L13: Normalization

CS3200 Database design (sp18 s2)

https://course.ccs.neu.edu/cs3200sp18s2/

2/26/2018

Announcements!

- Keep bringing your name plates ☺
- Page Numbers now bigger (may change slightly)
- Exam 1 discussion: solutions posted on BB, questions on grading: Piazza, send to instructors only
- Project part 1 discussion likely Thursday in class
- Outline
  - Continue with ER modeling and Normalization
  - Transactions after Spring Break

Practice

Exercise 1

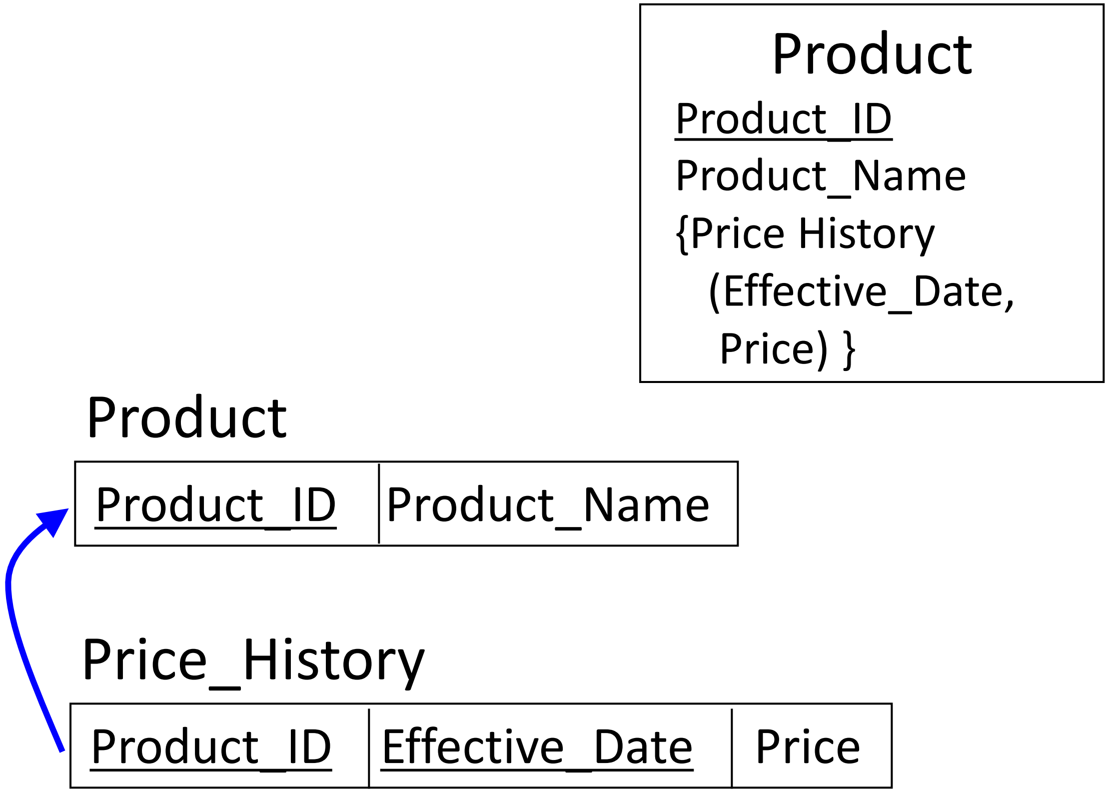
- Create a relational schema to represent the following E-R Diagram:

Product

<u>Product_ID</u>
Product_Name
{Price History
(Effective_Date,
Price) }

Exercise 1

- Create a relational schema to represent the following E-R Diagram:



Exercise 2

- Create a relational schema to represent the following E-R Diagram:

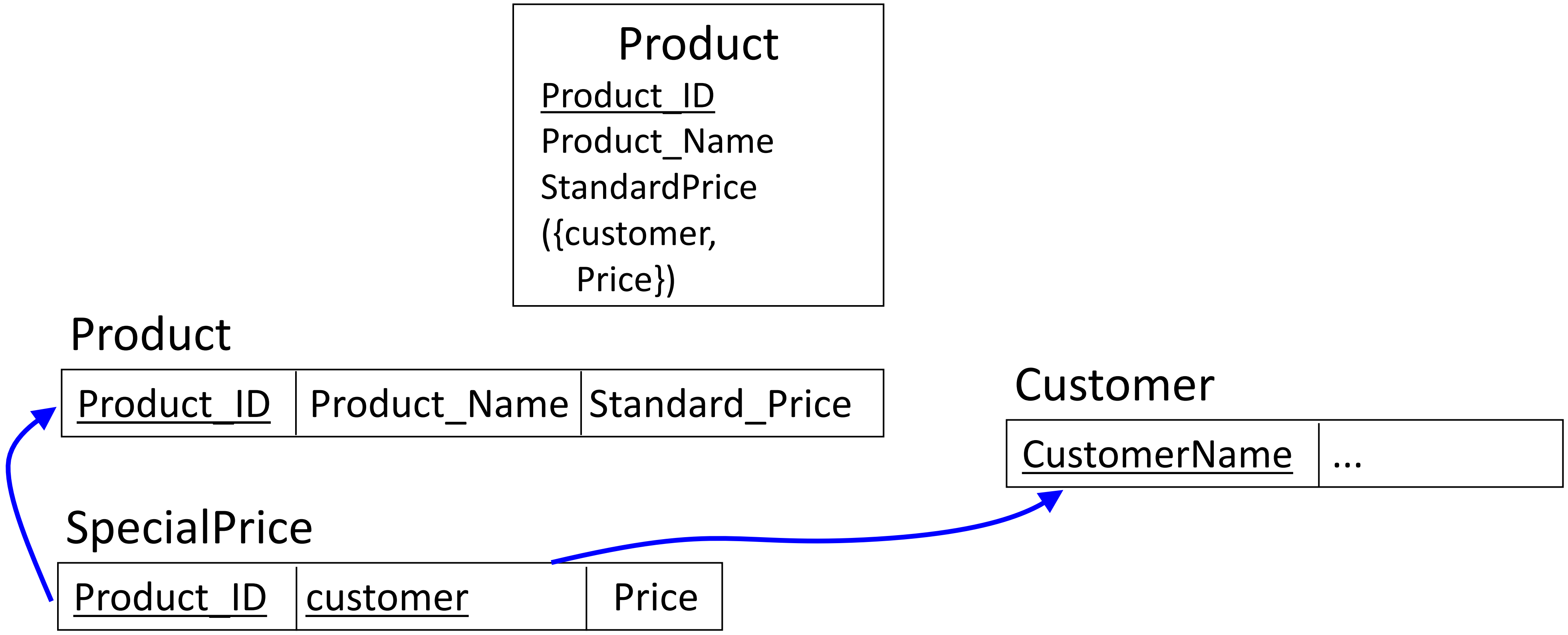
Product
<u>Product_ID</u>
Product_Name
StandardPrice
({customer,
Price})

Product

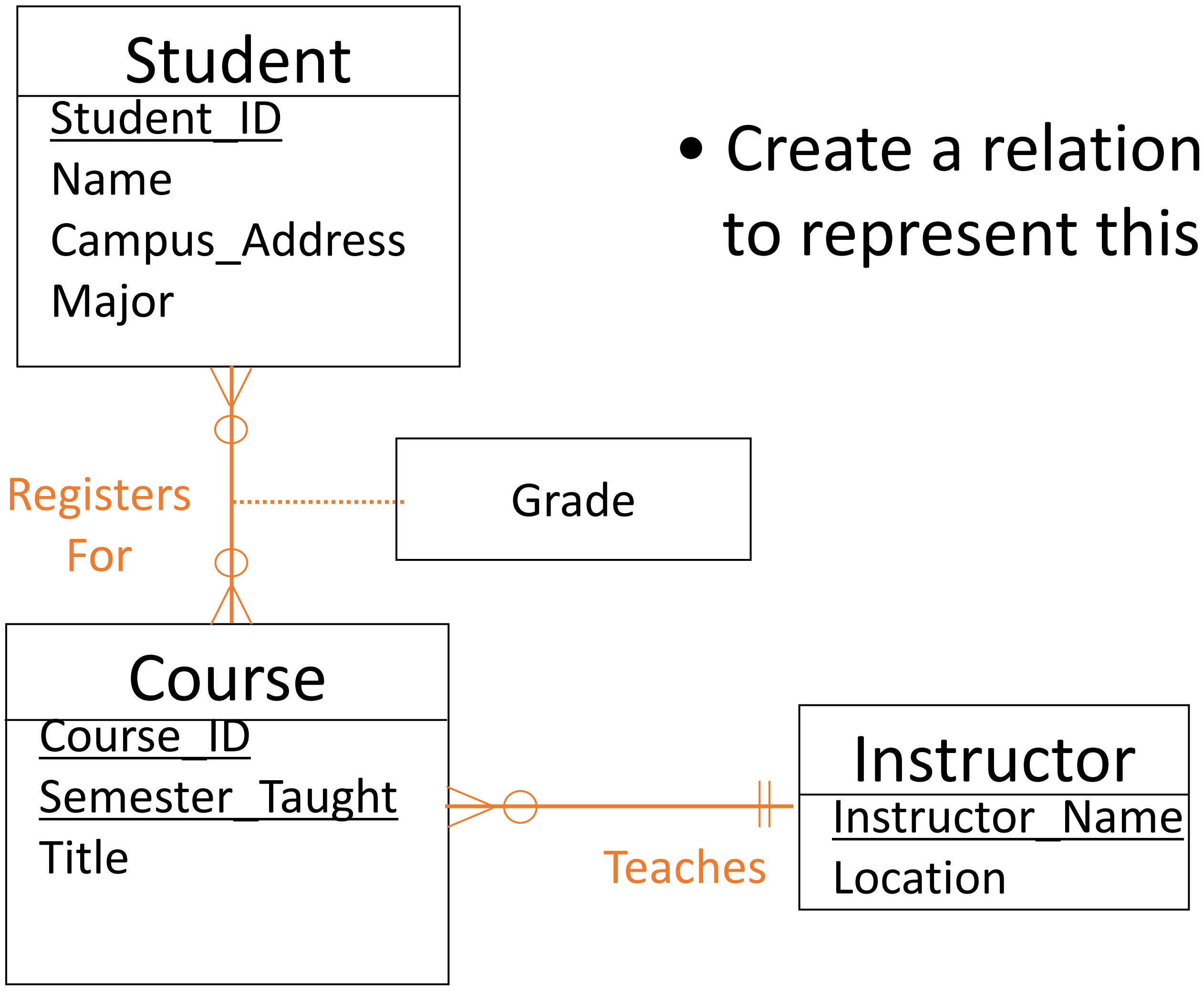
<u>Product_ID</u>	Product_Name

Exercise 2

- Create a relational schema to represent the following E-R Diagram:

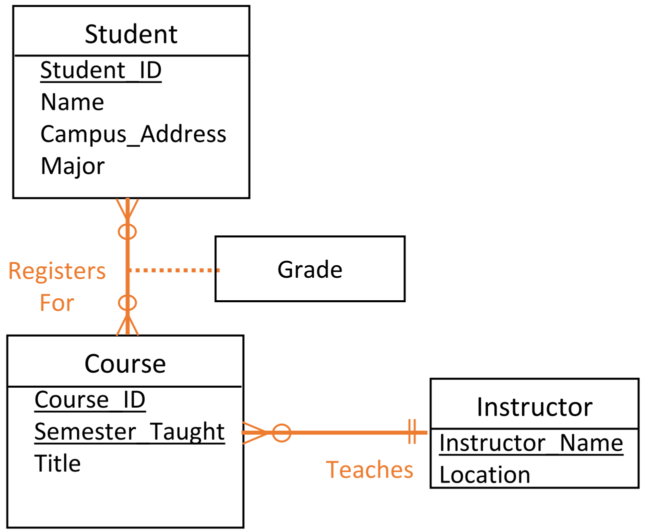


Exercise 3



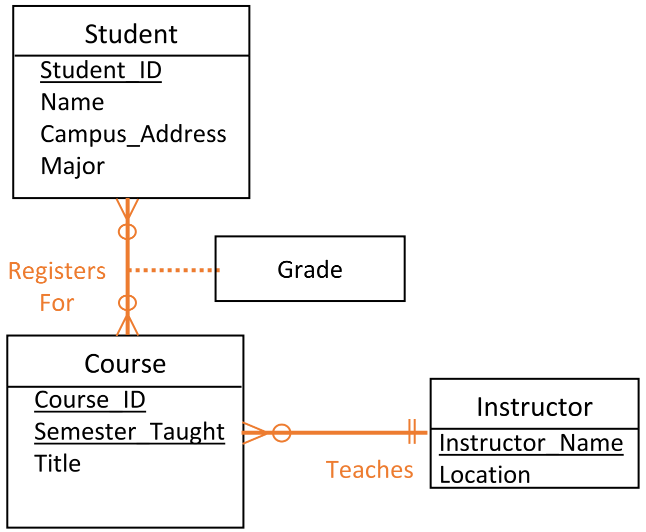
- Create a relational schema to represent this E-R Diagram:

Exercise 3



Student
Student_ID
Name
Campus_Address
Major
Registers
For
Grade
Course
Course_ID
Semester_Taught
Title
Teaches
Instructor
Instructor_Name
Location

Exercise 3: Solution



Student

Student_ID	Name	Campus_Address	Major

Course_Registration

@Student_ID	@Course_ID	@Semester_Taught	Grade

Course

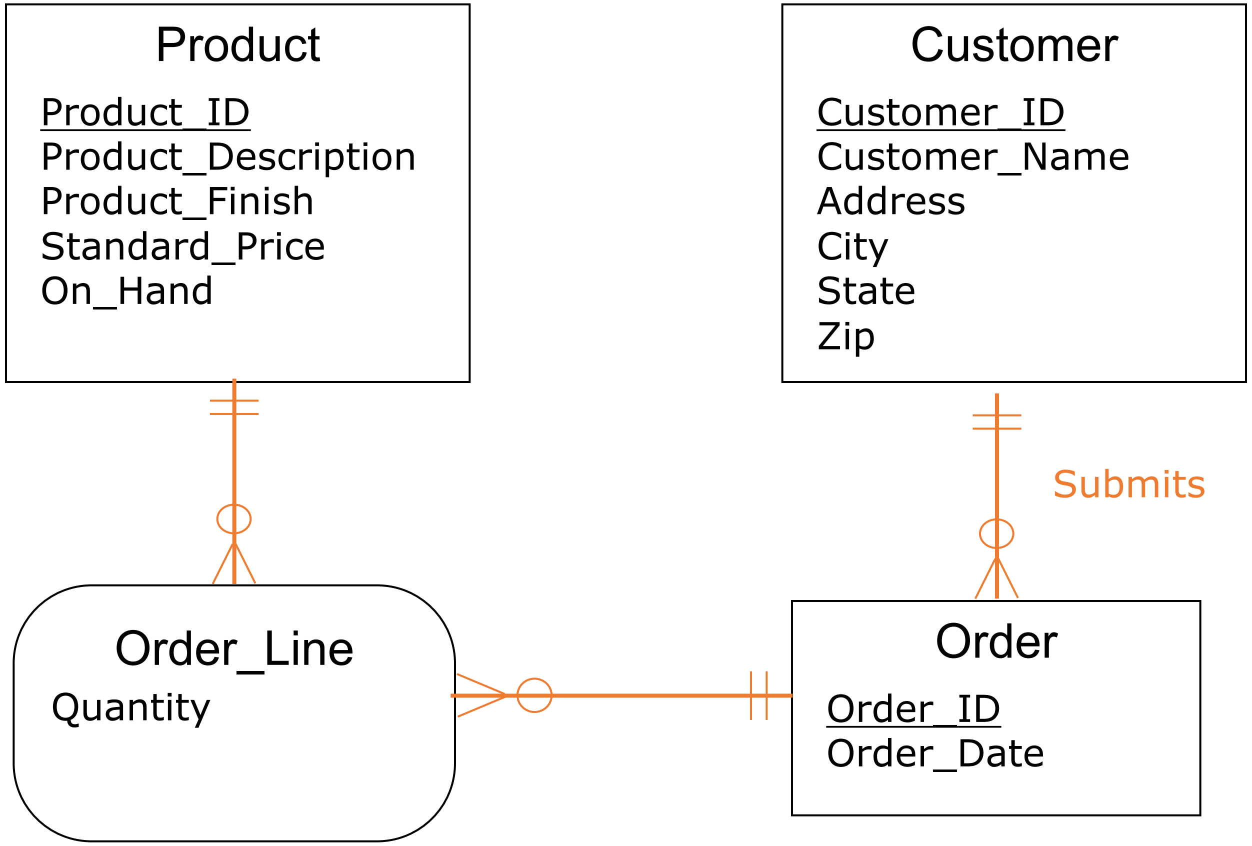
Course_ID	Semester_Taught	Title	@Instructor_Name

Instructor

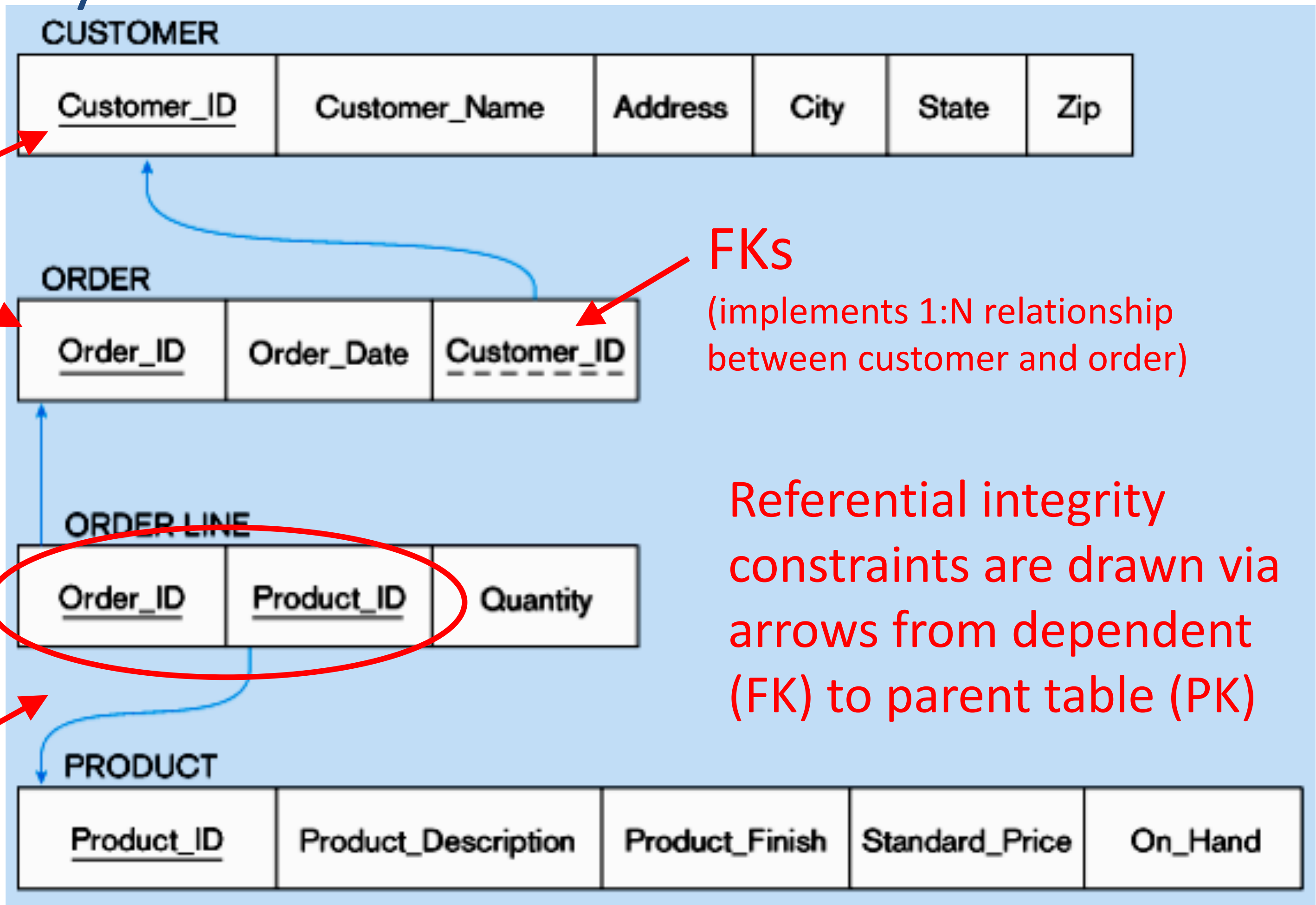
Instructor_Name	Location

don't forget:
"not null" constraint

Example: Pine Valley Furniture Company



Example: Pine Valley Furniture Referential Integrity Constraints



PKs

FKs

(implements 1:N relationship between customer and order)

Combined, these are a *composite primary key* (uniquely identifies the order line)…individually they are *foreign keys* (together implement M:N relationship between order and product)

Referential integrity constraints are drawn via arrows from dependent (FK) to parent table (PK)

Source: Compare with Fig 4-30: Hoffer, Ramesh, Topi, "Modern database management," 10th ed, 2010.

1. Normal forms and Functional Dependencies

Design Theory

- Design theory is about how to represent your data to avoid anomalies.

- It is a mostly mechanical process
  - Tools can carry out routine portions

- We have a notebook implementing all algorithms!
    - We'll play with it in the activities!

Data Normalization

- Data normalization is the process of decomposing relations with anomalies to produce smaller, well-structured relations

- Goals of normalization include:
  - Minimize data redundancy
  - Simplifying the enforcement of referential integrity constraints
  - Simplify data maintenance (inserts, updates, deletes)
  - Improve representation model to match "the real world"

Well-Structured Relations

- A well-structured relation contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies

- Anomalies are errors or inconsistencies that may result when a user attempts to update a table that contains redundant data.

- Three types of 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

Normal Forms

Normal Form: a state of a relation that results from applying simple rules regarding FDs to that relation

- 1st Normal Form (1NF) = All tables are flat

- 2nd Normal Form = not used anymore
  - no more "partial FDs" (those are part of the "bad" FDs)

- **3rd Normal Form (3NF)**
  - **no more transitive FDs (also "bad")**

- **Boyce-Codd Normal Form (BCNF)**
  - **every determiniant is a candidate key**

DB designs based on FDs (*functional dependencies*), intended to prevent data ***anomalies***

Our focus next

- 4^{th}: any multivalued dependencies have been removed (see textbook)
- 5^{th}: any remaining anomalies have been removed (see text book)

1st Normal Form (1NF)

Student	**Courses**
Mary	{CS3200, CS4240}
Joe	{CS3200, CS4240}
…	…

Violates 1NF.

1st Normal Form (1NF)

Student	Courses
Mary	{CS3200, CS4240}
Joe	{CS3200, CS4240}
…	…

Violates 1NF.

Student	Courses
Mary	CS3200
Mary	CS4240
Joe	CS3200
Joe	CS4240

In 1^{st} NF

1NF Constraint: Types must be atomic!

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	**Course**	**Room**
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

If every course is in only one room, contains ***redundant*** information!

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	B12
Sam	CS3200	WVF20
..	..	..

If we update the room number for one tuple, we get inconsistent data = an ***update anomaly***

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

Student	**Course**	**Room**
…	…	…

If everyone drops the class, we lose what room the class is in! = a ***delete* anomaly**

Constraints Prevent (some) Anomalies in the Data

A poorly designed database causes ***anomalies***:

...	CS4240	B12

Student	**Course**	**Room**
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

Similarly, we can't reserve a room without students = an ***insert* anomaly**

Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CS3200
Joe	CS3200
Sam	CS3200
..	..

Course	Room
CS3200	WVF20
CS4240	B12

Is this form better?

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

Next: develop theory to understand why this design may be better **and** how to find this *decomposition*…

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Is This Table Well Structured?

EMPLOYEE2

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

- Does it contain anomalies?
  - Insertion: if an employee takes a new class we need to add duplicate data (Name, Dept_Name, Salary)
  - Deletion: If we remove employee 140, we lose information about the existence of a Tax Acc class
  - Modification: Giving a salary increase to employee 100 forces us to update multiple records

- Why do these anomalies exist?
  - Because there are <u>two themes (entity types) in one relation</u>. This results in duplication, and an unnecessary dependency between the entities

Normalizing Previous Employee/Class Table

Employee

Emp_ID	Name	Dept_Name	Salary
100	Margaret Simpson	Marketing	48000
140	Alan Beeton	Accounting	52000
110	Chris Lucero	Info Sys	43000
190	Lorenzo Davis	Finance	55000
150	Susan Martin	Marketing	42000

Course_Completion

Emp_ID	Course_ID	Date_Completed
100	1	6/19/2005
100	2	10/7/2004
140	3	12/8/2004
110	1	1/12/2004
110	4	4/22/2003
150	1	6/19/2005
150	5	8/12/2002

Course

Course_ID	Course_Title
1	SPSS
2	Surveys
3	Tax Acc
4	C++
5	Java

This seems more complicated

Why might this approach be superior to the previous one?

Functional Dependencies ("FDs")

Definition:

If two tuples agree on the attributes

$A_1, A_2, \ldots, A_n$

then they must also agree on the attributes

$B_1, B_2, \ldots, B_m$

Formally:

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

Functional Dependencies ("FDs")

Def: Let A,B be *sets* of attributes

We write A → B or say A ***functionally determines*** B if, for any tuples t_1 and t_2:

$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$

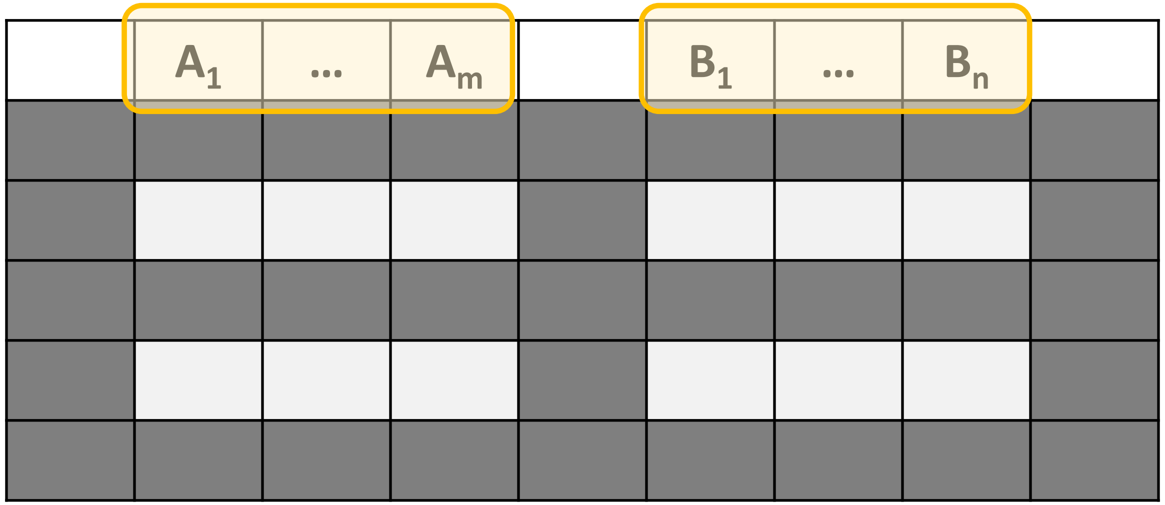
and we call A → B a **functional dependency**

A (determinant) →B (dependent)

A->B means that

"whenever two tuples agree on A then they agree on B."

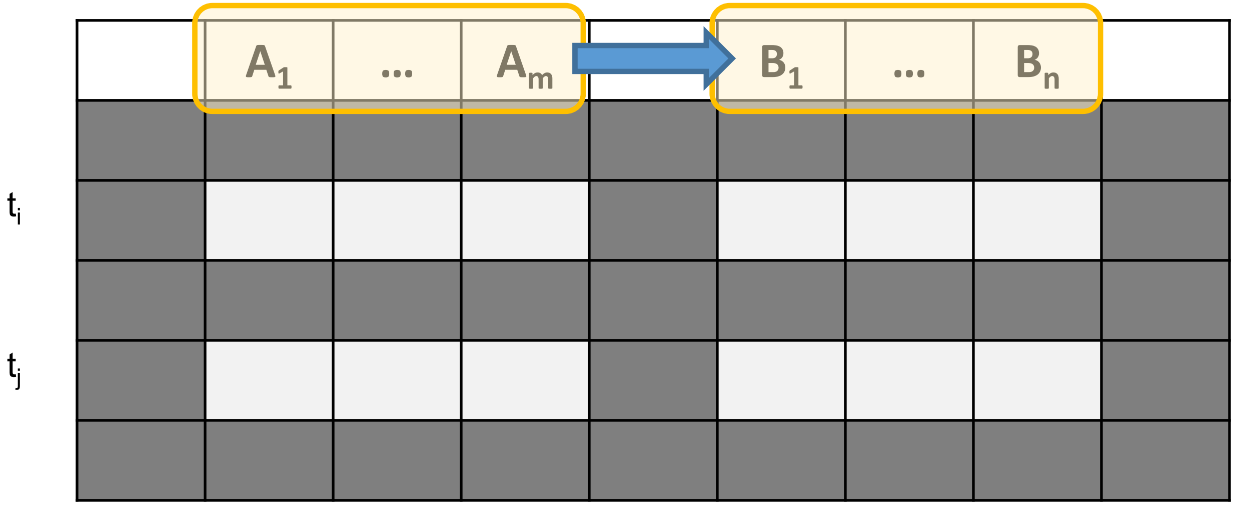
A Picture Of FDs



Defn (again):

Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

A Picture Of FDs

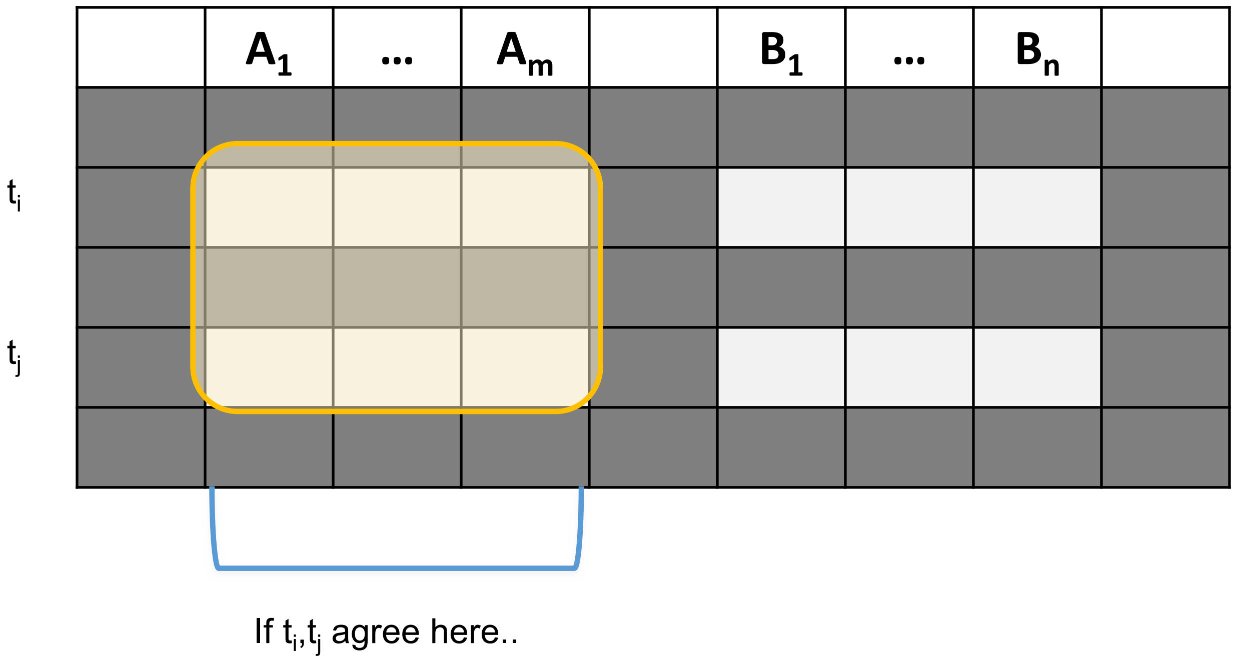


<u>Defn (again):</u>

Given attribute sets **A=**$\{A_1,\ldots,A_m\}$ and **B** = $\{B_1,\ldots B_n\}$ in **R**,

The ***functional dependency*** **A→ B on R** holds if for ***any*** t_i,t_j in R:

A Picture Of FDs



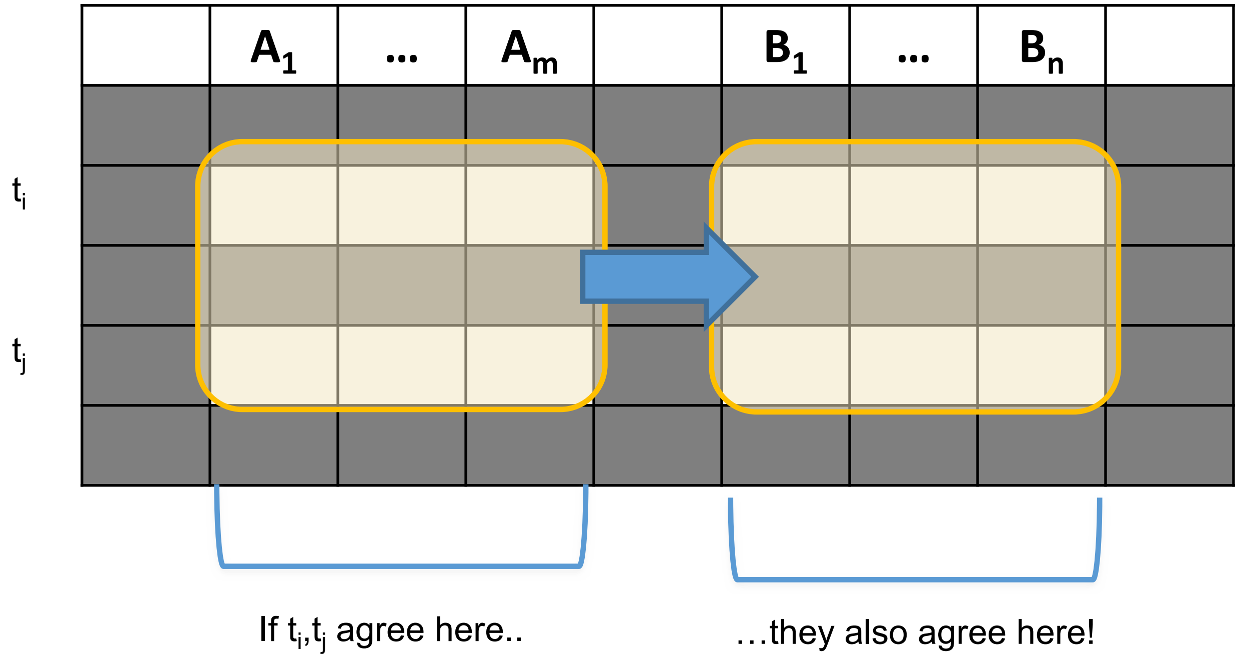
Defn (again):

Given attribute sets $A=\{A_1,\ldots,A_m\}$ and $B = \{B_1,\ldots B_n\}$ in R,

The *functional dependency* A→ B on R holds if for *any* t_i,t_j in R:

if $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

A Picture Of FDs



Defn (again):

Given attribute sets **A={A_1,…,A_m}** and **B = {B_1,…B_n}** in **R,**

The ***functional dependency*** **A→ B** on **R** holds if for ***any*** t_i,t_j in R:

if $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

then $t_i[B_1] = t_j[B_1]$ AND $t_i[B_2]=t_j[B_2]$ AND … AND $t_i[B_n] = t_j[B_n]$

FDs for Relational Schema Design

- High-level idea: why do we care about FDs?
  - Start with some relational schema
  - Find out its functional dependencies (FDs)
  - Use these to design a better schema
    - One which minimizes the possibility of anomalies

Functional Dependencies as Constraints

A **functional dependency** is a form of **constraint**

- *Holds* on some instances (but not others) – can check whether there are violations
- Part of the schema, helps define a *valid* instance

*Recall: an __instance__ of a schema is a multiset of tuples conforming to that schema, **i.e. a table***

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

Note: The FD {Course} -> {Room} ***holds on this instance***

Functional Dependencies as Constraints

Note that:

- You can check if an FD is **violated** by examining a single instance;
- However, you **cannot prove** that an FD is part of the schema by examining a single instance.
  - *This would require checking every valid instance*

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

However, cannot *prove* that the FD {Course} -> {Room} is ***part of the schema***

More Examples

An FD is a constraint which <u>holds</u>, or <u>does not hold</u> on an instance:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

EID → NAME

More Examples

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876 ←	Salesrep
E1111	Smith	9876 ←	Salesrep
E9999	Mary	1234	Lawyer

{Position} → {Phone}

More Examples

EmpID	Name	Phone	Position
E0045	Smith	1234 →	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234 →	Lawyer

but *not* {Phone} → {Position}

Practice

A	B	C	D	E
1	2	4	3	6
3	2	5	1	8
1	4	4	5	7
1	2	4	3	6
3	2	5	1	8

Find at least *three* FDs which are violated on this instance:

{ C } → { D }
{ C } → { B }
{ A } → { B }

2. Finding FDs

What you will learn about next

- “Good” vs. “Bad” FDs: Intuition
- Finding FDs
- Closures
- PRACTICE: Compute the closures

1NF

- First normal form: A relation that has a primary key and in which there are no repeating groups
  - No multivalued attributes
  - Every attribute value is atomic (single fact in each table cell)

- All relations are in 1NF

- Normalization steps (from tabular view of data):
  - Goal: create a relation from the tabular view
  - Action: remove repeating groups
  - Action: select the primary key

Example: Convert To 1NF

Order_ID	Order_Date	Customer_ID	Customer_Name	Customer_Address	Product_ID	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity
1006	10/24/2004	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	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

- Normalization steps (from tabular view of data):
  - Goal: create a relation from the tabular view
  - Action: remove repeating groups
  - Action: select the primary key

Action: Remove Repeating Groups

Order_ID	Order_Date	Customer_ID	Customer_Name	Customer_Address	Product_ID	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity
1006	10/24/2004	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

- Is the data view a relation now?
  - Answer: yes
- Is it well-structured?
  - Answer: no

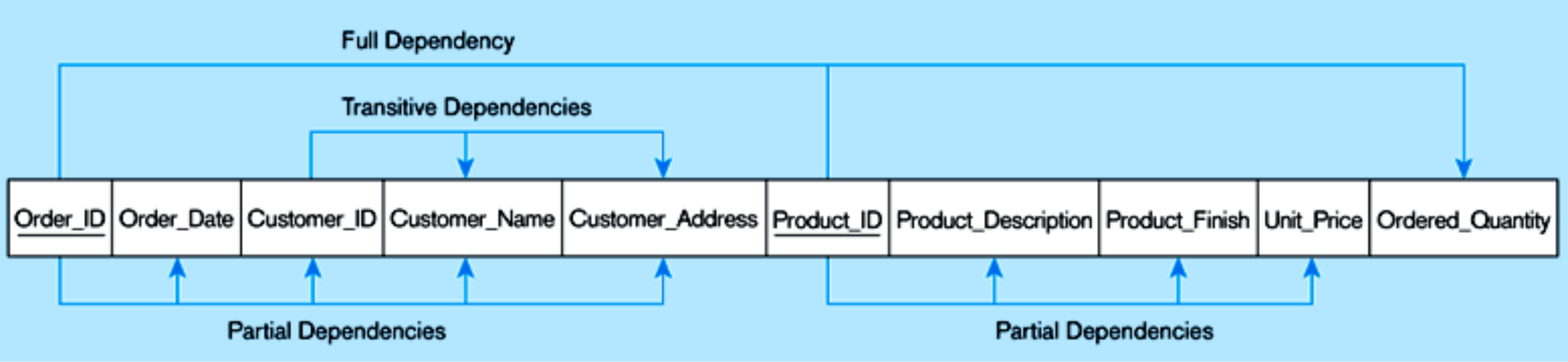
What are the anomalies in this table?

Order_ID	Order_ Date	Customer_ ID	Customer_ Name	Customer_ Address	Product_ID	Product_ Description	Product_ Finish	Unit_ Price	Ordered_ Quantity
1006	10/24/2004	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4–Dr Dresser	Oak	500.00	4
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

- Insertion: If new product is ordered for order 1007 of existing customer, customer data must be re-entered, causing duplication
- Deletion: If we delete the Dining Table from Order 1006, we lose information concerning this item's finish and price
- Update: Changing the price of product ID 4 requires update in several records
- Why do these anomalies exist? Because there are multiple themes (entity types) in one relation. -> duplication, and unnecessary dependency between entities

Action: Select A Primary Key

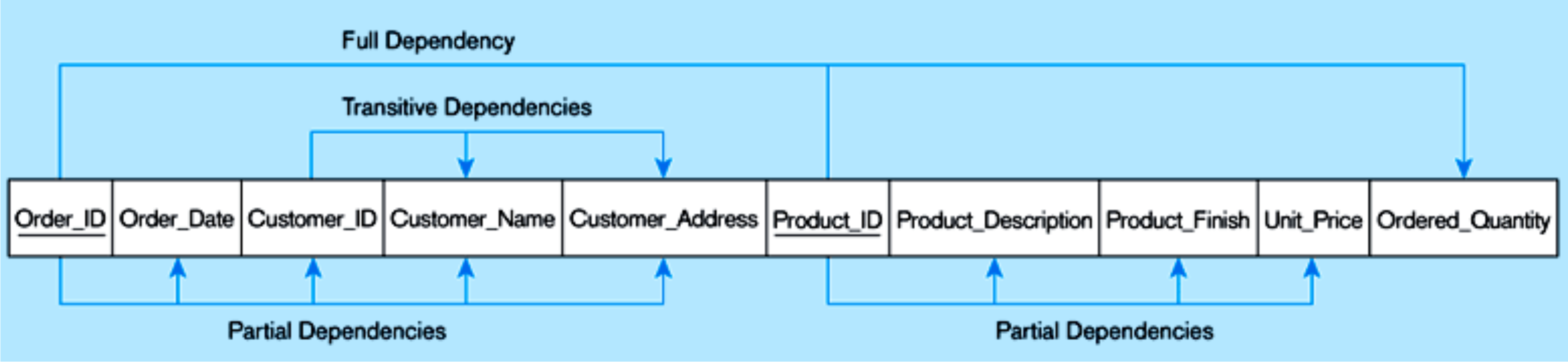
- Identify FDs and CKs (candidate keys = minimal superkeys)
- Four determinants and functional dependencies
  - Order_ID → Order_Date, Customer_ID, Customer_Name, Customer_Address
  - Customer_ID → Customer_Name, Customer_Address
  - Product_ID → Product_Description, Product_Finish, Unit_Price
  - Order_ID, Product_ID → Ordered_Quantity
- Select a PK from CKs
  - (Order_ID, Product_ID)



Next Step: Convert To 2NF

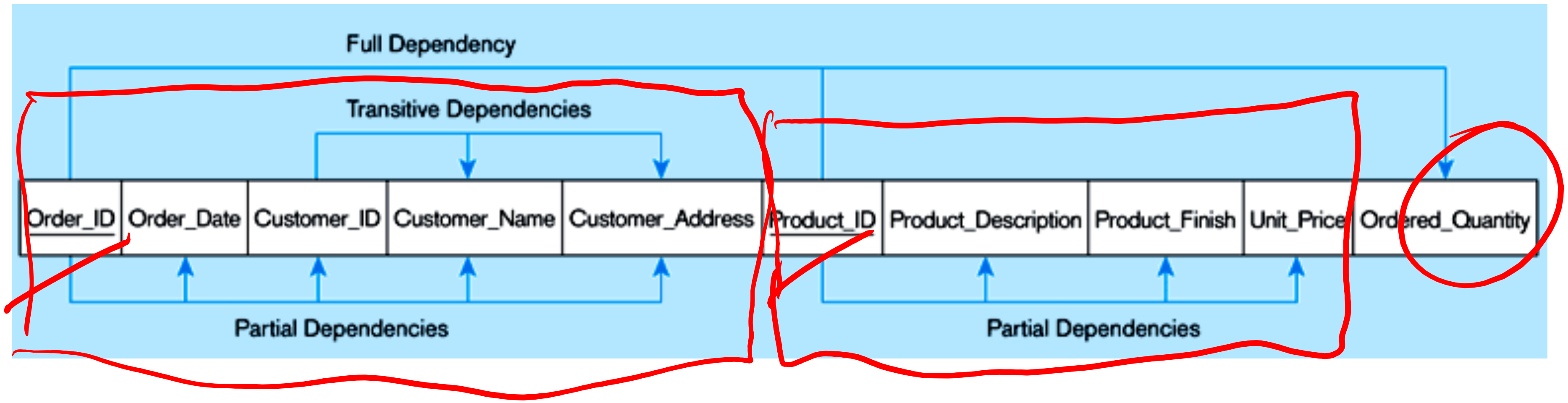
- 2NF: A relation in 2NF in which every non-key attribute is fully functionally dependent on the primary key

- Partial FD: A FD in which one or more nonkey attributes are functionally dependent on part (but not all) of the PK



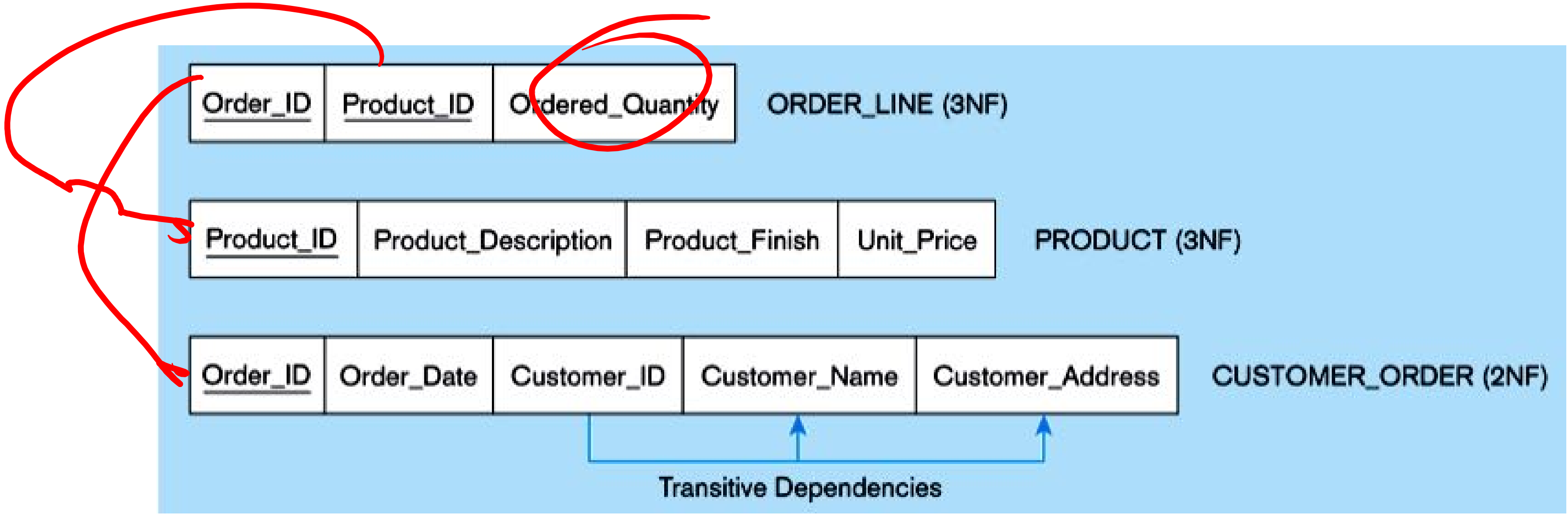
Getting A Relation To 2NF

- Create a new relation for each primary key attribute that is a determinant in a partial dependency
  - That attribute is the primary key in the new relation
- Move the nonkey attributes that are dependent on this primary key attribute(s) from the old relation to the new relation
- Exercise: Convert 1NF relation to 2NF



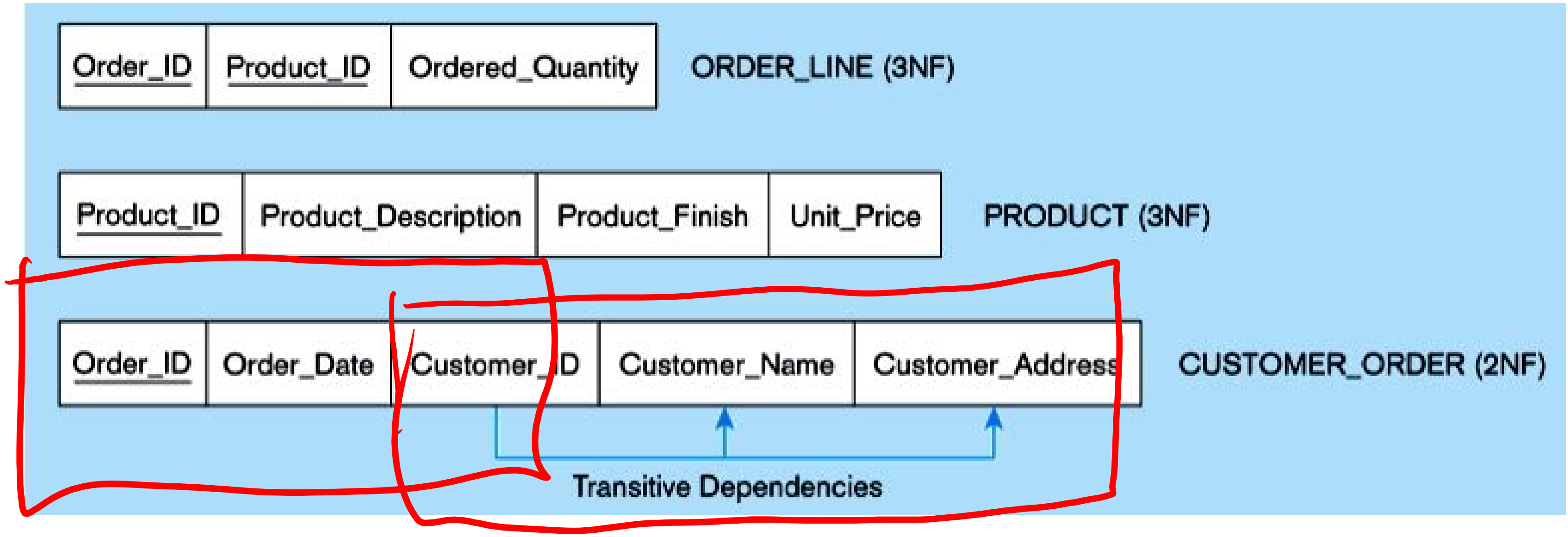
A 1NF Relation Is In 2NF if

- The PK consists of only one attribute. There cannot be a partial dependency in such a relation
- (or) no nonkey attributes exist in the relation (thus all attributes in the relation are components of the PK). There are no FDs in such a relation
- (or) every nonkey attribute is functionally dependent on the full set of PK attributes.



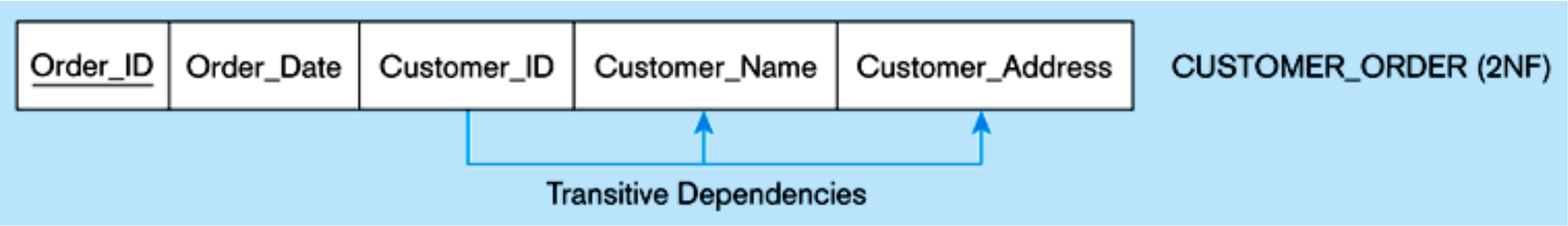
3NF

- 3NF: A relation that is in 2NF and has no transitive dependencies present
- Transitive dependency: An FD between two (or more) nonkey attributes
  - FD between the PK and one or more nonkey attributes that are dependent on the PK via another nonkey attribute
- Transitive dependency example:



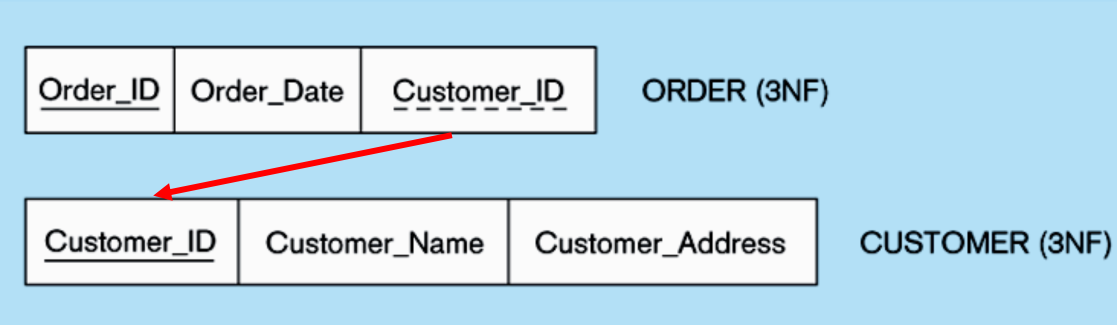
Removing Transitive Dependencies

- For each nonkey attribute(s) that is a determinant in a relation, create a new relation.
  - That attribute becomes the PK of the new relation
- Move all of the attributes that are functionally dependent on the attribute from the old to the new relation
- Leave the attribute (which serves as a PK in the new relation in the old relation to serve as a FK that allows us to associate the two relations
- Exercise: Convert relation below to 3NF

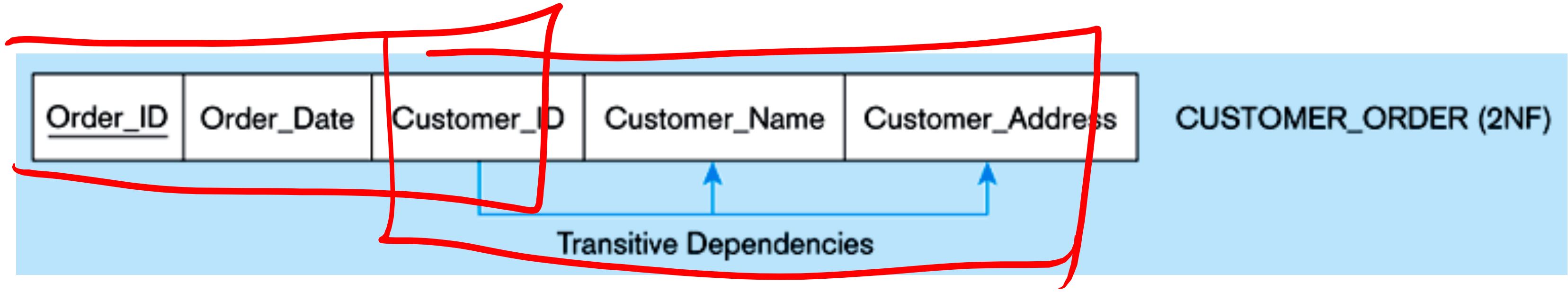


Third Normal Form

- Example converted to 3NF:



- Original example in 2NF:



Full Example: From 1NF to 3NF

Before (3NF):

Order_ID	Order_Date	Customer_ID	Customer_Name	Customer_Address	Product_ID	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity

After (3NF):

Order_ID	Product_ID	Ordered_Quantity

ORDER_LINE (3NF)

Product_ID	Product_Description	Product_Finish	Unit_Price

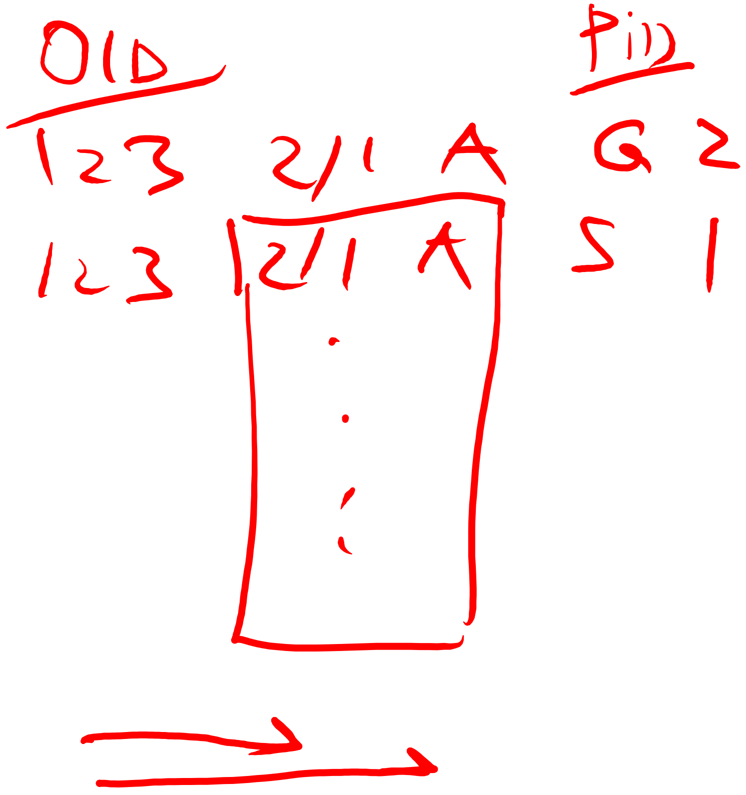
PRODUCT (3NF)

Order_ID	Order_Date	Customer_ID

ORDER (3NF)

Customer_ID	Customer_Name	Customer_Address

CUSTOMER (3NF)



Normalization Summary

- Data normalization is the process of decomposing relations with anomalies to produce smaller, well-structured relations

- Goals of normalization include:
  - Minimize data redundancy
  - Simplifying the enforcement of referential integrity constraints
  - Simplify data maintenance (inserts, updates, deletes)
  - Improve representation model to match "the real world"

"Good" vs. "Bad" FDs

We can start to develop a notion of **good** vs. **bad** FDs:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Intuitively:

EmpID -> Name, Phone, Position is *"good FD"*

- ***Minimal redundancy, less possibility of anomalies***

“Good” vs. “Bad” FDs

We can start to develop a notion of **good** vs. **bad** FDs:

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

<u>Intuitively:</u>

EmpID -> Name, Phone, Position is *“good FD”*

But Position -> Phone *is a “bad FD”*

- ***Redundancy! Possibility of data anomalies***

“Good” vs. “Bad” FDs

Student	Course	Room
Mary	CS3200	WVF20
Joe	CS3200	WVF20
Sam	CS3200	WVF20
..	..	..

Returning to our original example… can you see how the “bad FD” {Course} -> {Room} could lead to an:

- Update Anomaly
- Insert Anomaly
- Delete Anomaly
- …

Given a set of FDs (from user) our goal is to:

1. Find all FDs, and
2. Eliminate the “Bad Ones".

FDs for Relational Schema Design

- High-level idea: why do we care about FDs?

1. Start with some relational schema

2. Find out its functional dependencies (FDs)

This part can be tricky!

3. Use these to design a better schema
    - One which minimizes possibility of anomalies

Finding Functional Dependencies

- There can be a very large number of FDs…
  - How to find them all efficiently?

- We can't necessarily show that any FD will hold on all instances…
  - How to do this?

We will start with this problem:
Given a set of FDs, F, what other FDs ***must*** hold?

Finding Functional Dependencies

- Equivalent to asking: Given a set of FDs, $F = \{f_1,\ldots f_n\}$, does an FD g hold?
  - Inference problem: How do we decide?

Example:

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Given the provided FDs, we can see that {Name, Category} → {Price} must also hold on **any instance**…

Which / how many other FDs do?!?

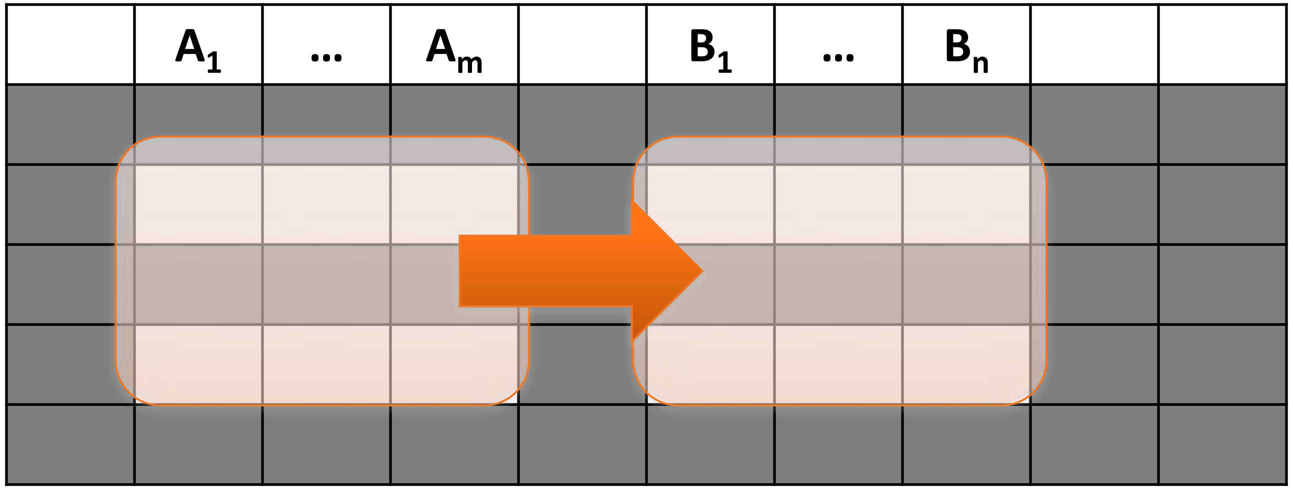
Finding Functional Dependencies

- Equivalent to asking: Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?
  - Inference problem: How do we decide?

Answer: Three simple rules called **Armstrong's Rules.**

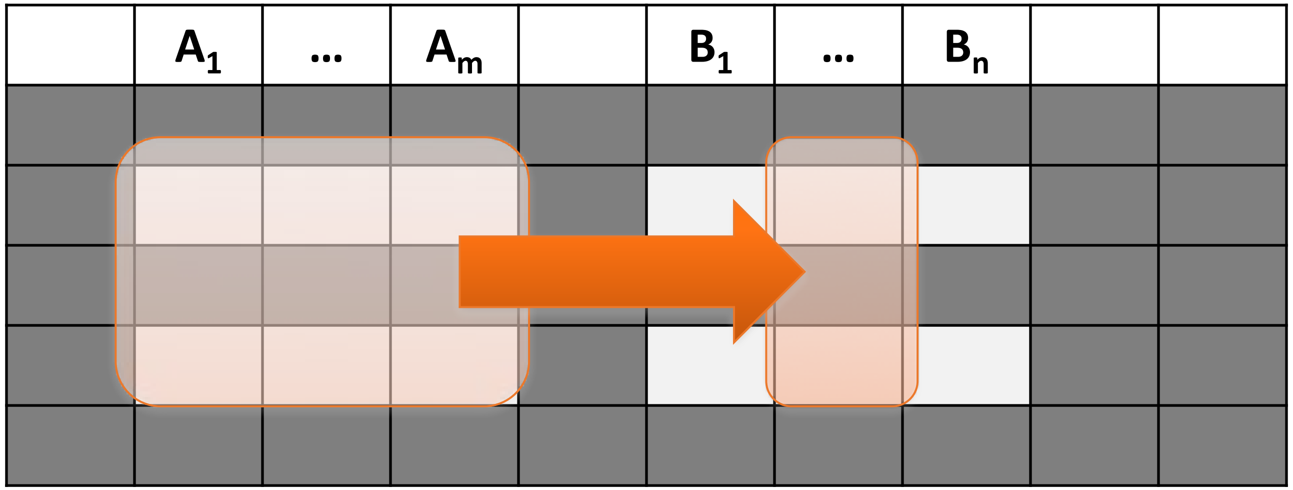
1. **Split/Combine,**
2. **Reduction, and**
3. **Transitivity…** *ideas by picture*

1. Split/Combine



$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$

1. Split/Combine

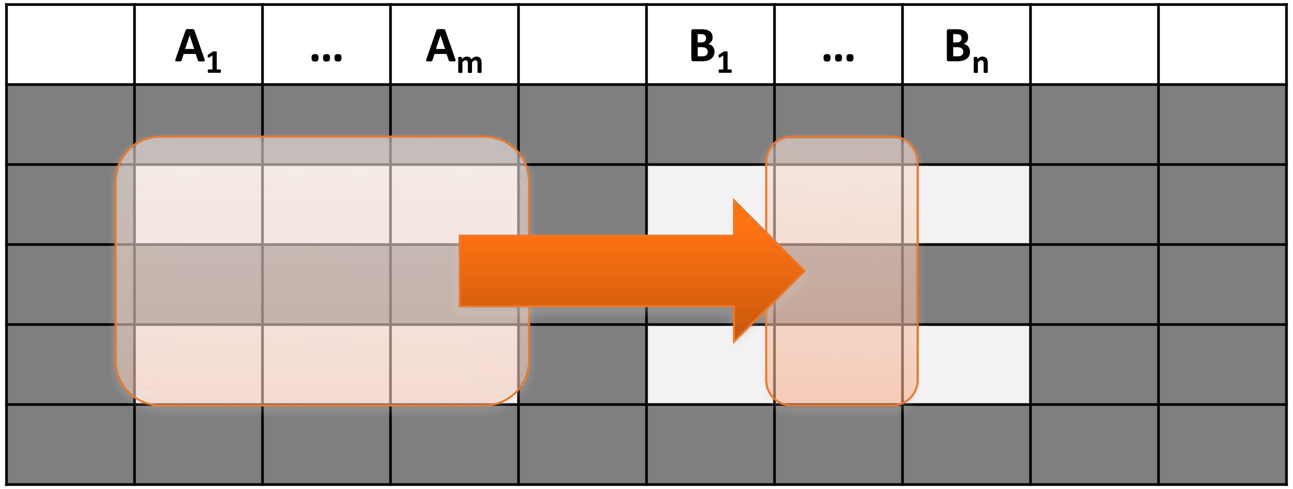


$$A_1, \ldots, A_m \rightarrow B_1, \ldots, B_n$$

… is equivalent to the following *n* FDs…

$$A_1, \ldots, A_m \rightarrow B_i \text{ for } i=1,\ldots,n$$

1. Split/Combine



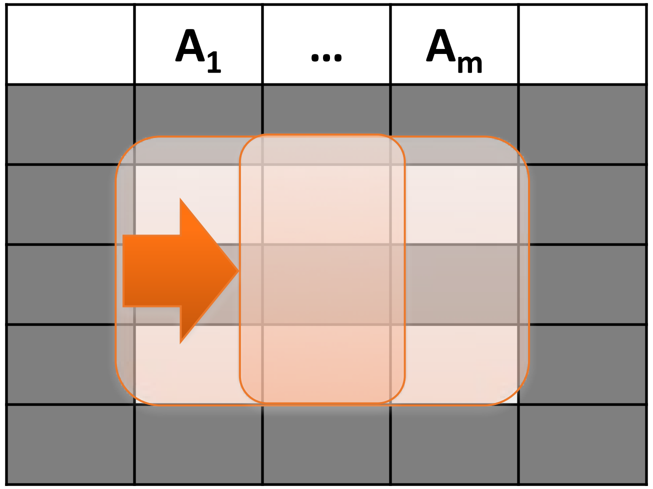
And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

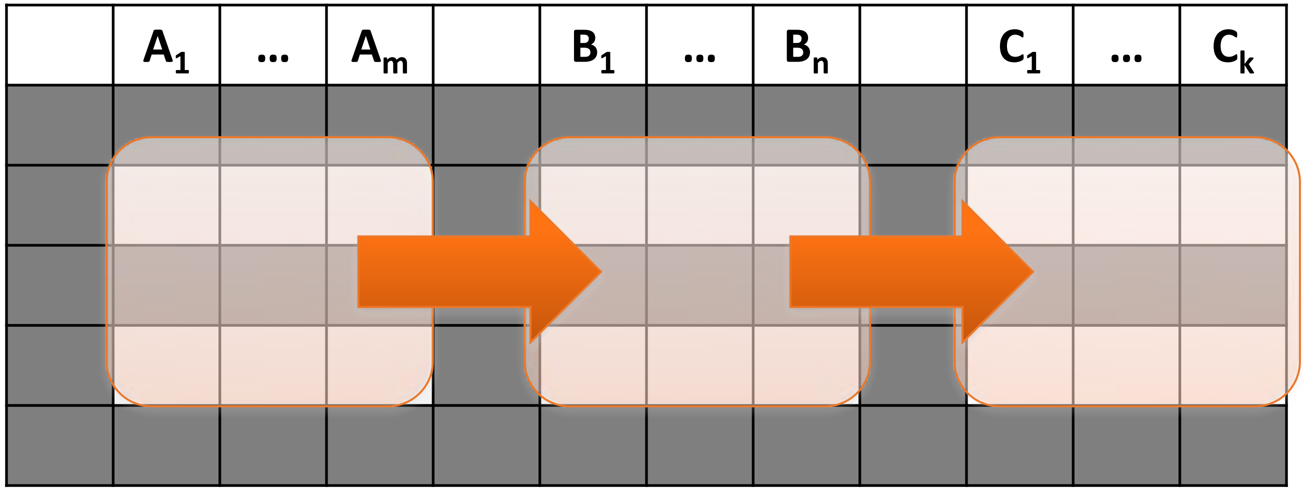
2. Reduction (Trivial)

(B, P)



$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$

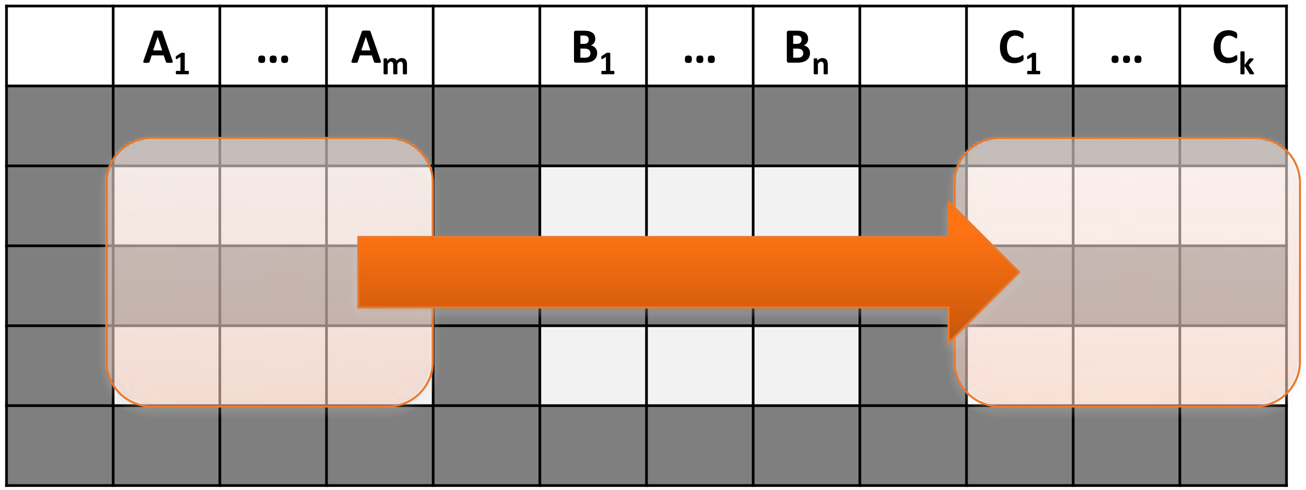
3. Transitive Closure



$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$ and

$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$

3. Transitive Closure



$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$ and
$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$

implies

$A_1,\ldots,A_m \rightarrow C_1,\ldots,C_k$

Finding Functional Dependencies

Example:

Products

Name	Color	Category	Dep	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Department}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	?
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	?
6. {Name, Category} -> {Category}	?
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	?
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category -> {Color, Category}	?
8. {Name, Category} -> {Price}	?

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category -> {Color, Category}	Split/combine (5 + 6)
8. {Name, Category} -> {Price}	?

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Finding Functional Dependencies

Can we find an algorithmic way to do this?

Example:

Inferred FDs:

Inferred FD	Rule used
4. {Name, Category} -> {Name}	Trivial
5. {Name, Category} -> {Color}	Transitive (4 -> 1)
6. {Name, Category} -> {Category}	Trivial
7. {Name, Category -> {Color, Category}	Split/combine (5 + 6)
8. {Name, Category} -> {Price}	Transitive (7 -> 3)

Provided FDs:

1. {Name} → {Color}
2. {Category} → {Dept.}
3. {Color, Category} → {Price}

Which / how many other FDs hold?

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F**:
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F:**
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = ?
{name, category}+ = ?

{color}+ = ?
```

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F**:
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}⁺ = {name, color}
{name, category}⁺ = ?

{color}⁺ = ?
```

Closure of a set of Attributes

Given a set of attributes $A_1, ..., A_n$ and a set of FDs **F:**
Then the <u>**closure**</u>, $\{A_1, ..., A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, ..., A_n\} \rightarrow B$

<u>Example:</u> F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = {name, color}
{name, category}+ =
  {name, category, color, ...}
{color}+ = ?
```

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F:**
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = {name, color}
{name, category}+ =
  {name, category, color, dept, price}
{color}+ = ?
```

Closure of a set of Attributes

Given a set of attributes $A_1, \ldots, A_n$ and a set of FDs **F**:
Then the **closure**, $\{A_1, \ldots, A_n\}^+$ is the set of attributes **B** s.t. $\{A_1, \ldots, A_n\} \rightarrow B$

Example: F =

```
{name} → {color}
{category} → {department}
{color, category} → {price}
```

Example Closures:

```
{name}+ = {name, color}
{name, category}+ =
  {name, category, color, dept, price}
{color}+ = {color}
```

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$ and set of FDs F.

Repeat until X doesn't change; **do**:

if $\{B_1, \ldots, B_m\} \rightarrow C$ is entailed by F

and $\{B_1, \ldots, B_m\} \subseteq X$

then add C to X.

Return X as X^+

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_m\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_m\} \subseteq X$:
then add C to X.
Return X as X^+

`{name, category}`$^+$ =
`{name, category}`

F =

```
{name} → {color}

{category} → {dept}

{color, category} → {price}
```

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_m\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_m\} \subseteq X$:
then add C to X.
Return X as X^+

$\{name, category\}^+ = \{name, category\}$

$\{name, category\}^+ = \{name, category, color\}$

F =

{name} → {color}

{category} → {dept}

{color, category} → {price}

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_m\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_m\} \subseteq X$:
then add C to X.
Return X as X^+

F =

```
{name} → {color}

{category} → {dept}

{color, category} →
{price}
```

```
{name, category}+ =
{name, category}
```

```
{name, category}+ =
{name, category, color}
```

```
{name, category}+ =
{name, category, color, dept}
```

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change; **do**:
if $\{B_1, \ldots, B_m\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_m\} \subseteq X$:
then add C to X.
Return X as X^+

F =

```
{name} → {color}

{category} → {dept}

{color, category} → {price}
```

```
{name, category}+ =
{name, category}
```

```
{name, category}+ =
{name, category, color}
```

```
{name, category}+ =
{name, category, color, dept}
```

```
{name, category}+ =
{name, category, color, dept,
price}
```

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B,$ $\}$

Compute $\{A, F\}^+ = \{A, F,$ $\}$

Example

R(A,B,C,D,E,F)

{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}

Compute $\{A,B\}^+ = \{A, B, C, D \quad\quad\quad\quad \}$

Compute $\{A, F\}^+ = \{A, F, \quad\quad\quad\quad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, F, \qquad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, B, F, \qquad\qquad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, B, C, F, \qquad \}$

Example

```
R(A,B,C,D,E,F)
```

```
{A,B} → {C}
{A,D} → {E}
{B} → {D}
{A,F} → {B}
```

Compute $\{A,B\}^+ = \{A, B, C, D, E\}$

Compute $\{A, F\}^+ = \{A, B, C, D, E, F\}$

3. Closures, Superkeys, and (Candidate) Keys