London Business School

AM05 Data Management

02. Logical Database Design

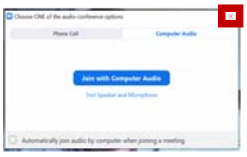
Dr. David Tilson

London Business School

AM05: Data Management

David Tilson

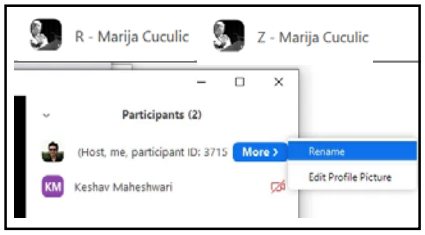
Welcome!
Class starts at 12:45 (London). Thank you for being early!



Roomies
When prompted, click "X"

Zoom Classroom Etiquette

1. Roomies: Please **turn on** your cameras and **join without audio**



2. Rename yourself
   a. Add “R” (for Roomie) in front of your name if you’re in the LT
   b. Add “Z” (for Zoomie) in front of your name if you’re remote

3. Questions
   a. Raise you (digital) hand if you want to speak or ask a question that needs an answer ASAP
   b. Use chat to ask questions that can wait for a few minutes
   c. I will ask you to answer questions using options on “participants” panel
   d. Technical issues - message the facilitator privately in Zoom chat

4. If you are in a breakout room, engage with your colleagues to extract the most out the class

5. Session will be recorded

¶ Components of Relational Data Model
- **Data structure** – Relations (tables), rows, and columns
- **Data manipulation** – SQL operations for retrieving/modifying data
- **Data integrity** – Mechanisms for implementing business rules that maintain integrity of manipulated data

¶ Converting ERD into Database

¶ Database Normalization

¶ Hands On

¶ For next time

In the relational model data is represented in **relations (or tables)** which have certain properties

1. Each **relation (table)** has a unique name

2. Each **attribute (column)** has a unique name

3. An entry at an intersection of row and column is **atomic** (not multivalued).

Employee

EmpNo	EmpName	Department	Email	Phone
E100	Chuck Coordinator	Administration	chuck@colorado.edu	3-1111
E101	Mary Manager	Football	mary@colorado.edu	5-1111
E102	Sally Supervisor	Planning	sally@colorado.edu	3-2222
E103	Alan Administrator	Administration	alan@colorado.edu	3-3333

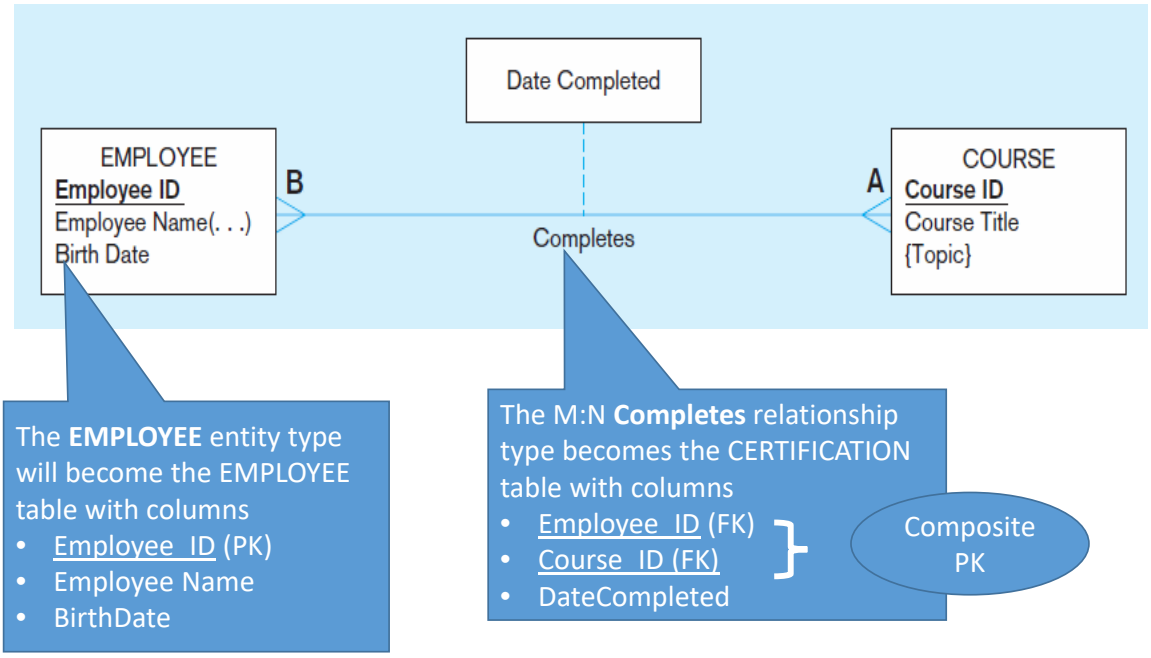
4. Each **row (record / tuple)** is unique
- Implies that relation must have a **Primary Key (PK)** – an attribute (or combination of attributes) that *uniquely identifies* each row.
- Corresponds to the identifier in an ERD

5. Sequence is NOT significant for
- Columns (left-to-right)
- Rows (top-to-bottom)

Data Structure

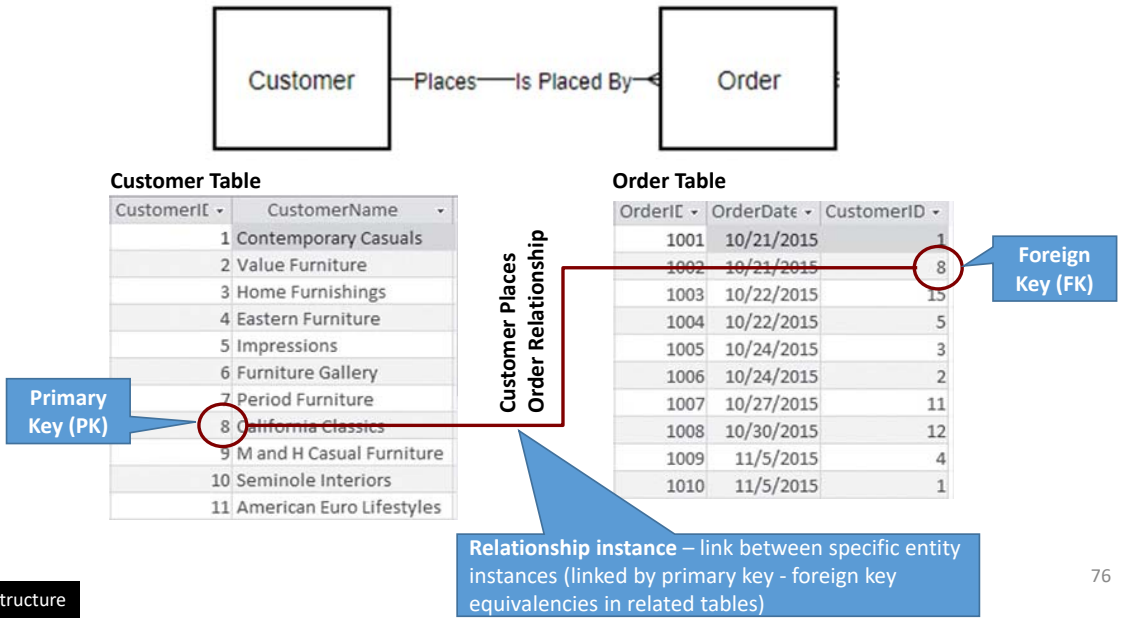
- All relations are tables. . . but not all tables are relations
- Relations can be thought of as **sets** of tuples
- Relations that follow these rules are in 1st Normal Form (will see soon)

Relations (tables) correspond with **entity types** and with **many-to-many relationship types** (or associative entities)



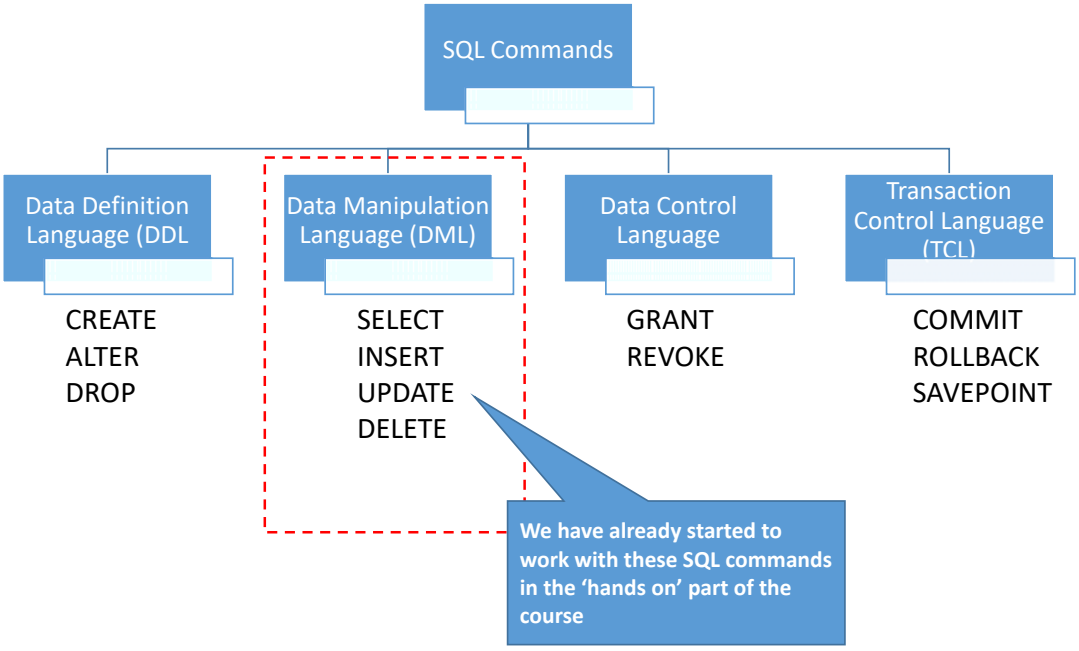
Data Structure NOTE: The word **relation** (in a relational database) is NOT the same as the word **relationship** in an ERD

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)



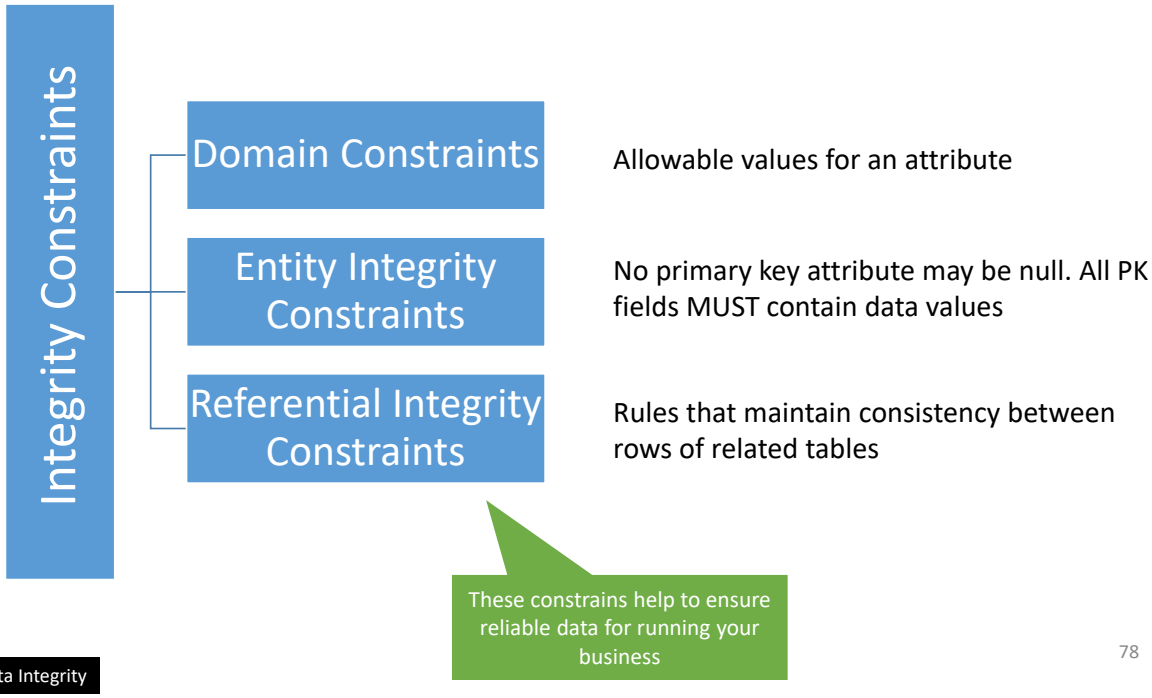
Data Structure

SQL provides powerful operations for retrieving and modifying data



Data Manipulation

Relational model includes **constraints** (rules) limiting acceptable values and actions to help **maintain accuracy and integrity** of the data in a database



Data Integrity

Data Domain refers to all values a data element may contain. Rule can be as simple as data type or enumerated list of values

Domain definitions help ensure that data in a database is valid

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

Enforced by
- Specifying datatypes and length when creating tables
- Enumerating values in another relation (e.g. list of valid US state abbreviations)
- Checks in applications (e.g. ensure that prices and quantities are positive)

Data Integrity – Domain Constraints

Entity integrity rule is to ensure that relations have a primary key that is unique and cannot contain NULLs

Example of how to create a relation in SQL DDL

Specify that the CustomerID cannot be NULL

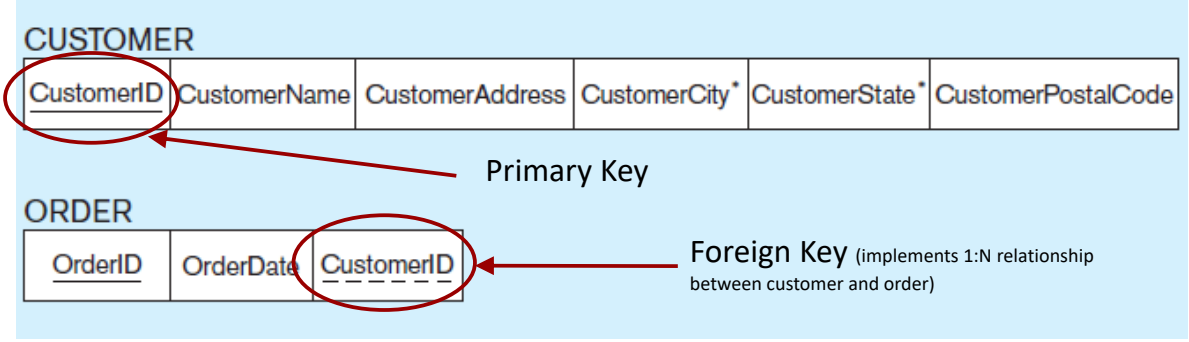
```
CREATE TABLE Customer_T
        (CustomerID             NUMBER(11,0)    NOT NULL,
        CustomerName            VARCHAR2(25)    NOT NULL,
        CustomerAddress         VARCHAR2(30),
        CustomerCity            VARCHAR2(20),
        CustomerState           CHAR(2),
        CustomerPostalCode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

This is one way a Primary Key is specified in SQL

Enforced by
- DBMS not allowing operations (INSERT, UPDATE) to produce an invalid primary key
- Any operation that creates a duplicate primary key or one containing nulls is rejected

Data Integrity – Entity Integrity Constraint

Referential integrity rules maintain consistency among the rows of relations



Enforced by

- DBMS not allowing insertion of new row in a table with a Foreign Key (FK) unless the value of the FK matches a Primary Key (PK) in the other table. For example, we would not be allowed to insert a new ORDER unless the CustomerID existed in the CUSTOMER table
- Handling issues with deletion of rows in parent table (e.g. CUSTOMER above)*
  - **Restrict**–don't allow delete of *parent* side if related rows exist in *dependent* side
  - **Cascade**–automatically delete *dependent* side rows corresponding to *parent* side row to be deleted
  - **Set-to-Null**–set FK on dependent side to null if deleting from the parent side (but not allowed for weak entities)

Data Integrity – Referential Integrity Constraint

See: https://www.mysqltutorial.org/mysql-foreign-key/ for more information

Referential integrity constraints implemented with foreign key to primary key references

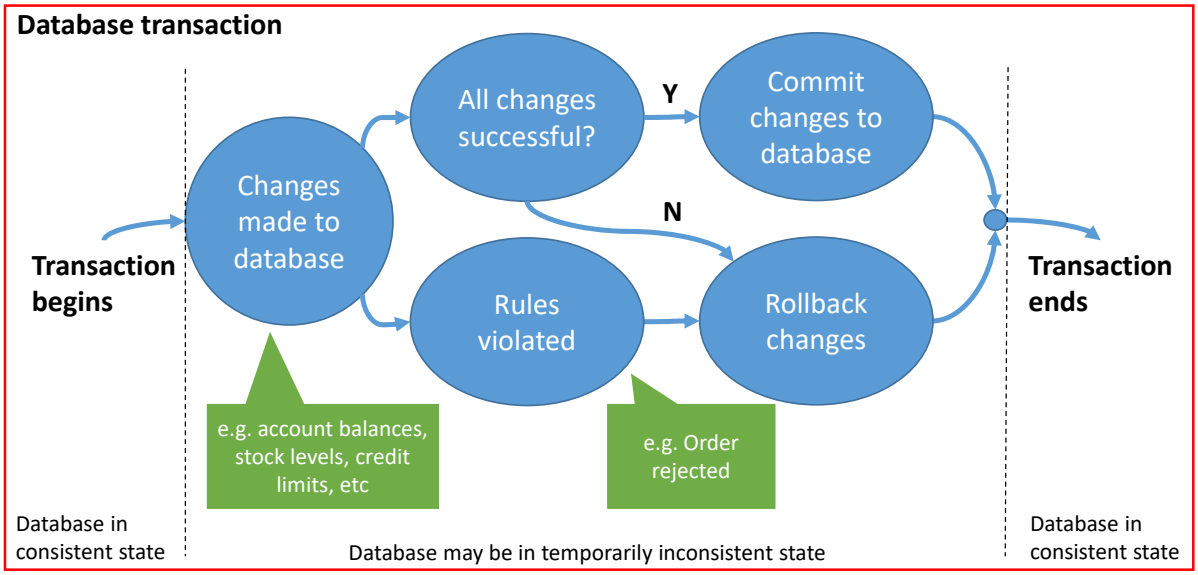
```
CREATE TABLE Customer_T
        (CustomerID                 NUMBER(11,0)     NOT NULL,
        CustomerName                VARCHAR2(25)     NOT NULL,
        CustomerAddress             VARCHAR2(30),
        CustomerCity                VARCHAR2(20),
        CustomerState               CHAR(2),
 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

Quick questions . . . . Answer via Zoom chat

1. Why is it important to have "referential integrity constraints" for the relationship between an order and a customer?

2. Would it be a good idea to use VARCHAR as the data type for OrderQuantity? Why/why not?

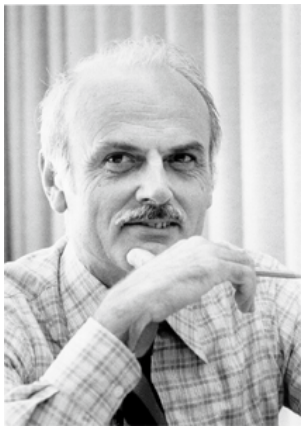
RDMBS can ensure database consistency across several operations using **transactions**



ACID Properties – to ensure accuracy, completeness, and data integrity

Atomicity	Each transaction is all or nothing
Consistency	Data should be consistent according to all rules
Isolation	Transactions do not affect each other
Durability	Committed data not lost (even if there was a power failure)

E.F. Codd invented the theoretical basis for the relational model



Edgar Frank "Ted" Codd (1923–2003) was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems.

- His 1970 paper **"A Relational Model of Data for Large Shared Data Banks"** introduced the concepts behind relational DBMSs.
- He also coined the term OLAP (will discuss this later)

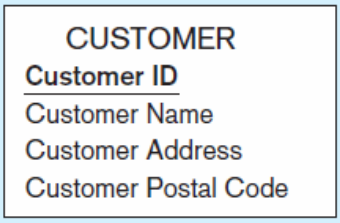
The relational model
- Has its own mathematical underpinning (Relational Algebra)
- Is the basis of most databases in use
- Ensures accurate, consistent data in innumerable mission critical systems

https://en.wikipedia.org/wiki/File:Edgar_F_Codd.jpg

- Components of Relational Data Model
- Converting ERD into Database
- Database Normalization
- Hands On
- For next time

Mapping a regular entity to a relation is very simple

CUSTOMER entity type with simple attributes



- Each attribute of the entity becomes a column (field) of the resulting relation
- Identifier of the entity becomes a primary key (PK) in the relation

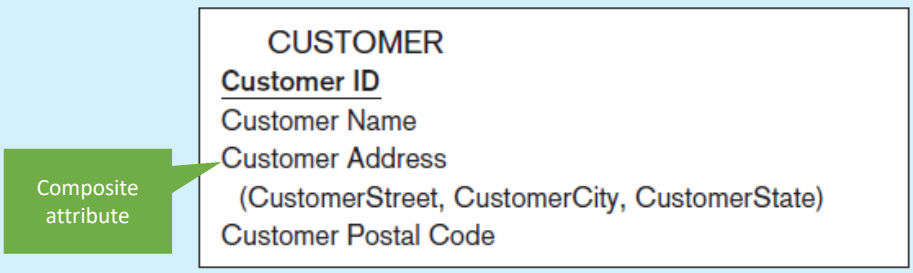
CUSTOMER relation

CUSTOMER

CustomerID	CustomerName	CustomerAddress	CustomerPostalCode

Dealing with a composite attribute is also easy

CUSTOMER entity type with composite attribute



Components of composite attribute become individual columns

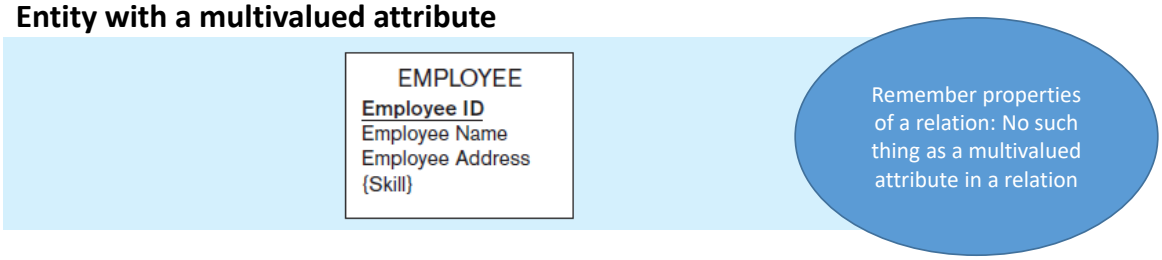
CUSTOMER relation with address detail

CUSTOMER

CustomerID	CustomerName	CustomerStreet	CustomerCity	CustomerState	CustomerPostalCode

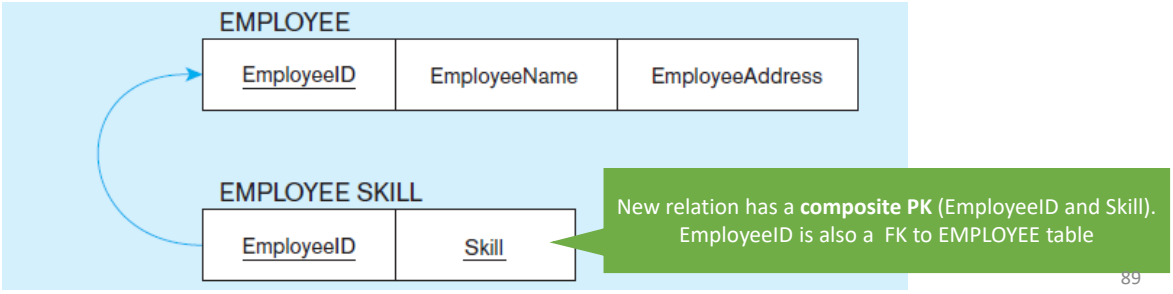
Mapping entity with multivalued attribute achieved with an additional relation

Entity with a multivalued attribute



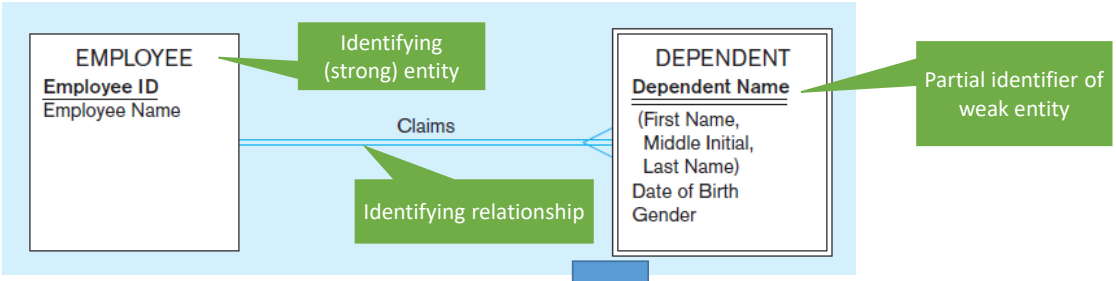
Multivalued attribute becomes a new relation with a 1:M relationship with the original entity

1:M relationship between original entity and new relation

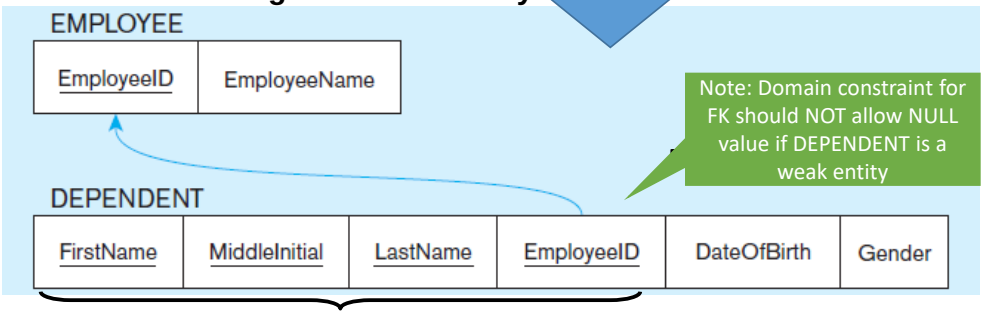


Weak entity needs a composite key with PK of owner entity

Weak entity exists through *identifying relationship* with owner entity



Relations resulting from weak entity

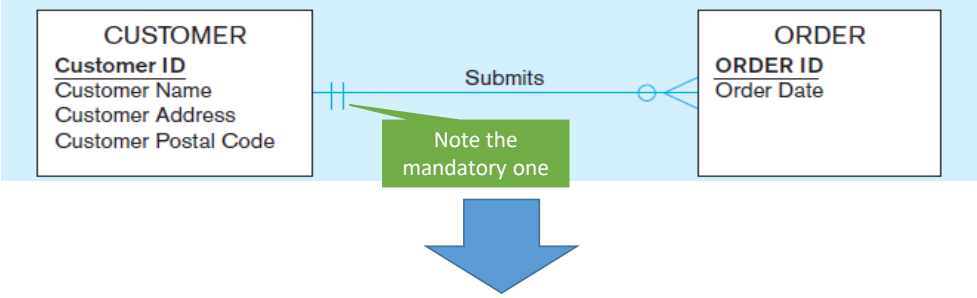


Composite PK composed of

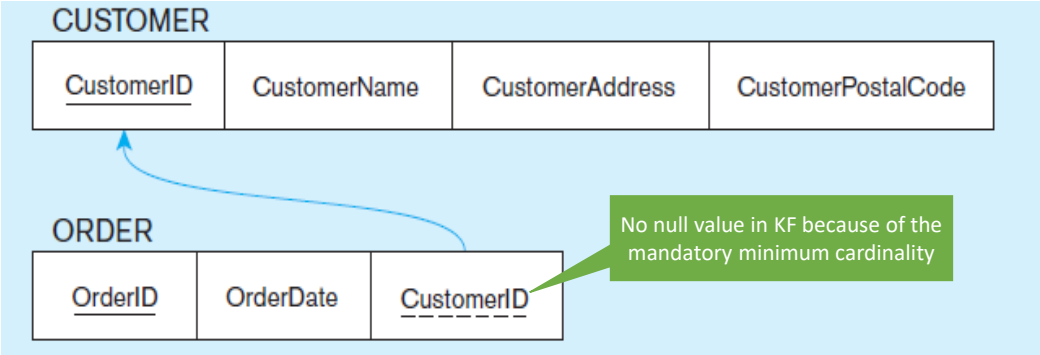
- Partial identifier of weak entity
- PK of identifying relation (strong entity)

For 1:M relationship PK on the *one side* becomes a FK on the *many side*

1:M Relationship between customers and orders

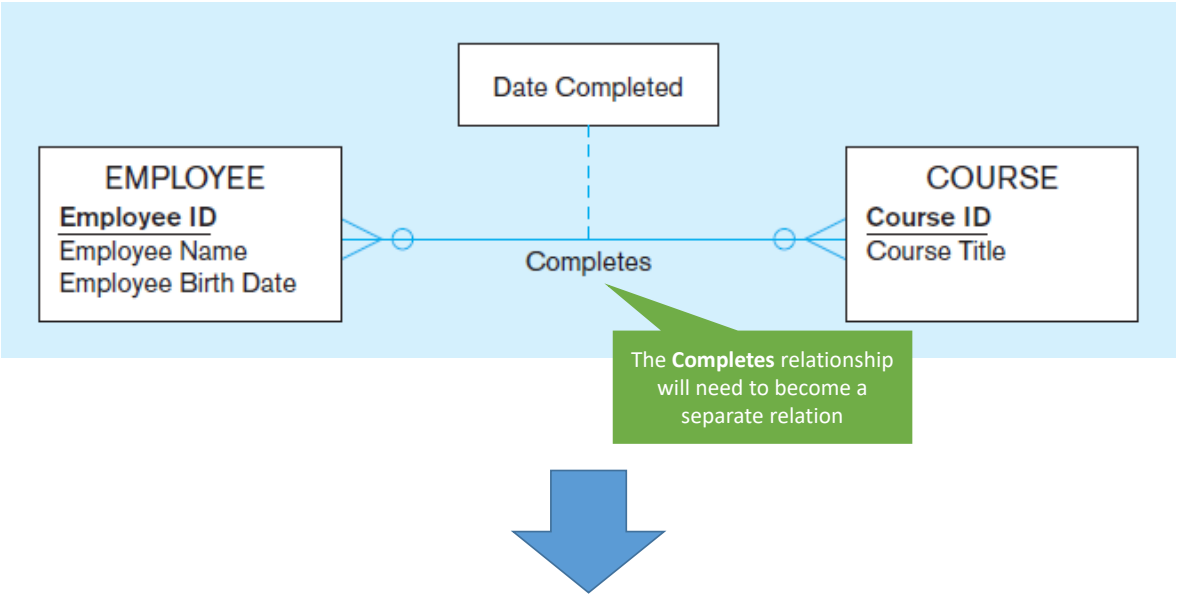


1:M Relationship mapped with FK on the many side



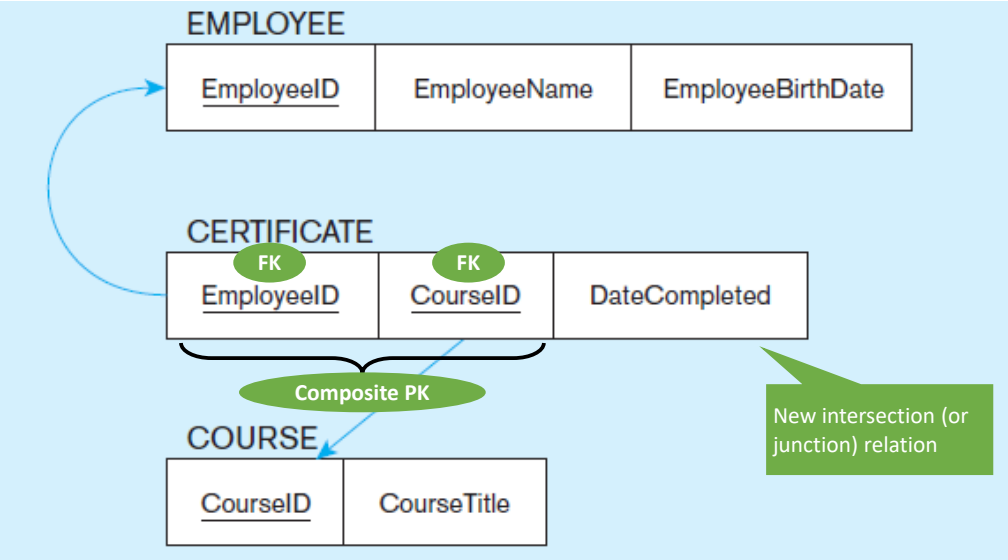
M:N relations results in the creation of a new relation with the PKs of the two entities as its PK (1/2)

Completes relationship (M:N)



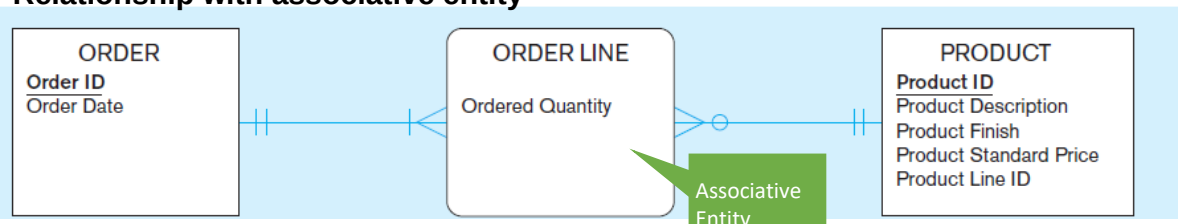
M:N relations results in the creation of a new relation with the PKs of the two entities as its PK (2/2)

Three resulting relations

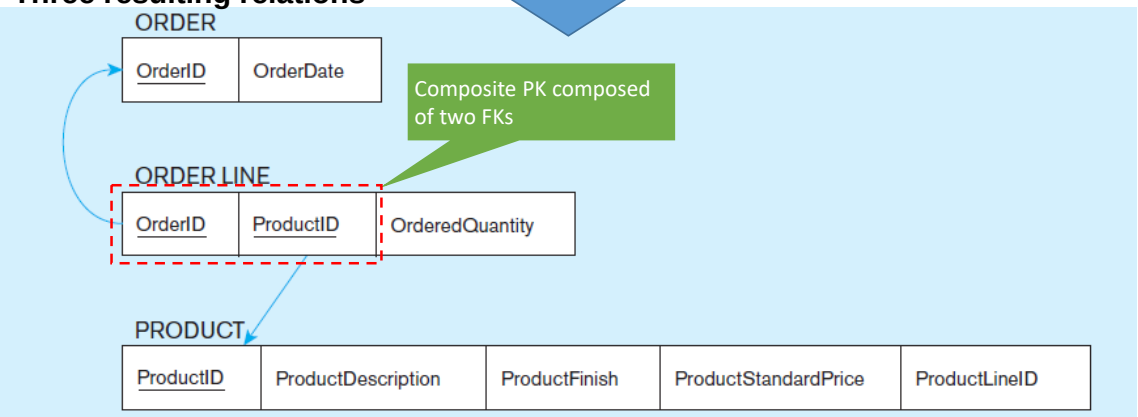


Mapping associative entities to relational databases is often identical to mapping M:N relationships

Relationship with associative entity

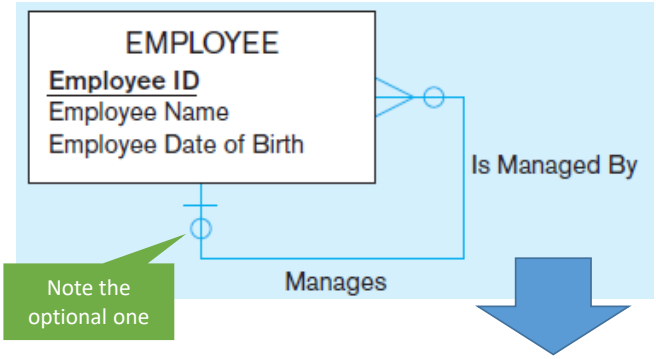


Three resulting relations

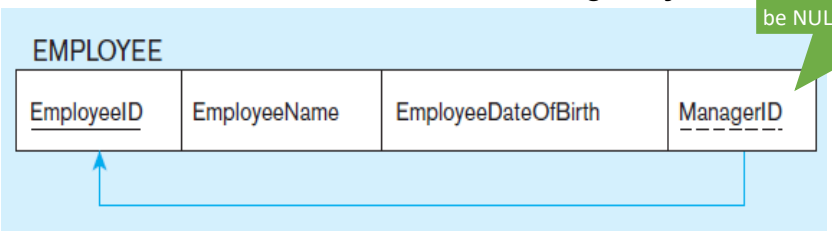


Unary 1:N relationship can be used to store a hierarchy… in this case supervisor has subordinates, and they in turn could manage others

EMPLOYEE entity with unary 1:N relationship



EMPLOYEE relation with recursive foreign key



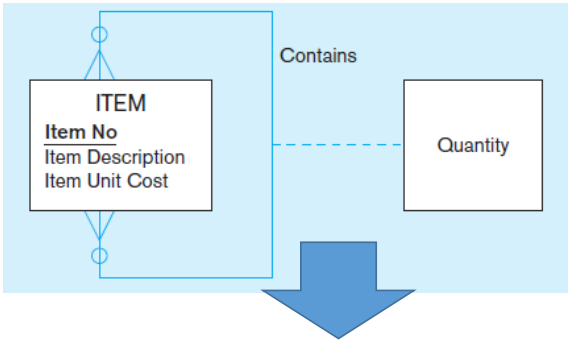
Foreign key (FK) points at the Primary key (PK) in the same table … a **recursive** key

As it is optional ManagerID could be NULL

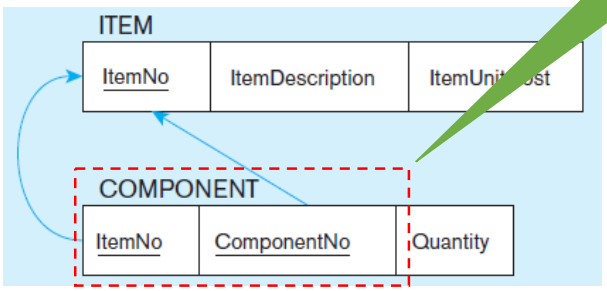
Unary M:N relationship requires two relations

- One for the entity
- One for the associative relation

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



ITEM and COMPONENT relations



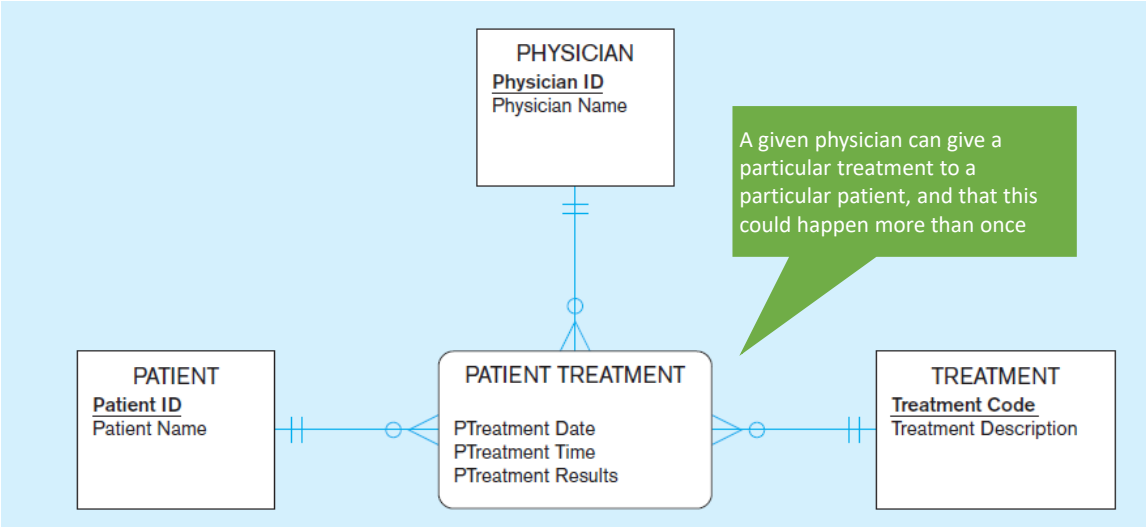
PK has two attributes, both taken from the PK of the ITEM entity

Also example of representing hierarchy

- An item may have sub-items, which could in turn have other sub-items, etc.
- Different from employees example
  - Given employee can have only one direct manager (1:N)
  - A component could be part of many other items (M:N)

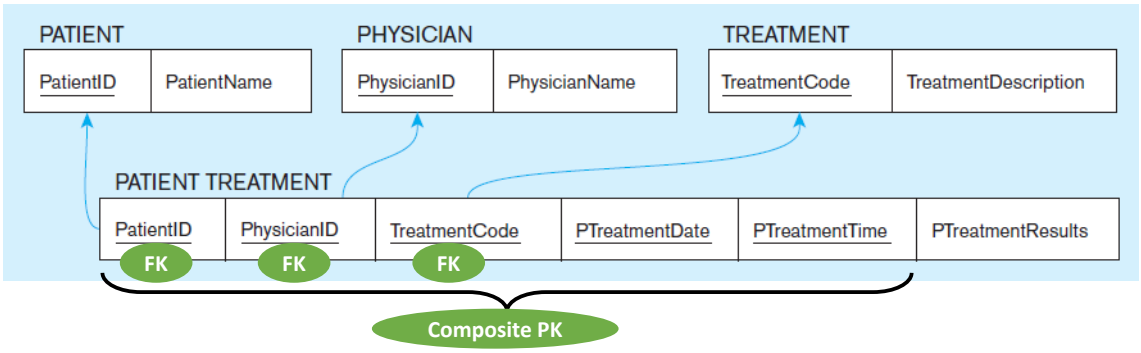
Mapping **Ternary (and n-ary) Relationships** requires one relation for each entity and one for the associative entity (1/2)

PATIENT TREATMENT Ternary relationship with associative entity



Mapping **Ternary (and n-ary) Relationships** requires one relation for each entity and one for the associative entity (2/2)

Mapping the ternary relationship PATIENT TREATMENT



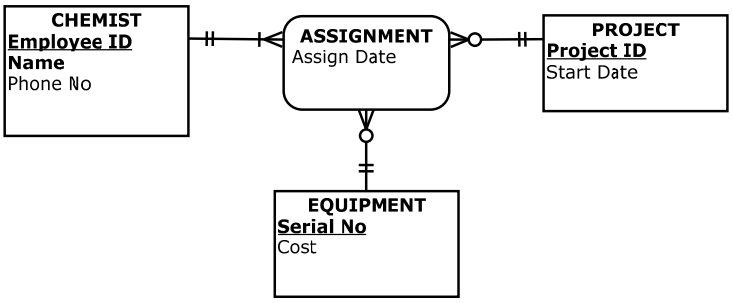
Remember that the **PK MUST be unique**… this is why treatment date and time are included in the composite PK

But this makes a very cumbersome key… it would be better to create a surrogate key like TreatmentID

In-class exercise: ERD to Schema

10-minute breakout

- The following ERD represents a data model for tracking the allocation of laboratory equipment to chemists working on projects.



- Convert the ERD into a set of relational schemas

Use this format (italics to show FKs) ORDER **OrderID** OrderDate *CustomerID*

Which is equivalent to

- Capture your answer on a PowerPoint page or Document
- Submit to https://forms.gle/ts1nwV3m58WXRqRX9 One submission per breakout team

¶Components of Relational Data Model

¶Converting ERD into Databases

¶Database Normalization

¶Hands On

¶For next time

This relation captures employee attributes along with their training history

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

Is this a relation? → Yes: Unique rows and no multivalued attributes

What's the primary key? → Composite: EmpID, CourseTitle

This relation has the potential of suffering from insertion anomalies

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

Insertion anomalies since we cannot insert information on

- A new employee without having an employee taking a class (otherwise CourseTitle would be NULL)
- A new course without an employee taking it (otherwise EmpID would be NULL)

It can also suffer from **deletion anomalies . . .**

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

Deletion anomaly if we remove employee 140, we lose information about the existence of a Tax Acc class

. . . as well as **modification anomalies**

EMPLOYEE2

EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

Modification anomaly since giving a salary increase to employee 100 forces us to update multiple records i.e. there are opportunities for data inconsistencies

We get the anomalies because there are two themes (entity types) in this relation. This results in **data duplication** and an unnecessary dependency between the entities

While EMPLOYEE2 is a relation . . . But it is **NOT** a **Well-Structured Relation**

EMPLOYEE2

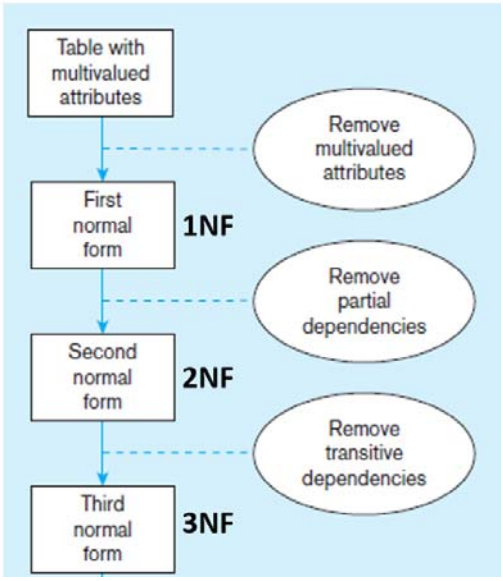
EmpID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

Fields pertaining to employees

Fields pertaining to courses

Rule of thumb: A table should not pertain to more than one entity type

Data Normalization improves a logical design to *avoid unnecessary duplication of data*



3NF generally considered to be sufficiently well-structured relations

A **Well-Structured Relation** is one that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies (anomalies)

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

The **normalization process** involves decomposing relations with anomalies to produce smaller, well-structured relations

This table is NOT a relation

Violating the principle of "no multivalued attributes". It's as if OrderID 1006 has "ProductID" equaling 7, 5, and 4. You can't have that in a true relation

Invoice data

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2015	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

Issues
- OrderID blank in several rows.
- Don't explicitly see customer fields or order date for the 2nd, 3rd, and 5th rows. You might infer the values if this was a spreadsheet. This is not sufficient for a database.

Eliminating multivalued attributes creates a relation in First Normal Form (1NF)

INVOICE relation with unique rows and no multivalued attributes (in 1NF)

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2015	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

Gaps filled

This is a relation, but not a well-structured one . . .
Note: the extensive data duplication (will cause anomalies)

***All relations* are in 1st Normal Form (1NF)**

Relation in First Normal Form (1NF) can still suffer from anomalies*

INVOICE relation with unique rows and no multivalued attributes (in 1NF)

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2015	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

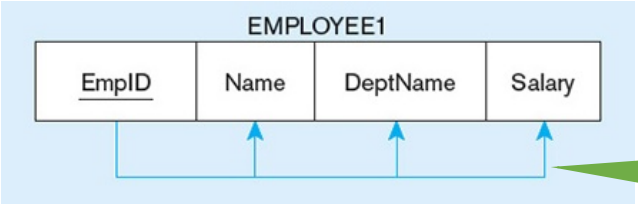
Fields pertaining to Order | **Fields pertaining to Customer** | **Fields pertaining to Product**

These anomalies exist because there are multiple themes (entity types) in one relation… results in duplication and unnecessary dependency between the entities

* It can suffer from insertion, deletion, and update anomalies just like the EMPLOYEES2 relation we already saw

Functional Dependence exists when an attribute uniquely determines the value of other attributes

Dependency Diagram for EMPLOYEE1



An employee's name, department and salary are **functionally dependent** on his/her EmpID. That is, there can only be one name, department, and salary for each EmpID

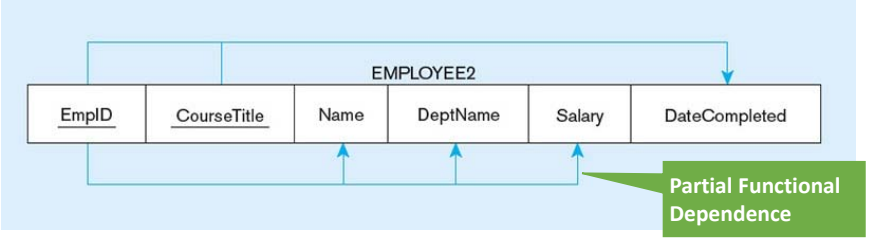
Functional dependencies also written as follows

Determinant		***Dependents***
EmpID	→	**Name, DeptName, Salary**
VIN	→	**Make, Model, Colour**
ISBN	→	**Title, FirstAuthorName, Publisher**
EmpID, CourseID	→	**DateCompleted**

Finding functional dependencies is also a way of finding **candidate PKs**

Partial Functional Dependence is associated with data duplication and data anomalies

Dependency Diagram for EMPLOYEE2

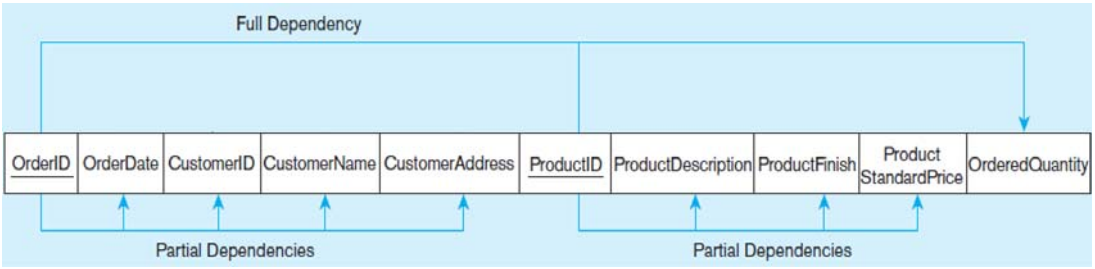


Functional Dependency	Implication
EmpID, CourseTitle ➔ DateCompleted	Only EmpID, CourseTitle is a candidate key and therefore must be the PK for this relation
EmpID ➔ Name, DeptName, Salary	Three attributes are determined by only part of the PK for the whole relation This is called **partial functional dependence**

Second Normal Form (2NF) is 1NF PLUS every non-key attribute FULLY functionally dependent on ENTIRE PK

(i.e. no partial functional dependencies)

Functional dependency diagram for INVOICE

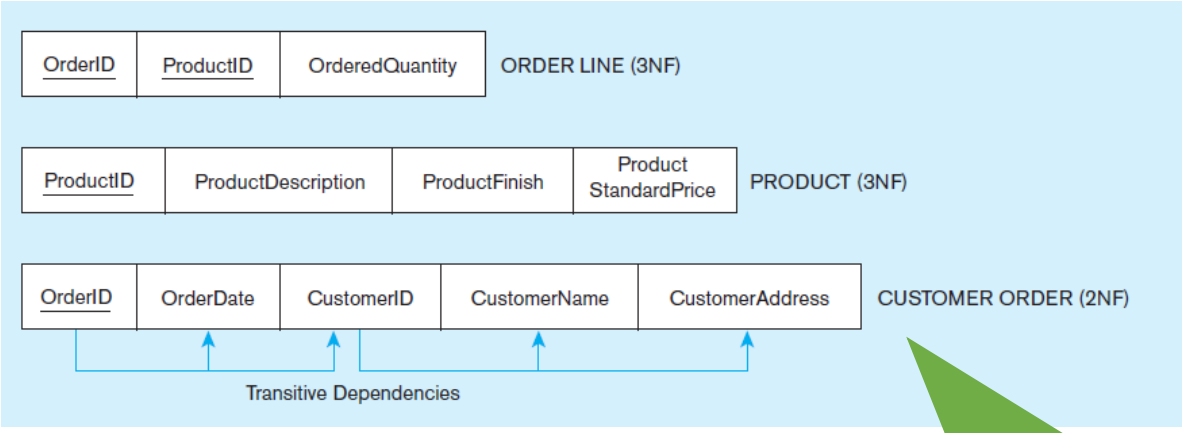


OrderID, ProductID ➔ OrderQuantity

OrderID ➔ OrderDate, CustomerID, CustomerName, CustomerAddress
ProductID ➔ ProductDescription, ProductFinish, ProductStandardPrice

Two **partial functional dependencies**.
So, this relation is **NOT in 2NF**

Removing partial dependencies by splitting table into three (customer order, product, order line) gets us to 2NF

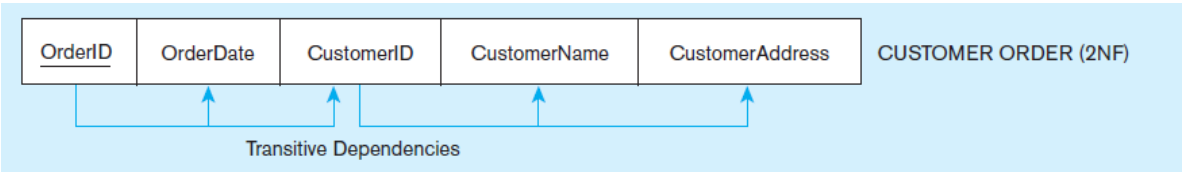


No **partial functional dependencies**. So, this relation is **in 2NF**

Partial dependencies are removed, but there are still transitive dependencies

Third Normal Form (3NF) is 2NF PLUS no transitive dependencies (functional dependencies on non-PK attributes)

Functional dependency diagram for CUSTOMER ORDER



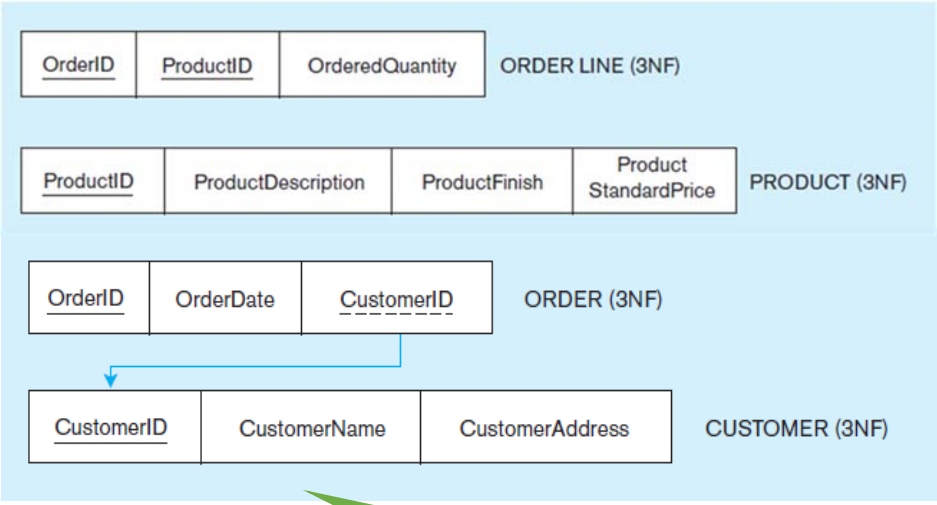
OrderID ➔ CustomerID ➔ CustomerName
OrderID ➔ CustomerID ➔ CustomerAddress

There are still **transitive dependencies**. **So, this relation is not in 3NF**

Called **transitive**, because PK is determinant for another attribute, which in turn is a determinant for a third attribute

Still have data redundancy since we would have to store customer name and address for each order

We reach Third Normal form (3NF)… by splitting out the customer attributes*

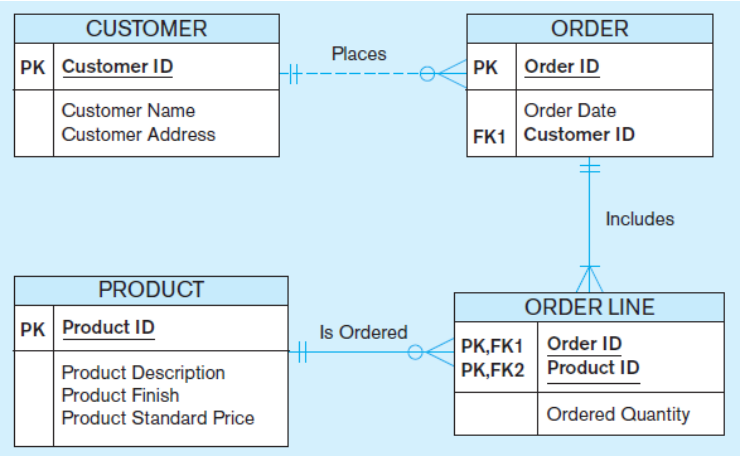


Transitive functional dependencies removed. So, this relation is in **3NF**

* More generally, non-key determinant with transitive dependencies goes into a new table; non-key determinant becomes PK in new table and stays as FK in old table

Finally we reach Third Normal form (3NF)… now we have a well-structured database

Normalization, yielded four separate relations where there was initially only one



Original table
- Had three separate entities wrapped into single relation (customer, product, and order)
- Hid M:N relationship between orders and products

If we had started with a thorough ERD, we would not have wound up with one big table containing multiple themes

Illustrates danger of going straight to logical database design without performing conceptual model analysis…. but it happens often

In-class exercise: Normalization

15-minute breakout

¶ The following is a sample of a spreadsheet used to track traffic violation at a college

Parking Ticket Table									
St ID	L Name	F Name	Phone No	St Lic	Lic No	Ticket #	Date	Code	Fine
38249	Brown	Thomas	111-7804	FL	BRY 123	15634	10/17/2018	2	$25
						16017	11/13/2018	1	$15
82453	Green	Sally	391-1689	AL	TRE 141	14987	10/05/2018	3	$100
						16293	11/18/2018	1	$15
						17892	12/13/2018	2	$25

¶ Note that

- Violation code 1 – expired traffic meter
- Violation code 2 – no parking permit
- Violation code 3 – accessible violation

¶ Develop a set of relations in 3NF

Use this format: ORDER **OrderID** OrderDate *CustomerID*

¶ Capture your answer on a PowerPoint page

¶ Submit your solution to https://forms.gle/Kr9TjbQGhwwAGgwMA One submission per breakout team

¶Components of Relational Data Model

¶Converting ERD into Database

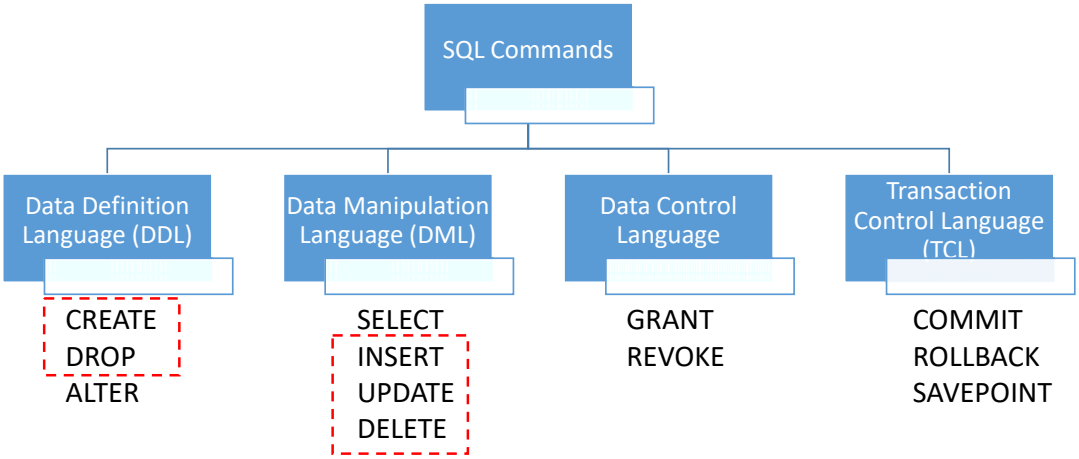
¶Database Normalization

¶Hands On

- SQL – Data Definition Language (DDL)
- SQL – INSERT, UPDATE and DELETE queries

¶For next time

Basics of SQL DDL as well as INSERT, UPDATE, and DELETE queries covered in today's hands on scripts and videos



In computer programming, create, read, update, and delete (**CRUD**) are the four basic functions of persistent storage. This is how SQL DML queries map to CRUD

CRUD	SQL
C - Create	Insert
R - Read	Select
U - Update	Update
D - Delete	Delete

The script that creates the AP database (part 1)

```
-- create the database
DROP DATABASE IF EXISTS ap;
CREATE DATABASE ap;

-- select the database
USE ap;

-- create the tables
CREATE TABLE general_ledger_accounts
(
  account_number        INT             PRIMARY KEY,
  account_description   VARCHAR(50)     UNIQUE
);

CREATE TABLE terms
(
  terms_id              INT             PRIMARY KEY
                                        AUTO_INCREMENT,
  terms_description     VARCHAR(50)     NOT NULL,
  terms_due_days        INT             NOT NULL
```

The script that creates the AP database (part 2)

```
CREATE TABLE vendors
(
  vendor_id                     INT             PRIMARY KEY
                                                AUTO_INCREMENT,
  vendor_name                   VARCHAR(50)     NOT NULL
                                                UNIQUE,
  vendor_address1               VARCHAR(50),
  vendor_address2               VARCHAR(50),
  vendor_city                   VARCHAR(50)     NOT NULL,
  vendor_state                  CHAR(2)         NOT NULL,
  vendor_zip_code               VARCHAR(20)     NOT NULL,
  vendor_phone                  VARCHAR(50),
  vendor_contact_last_name      VARCHAR(50),
  vendor_contact_first_name     VARCHAR(50),
  default_terms_id              INT             NOT NULL,
  default_account_number        INT             NOT NULL,
  CONSTRAINT vendors_fk_terms
    FOREIGN KEY (default_terms_id)
    REFERENCES terms (terms_id),
  CONSTRAINT vendors_fk_accounts
    FOREIGN KEY (default_account_number)
    REFERENCES general_ledger_accounts (account_number)
);
```


The script that creates the AP database (part 3)

```
CREATE TABLE invoices
(
  invoice_id            INT             PRIMARY KEY
                                        AUTO_INCREMENT,
  vendor_id             INT             NOT NULL,
  invoice_number        VARCHAR(50)     NOT NULL,
  invoice_date          DATE            NOT NULL,
  invoice_total         DECIMAL(9,2)    NOT NULL,
  payment_total         DECIMAL(9,2)    NOT NULL      DEFAULT 0,
  credit_total          DECIMAL(9,2)    NOT NULL      DEFAULT 0,
  terms_id              INT             NOT NULL,
  invoice_due_date      DATE            NOT NULL,
  payment_date          DATE,
  CONSTRAINT invoices_fk_vendors
    FOREIGN KEY (vendor_id)
    REFERENCES vendors (vendor_id),
  CONSTRAINT invoices_fk_terms
    FOREIGN KEY (terms_id)
    REFERENCES terms (terms_id)
);
```

Variety of datatypes used in examples. See full details of MySQL datatypes here https://dev.mysql.com/doc/refman/8.0/en/data-types.html

11.1 Data Type Overview
11.2 Numeric Types
11.3 Date and Time Types
11.4 String Types
11.5 Spatial Data Types
11.6 The JSON Data Type

The script that creates the AP database (part 4)

```
CREATE TABLE invoice_line_items
(
  invoice_id              INT            NOT NULL,
  invoice_sequence        INT            NOT NULL,
  account_number          INT            NOT NULL,
  line_item_amount        DECIMAL(9,2)   NOT NULL,
  line_item_description   VARCHAR(100)   NOT NULL,
  CONSTRAINT line_items_pk
    PRIMARY KEY (invoice_id, invoice_sequence),
  CONSTRAINT line_items_fk_invoices
    FOREIGN KEY (invoice_id)
    REFERENCES invoices (invoice_id),
  CONSTRAINT line_items_fk_acounts
    FOREIGN KEY (account_number)
    REFERENCES general_ledger_accounts (account_number)
);

-- create an index
CREATE INDEX invoices_invoice_date_ix
  ON invoices (invoice_date DESC);
```


Insert a single row without using a column list

```
INSERT INTO invoices VALUES
(115, 97, '456789', '2018-08-01', 8344.50, 0, 0, 1,
'2018-08-31', NULL)

(1 row affected)
```

Insert a single row using a column list

```
INSERT INTO invoices
    (vendor_id, invoice_number, invoice_total, terms_id,
    invoice_date, invoice_due_date)
VALUES
    (97, '456789', 8344.50, 1, '2018-08-01',
    '2018-08-31')

(1 row affected)
```

Update one column for multiple rows

```
UPDATE invoices
SET terms_id = 1
WHERE vendor_id = 95

(6 rows affected)
```

Update one column for one row

```
UPDATE invoices
SET credit_total = credit_total + 100
WHERE invoice_number = '97/522'

(1 row affected)
```

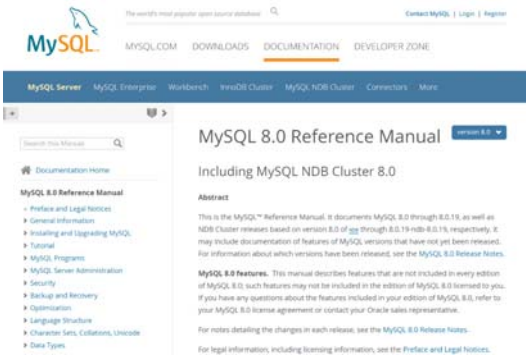

Delete one row

```
DELETE FROM general_ledger_accounts
WHERE account_number = 306

(1 row affected)
```

Delete one row using a compound condition

```
DELETE FROM invoice_line_items
WHERE invoice_id = 78 AND invoice_sequence = 2

(1 row affected)
```

IMPORTANT
See 'hands on' scripts and videos for more examples… you will need those examples to complete the homework



¶ Impossible to cover all aspects of SQL in a short course… not even all the options available for the statements we do use. It is worthwhile getting comfortable using the documentation
- https://dev.mysql.com/doc/
- https://dev.mysql.com/doc/refman/8.0/en/

¶ Still having trouble with MySQL on your system? How many of you?
- Your root password for the MySQL server should be 'sesame80'
- See p46-47 in free pdf of chapter 2 of Murach's MySQL book for guide to starting MySQL server if it is not running
- LBS IT can provide support in getting you going (Philip Dainton-Smith?)

¶Components of Relational Data Model

¶Converting ERD into Database

¶Database Normalization

¶Hands On

¶For next time

For next time

- Hands on activities (see Canvas page for session 2)
- Attend Thursday workshop (optional) if you think you might need more support before completing the homework
- Check out readings for this session and next
- Submit Homework Assignment #2 (team)
  - Covers more SQL queries and aspects of logical database design (details on Canvas)
  - Submit via Canvas by 10pm on Sunday

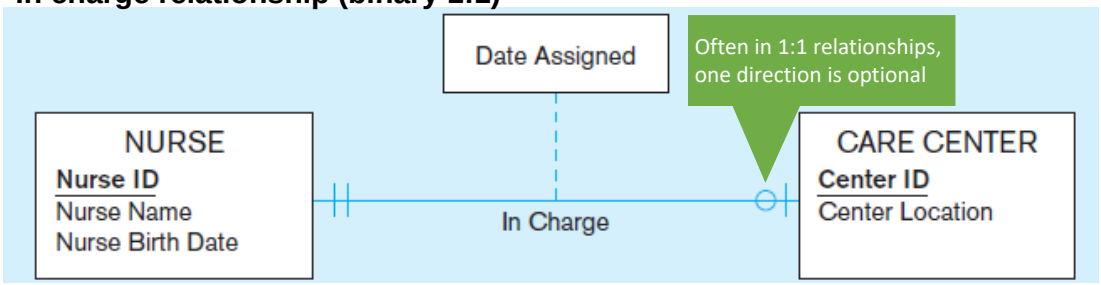
Appendix

Additional Examples of Converting ERDs into Database

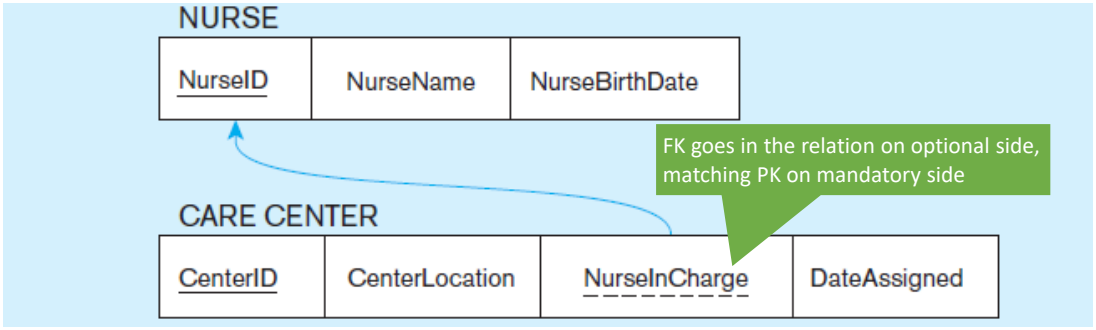
Hint: Pages on Supertype/subtype might help with homework assignment

With a **1:1 relationship** PK on mandatory side becomes a FK on optional side

In charge relationship (binary 1:1)



Resulting relations

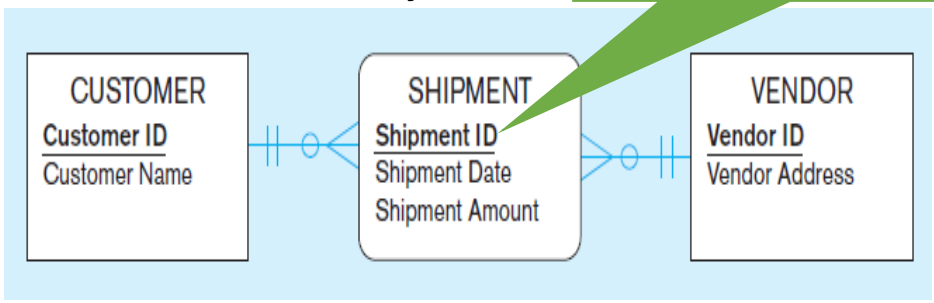


Mapping **associative entity with assigned identifier** is slightly different (1/2)

In this case, associative entity has its own identifier

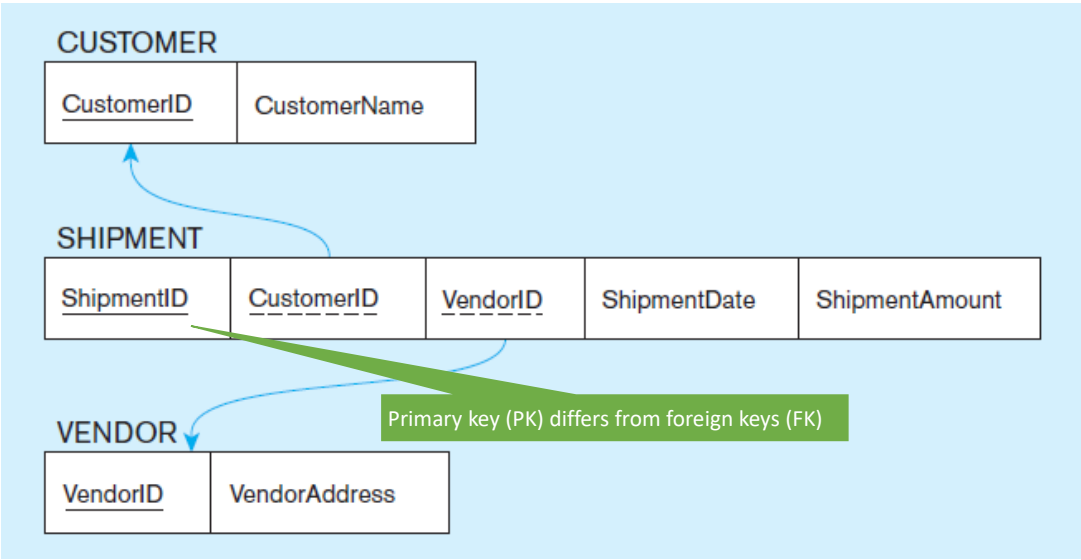
Happens if
- It is natural and familiar to end-users
- Default identifier may not be unique

SHIPMENT associative entity

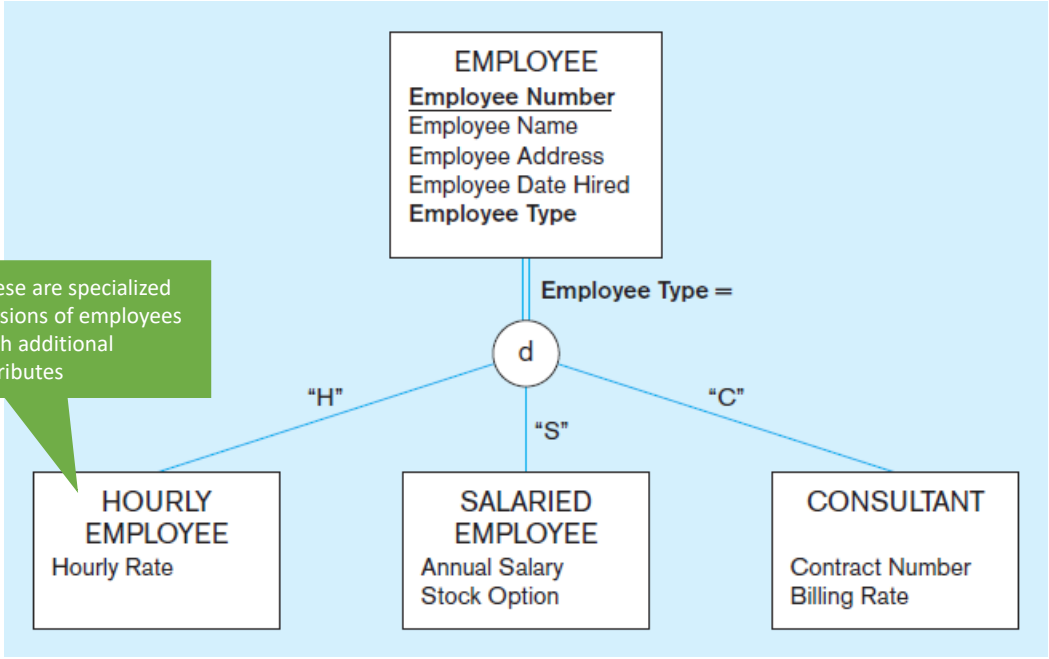


Mapping **associative entity with assigned identifier** is slightly different (2/2)

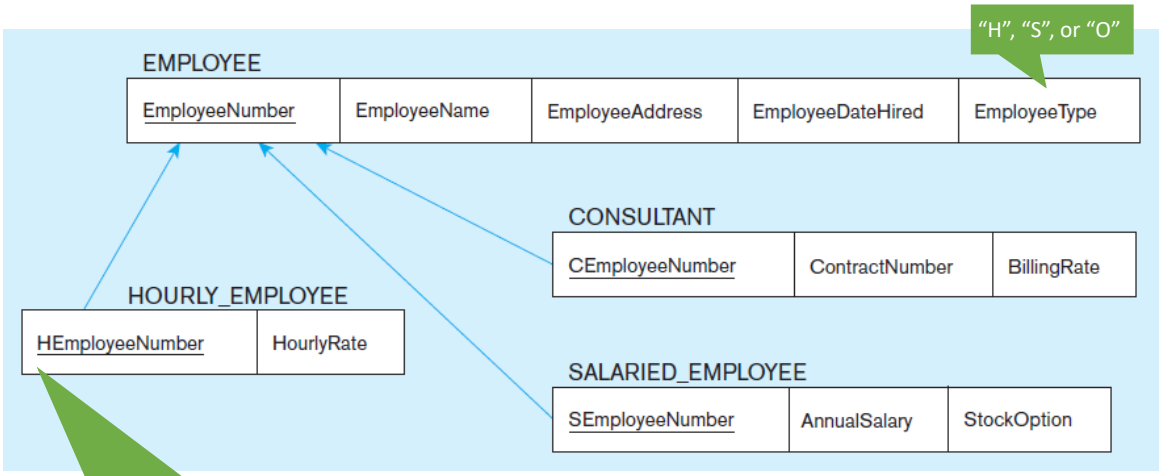
Three resulting relations



Supertype/subtype relationships can be modeled in ERDs



Supertype/subtypes are implemented in databases as one-to-one relationships



PK of the subtype relation is also a FK to the supertype relation.

If you see the entire PK is a FK in a database it implies a supertype-subtype relationship.