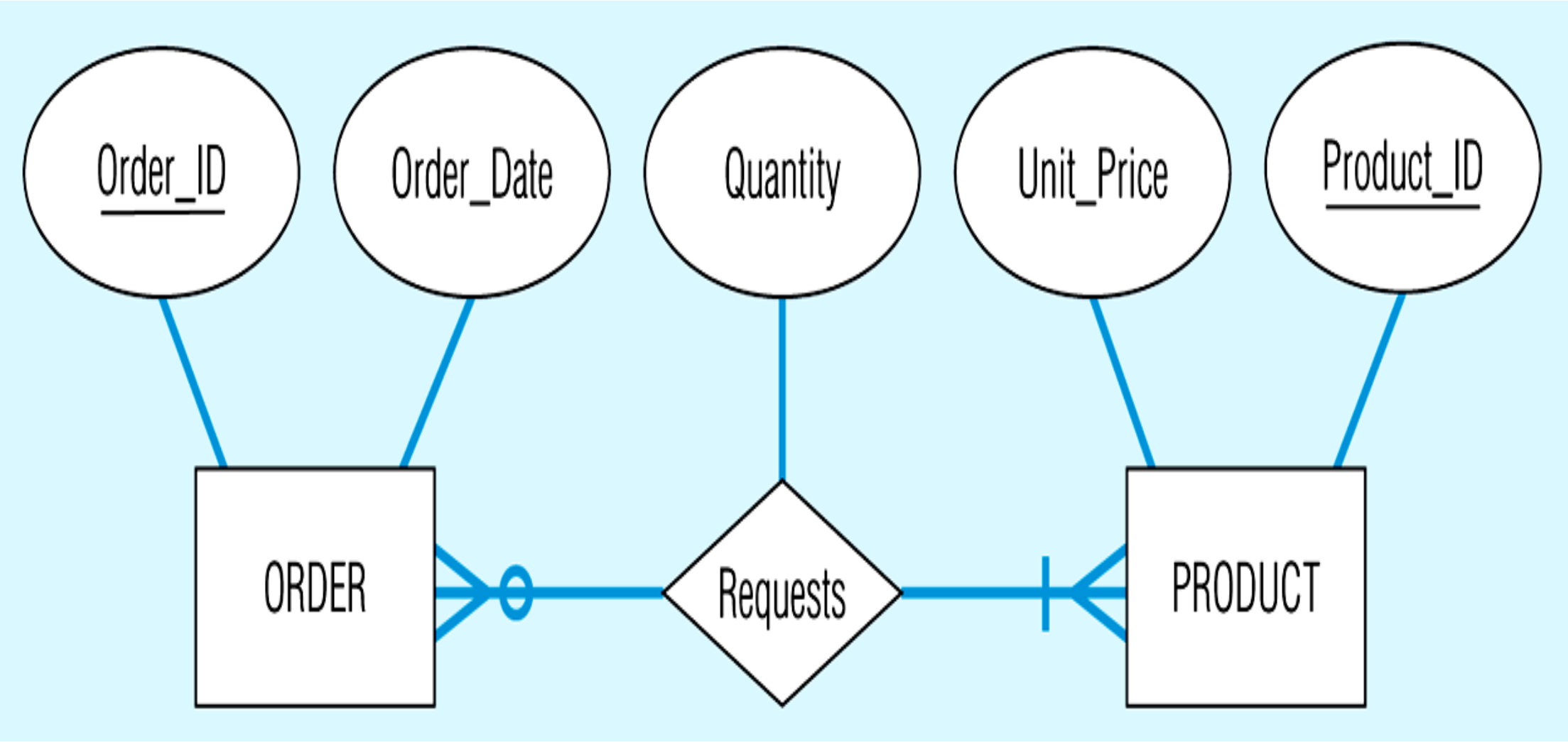
Example of mapping a 1:M relationship



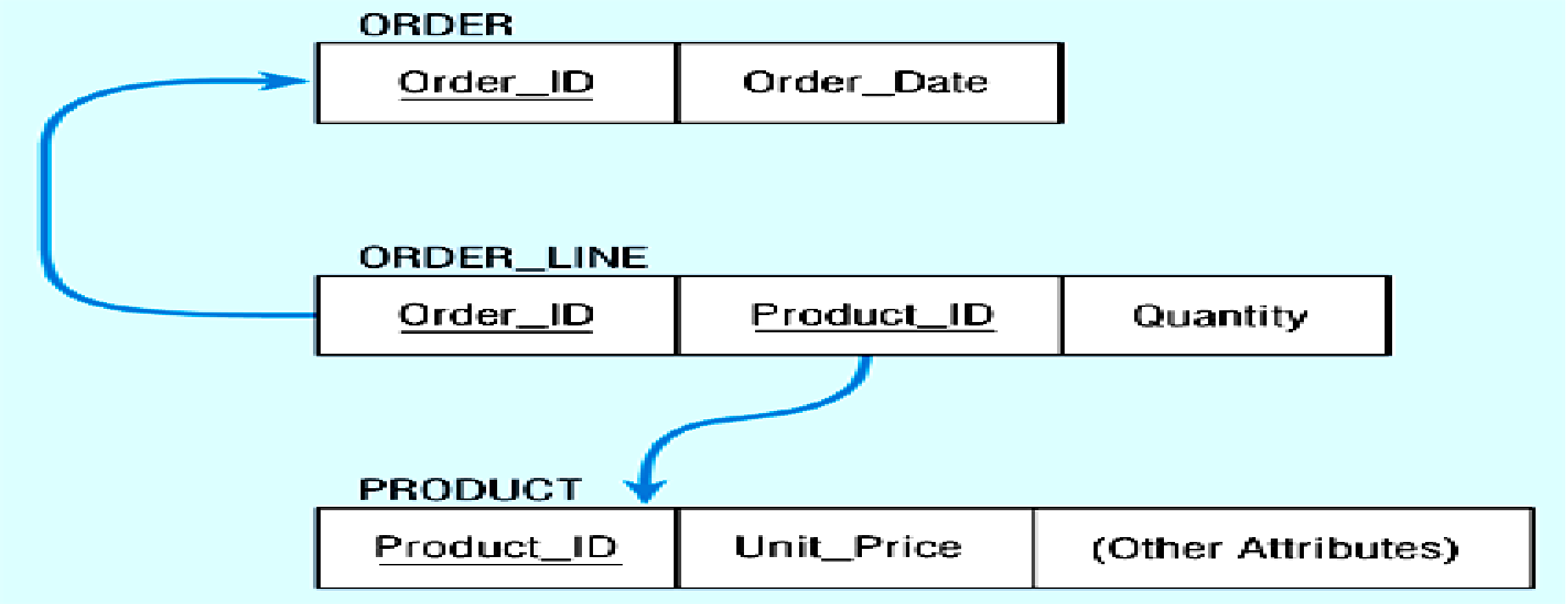
Looks like this using relational schema notation



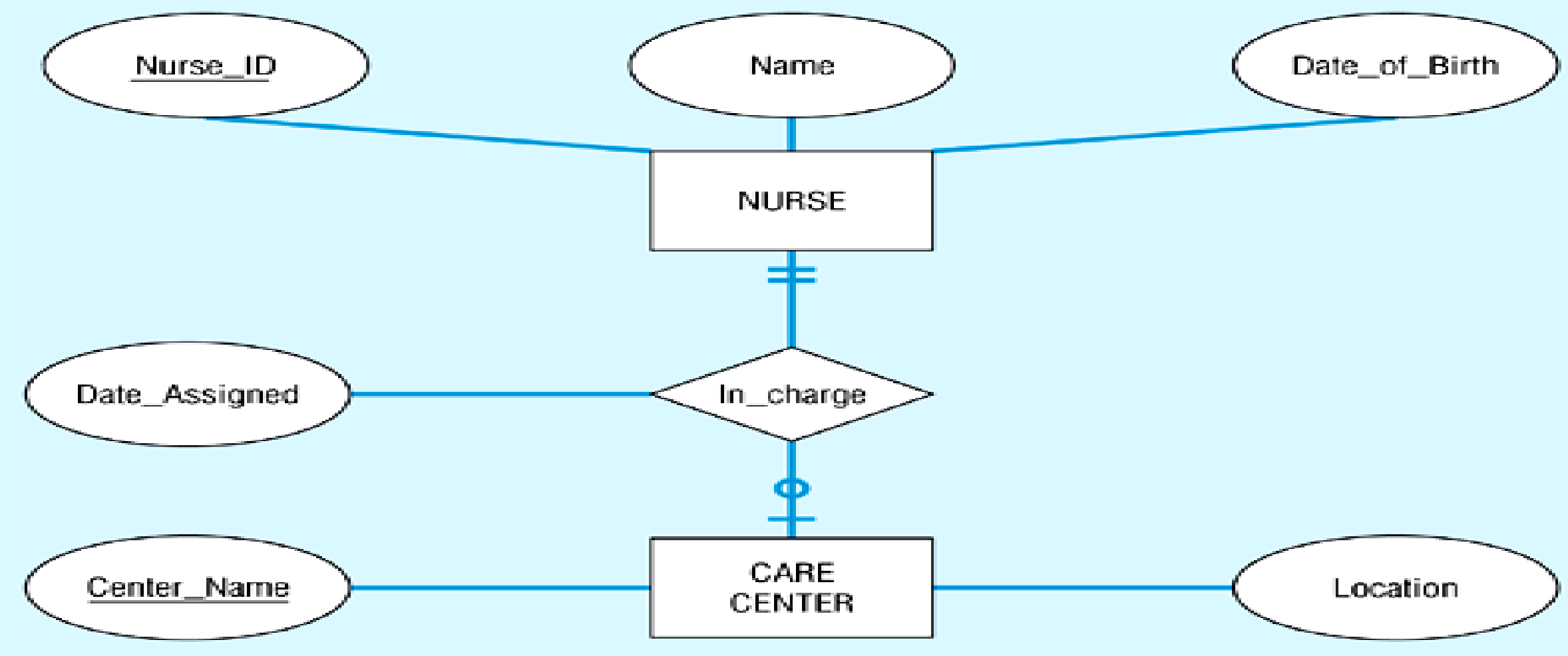
Example of mapping an M:M relationship



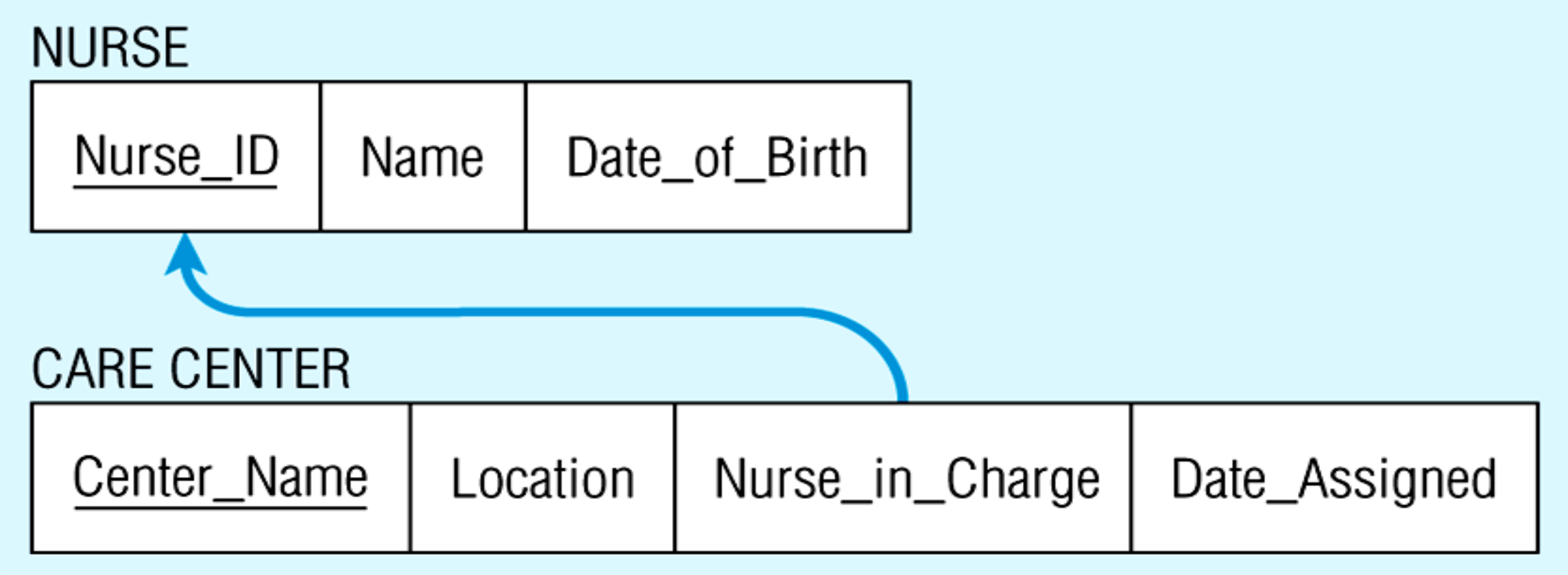
Looks like this using relational schema notation



Mapping a binary 1:1 relationship



Looks like this using relational schema notation

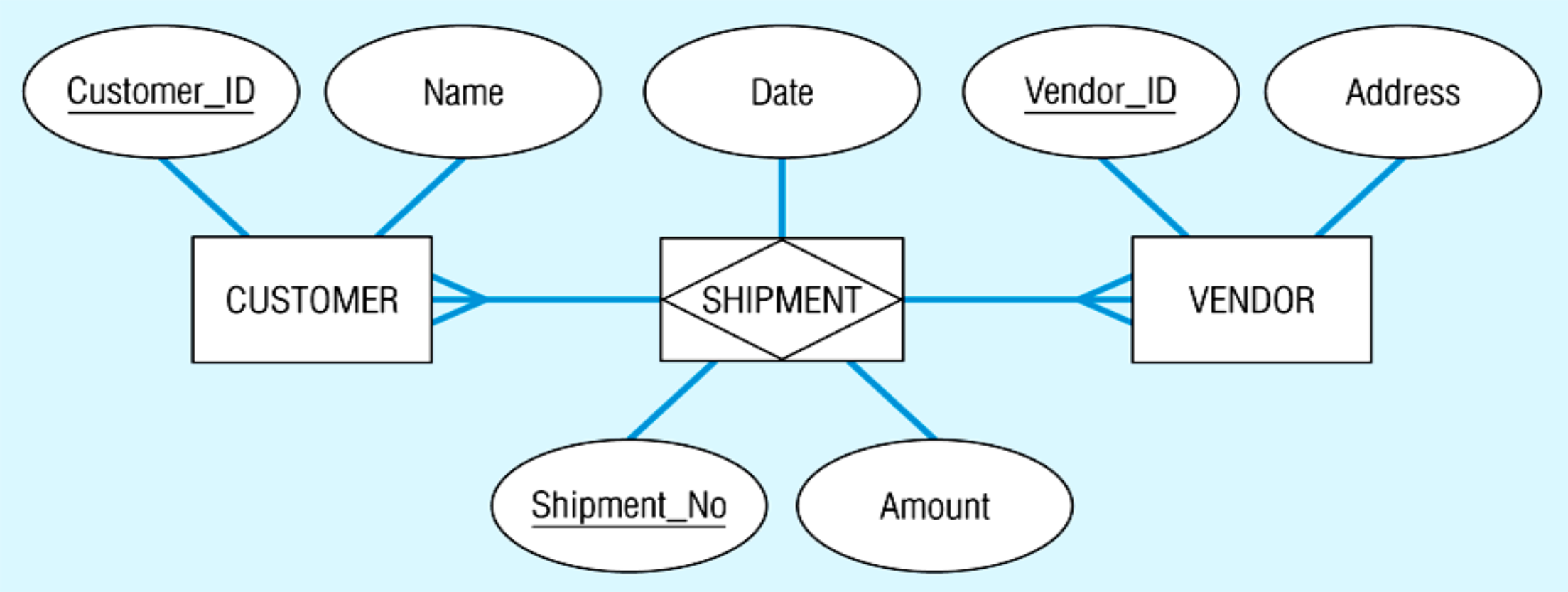


Transforming E-R diagrams into relations

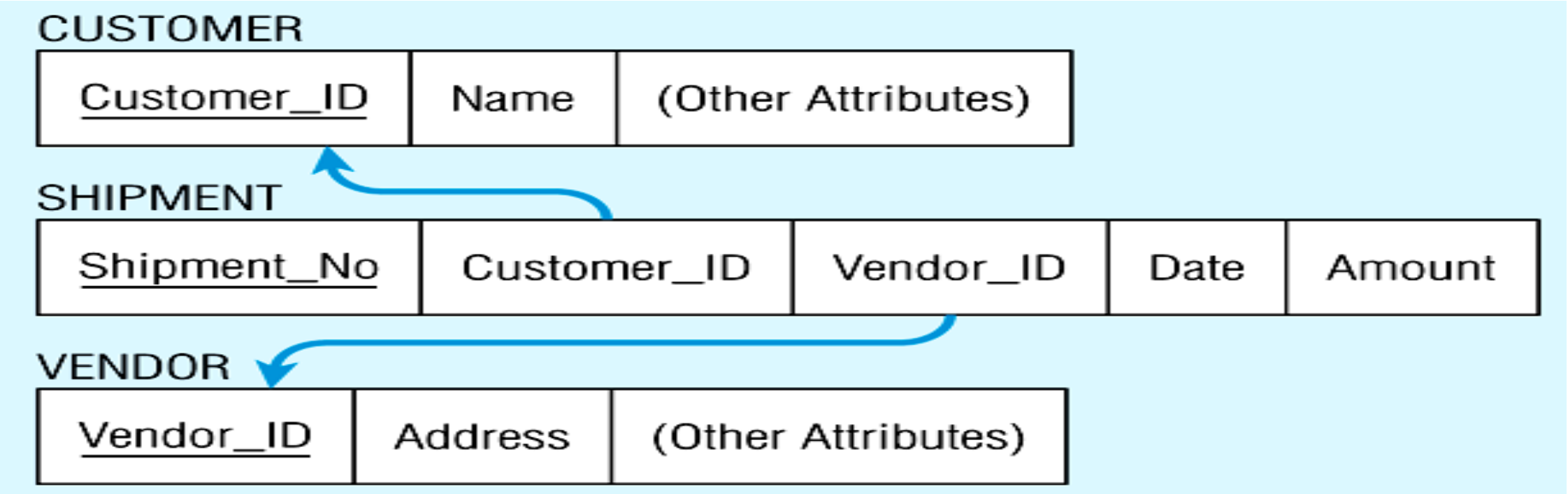
Mapping associative entities

- Identifier not assigned
  - Default primary key for the association relation is the primary keys of the two entities
- Identifier assigned
  - It is natural and familiar to end-users
  - Default identifier may not be unique

Mapping an associative entity with an identifier



Looks like this using relational schema notation



Transforming E-R diagrams into relations

Mapping unary relationships

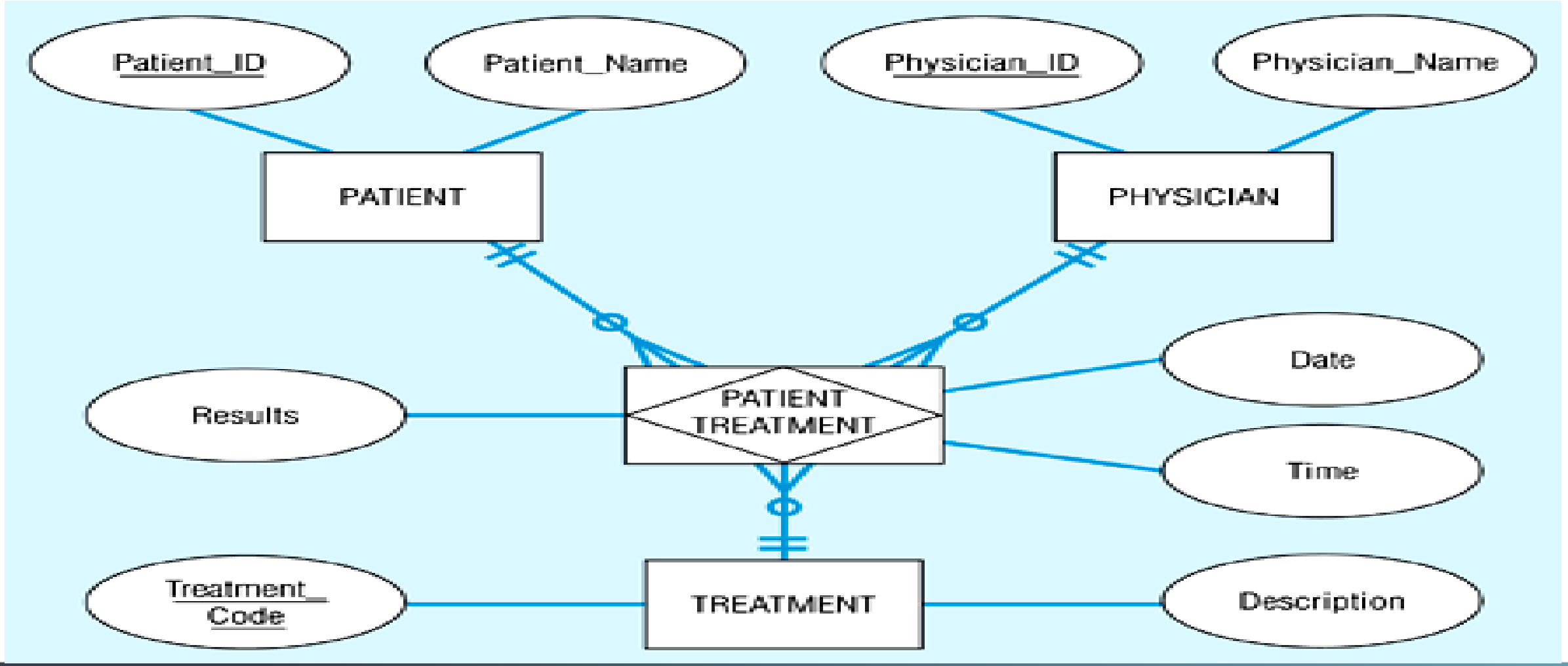
- One-to-many - recursive foreign key in the same relation
- Many-to-many - two relations:
  - One for the entity type
  - One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

Transforming E-R diagrams into relations

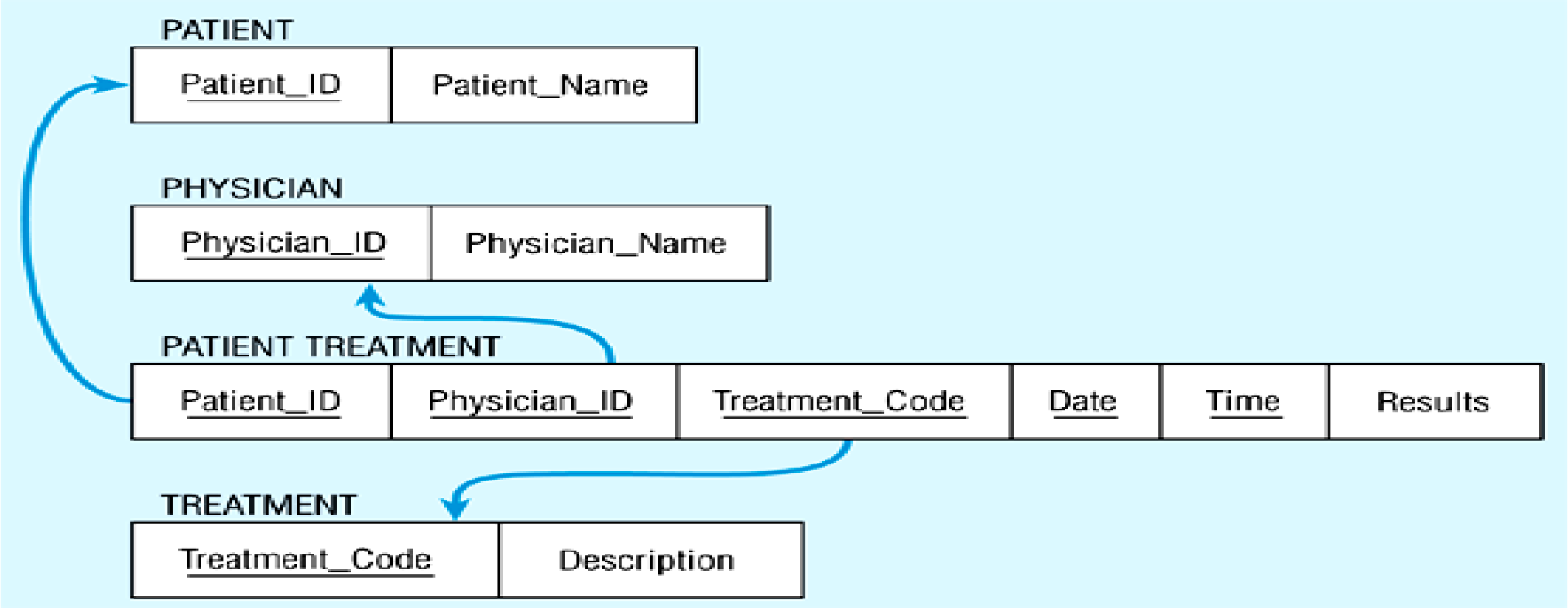
Mapping ternary (and n-ary) relationships

- **One relation for each entity and one for the associative entity**

Mapping a ternary relationship



Looks like this using relational schema notation

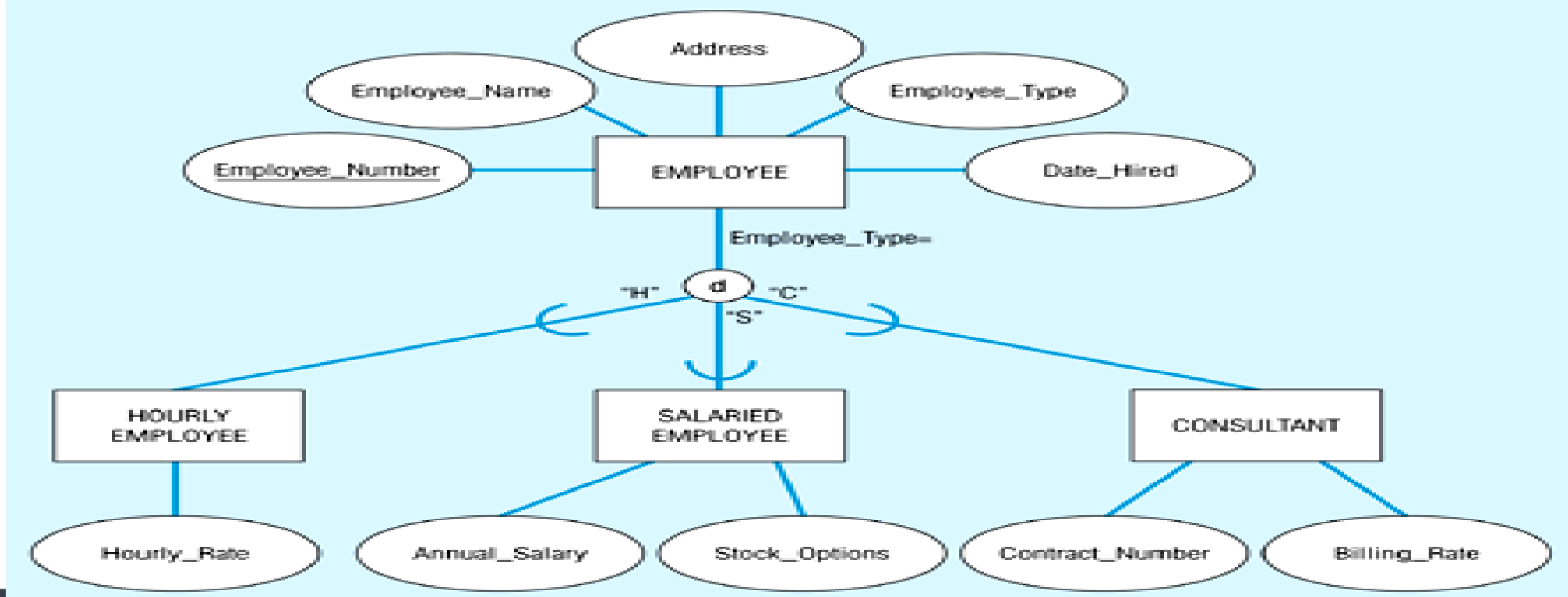


Transforming E-R diagrams into relations

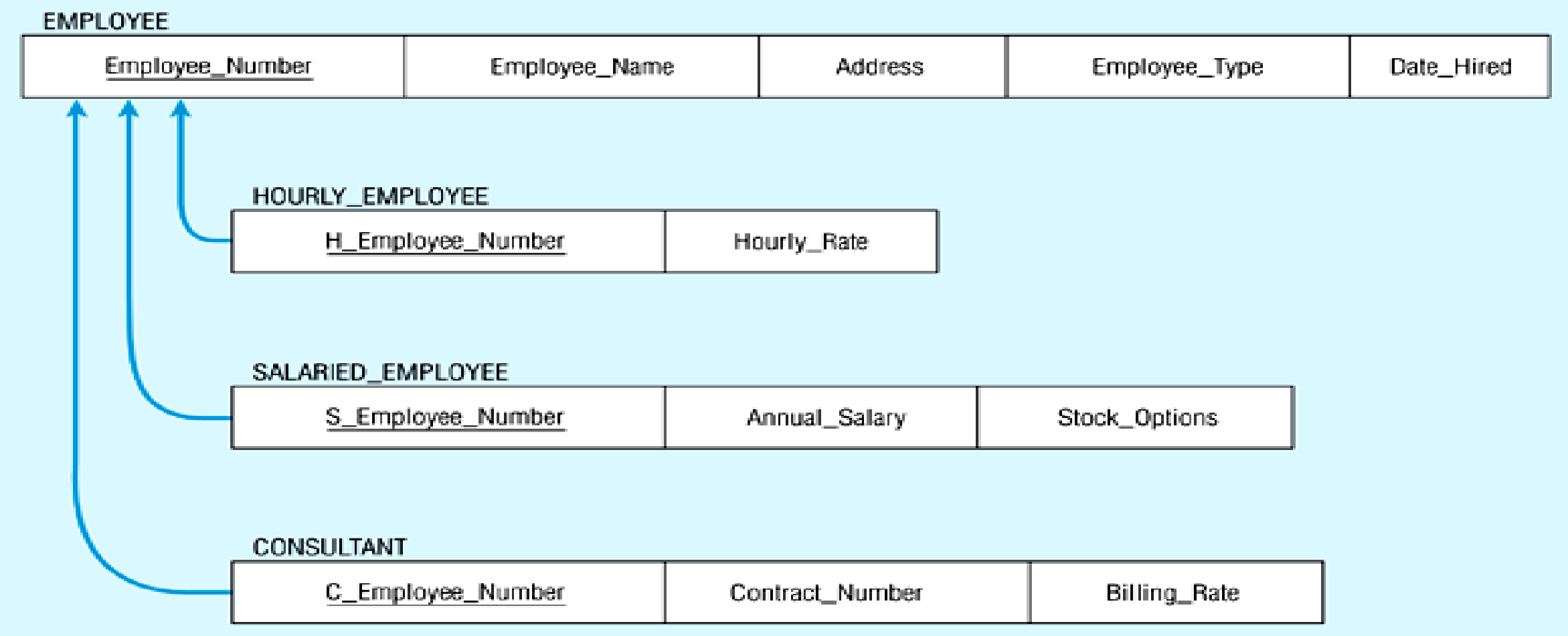
Mapping Supertype/subtype relationships

- Create a separate relation for the supertype and each of the subtypes
- Assign common attributes to supertype
- Assign primary key and unique attributes to each subtype
- Assign an attribute of the supertype to act as subtype discriminator

Mapping Supertype/subtype relationships



Would look like this...

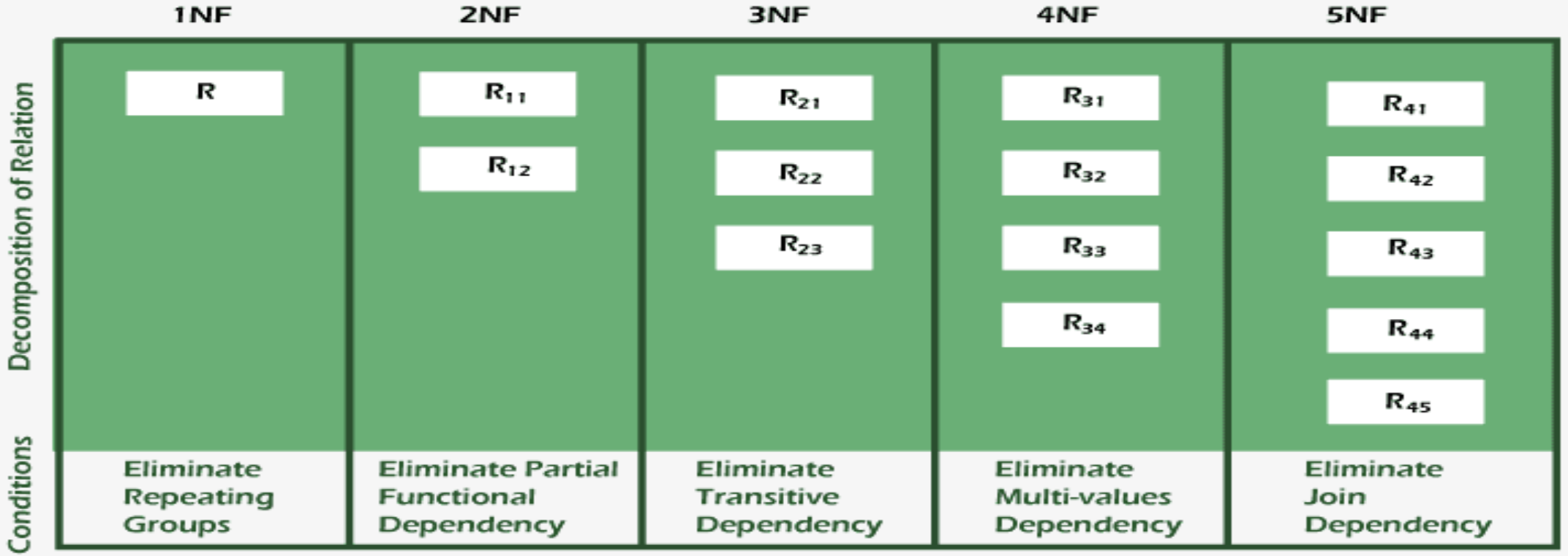


Well-structured relations

- Well-structured relations contain minimal redundancy and allow insertion, modification, and deletion without errors or inconsistencies
- Anomalies are errors or inconsistencies resulting from redundancy
  - Insertion anomaly
  - Deletion anomaly
  - Modification anomaly

Data normalization

- Normalization is a formal process for deciding which attributes should be grouped together in a relation
  - Objective: to validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data
  - Definition: the process of decomposing relations with anomalies to produce smaller, well-structured relations



Functional dependencies and keys

- Functional dependency: the value of one attribute (the *determinant*) determines the value of another attribute
  - A -> B, for every valid instance of A, that value of A uniquely determines the value of B
- Candidate key: an attribute or combination of attributes that uniquely identifies an instance
  - Uniqueness: each non-key field is functionally dependent on every candidate key
  - Non-redundancy

Let's consider a database normalization example, where a business needs to keep track of its employees' names and contact information. In the database, it makes a table that appears as follows:

Employee Code	Employee Name	Employee Phone Number
101	**Jolly**	98765623,998234123
101	**Jolly**	89023467
102	**Rehan**	76213908
103	**Snehi**	98132452

First normal form

- No multi-valued attributes.
- Every attribute value is atomic.

- Employee Phone Number is a multivalued characteristic in this case. This relationship is, therefore, not 1NF.
- As demonstrated below, we create new rows for each employee's phone number in order to transform this table into 1NF:

<EmployeeDetail>

Employee Code	Employee Name	Employee Phone Number
101	Jolly	998234123
101	Jolly	98765623
101	Jolly	89023467
102	Rehan	76213908
103	Snehi	98132452

Second normal form

- 1NF and every non-key attribute is fully functionally dependent on the primary key.
- Every non-key attribute must be defined by the entire key, not by only part of the key.
- No partial functional dependencies.

To better understand partial dependency and how to normalize the table to the second normal form, let's use the following "EmployeeProjectDetail" table as a database normalization example with the solution:

<EmployeeProjectDetail>

Employee Code	Project ID	Employee Name	Project Name
101	P03	Jolly	Project103
101	P01	Jolly	Project101
102	P04	Rehan	Project104
103	P02	Snehi	Project102

- Employee Code and Project ID are the table's primary attributes in the aforementioned table. Because Employee Code may identify Employee Name and Project Name can be determined by Project ID, we have partial dependencies in this table. As a result, the relational table mentioned above breaks the 2NF constraint.

- The key characteristics that make up one or more candidate keys are the prime characteristics.

- The EmployeeProjectDetail table can be divided into the following three tables to eliminate partial dependencies and normalize it into the second normal form:

Employee Code	Employee Name
101	Jolly
101	Jolly
102	Rehan
103	Snehi

<ProjectDetail>

Project ID	Project Name
P03	Project103
P01	Project101
P04	Project104
P02	Project102

<EmployeeProject>

Employee Code	Project ID
101	P03
101	P01
102	P04
103	P02

Third normal form

- 2NF and no transitive dependencies (functional dependency between non-key attributes.)

- Applying the third normal form to regularly changing data may be more practical. If any dependent fields still exist, build your application to ask the user to confirm changes to all linked fields.

Employee Code	Employee Name	Employee Zipcode	Employee City
101	Jolly	110033	Model Town
101	Jolly	110044	Badarpur
102	Rehan	110028	Naraina
103	Snehi	110064	Hari Nagar

•Employee Code -> Employee City transitive dependency prevents the table mentioned above from being in 3NF because:
•Customer Code -> Customer Zipcode
•Employee City -> Employee Zipcode

- Additionally, neither Employee City nor Employee Zipcode is the primary attribute.
- We can divide the "EmployeeDetail" table into the following two tables to eliminate transitive dependency from it and normalize it into the third normal form:

Employee Code	Employee Name	Employee Zipcode
101	Jolly	110033
101	Jolly	110044
102	Rehan	110028
103	Snehi	110064

<EmployeeLocation>

Employee Zipcode	Employee City
110033	Model Town
110044	Badarpur
110028	Naraina
110064	Hari Nagar

- By breaking the "EmployeeDetail" data down into the "EmployeeDetail" and "EmployeeLocation" tables, which are both in 2NF and do not have any transitive dependencies, we were able to transform the "EmployeeDetail" table into 3NF.

- The 2NF and 3NF remove any redundant dependence on candidate keys and set certain additional restrictions on them. There may, however, still be certain dependencies that result in database redundancy. A stricter normal form called BCNF eliminates these redundant elements.

4. BCNF (Boyce-Codd Normal Form)

- Boyce-Codd As it has more limitations than 3NF, Normal Form is an improved form of 3NF.
- A relational table must conform to the following conditions to be in Boyce-Codd normal form:
- The third normal form of the table is required.
- X is the table's superkey for every non-trivial functional dependence X -> Y. As a result, if Y is a prime attribute, X cannot be a non-prime attribute.
- A group of one or more attributes known as a superkey can be used to identify a row in a database table specifically.

- To better understand how to normalize the table to the BCNF, let's use the following "EmployeeProjectLead" table as an example:

Employee Code	Project ID	Project Leader
101	P03	Grey
101	P01	Christian
102	P04	Hudson
103	P02	Petro

- The preceding table satisfies all conventional forms up to 3NF. However, since its candidate key is "Employee Code, Project ID," it defies BCNF's criteria. Project Leader is a non-prime property for the non-trivial functional dependence Project Leader -> Project ID, whereas Project ID is a prime attribute. In BCNF, this is not permitted.

We divide the given table into three tables and then translate them into BCNF.

<EmployeeProject>

Employee Code	Project ID
101	P03
101	P01
102	P04
103	P02

<ProjectLead>

Project Leader	Project ID
Grape	P03
Christiano	P01
Harry	P04
Petric	P02

4NF (Fourth Normal Form)

- It is in fourth normal form if no database table instance has two or more independent, multivalued pieces of data characterizing the relevant object.
- If a relationship has no multivalued dependencies and is in Boyce Codd's normal form, it is said to be in 4NF.
- The relationship is said to be a multivalued dependency if there are multiple values of B for a given value of A in a dependency A B.
- You saw types of normalization in DBMS and how to perform normalization in DBMS with examples. It should give a good idea of going about it and making a better database model.

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

What is the Purpose of DBMS Normalization?

- As data's usefulness to various types of businesses rises, the purpose of normalization in DBMS, the manner that data is organized when it is present in huge quantities, becomes even more critical. It is evident that good Data normalization of the database is used to get better results like:
- Overall business performance increases.
- Improving group analysis without being concerned with redundancy.
- Imagine the consequences if you failed to arrange your data and missed out on crucial growth opportunities because a website wouldn't load or a vice president didn't get your notes.
- A business can gather all the information it needs from any source. However, without data normalization, most of it would just be wasted and not be of any real use of normalization in DBMS to the organization.

Types of Anomalies in DBMS

1. Insertion Anomaly

- Imagine that we have a table with four columns. Student identification number, name, address, and grades. Now, even if the first three properties can be filled when a new student enrolls in school, the fourth attribute will have a NULL value because he doesn't yet have any grades.

2. Deletion Anomaly

- This oddity suggests that crucial data was unnecessarily removed from the table. Suppose we have the following student data and a list of the courses they have taken: (student ID, Student Name, Course, address). Any entry pertaining to a student who leaves the school will be removed. Nevertheless, even if the course depends on the school and not the student, that deletion will also erase the course information.
- The goal of normalization is to make the tables as granular as possible to avoid these problems. To put it simply, it attempts to divide tables into several tables and establish associations between them using keys.

3. Updation Anomaly

- Consider a table with 10 columns, of which 2 are designated as employee Name and employee Address. Now, we need to update the table whenever an employee changes location. However, if the table is not normalized, one employee may have many entries, and if all of those entries are updated at once, one of them may be overlooked.