BỘ MÔN HỆ THỐNG THÔNG TIN
KHOA CÔNG NGHỆ THỐNG TIN – ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP HCM

CƠ SỞ DỮ LIỆU NÂNG CAO

Chương 08:
THIẾT KẾ CƠ SỞ DỮ LIỆU MỨC LOGIC (LOGICAL DATABASE DESIGN)

Giảng viên: TS. Nguyễn Trần Minh Thư

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Objectives

- Describe logical database design
- Transform an E-R (or EER) diagram into a logically equivalent set of relations.
- Use normalization to decompose a relation with anomalies into well-structured relations.
- Briefly describe four problems that may arise when merging relations.
- Create relational tables that incorporate entity integrity and referential integrity constraints.
- Validate the logical model against user requirements

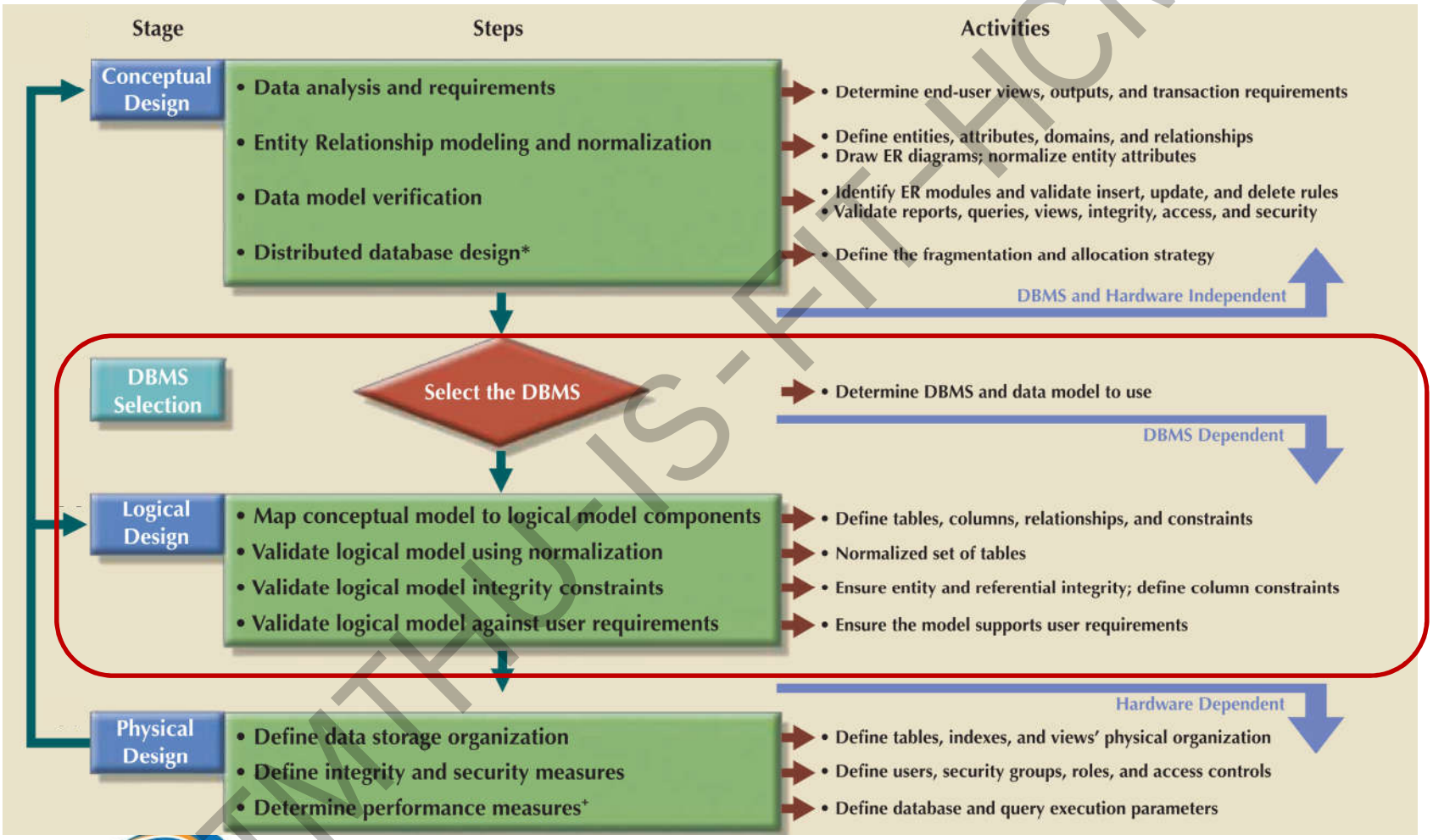
Outline

- Logical Database Design
- Overview: Relational Data Model
- Logical Design Steps
  - ER-/EER-to-Relational Mapping
  - Validate the logical model using normalization
  - Validate the logical model integrity constraints
  - Validate the logical model against user requirements

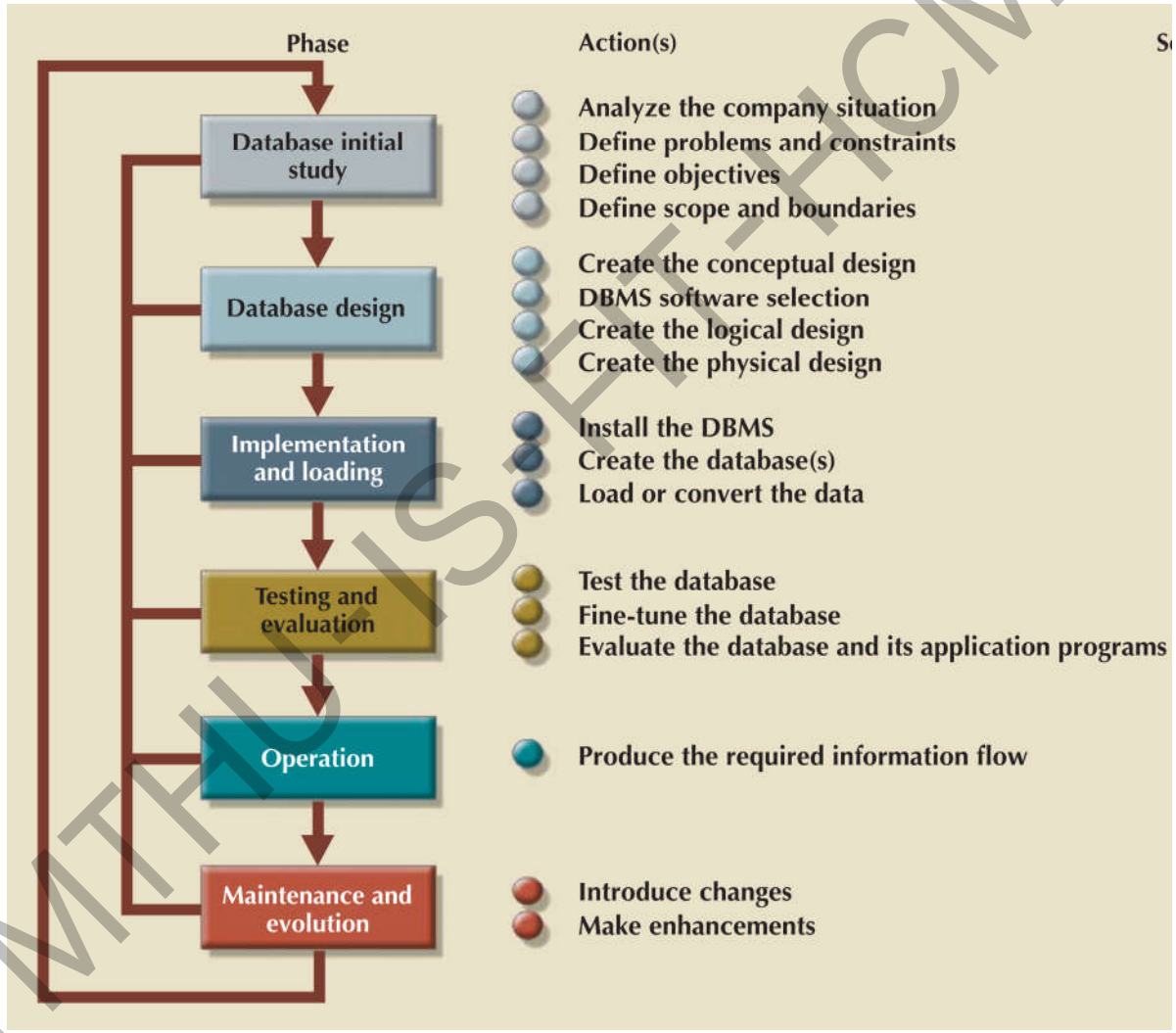
Outline

- Logical Database Design
- Overview: Relational Data Model
- Logical Design Steps
  - ER-/EER-to-Relational Mapping
  - Validate the logical model using normalization
  - Validate the logical model integrity constraints
  - Validate the logical model against user requirements

Database Design Process



The Database Life Cycle (DBLC)



Database Design Process

- A Conceptual model represents reality in an abstracted form that can be used in developing an information system in a wide variety of formats (e.g. relational, object-oriented, flat-file, etc.)
  - It is hardware and software independent
  - It is independent of any logical model type
- A Logical model represents reality in the format required by a particular database model (e.g. relational or object-oriented)
  - Is still hardware and software independent
  - Depends on the chosen logical model type
- A Physical model is created specifically for a particular database software package
  - Is dependent on hardware, software, and on the chosen logical model type

Objectives of Logical Design

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

Logical Design

- **Logical design**:

Designs an enterprise-wide database that is based on a specific data model but independent of physical-level details

- **Validates logical model**:
  - Using normalization
  - Integrity constraints
  - Against user requirements

Outline

- Logical Database Design
- Overview: Relational Data Model
- Logical Design Steps
  - ER-/EER-to-Relational Mapping
  - Validate the logical model using normalization
  - Validate the logical model integrity constraints
  - Validate the logical model against user requirements

Components of relational model

- Data structure
  - Tables (relations), rows, columns
- Data manipulation
  - Powerful SQL operations for retrieving and modifying data
- Data integrity
  - Mechanisms for implementing business rules that maintain integrity of manipulated data

Relation

- A relation is a named, two-dimensional table of data.
- A table consists of rows (records) and columns (attribute or field).
- Requirements for a table to qualify as a relation:
  - It must have a unique name.
  - Every attribute value must be atomic (not multivalued, not composite).
  - Every row must be unique (can't have two rows with exactly the same values for all their fields).
  - Attributes (columns) in tables must have unique names.
  - The order of the columns must be irrelevant.
  - The order of the rows must be irrelevant.

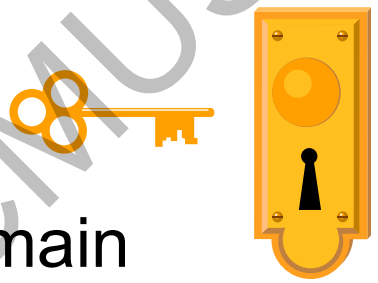
NOTE: All *relations* are in ***1st Normal form.***

Correspondence with E-R Model

- **Relations** (tables) correspond with **entity types** and with **many-to-many relationship types**.
- **Rows** correspond with **entity instances** and with **many-to-many relationship instances**.
- **Columns** correspond with **attributes**.

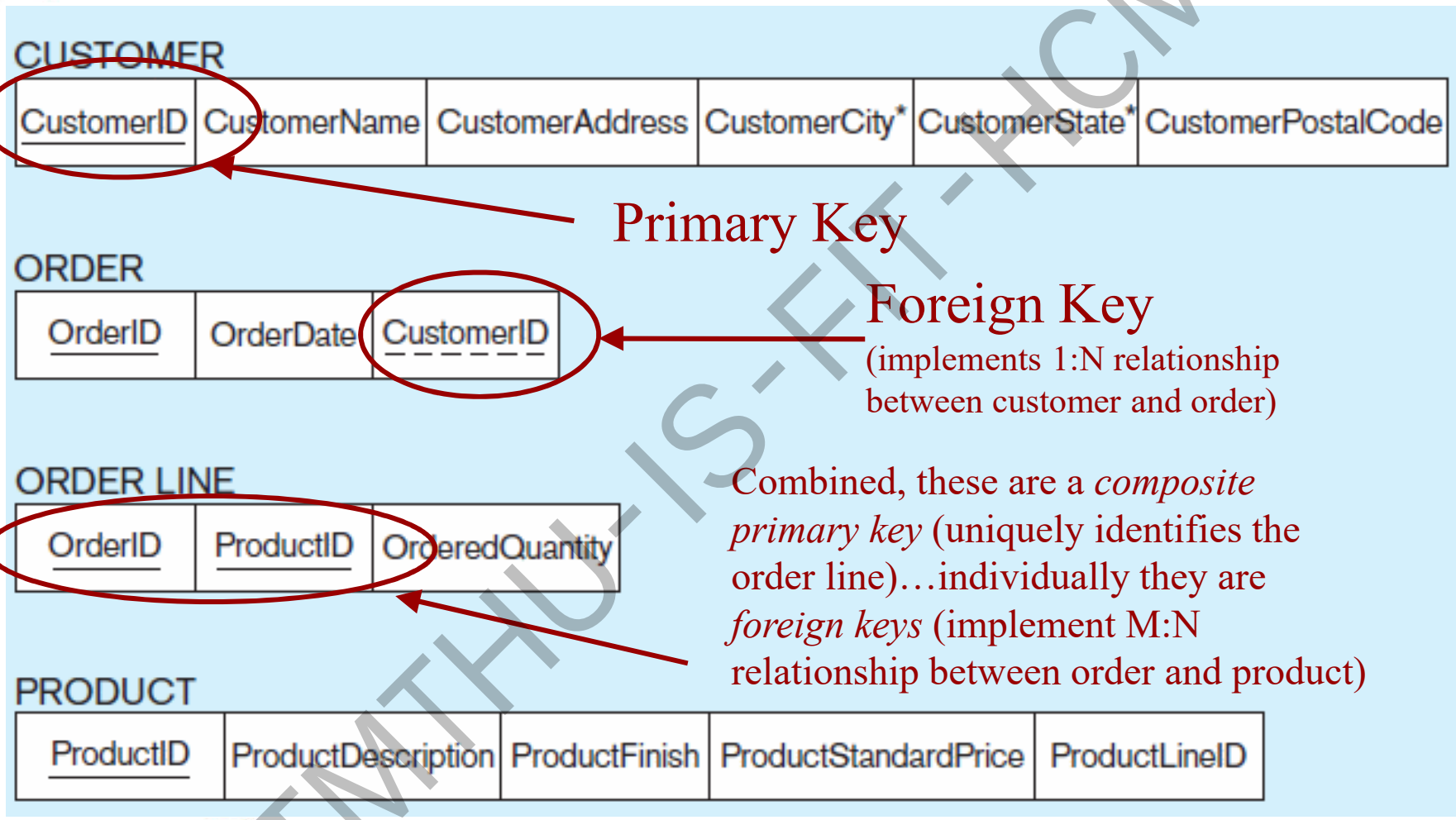
- **NOTE**: The word relation (in relational database) is NOT the same as the word relationship (in E-R model).

Key Fields



- Keys are special fields that serve two main purposes:
  - ***Primary keys*** are unique identifiers of the relation. Examples include employee numbers, social security numbers, etc. *This guarantees that all rows are unique.*
  - ***Foreign keys*** are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship).
- Keys can be **simple** (a single field) or **composite** (more than one field).
- Keys usually are used as indexes to speed up the response to user queries (more on this in Chapter 09).

Schema for four relations (Pine Valley Furniture Company)



Integrity Constraints

- Domain Constraints
  - + Allowable values for an attribute
- Entity Integrity
  - +No primary key attribute may be null. All primary key fields **MUST** have data.
- Action Assertions
  - +Business rules (Recall from Chapter 2)

Domain definitions enforce domain integrity constraints.

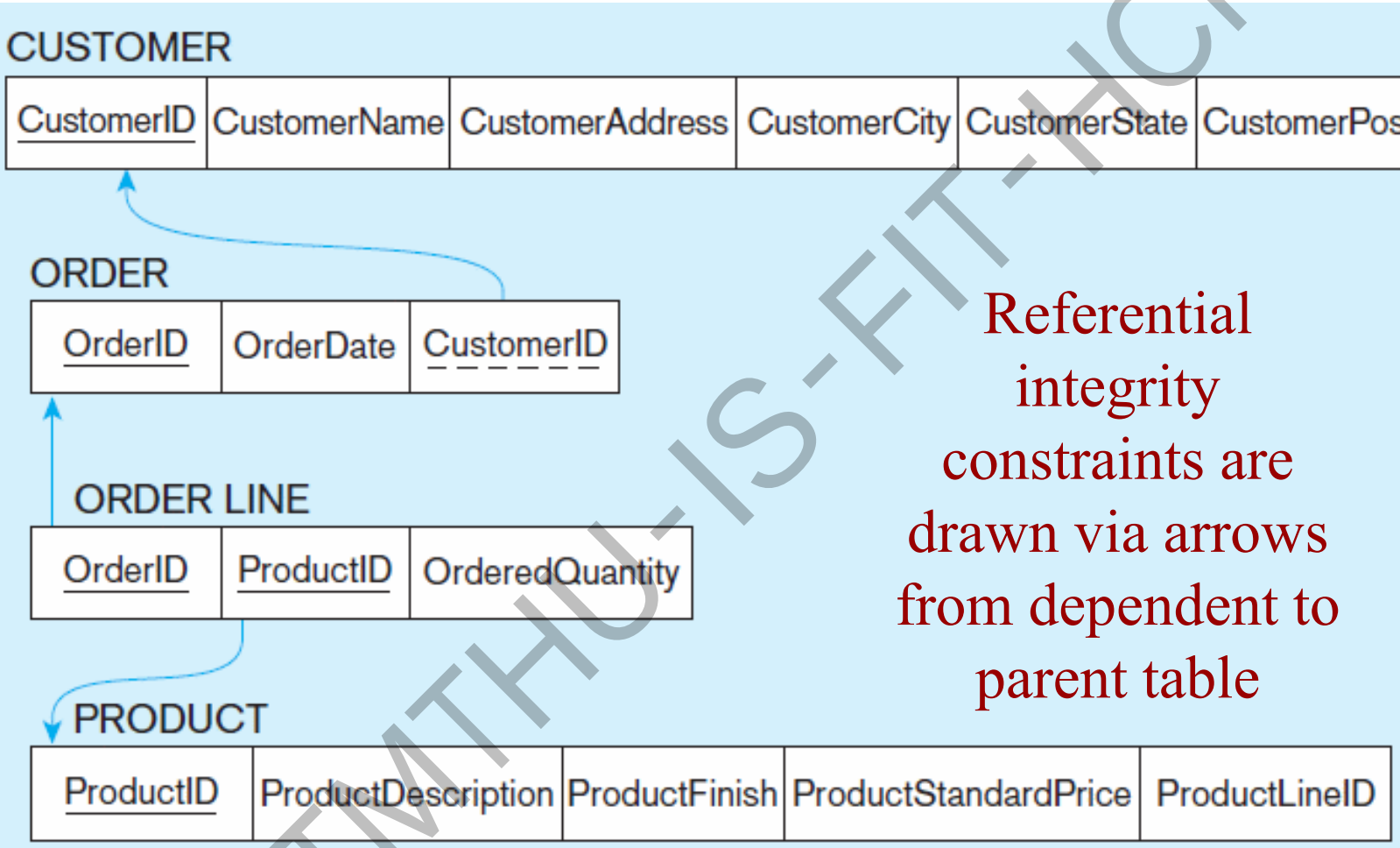
TABLE 4-1 Domain Definitions for INVOICE Attributes

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25
CustomerAddress	Customer Addresses	Set of all possible customer addresses	character: size 30
CustomerCity	Cities	Set of all possible cities	character: size 20
CustomerState	States	Set of all possible states	character: size 2
CustomerPostalCode	Postal Codes	Set of all possible postal zip codes	character: size 10
OrderID	Order IDs	Set of all possible order IDs	character: size 5
OrderDate	Order Dates	Set of all possible order dates	date: format mm/dd/yy
ProductID	Product IDs	Set of all possible product IDs	character: size 5
ProductDescription	Product Descriptions	Set of all possible product descriptions	character: size 25
ProductFinish	Product Finishes	Set of all possible product finishes	character: size 15
ProductStandardPrice	Unit Prices	Set of all possible unit prices	monetary: 6 digits
ProductLineID	Product Line IDs	Set of all possible product line IDs	integer: 3 digits
OrderedQuantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

Integrity Constraints

- **Referential Integrity**–rule states that any foreign key value (on the relation of the many side) MUST match a primary key value in the relation of the one side. (Or the foreign key can be null)
  - For example: **Delete Rules**
    - **Restrict**–don't allow delete of "parent" side if related rows exist in "dependent" side
    - **Cascade**–automatically delete "dependent" side rows that correspond with the "parent" side row to be deleted
    - **Set-to-Null**–set the foreign key in the dependent side to null if deleting from the parent side → not allowed for weak entities

Referential integrity constraints (Pine Valley Furniture)



Referential integrity constraints are drawn via arrows from dependent to parent table

SQL table definitions

```
CREATE TABLE Customer_T
        (CustomerID                 NUMBER(11,0)     NOT NULL,
        CustomerName                VARCHAR2(25)     NOT NULL,
        CustomerAddress             VARCHAR2(30),
        CustomerCity                VARCHAR2(20),
        CustomerState               CHAR(2),
        CustomerPostalCode          VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));

CREATE TABLE Order_T
        (OrderID                    NUMBER(11,0)     NOT NULL,
        OrderDate                   DATE DEFAULT SYSDATE,
        CustomerID                  NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T (CustomerID));

CREATE TABLE Product_T
        (ProductID                  NUMBER(11,0)     NOT NULL,
        ProductDescription          VARCHAR2(50),
        ProductFinish               VARCHAR2(20),
        ProductStandardPrice        DECIMAL(6,2),
        ProductLineID               NUMBER(11,0),
CONSTRAINT Product_PK PRIMARY KEY (ProductID));

CREATE TABLE OrderLine_T
        (OrderID                    NUMBER(11,0)     NOT NULL,
        ProductID                   NUMBER(11,0)     NOT NULL,
        OrderedQuantity             NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T (OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T (ProductID));
```

Referential integrity constraints are implemented with foreign key to primary key references.

Outline

- Logical Database Design
- Overview: Relational Data Model
- Logical Design Steps
  - ER-/EER-to-Relational Mapping
  - Validate the logical model using normalization
  - Validate the logical model integrity constraints
  - Validate the logical model against user requirements

Logical Design Steps

Step 1
- Map the conceptual Model to logical model components

Step 2
- Validate the logical model using normalization

Step 3
- Validate the logical model integrity constraints

Step 4
- Validate the logical model against user requirements

Logical Design Steps

Step 1
- **Map the conceptual Model to logical model components**

Step 2
- **Validate the logical model using normalization**

Step 3
- **Validate the logical model integrity constraints**

Step 4
- **Validate the logical model against user requirements**

Transforming EER Diagrams into Relations

* **A conceptual model MUST NOT include FK information** *

- An entity turns into a table.
- Each attribute turns into a column in the table.
- The (unique) identifier of the entity turns into a PK of the table.

Transforming EER Diagrams into Relations

SOME RULES.

* Remember! The Relational DB Model does not like any type of redundancy.

- Every table must have a unique name.
- Attributes in tables must have unique names.
- Every attribute value is atomic.
- The order of the columns is irrelevant.
- The order of the rows is irrelevant.

Transforming EER Diagrams into Relations

Mapping Regular Entities to Relations

- **Simple attributes**: E-R attributes map directly onto the relation
- **Composite attributes**: Use only their simple, component attributes
- **Multivalued Attribute**: Becomes a separate relation with a foreign key taken from the superior entity

Mapping a regular entity

(a) CUSTOMER entity type with simple attributes

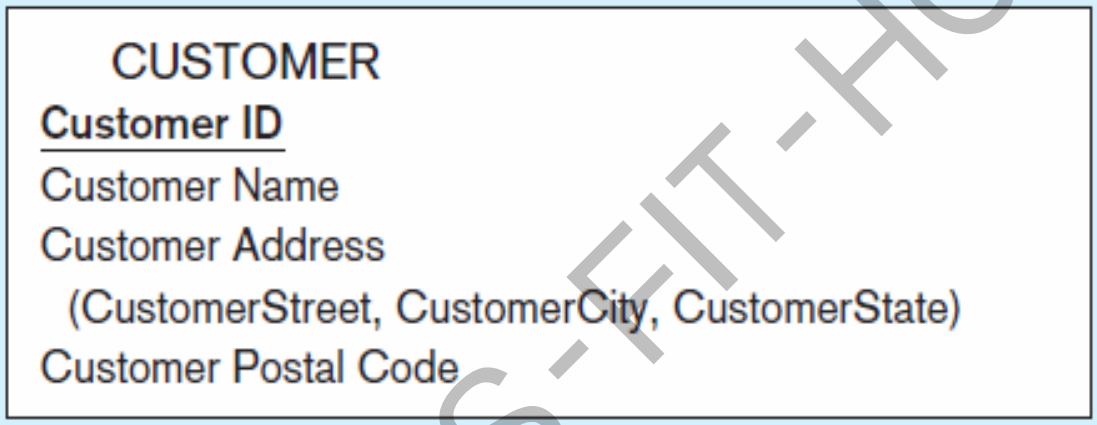
(b) CUSTOMER relation

CUSTOMER

CustomerID	CustomerName	CustomerAddress	CustomerPostalCode

Mapping a composite attribute

(a) CUSTOMER entity type with composite attribute



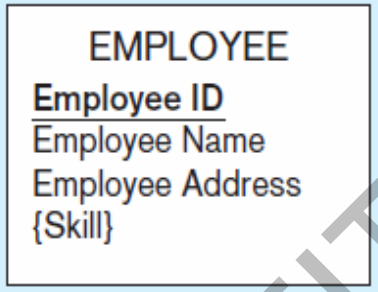
(b) CUSTOMER relation with address detail

CUSTOMER

CustomerID	CustomerName	CustomerStreet	CustomerCity	CustomerState	CustomerPostalCode

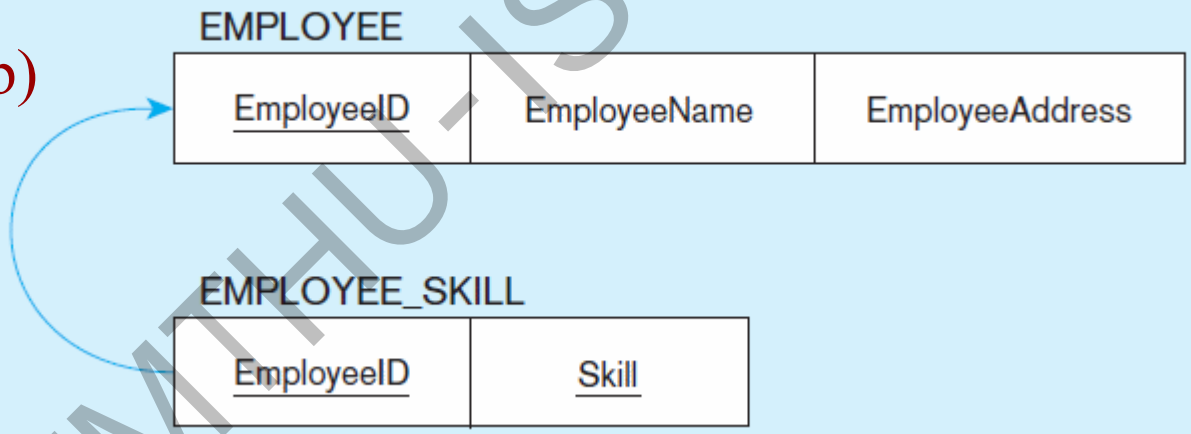
Mapping an entity with a multivalued attribute

(a)



Multivalued attribute becomes a separate relation with foreign key

(b)



One–to–many relationship between original entity and new relation

Transforming EER Diagrams into Relations (cont.)

Mapping Weak Entities

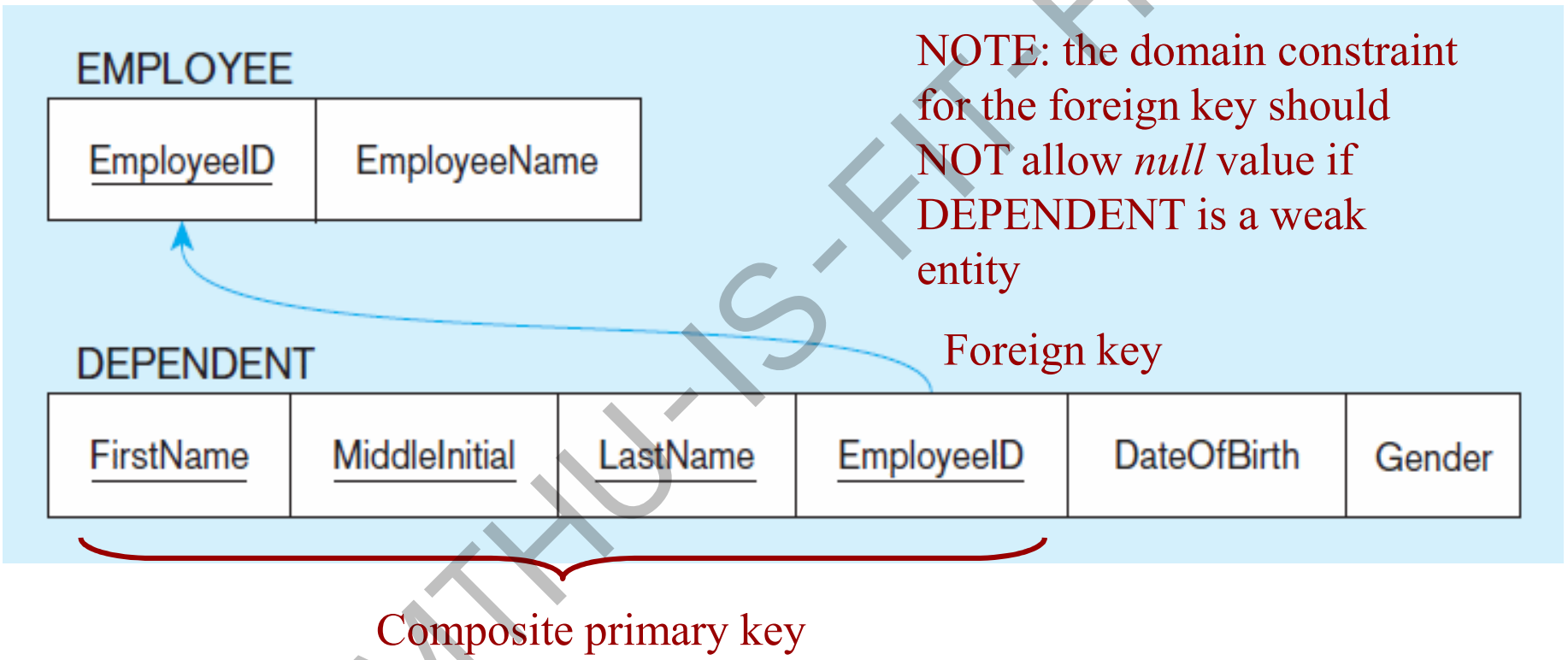
- Becomes a separate relation with a foreign key taken from the superior entity
- Primary key composed of:
  - Partial identifier of weak entity
  - Primary key of identifying relation (strong entity)

Example of mapping a weak entity

a) Weak entity DEPENDENT

Example of mapping a weak entity (cont.)

b) Relations resulting from weak entity



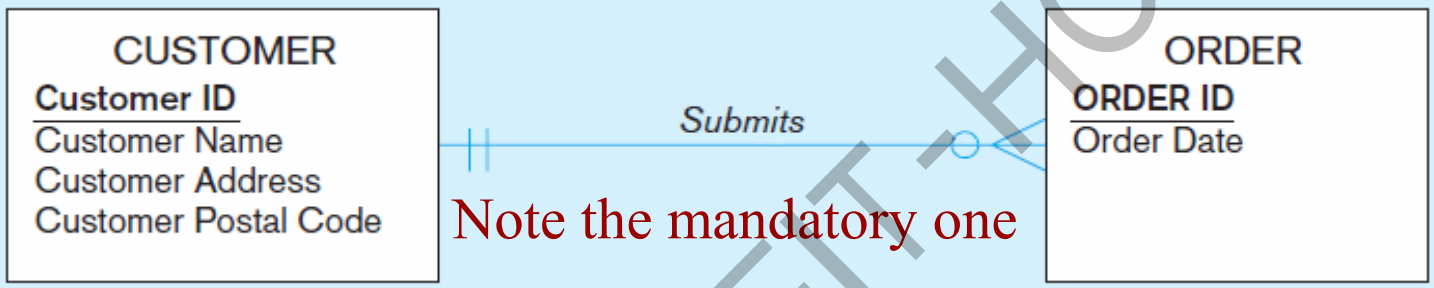
Transforming EER Diagrams into Relations (cont.)

Mapping Binary Relationships

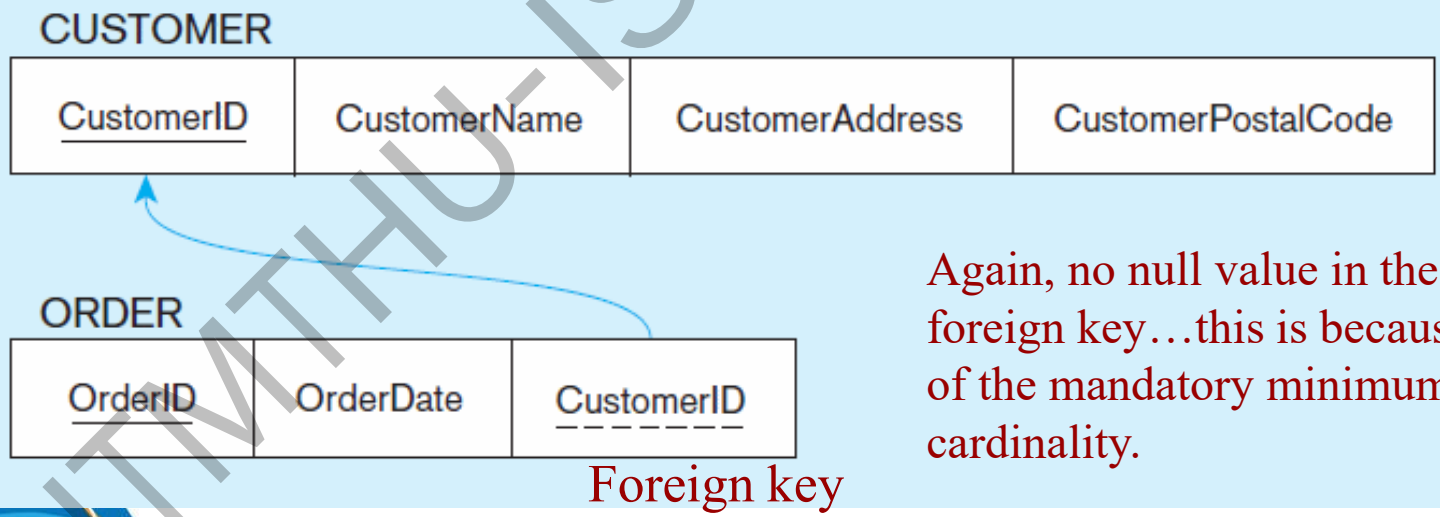
- **One-to-Many**–Primary key on the one side becomes a foreign key on the many side
- **Many-to-Many**–Create a new relation with the primary keys of the two entities as its primary key
- **One-to-One**–Primary key on mandatory side becomes a foreign key on optional side

Example of mapping a 1:M relationship

a) Relationship between customers and orders

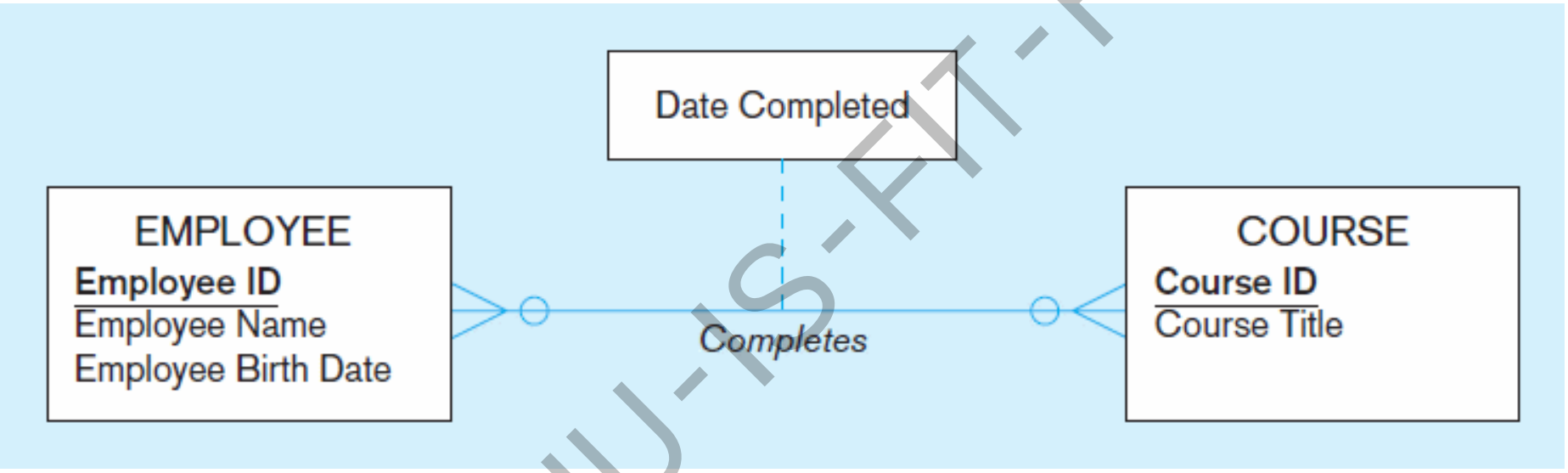


b) Mapping the relationship



Example of mapping an M:N relationship

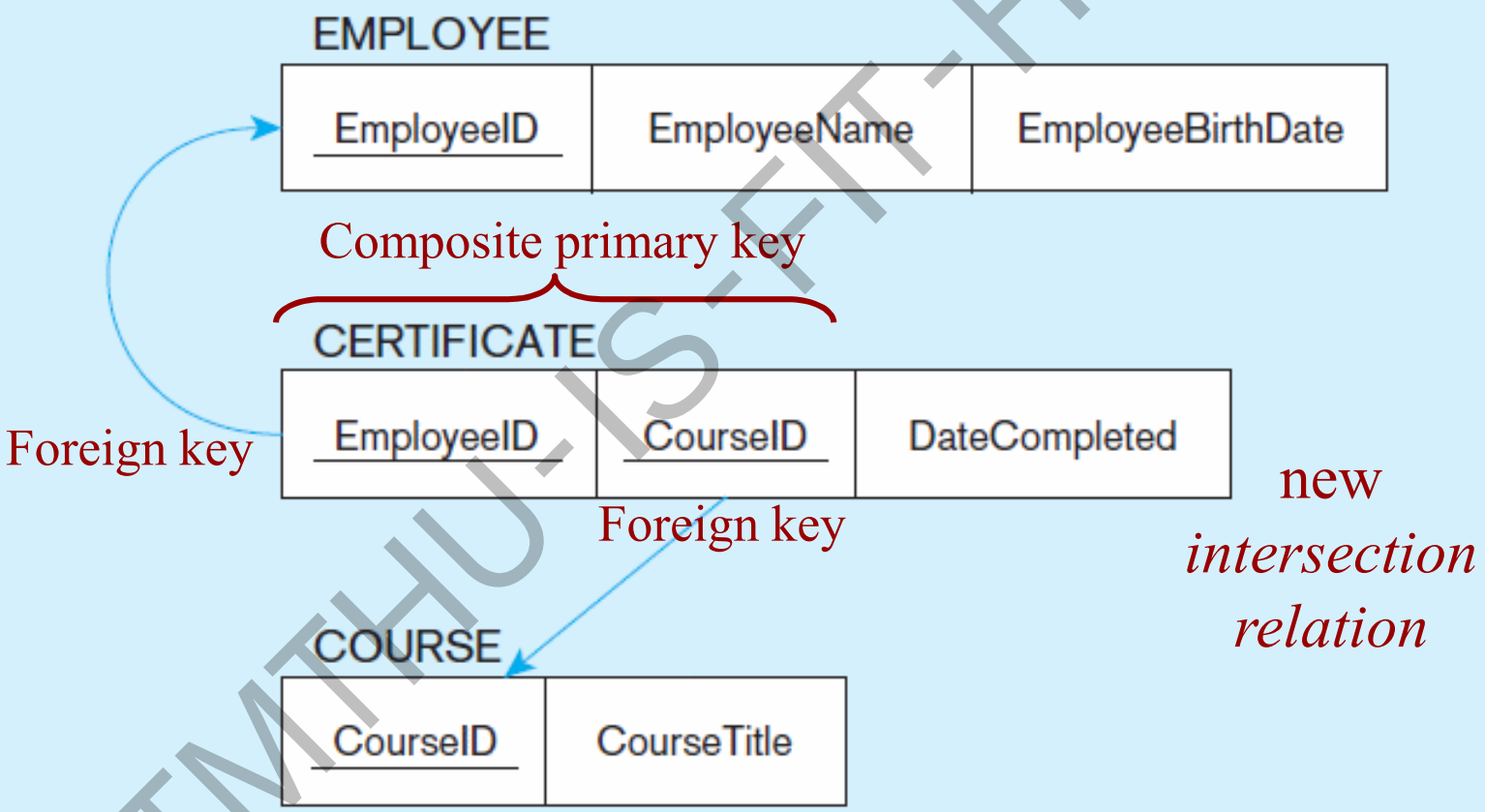
a) Completes relationship (M:N)



The *Completes* relationship will need to become a separate relation.

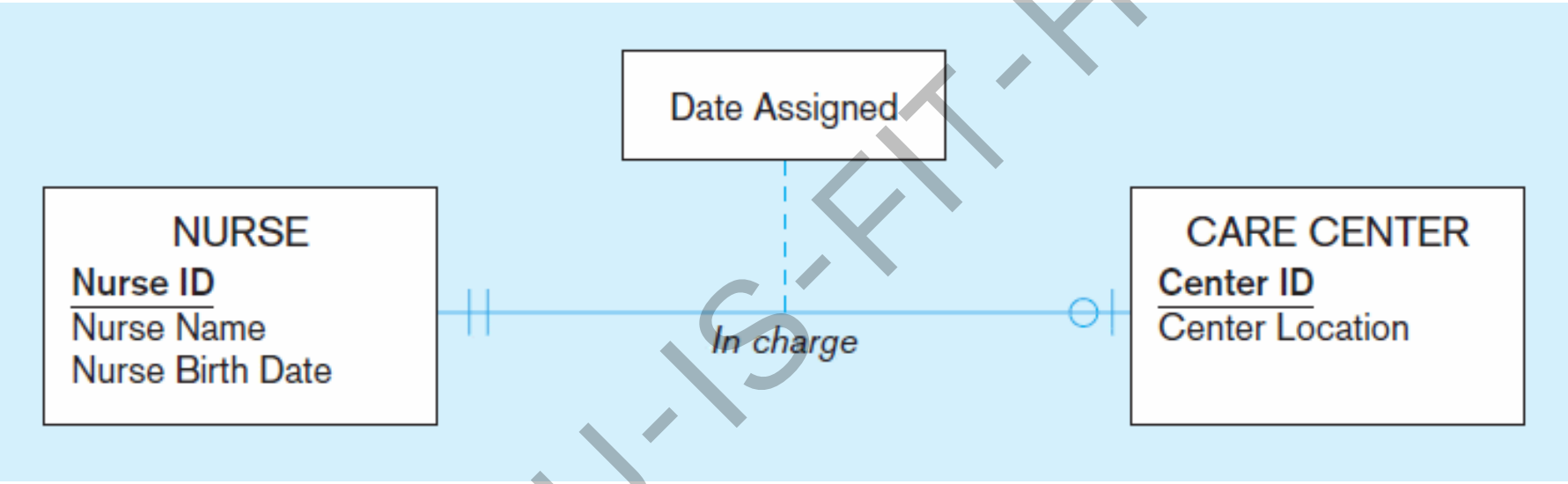
Example of mapping an M:N relationship (cont.)

b) Three resulting relations



Example of mapping a binary 1:1 relationship

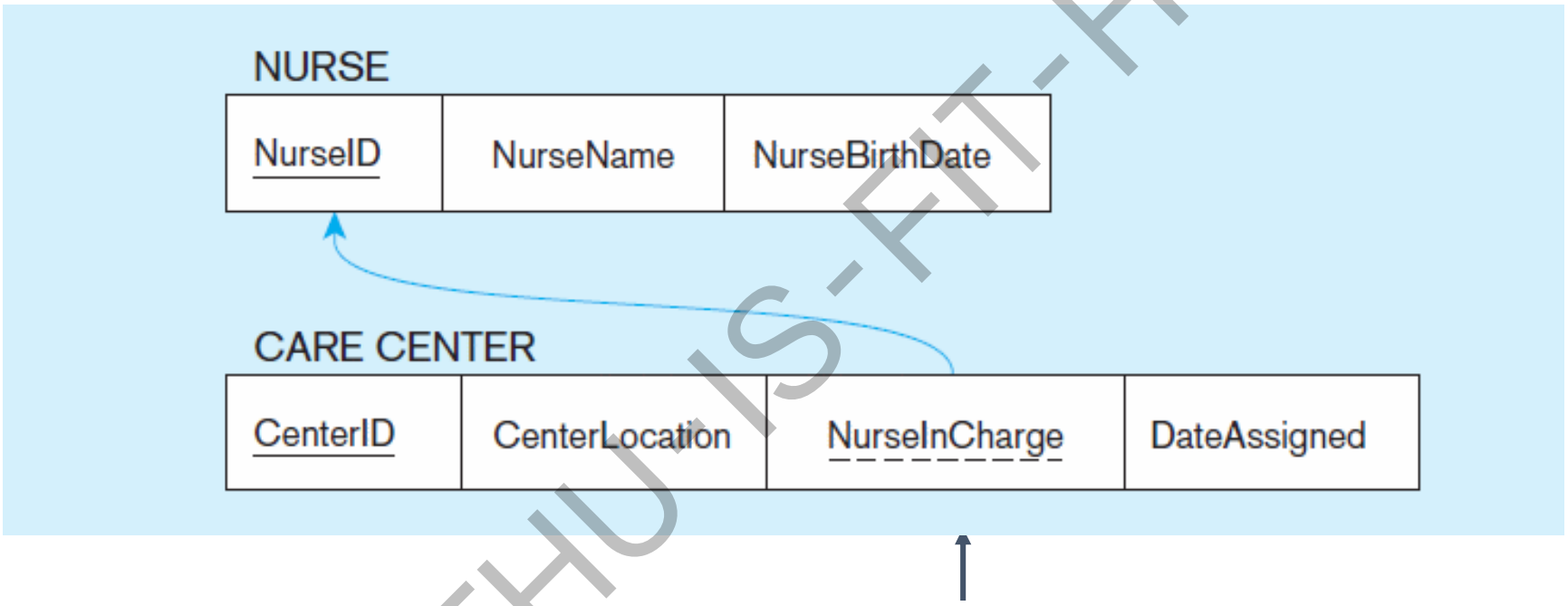
a) In charge relationship (1:1)



Often in 1:1 relationships, one direction is optional

Example of mapping a binary 1:1 relationship (cont.)

b) Resulting relations



Foreign key goes in the relation on the optional side, matching the primary key on the mandatory side

Transforming EER Diagrams into Relations (cont.)

Mapping Associative Entities

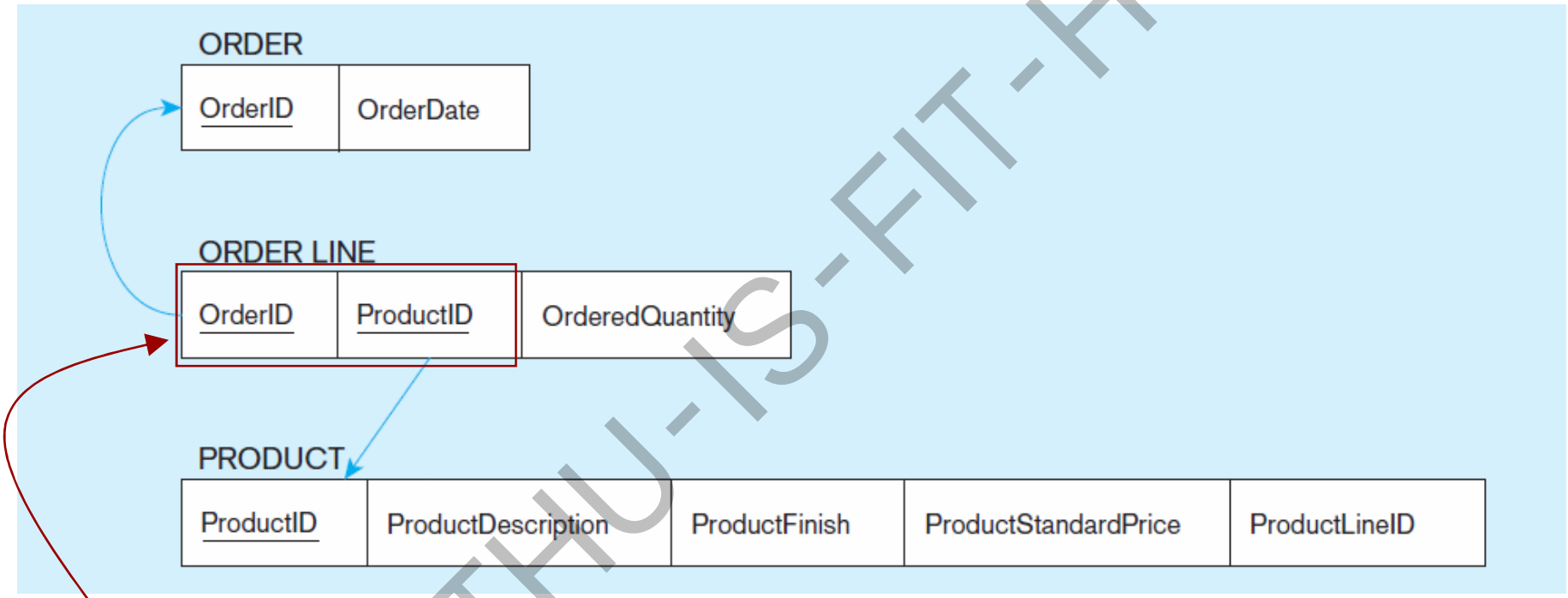
- Identifier Not Assigned
  - Default primary key for the association relation is composed of the primary keys of the two entities (as in M:N relationship)
- Identifier Assigned
  - It is natural and familiar to end-users
  - Default identifier may not be unique

Example of mapping an associative entity

a) An associative entity

Example of mapping an associative entity (cont.)

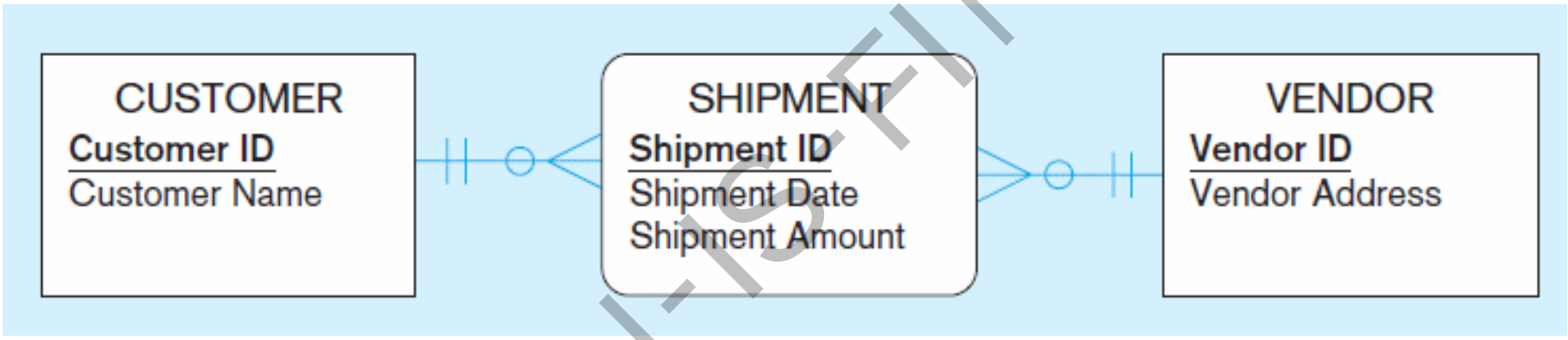
b) Three resulting relations



Composite primary key formed from the two foreign keys

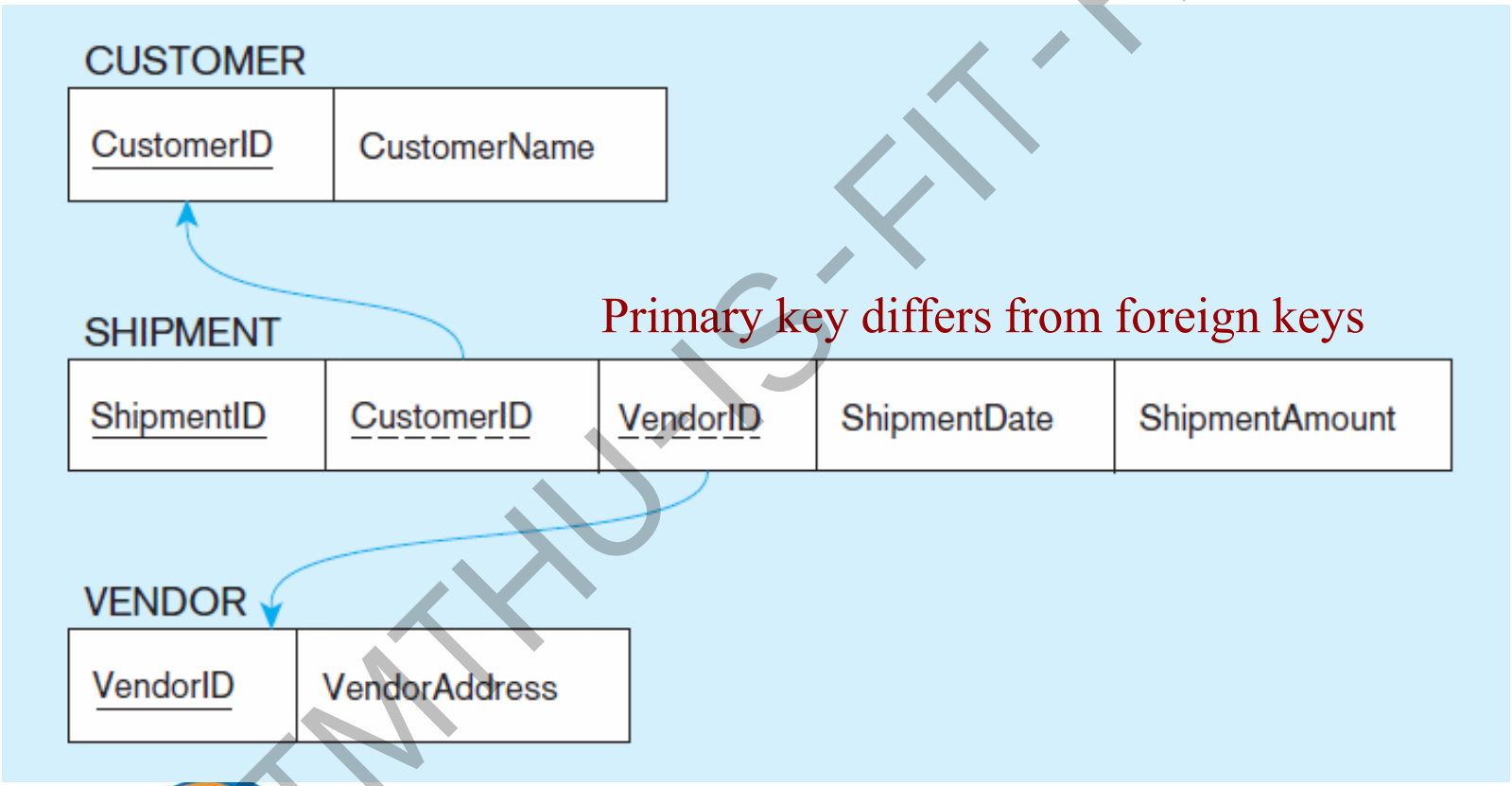
Example of mapping an associative entity with an identifier

a) SHIPMENT associative entity



Example of mapping an associative entity with an identifier (cont.)

b) Three resulting relations

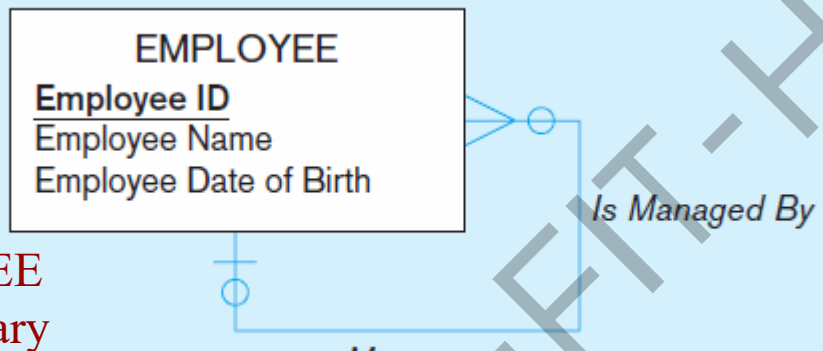


Transforming EER Diagrams into Relations (cont.)

Mapping Unary Relationships

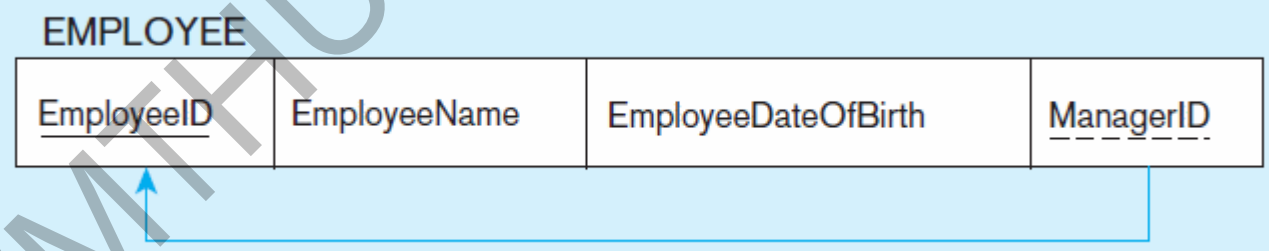
- One-to-Many–Recursive foreign key in the same relation
- Many-to-Many–Two relations:
  - One for the entity type
  - One for an associative relation in which the primary key has two attributes, both taken from the primary key of the entity

Mapping a unary 1:N relationship

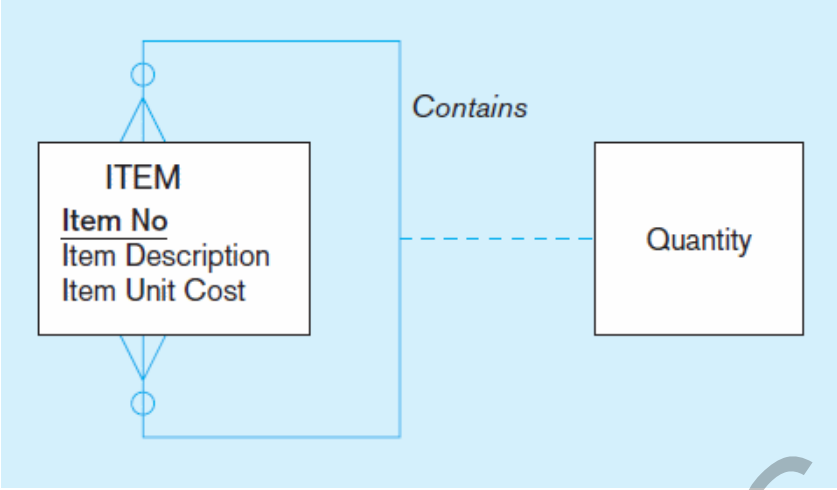


(a) EMPLOYEE entity with unary relationship

(b) EMPLOYEE relation with recursive foreign key

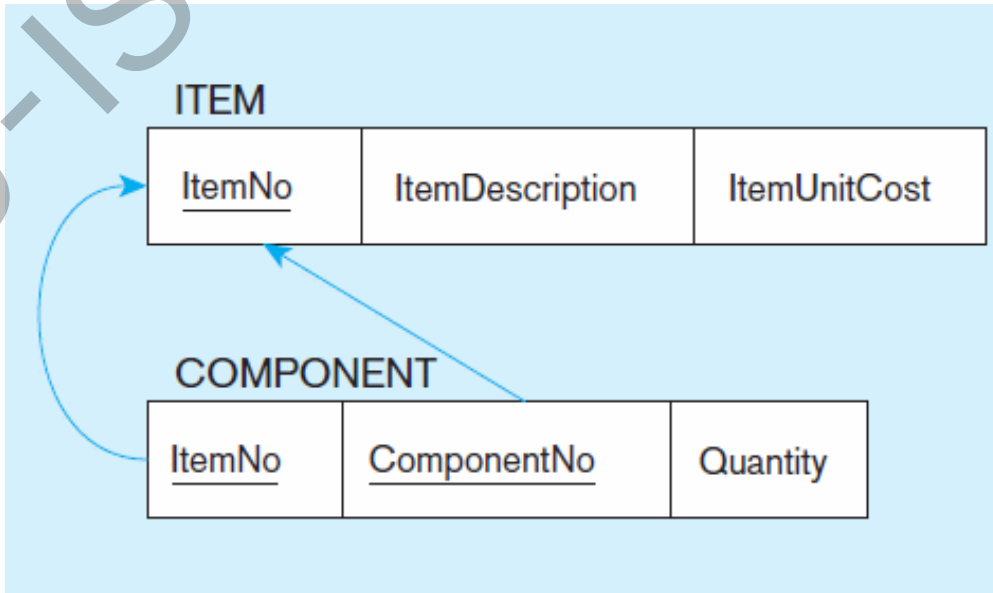


Mapping a unary M:N relationship



(a) Bill-of-materials relationships (M:N)

(b) ITEM and COMPONENT relations



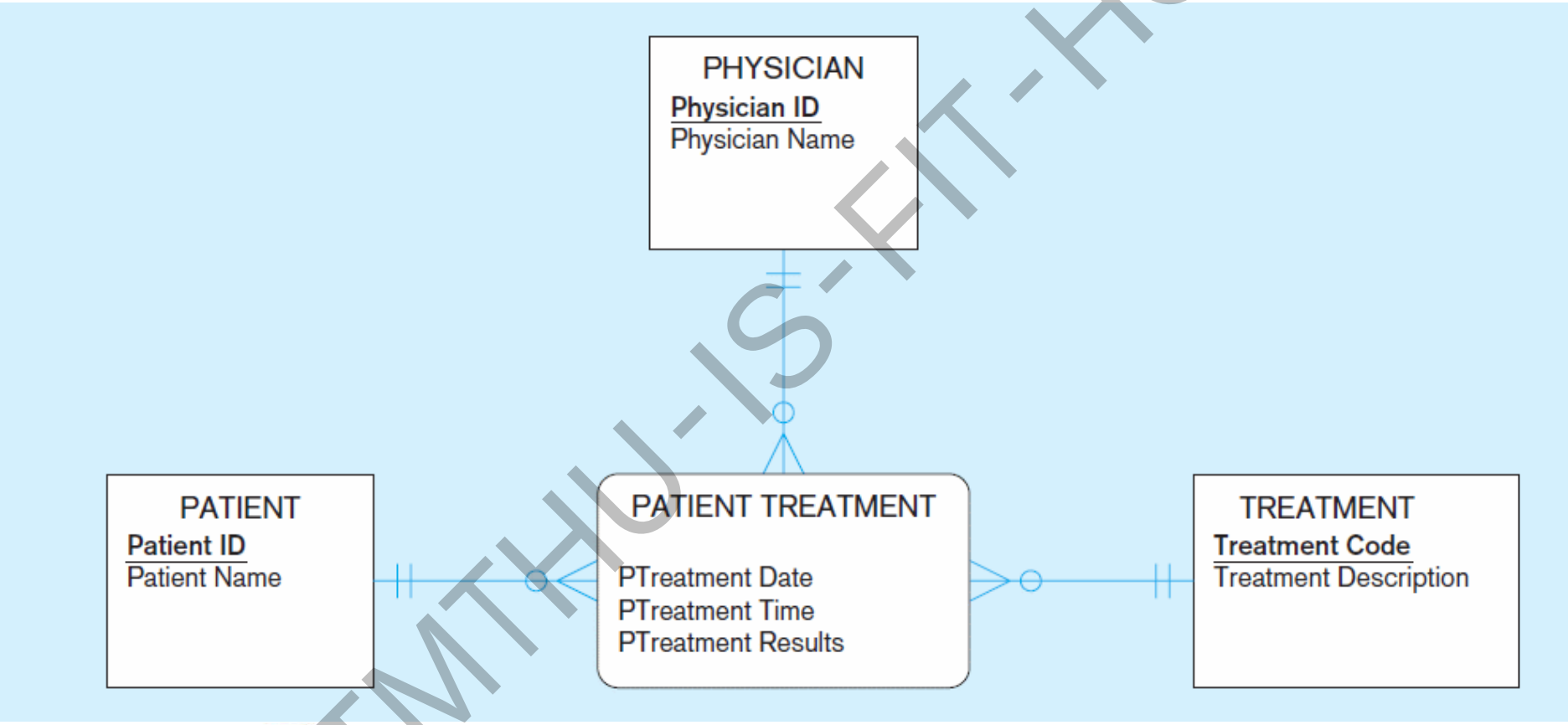
Transforming EER Diagrams into Relations (cont.)

Mapping Ternary (and n-ary) Relationships

- One relation for each entity and one for the associative entity
- Associative entity has foreign keys to each entity in the relationship

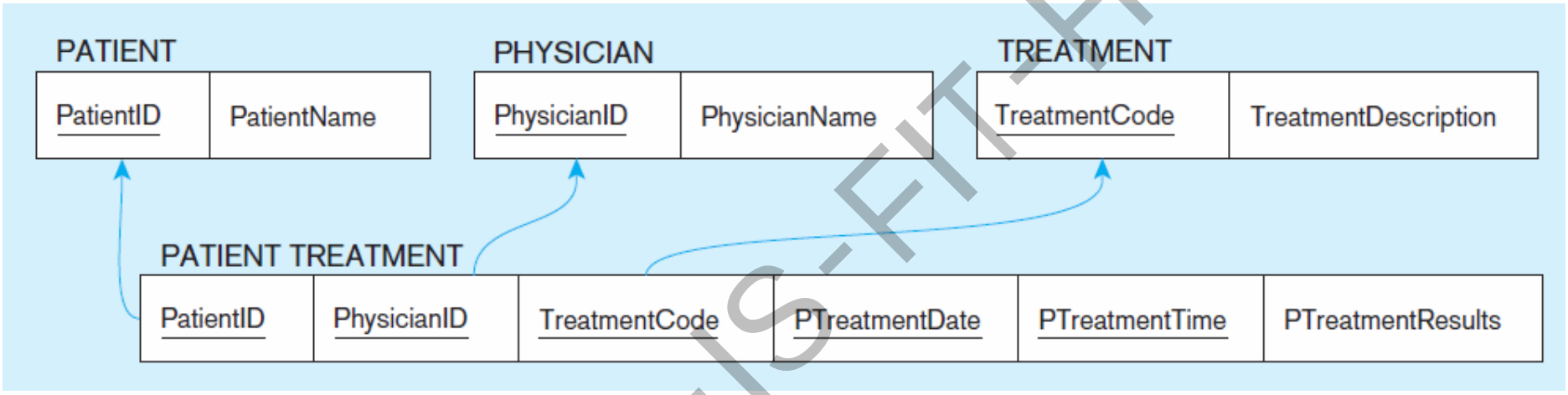
Mapping a ternary relationship

a) PATIENT TREATMENT Ternary relationship with associative entity



Mapping a ternary relationship (cont.)

b) Mapping the ternary relationship PATIENT TREATMENT



Remember that the primary key MUST be unique.

This is why treatment date and time are included in the composite primary key.

But this makes a very cumbersome key…

It would be better to create a surrogate key like Treatment#.

Transforming EER Diagrams into Relations (cont.)

Mapping Supertype/Subtype Relationships

- One relation for supertype and for each subtype
- Supertype attributes (including identifier and subtype discriminator) go into supertype relation
- Subtype attributes go into each subtype; primary key of supertype relation also becomes primary key of subtype relation
- 1:1 relationship established between supertype and each subtype, with supertype as primary table

Transforming EER Diagrams into Relations (cont.)

- **Options for Mapping Specialization or Generalization.**

  Convert each specialization with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and generalized superclass C, where the attributes of C are $\{k, a_1, \ldots a_n\}$ and k is the (primary) key, into relational schemas using one of the four following options:

  →Option A: Multiple relations-Superclass and subclasses
  →Option B: Multiple relations-Subclass relations only
  →Option C: Single relation with one type attribute
  →Option D: Single relation with multiple type attributes

Transforming EER Diagrams into Relations (cont.)

- **Option A: Multiple relations-Superclass and subclasses**

  Create a relation L for C with attributes Attrs(L) = $\{k,a_1,\ldots a_n\}$ and PK(L) = k. Create a relation L_i for each subclass S_i, $1 < i < m$, with the attributesAttrs(L_i) = {k} U {attributes of S_i} and PK(L_i)=k. This option works for any specialization (total or partial, disjoint or over-lapping).

- **Option B: Multiple relations-Subclass relations only**

  Create a relation L_i for each subclass S_i, $1 < i < m$, with the attributes Attr(L_i) = {attributes of S_i} U $\{k,a_1\ldots,a_n\}$ and PK(L_i) = k. This option only works for a specialization whose subclasses are total (every entity in the superclass must belong to (at least) one of the subclasses).

Transforming EER Diagrams into Relations (cont.)

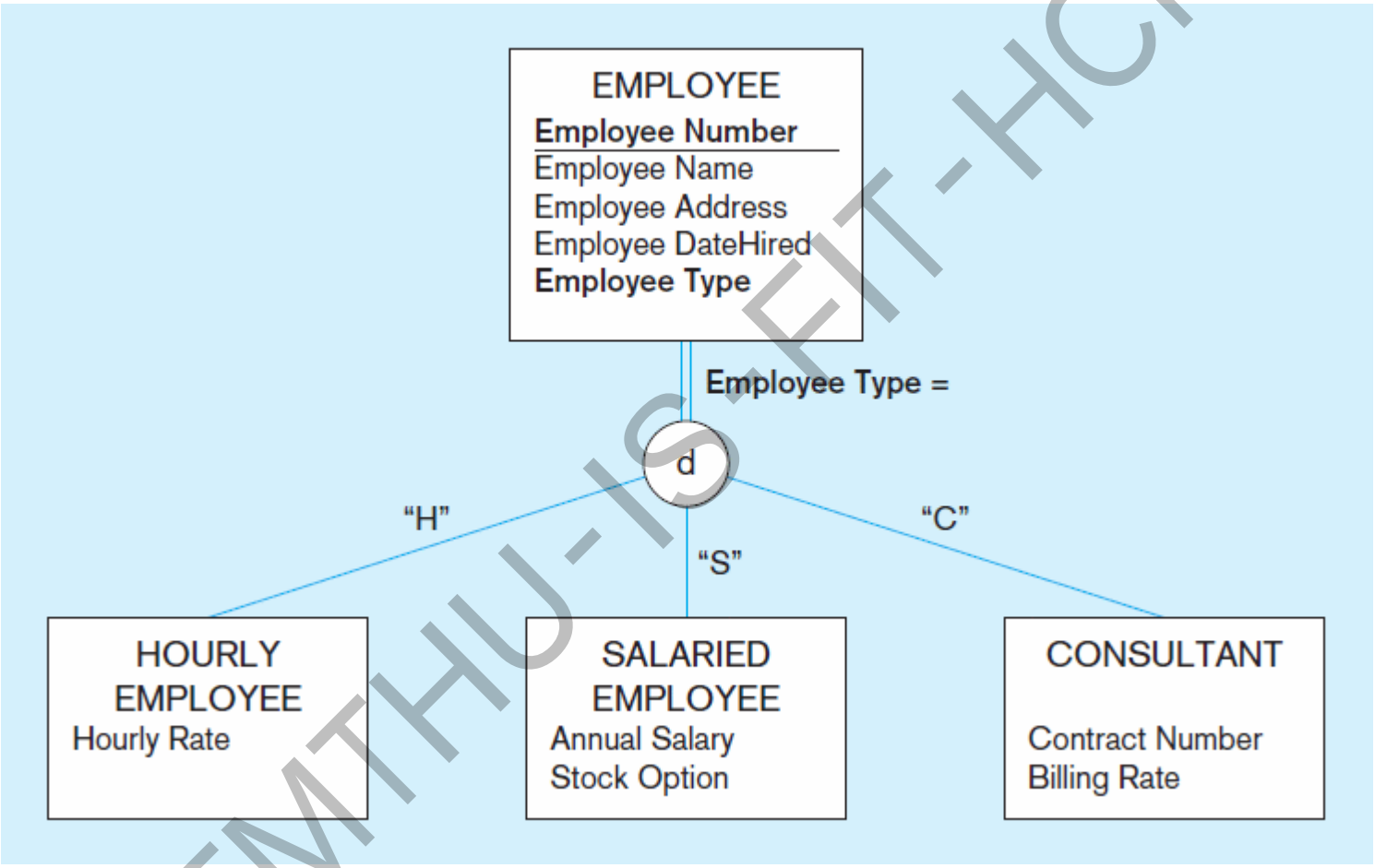
- **Option C: Single relation with one type attribute**

  Create a single relation L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t} and PK(L) = k. The attribute t is called a type (or **discriminating**) attribute that indicates the subclass to which each tuple belongs

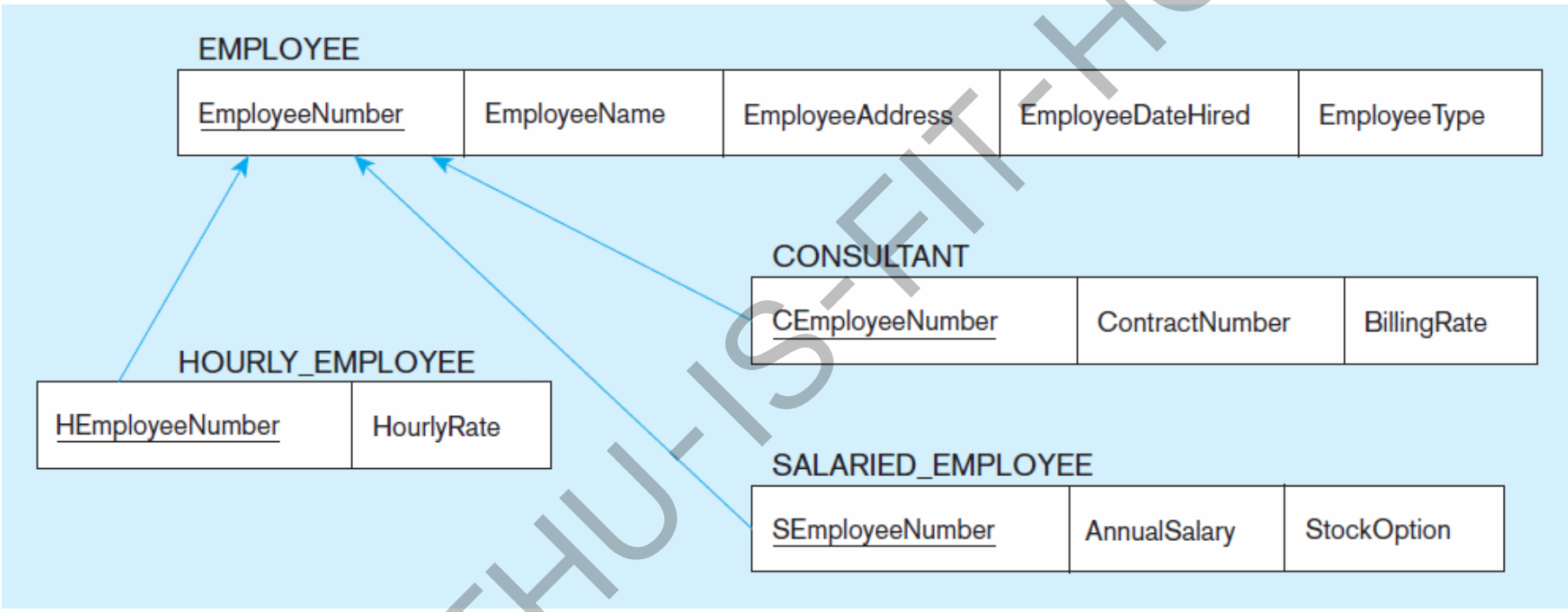
- **Option D: Single relation with multiple type attributes**

  Create a single relation schema L with attributes Attrs(L) = {k,a_1,…a_n} U {attributes of S_1} U…U {attributes of S_m} U {t_1, t_2,…,t_m} and PK(L) = k. Each t_i, 1 < I < m, is a Boolean type attribute indicating whether a tuple belongs to the subclass S_i.

Supertype/subtype relationships

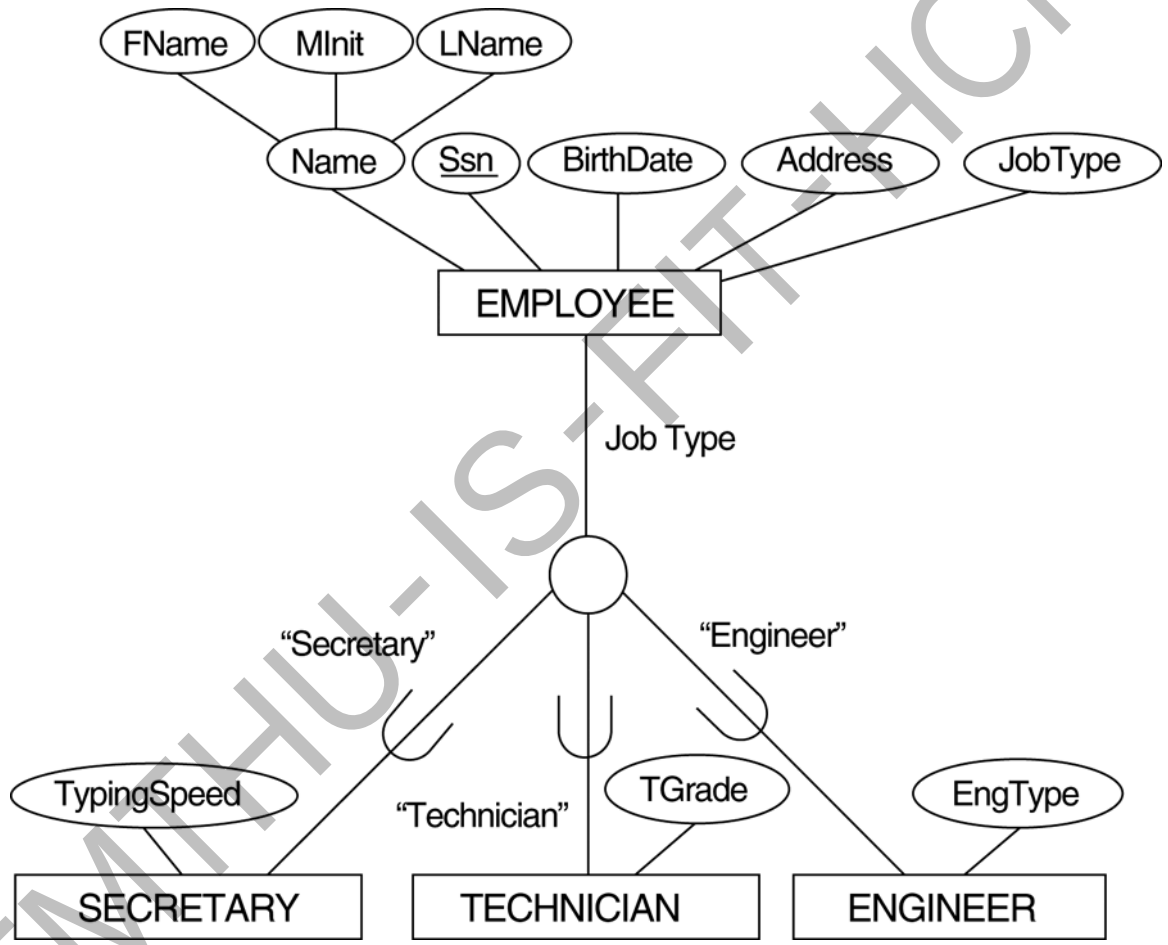


Mapping supertype/subtype relationships to relations

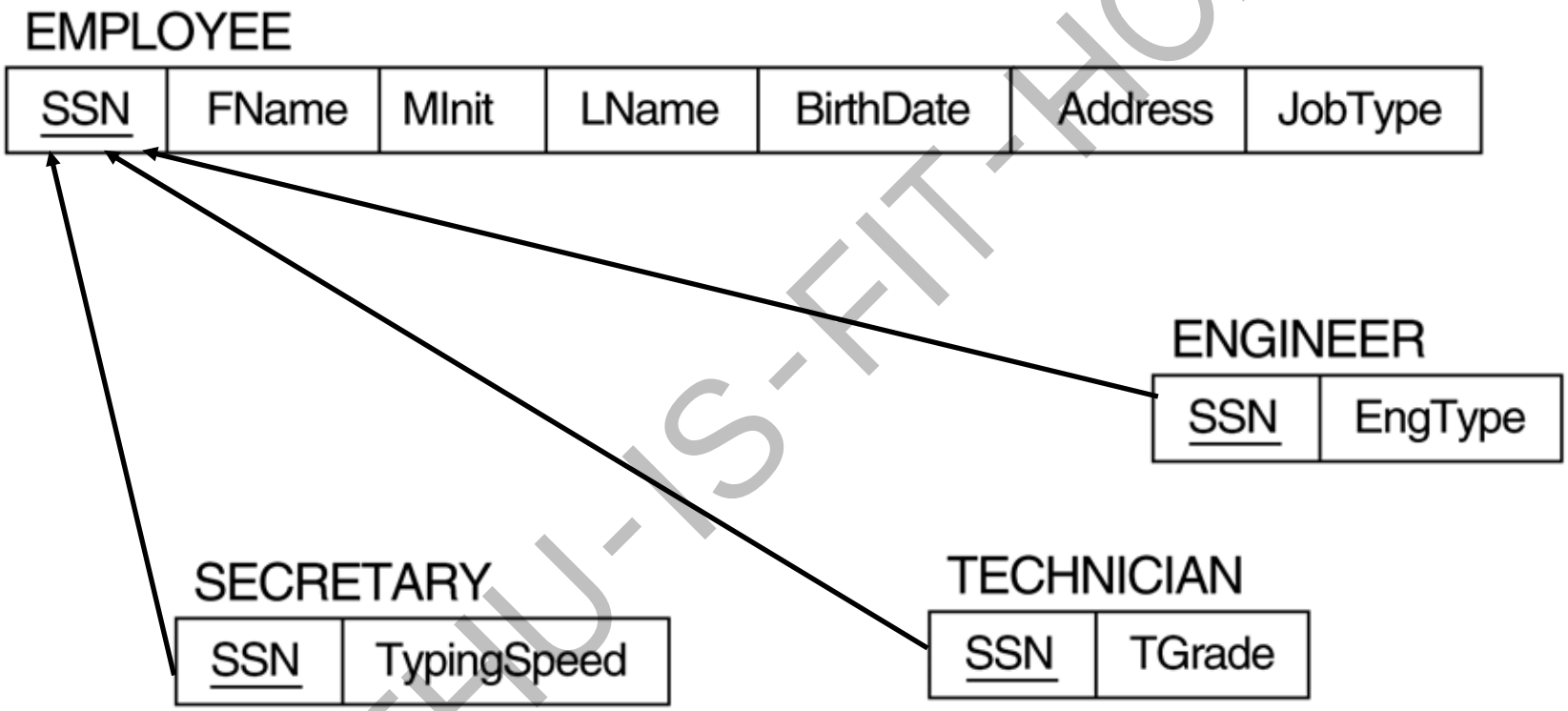


Option A: These are implemented as one-to-one relationships.

Supertype/subtype relationships



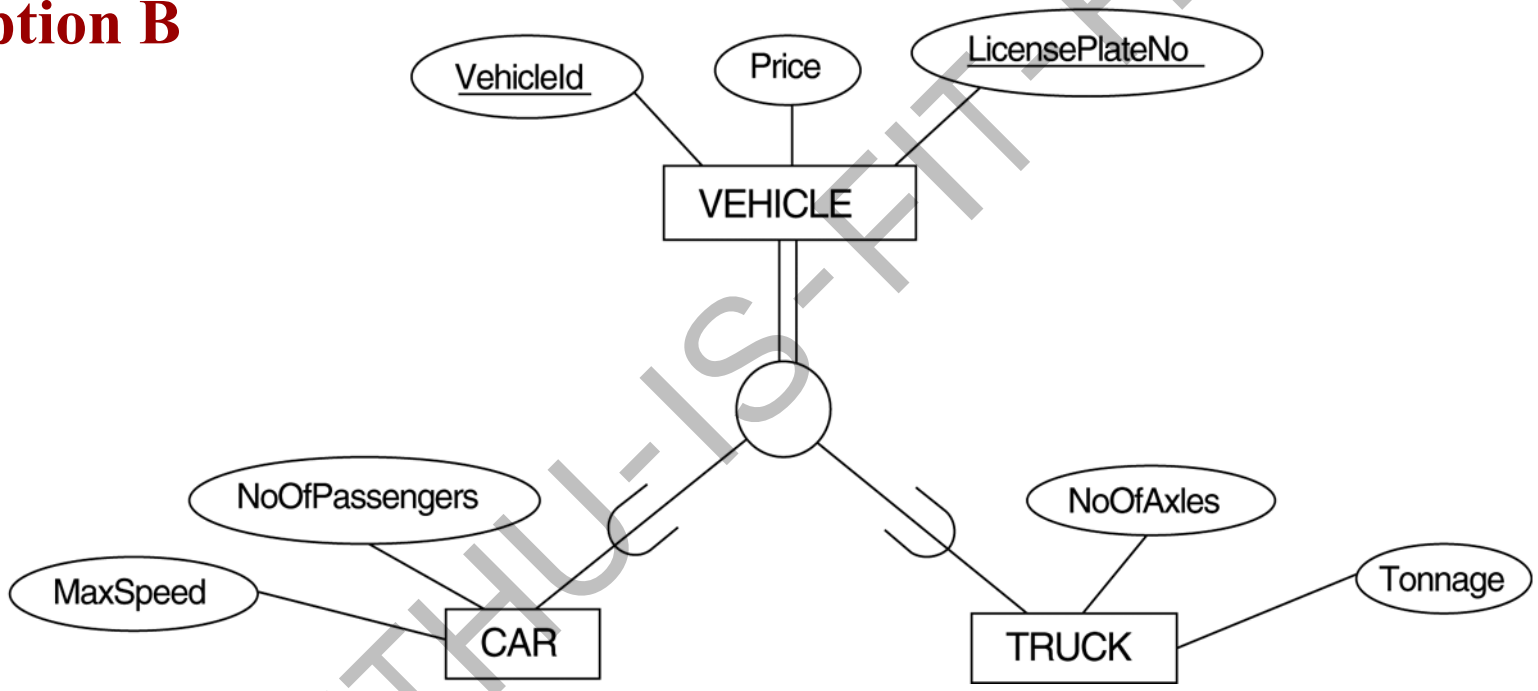
Supertype/subtype relationships



Option A: Multiple relations-Superclass and subclasses.

Supertype/subtype relationships

Option B



Supertype/subtype relationships

CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

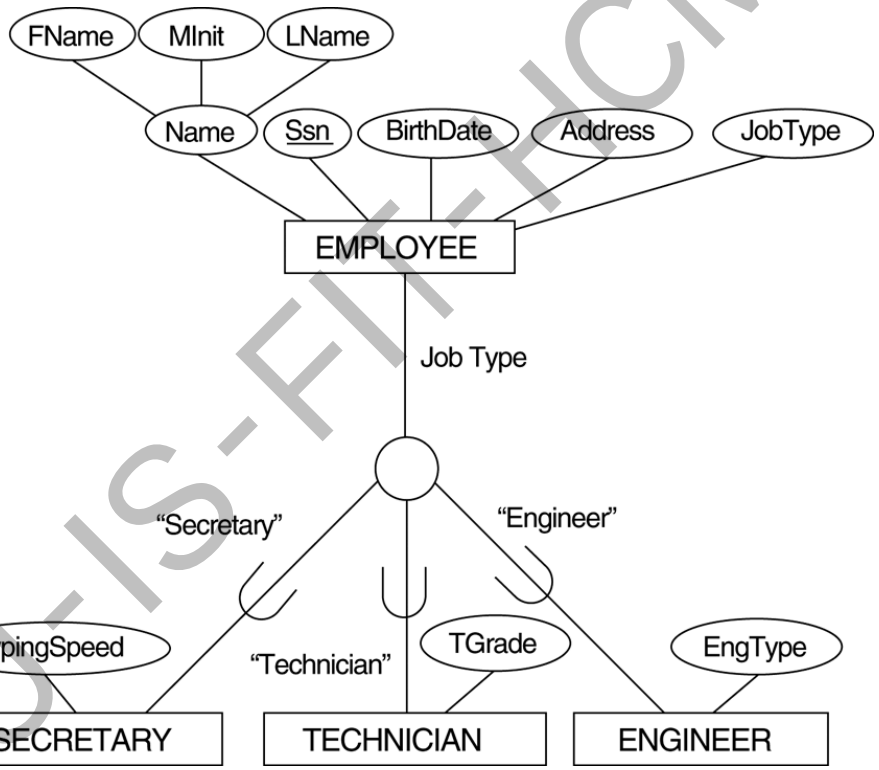
TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	

Option B. Multiple relations-Subclass relations only

Supertype/subtype relationships

Option C



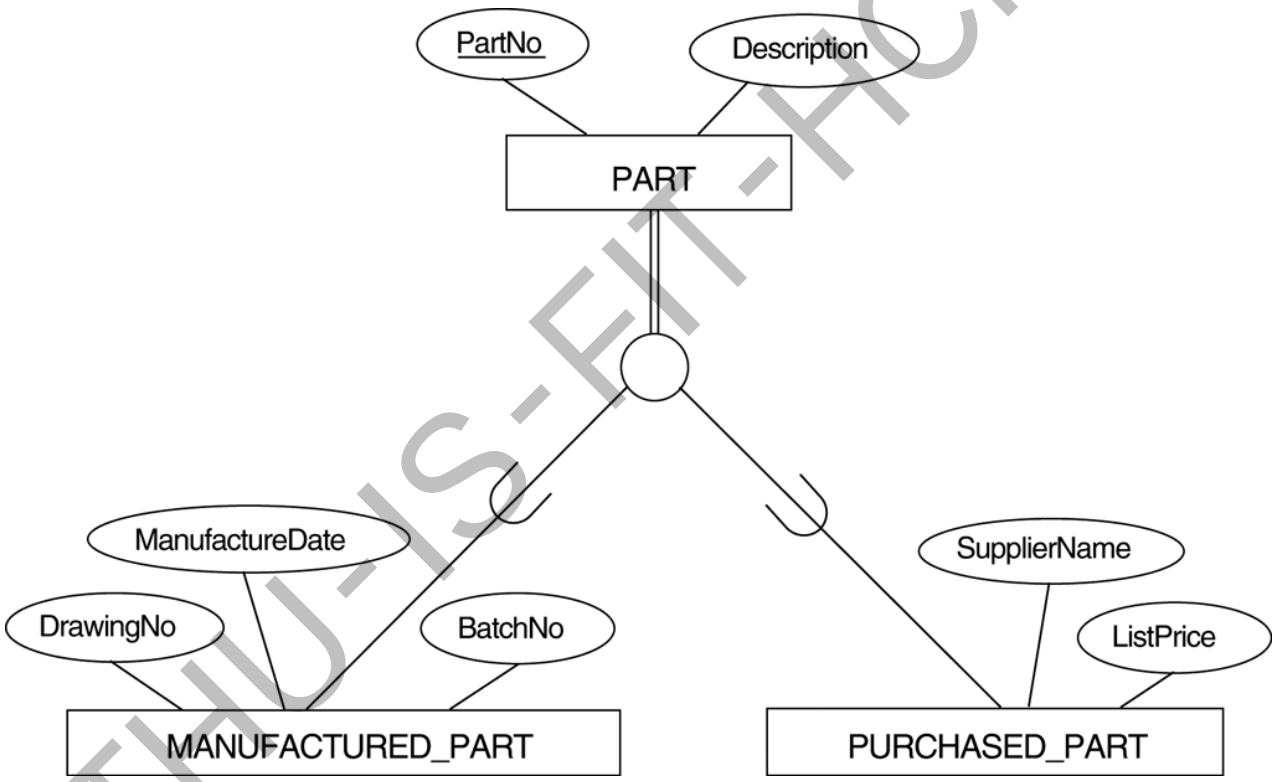
EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	EmpType

discriminating attribute

Supertype/subtype relationships

Option D



PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice

Boolean type attribute

Case Tools

- CASE tools can perform many of the transformation steps automatically, but..
  - Often CASE tools cannot model complexity of data and relationship (Ternary relationships, supertype/subtypes, i.e..)
  - You must be able to perform a quality check on CASE tool results

* Mapping a conceptual model to a relational schema is a straight-forward process...

Logical Design Steps

Step 1
- Map the conceptual Model to logical model components

Step 2
- Validate the logical model using normalization

Step 3
- Validate the logical model integrity constraints

Step 4
- Validate the logical model against user requirements

Data Normalization

- Primarily a tool to validate and improve a logical design so that it satisfies certain constraints that avoid unnecessary duplication of data
- The process of decomposing relations with anomalies to produce smaller, well-structured relations

Well-Structured Relations

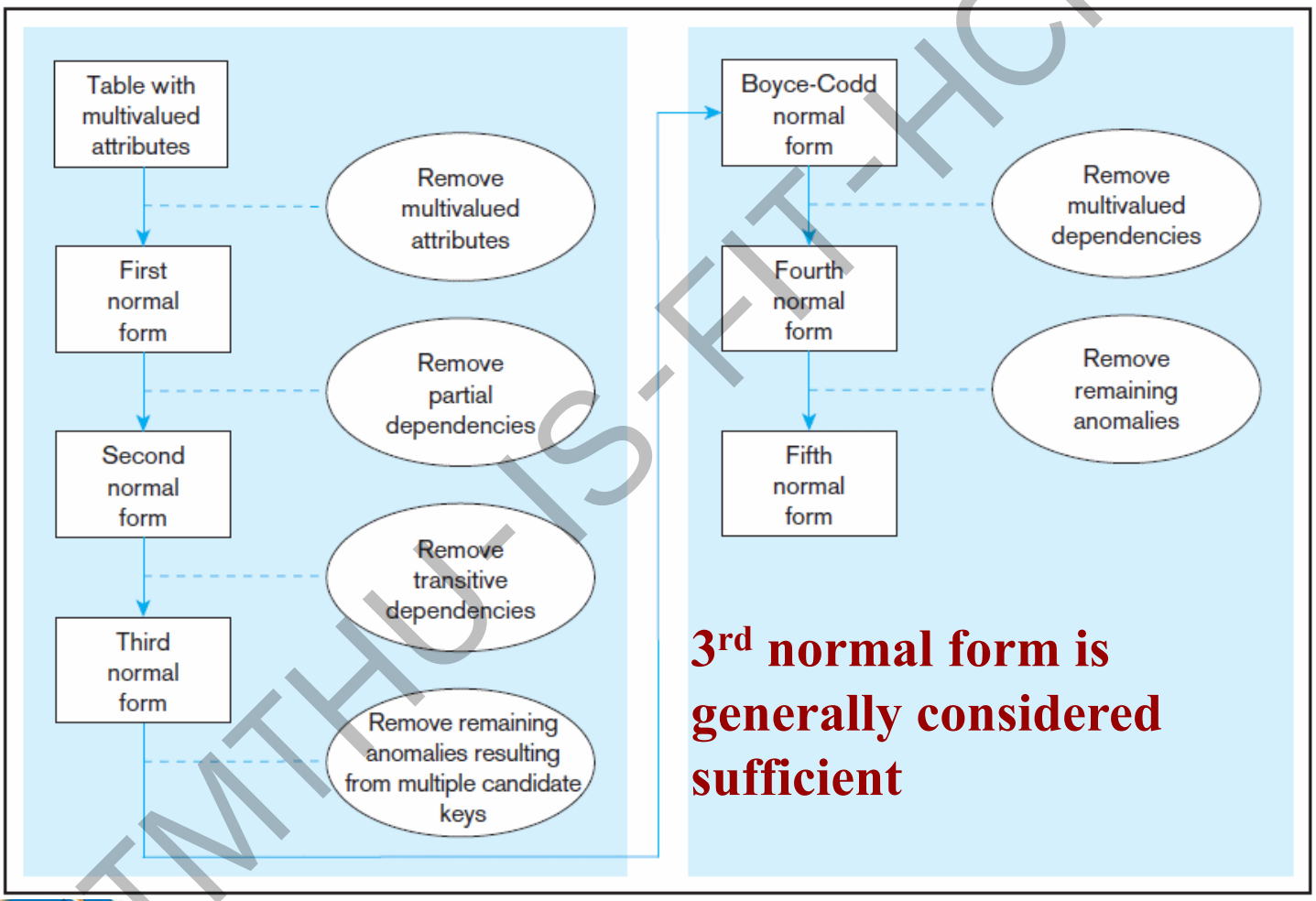
- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies
- Goal is to avoid anomalies
  - **Insertion Anomaly**–adding new rows forces user to create duplicate data
  - **Deletion Anomaly**–deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly**–changing data in a row forces changes to other rows because of duplication

General rule of thumb: A table should not pertain to more than one entity type.

Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in "good" form.
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving.

Normalization using FDs



Merging Relations (View Integration)

- **Why is merging Relations important?**
  1. On large projects, the work of several sub-teams comes together during logical design, so there is often a need to merge relations
  2. Integrating existing databases with new information requirements often leads to the need to integrate different views.
  3. New data requirements may arise during the life cycle, so there is a need to merge any new relations with what has already been developed.

Merging Relations (View Integration)

The first user view

EMPLOYEE1(EmployeeID, Name, Address, Phone)

The second user view

EMPLOYEE2(EmployeeID, Name, Address, Jobcode, NoYears)

User view integration

EMPLOYEE(EmployeeID, Name, Address, Phone, Jobcode, NoYears)

View integration Problems

- Synonyms
- Homonyms
- Transitive dependencies
- Supertype/subtype relationships.

Synonyms

Synonyms: Two (or more) attributes that have different names but the same meaning

STUDENT1(StudentID, Name)
STUDENT2(MatriculationNo, Name, Address)

STUDENT(StudentNo, Name, Address)

alias

An alternative name used for an attribute.

Homonyms

Homonym: An attribute that may have more than one meaning

STUDENT1(StudentID, Name, Address)
STUDENT2(StudentID, Name, PhoneNo, Address)

STUDENT(StudentID, Name, PhoneNo, CampusAddress, PermanentAddress)

Transitive dependencies

When two 3NF relations are merged to form a single relation, transitive dependencies may result

STUDENT(StudentID, MajorName, Advisor)

MajorName → Advisor

STUDENT(StudentID, MajorName)
MAJOR (MajorName, Advisor)

Supertype/subtype relationships

These relationships may be hidden in user views or relations

PATIENT1(PatientID, Name, Address)
PATIENT2(PatientID, RoomNo)

PATIENT(PatientID, Name, Address)
RESIDENTPATIENT(PatientID, RoomNo)
OUTPATIENT(PatientID, DateTreated)

Defining Relational Keys

- Primary Key
- Natural Key (or Natural Identifier)
- Desirable Primary Key Characteristics
- Enterprise Key

Primary Keys

- **Single attribute or a combination of attributes, which uniquely identifies each entity instance**
  - Guarantees entity integrity
  - Works with foreign keys to implement relationships

Natural Keys or Natural Identifier

- **Real-world identifier used to uniquely identify real-world objects**
  - Familiar to end users and forms part of their day-to-day business vocabulary
  - Also known as natural identifier
  - Used as the primary key of the entity being modeled

Desirable Primary Key Characteristics

- Non intelligent (không nên nhúng ngữ nghĩa vào)
- No change over time (không đổi theo thời gian)
- Preferably single-attribute (nên là thuộc tính đơn)
- Preferably numeric (nên là trường kiểu số)
- Security-compliant (tuân thủ tính bảo mật)

Surrogate Primary Keys

- **Primary key used to simplify the identification of entity instances are useful when:**
  - There is no natural key
  - Selected candidate key has embedded semantic contents or is too long
- **Require ensuring that the candidate key of entity in question performs properly**
  - Use unique index and not null constraints

Surrogate Primary Keys

- **Example: Primary Key of EVENT?**

EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF)

DATA USED TO KEEP TRACK OF EVENTS

DATE	TIME_START	TIME_END	ROOM	EVENT_NAME	PARTY_OF
6/17/2016	11:00a.m.	2:00p.m.	Allure	Burton Wedding	60
6/17/2016	11:00a.m.	2:00p.m.	Bonanza	Adams Office	12
6/17/2016	3:00p.m.	5:30p.m.	Allure	Smith Family	15
6/17/2016	3:30p.m.	5:30p.m.	Bonanza	Adams Office	12
6/18/2016	1:00p.m.	3:00p.m.	Bonanza	Boy Scouts	33
6/18/2016	11:00a.m.	2:00p.m.	Allure	March of Dimes	25
6/18/2016	11:00a.m.	12:30p.m.	Bonanza	Smith Family	12

(**DATE, TIME_START, ROOM**)? or (**DATE, TIME_END, ROOM**)?

Surrogate Primary Keys

- **Example: Primary Key of EVENT?**

EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF)

(**DATE, TIME_START, ROOM**)? or (**DATE, TIME_END, ROOM**)?

Assume each event (EVENT) uses many resources (RESOURCE): tables, computers, projectors, …

RESOURCE (**RSC_ID**, RSC_DESCRIPTION, RSC_TYPE, RSC_QTY, RSC_PRICE)

EVNTRSC (**DATE, TIME_START, ROOM, RSC_ID**, QTY_USED)

Surrogate Primary Keys

Lưu ý:

The surrogate key assures entity integrity, it does not guarantee the correct semantics of business constraints.
In conceptual design, it is not possible to have the surrogate & primary key (the original) at the same time, the surrogate key only appears when programming.

EVNTRSC (**ID**, **DATE**, **TIME_START**, **ROOM**, **RSC_ID**, QTY_USED)

Enterprise key

- A primary key whose value is unique across all relations.
- Corresponds with the concept of an object ID in object-oriented systems

EMPLOYEE(EmpID, EmpName, DeptName, Salary)
CUSTOMER(CustID, CustName, Address)

Suppose that employees can also be customers:

PERSON(PersonID, PersonName)
EMPLOYEE(PersonID, DeptName, Salary)
CUSTOMER(PersonID, Address)

PROBLEMS?

Enterprise key

Relations with enterprise key

OBJECT (OID, ObjectType)
EMPLOYEE (OID, EmpID, EmpName, DeptName, Salary)
CUSTOMER (OID, CustID, CustName, Address)

Sample data with enterprise key

OBJECT

OID	ObjectType
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER

EMPLOYEE

OID	EmpID	EmpName	DeptName	Salary
1	100	Jennings, Fred	Marketing	50000
4	101	Hopkins, Dan	Purchasing	45000
5	102	Huber, Ike	Accounting	45000

CUSTOMER

OID	CustID	CustName	Address
2	100	Fred's Warehouse	Greensboro, NC
3	101	Bargain Bonanza	Moscow, ID
6	102	Jasper's	Tallahassee, FL
7	103	Desks 'R Us	Kettering, OH

Logical Design Steps

Step 1	• Map the conceptual Model to logical model components
Step 2	• Validate the logical model using normalization
Step 3	• Validate the logical model integrity constraints
Step 4	• Validate the logical model against user requirements

Validate Logical Model Integrity Constraints

CLASS (CLASS_CODE, *EMP_NUM*, CLASS_TIME, CLASS_DAYS, *CRS_CODE*)

PRIMARY KEY: CLASS_CODE
FOREIGN KEYS: EMP_NUM REFERENCES PROFESSOR
CRS_CODE REFERENCES COURSE

CLASS_CODE	is a valid class code. Type: numeric Range: low value=1000 high value=9999 Display format: 9999 Length: 4
CLASS_DAYS	is a valid day code. Type: character Display format: XXX Valid entries: MWF, TR, M, T, W, R, F, S Length: 3
CLASS_TIME	is a valid time. Type: character Display format: 99:99 (24-hour clock) Display range: 06:00 to 22:00 Length: 5

Logical Design Steps

Step 1
- Map the conceptual Model to logical model components

Step 2
- Validate the logical model using normalization

Step 3
- Validate the logical model integrity constraints

Step 4
- Validate the logical model against user requirements

Validate the logical model against user requirements

1. Identify the Relational model's central relation
2. Identify each module's transaction requirements
   - Internal: updates/inserts/deletes/queries/reports
   - External: module interfaces
3. Verify all processes against system requirements
4. Make all necessary changes suggested in Step 4
5. Repeat Steps 2–5 for all modules