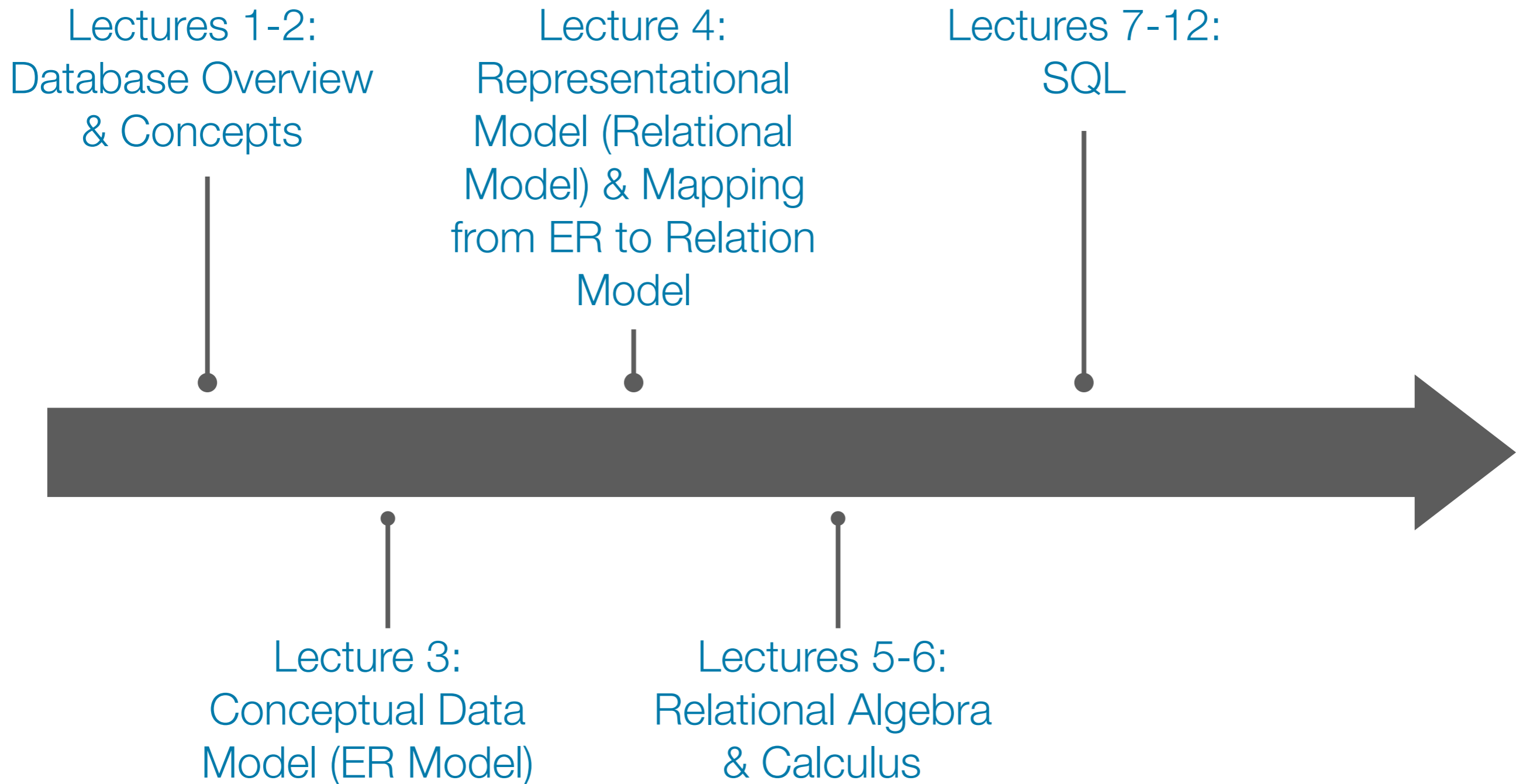


Database Design Theory and Normalization

CS 377: Database Systems

Recap: What Has Been Covered



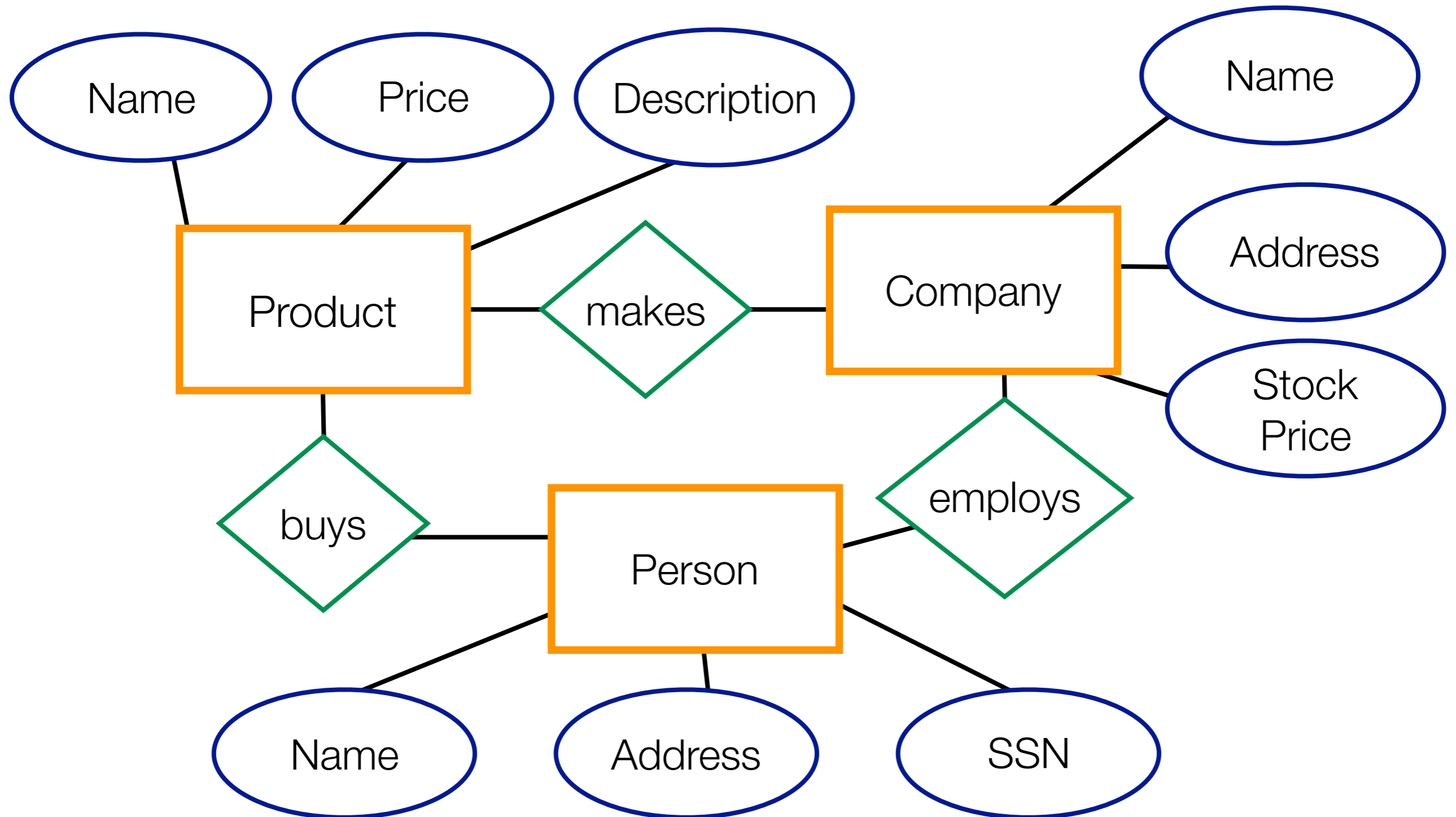
What's Left

- Database design: Schema normalization
 - Data storage & indexing
 - Query optimization
 - Transaction management & concurrency control
 - Big data systems
 - NoSQL
- Intention: Give you a taste of advanced database systems.
More details — take CS554**

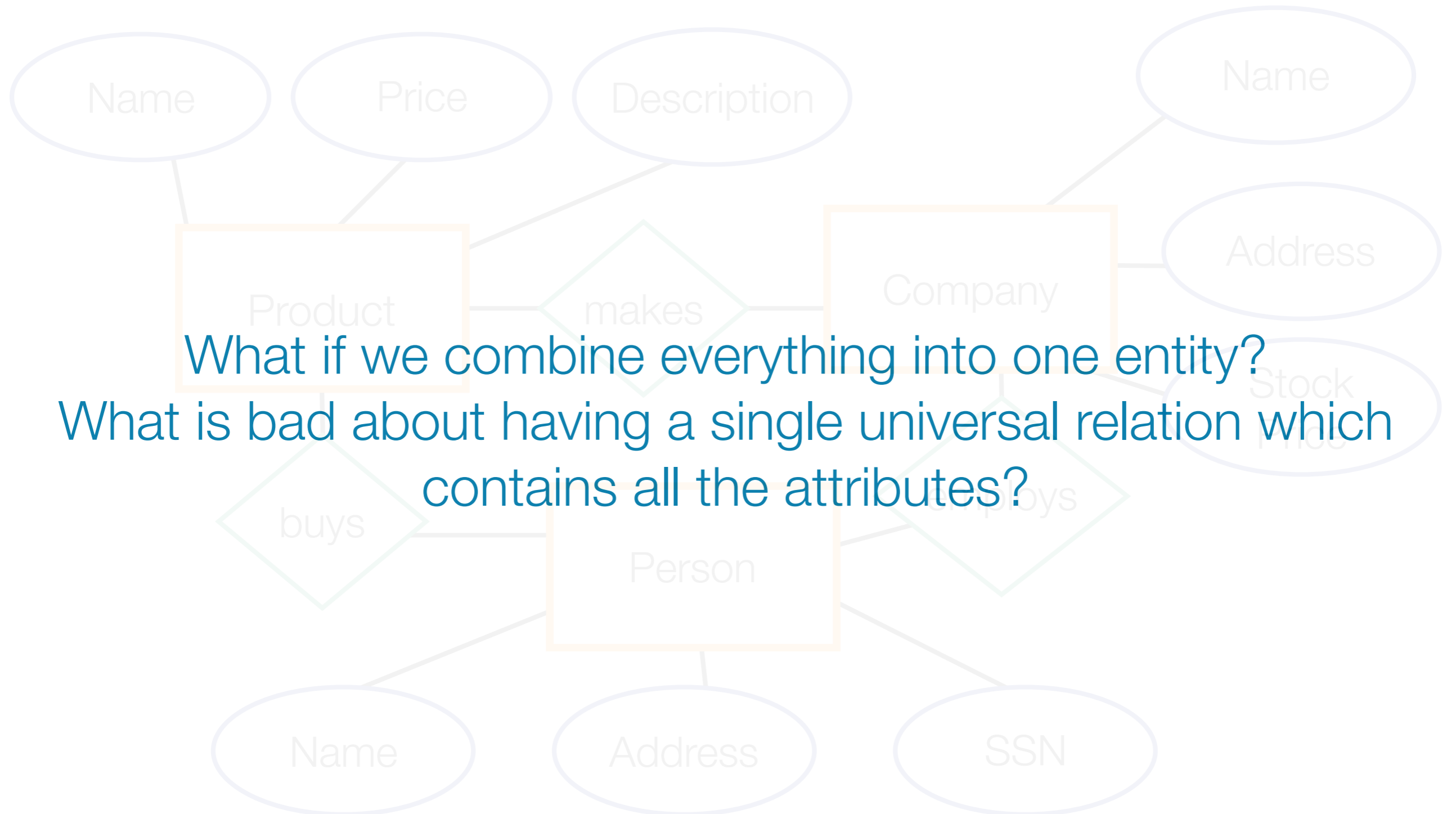
Today and Next Lecture

1. 1NF
2. Informal guidelines
3. Functional dependency
 1. Inference rules
 2. Closure algorithm
4. 2NF, 3NF, BCNF

Product Database (from Lecture 3)



Universal Relation



Are these Bad Designs?

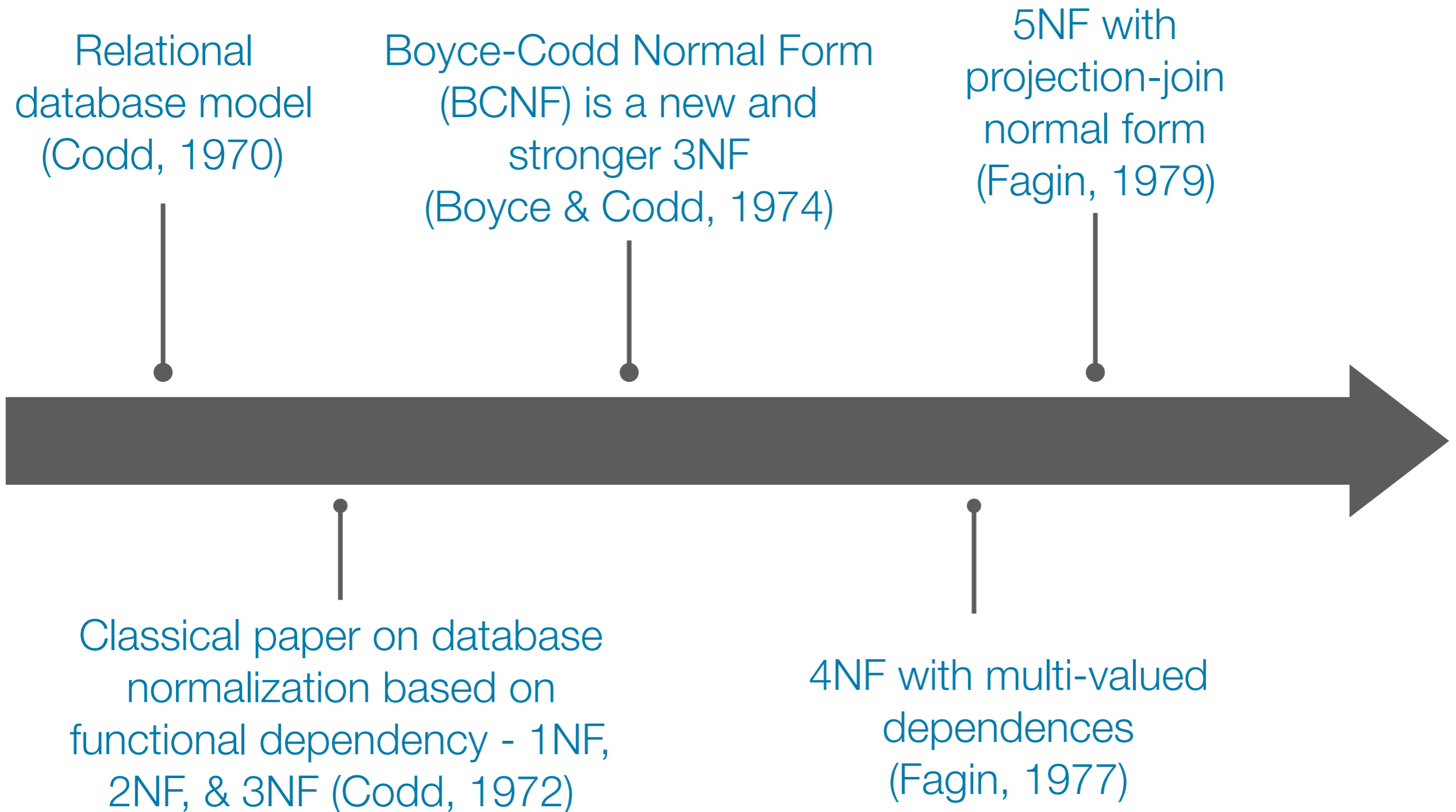
EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland

History of Database Design



Relationship amongst Normal Forms

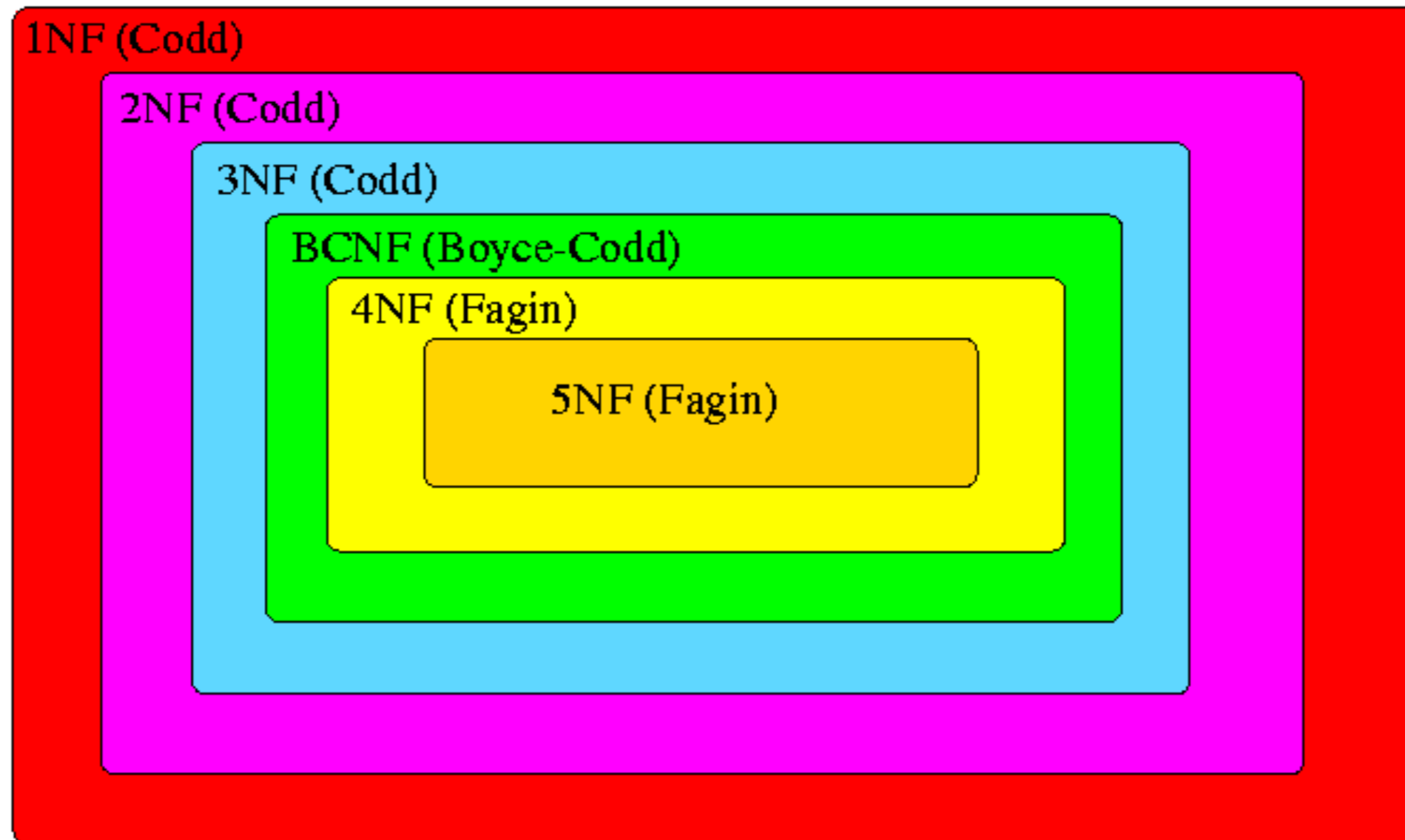


Image courtesy of Prof Cheung's notes

Each rectangle represents all possible relations

Normalization: General Idea

- Designers should aim for the “ultimate” 5NF
 - However, designers typically stop at 3NF or BCNF
- Designing a good database is a complex task
 - Normalization is useful aid but should not be panacea
- Normal forms can be violated deliberately to achieve better performance (less join operations)

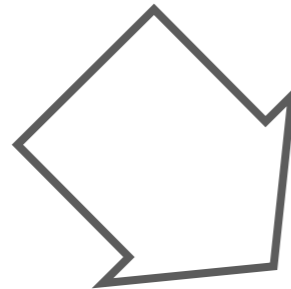
First Normal Form (1NF)

- Simplest one that does not depend on “functional dependency”
- Basic relational model where every attribute has atomic (single, not multi) values
- Techniques to achieve 1NF (if not already done)
 - Remove attribute violating 1NF and place in separate relation
 - Expand the key

Example: 1NF Conversion

DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}



DEPARTMENT

Dname	<u>Dnumber</u>	Dmgr_ssn	<u>Dlocation</u>
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Adapted from Figure 14.9 (Book)

When is a Relation “Good” or “Bad”?

- Number of bad properties called anomalies
- A relation exhibiting one or more of these anomalies is deemed bad

CAVEAT: “Good” relations can be inefficient!

DB designers may use “bad” relations for performance reasons.

Database Anomalies: Insert Anomaly

Inserting ONE item of information

- Normal: one tuple is introduced to one or more tables with no NULL values
- **Anomaly**: multiple tuples are added to some relation
- **Anomaly**: NULL values are added

Example: Insert Anomaly

Relation about employees and departments

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
111-11-1111	John	Smith	5	Research	123-45-6789
222-22-2222	Jane	Doe	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	1	Payroll	777-77-7777

What if a new department is created (dno = 6, dname = “Administration”) with no employees yet?

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
NULL	NULL	NULL	6	Administration	NULL

Database Anomalies: Delete Anomaly

- Normal behavior of deleting ONE item of information
 - One tuple is removed in one or more tables
 - Only intended information is deleted and does not cause loss of additional information
- Delete anomaly occurs when deleting ONE item of information
 - Deletes multiple tuples into some relation
 - Causes additional (unintended) information

Example: Delete Anomaly

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
111-11-1111	John	Smith	5	Research	123-45-6789
222-22-2222	Jane	Doe	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	1	Payroll	777-77-7777

What if Jack Rabbit leaves the company?

DELETE employee **WHERE** fname = 'Jack' **AND** lname = 'Rabbit';

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
111-11-1111	John	Smith	5	Research	123-45-6789
222-22-2222	Jane	Doe	5	Research	123-45-6789

Payroll department is also deleted!

Database Anomalies: Update Anomaly

- Normal behavior of updating ONE item of information
 - One tuple in one or more tables is updated
- Update anomaly occurs when updating ONE item of information
 - Updates multiple tuples from some relation

Example: Update Anomaly

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
111-11-1111	John	Smith	5	Research	123-45-6789
222-22-2222	Jane	Doe	5	Research	123-45-6789
333-33-3333	Jack	Rabbit	1	Payroll	777-77-7777

What if manager of research department changes?

**UPDATE employee SET MgrSSN = '888-88-8888'
WHERE DName = 'Research';**

<u>SSN</u>	FName	LName	DNo	DName	MgrSSN
111-11-1111	John	Smith	5	Research	888-88-8888
222-22-2222	Jane	Doe	5	Research	888-88-8888
333-33-3333	Jack	Rabbit	1	Payroll	777-77-7777

Operation has modified multiple tuples in single relation!

Generation of Spurious Tuples

- Natural join results in more tuples than “expected”
- Represents spurious information that is not valid
- Example: What happens during a natural join?

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford

Example: Generation of Spurious Tuples

Ssn	Pnumber	Hours	Pname	Plocation	Ename
123456789	1	32.5	ProductX	Bellaire	Smith, John B.
* 123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
123456789	2	7.5	ProductY	Sugarland	Smith, John B.
* 123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
* 123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
* 666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
* 453453453	1	20.0	ProductX	Bellaire	Smith, John B.
453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
* 453453453	2	20.0	ProductY	Sugarland	Smith, John B.
453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
* 453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
* 333445555	2	10.0	ProductY	Sugarland	Smith, John B.
* 333445555	2	10.0	ProductY	Sugarland	English, Joyce A.

Asterisk denotes the tuples that don't make sense

Informal Design Guidelines

- Design relations where meaning of a relation's attributes can be easily explained — avoid combining multiple entity types and relationship types into a single relation
- Avoid insertion, deletion, and update anomalies — minimize redundant information

Informal Design Guidelines

- Reduce NULL values in tuples — use space efficiently and avoid joins with NULL values
- Design relation schemas to guarantee no spurious tuples — avoid relations that contain matching attributes that are not (foreign key, primary key) combinations

Formal Database Design Theory

- Normal forms
 - Set of properties that relations must satisfy
 - Successively higher degrees of stringency
- Database normalization
 - Certify whether a database design satisfies a certain normal form
 - Correct designs to achieve certain normal form

Functional Dependencies (FD)

- Definition:
 - Let X and Y be 2 sets of attributes of R
 - A functional dependency ($X \rightarrow Y$) occurs if for any two tuples t_1 and t_2 of the relation R , if $t_1[X] = t_2[X]$ (i.e., the attribute values for X is the same in both tuples) then $t_1[Y] = t_2[Y]$

$X \rightarrow Y$ means that whenever two tuples agree on X , then they agree on Y

FD Pictorially

X

Y

	A	B	C	D	E	F	G
t1
	...	b7	c4	...	e1	f3	g4
t2
	...	b7	c4	...	e1	f3	g4

If t1 and t2 agree here...

they also agree here!

FD for Relational Schema

- Constraint between two sets of attributes
- Generalize the concept of keys
- Why should we care?
 - Start with relational schema
 - Find FDs
 - Use these to design better schema

Example: Company Database

- Relation that represent information about employees and the projects they work on

<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

Example: Company Database

<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

- FDs in the relation
 - $SSN \rightarrow \text{fname, lname}$
 - $PNo \rightarrow PName$
 - $SSN, PNo \rightarrow \text{Hours}$

Example: Company Database

<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

- FDs can cause anomalies due to dependency between attributes
- Insert anomaly - new project (pj3) with no employees
- Delete anomaly - deleting John Smith from pj2 deletes information about pj2

Inferring FDs

- An FD is
 - Inherent property of an application
 - Defined based on the semantics of the attributes
 - Not something we can infer from a set of tuples

Inferring FDs from Table

- Given a table with a set of tuples
 - Can confirm that a FD **seems** to be valid
 - Infer a FD is **definitely invalid**
 - Can **never prove** that FD is valid

Example: Course Database

- Relation with courses, students, and instructors

<u>studentID</u>	name	semester	courseNo	section	instructor
123455	Bob	S17	CS377	0	Ho
234097	John	S17	CS377	0	Ho
234107	Alice	F16	CS377	0	Cheung
140701	Mary	F16	CS377	0	Cheung

Example: Course Database

<u>studentID</u>	name	level	semester	courseNo	instructor
123455	Bob	Junior	S17	CS377	Ho
234097	John	Senior	S17	CS377	Ho
234107	Alice	Junior	F16	CS377	Cheung
140701	Mary	Senior	F16	CS377	Cheung

- FDs in the relation
 - courseNo, semester \rightarrow instructor
 - studentID \rightarrow courseNo, semester
 - studentID \rightarrow name, level

“Good” vs “Bad” FDs: Intuition

<u>studentID</u>	name	level	semester	courseNo	instructor
123455	Bob	Junior	S17	CS377	Ho
234097	John	Senior	S17	CS377	Ho
234107	Alice	Junior	F16	CS377	Cheung
140701	Mary	Senior	F16	CS377	Cheung

- studentID → name, level is a “good” FD
- Minimal redundancy, less possibility of anomaly

“Good” vs “Bad” FDs: Intuition

<u>studentID</u>	name	level	semester	courseNo	instructor
123455	Bob	Junior	S17	CS377	Ho
234097	John	Senior	S17	CS377	Ho
234107	Alice	Junior	F16	CS377	Cheung
140701	Mary	Senior	F16	CS377	Cheung

- courseNo, semester \rightarrow instructor is a “bad” FD
- Redundancy! Possibility of anomalies

Refresher: Keys

- Set of attributes S is a super key of a relation R if S functionally determines all attributes in R

$$\forall t_1, t_2 \in R : t_1[SK] \neq t_2[SK]$$

- Set of attributes K is a key of a relation if and only if
 - K functionally determines all attributes in R
 - K is minimal superkey
 - None of its subsets functionally determines all attributes in R

“Good” vs “Bad” FDs

- A key of a relation functionally determines all attributes in that relation
 - This is called natural or trivial
- “Good” functional dependency is a natural or trivial functional dependency
- Functional dependencies other than natural dependencies will cause anomalies

Example: Company DB Revisited


<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
------------	-------	-------	------------	-------	-------

- $SSN, PNo \rightarrow Hours$ is a “good” functional dependency
 - $(SSN, PNo, Hours)$ should be in the same relation
- $SSN \rightarrow fname, lname$ is a “bad” functional dependency and should be taken out and put together in another relation on their own
- $PNo \rightarrow PName$ is a “bad” functional dependency and should be taken out and put in another relation on their own

“Bad” FDs Cause Anomalies

- Since the LHS of a functional dependency is not a key, you can have multiple tuples in the database
 - Leads to update anomalies as well as insert and delete anomalies
 - Duplication of information is guaranteed!
- Solution: break up the relation into multiple tuples

Relation Decomposition

- A decomposition of relation R is a collection of relations R_1, R_2, \dots, R_n such that every attribute of R appears in R_1, R_2, \dots, R_n at least once
- Some decompositions are useful and some aren't
- Example:
Employee(SSN, Fname, LName, PNo, PName, Hours) —
R1(SSN, PName, Hours)
R2(PNumber, Fname, LName) 
What does this mean?
- Decompose with a goal!

What is a Good Decomposition?

- Normal forms will be guiding criteria for better relations
- When a relation R violates the guiding criteria of normal form, we decompose the relation to comply with the guiding criteria of the normal form
- Use functional dependencies to determine if dependency is “good” or “bad”
 - Find all keys of the relation R via inference rules

Armstrong's Axioms

- Most basic inference rules
 - Given a set of functional dependencies, we can derive additional functional dependencies using inference rules
- Sound — any FD inferred using Armstrong's axioms will hold in R
- Complete — Every valid FD on R can be found by applying only Armstrong's axioms

Armstrong's Axiom 1: Reflexivity

- For attribute sets X, Y : If Y is subset of X , then $X \rightarrow Y$
- Examples:
 - $A, B \rightarrow B$
 - $A, B, C \rightarrow A, B$
 - $A, B, C \rightarrow A, B, C$

Armstrong's Axiom 2: Augmentation

- For attribute sets X, Y, Z : If $X \rightarrow Y$, then $X, Z \rightarrow Y, Z$
- Examples:
 - $A \rightarrow B$ implies $A, C \rightarrow B, C$
 - $A, B \rightarrow C$ implies $A, B, C \rightarrow C$

Armstrong's Axiom 3: Transitivity

- For attribute sets X, Y, Z : If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$, then $X \twoheadrightarrow Z$
- Examples:
 - $A \twoheadrightarrow B$ and $B \twoheadrightarrow C$ implies $A \twoheadrightarrow C$
 - $A \twoheadrightarrow C, D$ and $C, D \twoheadrightarrow E$ implies $A \twoheadrightarrow E$

Example: Armstrong's Axioms

- **Product**(name, category, color, department, price)
- Given initial set of FDs:
 - name \rightarrow color
 - category \rightarrow department
 - color, category \rightarrow price

Example: Armstrong's Axioms

- **Product**(name, category, color, department, price)
- Inferred FDs:
 - name, category \rightarrow price: augmentation & transitivity
 - name, category \rightarrow color: reflexivity & transitivity

Other Useful Inference Rules

- Derived from Armstrong's Axioms
- Decomposition rule: If $X \twoheadrightarrow Y, Z$ then $X \twoheadrightarrow Y, X \twoheadrightarrow Z$
- Union rule: If $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$, then $X \twoheadrightarrow Y, Z$
- Pseudo transitivity rule: If $X \twoheadrightarrow Y$ and $Y, W \twoheadrightarrow Z$ then $X, W \twoheadrightarrow Z$

Tedious to infer all the functional dependencies and check them all — is there algorithmic way to do this?

Finding Keys of Relation R

- Bad news: NP-complete problem
 - Running time of algorithm to solve the problem exactly is exponentially increasing with the problem size
 - Large NP-complete problems are difficult to solve!
 - No efficient solution to find all the keys

Finding Keys of Relation R

- Possible solutions
 - Brute force algorithm: Check every subset of attributes for super key strategy — tests every possible solution
 - Use heuristics to find all the keys of a relation — turn towards closures to help us find keys in a relation

Attribute Closure Set

- If X is an attribute set, the closure X^+ is the set of all attributes B such that $X \twoheadrightarrow B$
 - X is subset of X^+ since $X \twoheadrightarrow X$
 - X^+ includes all attributes that are functionally determined from X
- Importance: If $X^+ = R$, then X is a superkey
 - Closure can tell us if set of attributes X is a superkey

Example: Closure

- **Product**(name, category, color, department, price)
 - name \rightarrow color
 - category \rightarrow department
 - color, category \rightarrow price
- Attribute Closure:
 - $\{\text{name}\}^+ = \{\text{name}, \text{color}\}$
 - $\{\text{name}, \text{category}\}^+ = \{\text{name}, \text{color}, \text{category}, \text{department}, \text{price}\}$
 - $\{\text{color}\}^+ = \{\text{color}\}$

Finding a Key after Closure

- If X^+ not equal to the relation, we must augment more attributes to X to obtain a key
- If $X^+ = R$, then X is superkey — check for minimality
 - Remove one or more attributes A
 - Compute the closure of $X - A$ to see if $(X - A)^+ = R$
 - X is a key if $(X - A)^+$ not equal R for any attribute A

Closure Algorithm

- Input: A set F of FDs on a relation schema R , and a set of attributes X , which is a subset of R
- Algorithm:
Initialize $X^+ := X$
repeat
 old $X^+ := X^+$
 for each functional dependency $Y \rightarrow Z$ in F
 if X^+ superset Y , then $X^+ := X^+ \cup Z$
until ($X^+ = \text{old } X^+$)

Example: Closure Algorithm

EmpProj(SSN, FName, LName, PNo, PName, PLocation, Hours)

- SSN \rightarrow FName, LName
- PNo \rightarrow PName, PLocation
- SSN, PNo \rightarrow Hours

Example: Closure Algorithm

- Initialize $SSN^+ := SSN$
- Repeat loop (for each FD)
 - $SSN \rightarrow FName, LName$
 $\Rightarrow SSN^+ := SSN, FName, LName$
 - $PNo \rightarrow PName, PLocation$
 \Rightarrow no change
 - $SSN, PNo \rightarrow Hours$
 \Rightarrow no change
- Result: $SSN^+ := SSN, FName, LName$

Since there were changes,
repeat another loop
through FDs, which results
in no changes \Rightarrow done

Example: Closure Algorithm

- Initialize $PNo^+ := PNo$
- Repeat loop (for each FD)
 - $SSN \rightarrow FName, LName$
 \Rightarrow no change
 - $PNo \rightarrow PName, PLocation$
 $\Rightarrow PNo^+ := PNo, PName, PLocation$
 - $SSN, PNo \rightarrow Hours$
 \Rightarrow no change
- Result: $PNo^+ := PNo, PName, PLocation$

Since there were changes,
repeat another loop
through FDs, which
results in no changes
 \Rightarrow done

Example: Closure Algorithm

- Initialize $(SSN, PNo)^+ := SSN, PNo$
- Repeat loop (for each FD)
 - $SSN \rightarrow FName, LName$
 $\Rightarrow (SSN, PNo)^+ := SSN, PNo, FName, LName$
 - $PNo \rightarrow PName, PLocation$
 $\Rightarrow (SSN, PNo)^+ := SSN, PNo, FName, LName, PName, PLocation$
 - $SSN, PNo \rightarrow Hours$
 $\Rightarrow (SSN, PNo)^+ := SSN, PNo, FName, LName, PName, PLocation, Hours$
- Result: $(SSN, PNo)^+ := SSN, PNo, FName, LName, PName, PLocation, Hours$

Example: Closure Algorithm

- Summary of results:
 - $SSN^+ := SSN, FName, LName$
 - $PNo^+ := PNo, PName, PLocation$
 - $(SSN, PNo)^+ := SSN, PNo, FName, LName, PName, PLocation, Hours$
- (SSN, PNo) is a superkey!
- (SSN, PNo) is minimal superkey
 - $\{(SSN, PNo) - (SSN)\}^+ = (PNo)^+$
 - $\{(SSN, PNo) - (PNo)\}^+ = (SSN)^+$

Finding Keys: Heuristic 1

- Increase/decrease until you find keys
- Step 1: Compute closure of all functional dependencies in F
- Step 2:
 - If deficient, then add missing attributes to the LHS until the closure is equal to the relation
 - If sufficient, then remove extraneous attributes from the LHS until set is minimal

Example: Key Heuristic 1

- $R(A, B, C, D, E, F)$
 - $A \rightarrow B, C$
 - $B, D \rightarrow E, F$
 - $F \rightarrow A$

Example: Key Heuristic 1

- Step 1: Closure of all functional dependencies
 - $A^+ = A, B, C$
 - $(B, D)^+ = A, B, C, D, E, F$
 - $F^+ = F, A, B, C$

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

- $A \twoheadrightarrow B, C$
- $B, D \twoheadrightarrow E, F$
- $F \twoheadrightarrow A$

Example: Key Heuristic 1

- Step 2: Insert / remove attributes
 - $A^+ = A, B, C$ — insufficient so add
 - Add D: $(A, D)^+ = A, B, C, D, E, F \rightarrow$ key!
 - Add E: $(A, E)^+ = A, B, C, E$
 - Add F: $(A, F)^+ = A, B, C, F$
 - Add E, F: $(A, E, F)^+ = A, B, C, E, F$
 - No more so done

Example: Key Heuristic 1

- Step 2: Insert / remove attributes
 - $(B, D)^+ = A, B, C, D, E, F$ — sufficient so try deleting
 - Delete B: $(D)^+ = D$
 - Delete D: $(B)^+ = B$
 - No more so done

B, D is minimal and thus a key!

Example: Key Heuristic 1

- Step 2: Insert / remove attributes
 - $F^+ = F, A, B, C$ — insufficient so add
 - Add D: $(D, F)^+ = A, B, C, D, E, F \rightarrow$ key!
 - Add E: $(E, F)^+ = A, B, C, E, F$
 - No more so done

Keys are: (A, D), (B, D), and (D, F)!

Finding Keys: Heuristic 2

- Find necessary attributes first
- Find the irreplaceable attributes
 - Attribute is replaceable if it appears in the RHS of some functional dependency
- A key must include every irreplaceable attribute
- Base set is set of all irreplaceable attributes
- Add other attributes to base set until you have a key

Example: Key Heuristic 2

- $R(A, B, C, D, E, F)$
 - $A \rightarrow B, C$
 - $B, D \rightarrow E, F$
 - $F \rightarrow A$
- Step 1: Find irreplaceable attributes and construct base set
Base set = $\{D\}$

Example: Key Heuristic 2

- Step 2: Add other attributes until you have key
 - Add A: $(A, D)^+ = A, B, C, D, E, F \rightarrow \text{key!}$
 - Add B: $(B, D)^+ = A, B, C, D, E, F \rightarrow \text{key!}$
 - Add C: $(C, D)^+ = C, D$
 - Add E: $(D, E)^+ = D, E$
 - Add F: $(D, F)^+ = A, B, C, D, E, F \rightarrow \text{key!}$

Example: Key Heuristic 2

- Step 2: Add other attributes until you have key (do not expand known keys)
 - Add C: $(C, D, E)^+ = C, D, E$
 - No more to add, so done!

Second Normal Form (2NF)

- (Definition) A relation schema R is in 2NF if every non-prime attribute (i.e., not a member of any candidate key) A in R is not partially dependent on any key of R
- Relation is 1NF (attributes are atomic)
- No non-key attribute that is functionally determined by only a (proper) subset of a key

<u>A</u>	<u>B</u>	<u>C</u>	D	E	F	G	H
----------	----------	----------	---	---	---	---	---

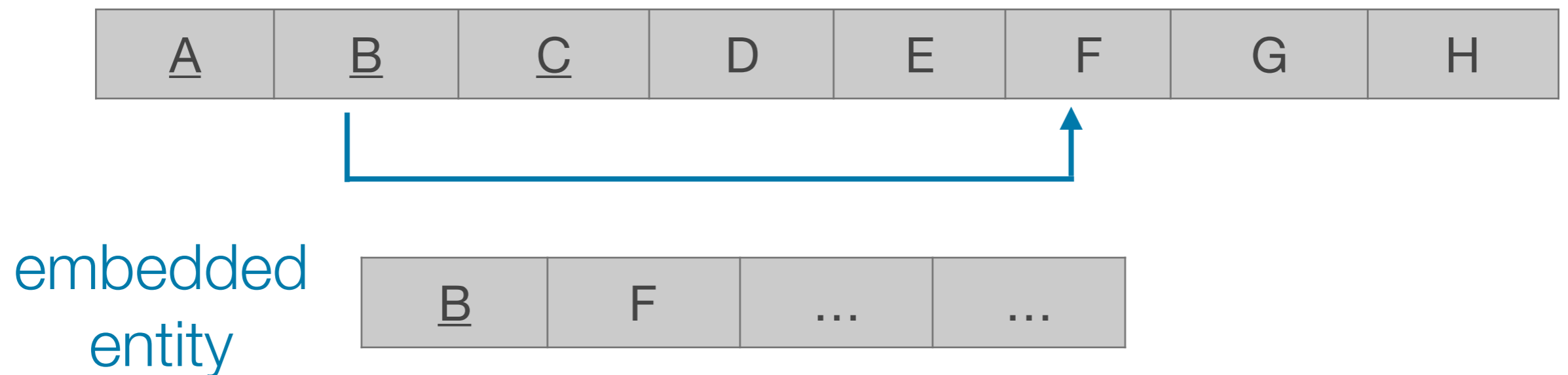
key
(A, B, C)



B \rightarrow F means F is functionally dependent on subset of key \Rightarrow violation of 2NF

2NF Meaning

A relation that violates 2NF contains another embedded autonomous entity



Example: Violation of 2NF

- **EmpProj**(SSN, FName, LName, PNo, PName, Hours)
 - SSN \rightarrow FName, LName
 - PNo \rightarrow PName
 - SSN, PNo \rightarrow Hours
- FName, LName are functionally dependent on SSN
- SSN is subset of a key (SSN, PNo)
- Violation since Employee entity is embedded (SSN, FName, LName)

Decomposition for Normal Form Violations

- Break a relation into two or more relations
- One possibility for **EmpProj**(SSN, FName, LName, PNo, PName, Hours):
 - R1(PNo, PName, Hours)
 - R2(SSN, FName, Lname)
- Another possibility for EmpProj
 - R3(SSN, FName, Lname)
 - R4(SSN, PNo, PName, Hours)

Are these good or bad decompositions?

Decomposition Effect

- Populate the new relations using data of the original relation
- Achieve this by using projection operation on the original relation
- Example:

$$R1 = \pi_{SSN, FName, LName}(\text{EmpProj})$$

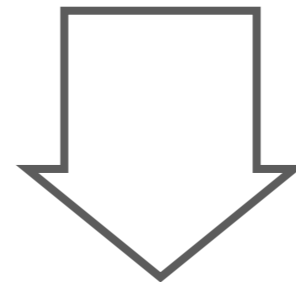
$$R2 = \pi_{PNo, PName, Hours}(\text{EmpProj})$$

Decomposition Effect (2)

- Can we obtain the same information stored in the original relation?
- Reconstruction algorithm:
If ($R1 \cap R2 \neq \emptyset$) {
 reconstruction = $R1 * R2$ // Natural join
} else {
 reconstruction = $R1 \times R2$ // Cartesian product
}

Example: Decomposition Effect

<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5



<u>SSN</u>	FName	LName
111-11-1111	John	Smith
333-33-3333	Jack	Rabbit

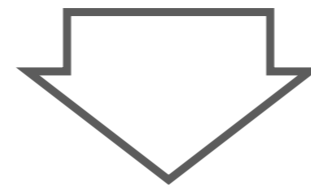
<u>PNo</u>	PName	Hours
pj1	ProjectX	20
pj2	ProjectY	10
pj1	ProjectX	5

Example: Reconstructing After Decomposition

<u>SSN</u>	FName	LName
111-11-1111	John	Smith
333-33-3333	Jack	Rabbit

 X

<u>PNo</u>	PName	Hours
pj1	ProjectX	20
pj2	ProjectY	10
pj1	ProjectX	5



<u>SSN</u>	FName	LName	<u>PNo</u>	PName	Hours
111-11-1111	John	Smith	pj1	ProjectX	20
111-11-1111	John	Smith	pj2	ProjectY	10
111-11-1111	John	Smith	pj1	ProjectX	5
333-33-3333	Jack	Rabbit	pj1	ProjectX	20
333-33-3333	Jack	Rabbit	pj2	ProjectY	10
333-33-3333	Jack	Rabbit	pj1	ProjectX	5

Extraneous tuples that weren't present in original relation!

Decomposition Relation Requirements

- Must be able to obtain all tuples in the original relation R using the reconstruction algorithm
 - Missing tuples means that we have lost information which is unacceptable
- Must not obtain extraneous tuples that were not present in the original relation R using the reconstruction algorithm
 - Invalid information in the relation which is also unacceptable

Lossless Decomposition

- A decomposition of relation R into 2 relations R_1 and R_2 is called lossless if and only if
 $\text{content}(R_1) * \text{content}(R_2) = \text{content}(R)$ or
 $\text{content}(R_1) \times \text{content}(R_2) = \text{content}(R)$
- 2 lemmas that provide needed guidelines to decompose R to guarantee lossless
- Lemma 1: $\text{content}(R) \subseteq \text{content}(R_1) * \text{content}(R_2)$
- Lemma 2: If either $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$, then $\text{content}(R) = \text{content}(R_1) * \text{content}(R_2)$

Example: 2NF via Lemma 2

- **EmpProj**(SSN, FName, LName, PNo, PName, Hours)

- SSN \rightarrow FName, LName

- PNo \rightarrow PName

- SSN, PNo \rightarrow Hours

- At least one violating FD

- SSN \rightarrow FName

- SSN \rightarrow LName

Remove all attributes
functionally dependent
on SSN \Rightarrow compute
closure of SSN

Example: 2NF via Lemma 2

- $R1(SSN^+) = \mathbf{R1}(SSN, FName, LName)$
- $R2(R - R1) = \mathbf{R2}(PNo, PName, Hours)$
- To satisfy lemma 2, add SSN to R2 \Rightarrow
 $\mathbf{R2}(SSN, PNo, PName, Hours)$
- $R1 \cap R2 = SSN$, and $SSN \rightarrow R1$

Are R1 and R2 in the 2NF?

Example: 2NF via Lemma 2

- **R1**(SSN, FName, LName)
 - $SSN \rightarrow FName, FName$ — key = good dependency
- **R2**(SSN, PNo, PName, Hours)
 - $SSN, PNo \rightarrow Hours$ — key = good dependency
 - $PNo \rightarrow PName$ — not key = bad!

Remove all attributes functionally dependent
on PNo => compute closure of PNo

Example: 2NF via Lemma 2

- $R_{21}(PNo^+) = \mathbf{R_{21}}(\underline{PNo}, PName)$
 - $R_{22}(R_2 - R_{21}) = \mathbf{R_{22}}(\underline{SSN}, Hours)$
 - To satisfy lemma 2, add PNo to R22 =>
 $\mathbf{R_{22}}(SSN, \underline{PNo}, Hours)$
 - Resulting decomposition:
 $\mathbf{R_1}(\underline{SSN}, FName, LName)$
 $\mathbf{R_{21}}(\underline{PNo}, PName)$
 $\mathbf{R_{22}}(\underline{SSN}, \underline{PNo}, Hours)$
- Are R1, R21, and R22
in the 2NF?

Example: 2NF Complaint

- **Employee2**(SSN, FName, LName, DNo, DName, MgrSSN)
 - SSN \rightarrow FName, LName, DNo
 - DNo \rightarrow DName, MgrSSN
- Employee2 is 2NF as DNo is not a subset of any key and neither of the functional dependencies violate 2NF criteria

Example: 2NF Complaint

- But...
 - Insert anomaly — adding new department results in NULL values
 - Delete anomaly — deleting an employee may delete information about department
 - Update anomaly — changing department name results in updates of multiple tuples

Transitive Functional Dependency

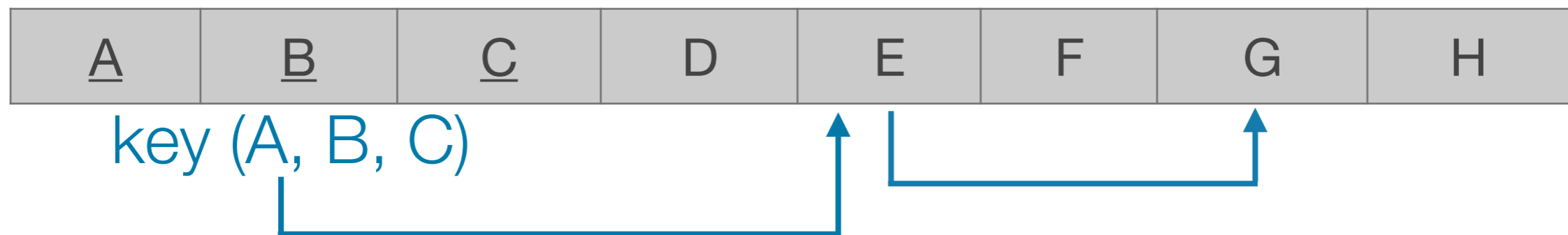
A functional dependency $A \rightarrow B$ is a transitive functional dependency in relation R if there is a set of attributes X such that:

- $A \rightarrow X$
- $X \rightarrow B$
- X is not a super key

Third Normal Form (3NF)

(Definition) A relation schema R is in 3NF if, whenever a nontrivial functional dependency $X \rightarrow A$ holds in R , either (a) X is a super key of R , or (b) A is a prime attribute of R

- R is in 2NF
- Every non-key attribute is non-transitively dependent on all the keys



If $E \rightarrow G$, then transitive dependency $(A, B, C) \rightarrow E \rightarrow G$

Example: 3NF Violation

- **Employee2**(SSN, FName, LName, DNo, DName, MgrSSN)
 - SSN \rightarrow FName, LName, DNo
 - DNo \rightarrow DName, MgrSSN
- Since DNo is not a super key, there is a transitive dependency SSN \rightarrow DNo \rightarrow DName, MgrSSN

Review: 3NF

- A relation R is 3NF if and only if for every functional dependency $X \rightarrow B$ in relation R , one of the following must be true:
 - X is a superkey, or
 - B is a key attribute (part of some key)

Simpler Form of 3NF

- Violation detection: Check every functional dependency $X \twoheadrightarrow B$ for:
 - B is a non-key attribute, and
 - X is not a superkey

Example: 3NF Violation Take 2

Employee2(SSN, FName, LName, DNo, DName, MgrSSN)

- SSN \rightarrow FName, LName, DNo
 - FName, LName, and DNO are non-key attributes \Rightarrow YES
 - SSN is not superkey \Rightarrow NO
 - FD is good

Example: 3NF Violation Take 2

Employee2(SSN, FName, LName, DNo, DName, MgrSSN)

- DNo \rightarrow DName, MgrSSN
- Name and MgrSSN are non-key attributes \Rightarrow YES
- DNo is not superkey \Rightarrow YES
- FD is bad and a 3NF violation

Example: 3NF Decomposition

- Solution: remove the violation by removing X^+ from the original relation
- $R(A, B, C, D, E, F)$
 - $A \twoheadrightarrow B, C, D$
 - $D \twoheadrightarrow E, F$
- Step 1: Find all keys
 - $A^+ = (A, B, C, D, E, F)$

Example: 3NF Decomposition

- Step 2: Is R 2NF?
 - Key(s): A
 - Non-key attributes: B, C, D, E, F
 - Is any of the non-key attributes functionally dependent on subset of (A)? NO
- Relation is 2NF

Example: 3NF Decomposition

- Step 3: Is R 3NF?
 - Key(s): A
 - Non-key attributes: B, C, D, E, F
 - Is any of the non-key attributes functionally dependent on attributes that are not super key? YES!
 - $D \twoheadrightarrow E, F$ where D is not a superkey

Example: 3NF Decomposition

- Step 4: Extract offending functional dependence
 - $D^+ = (D, E, F)$
 - $R1(\underline{D}, E, F)$
 $R2(\underline{A}, B, C, D)$
- Step 5: Check the new relations if they are 3NF?
 - $R1: D \twoheadrightarrow E, F$ doesn't violate 3NF criteria
 - $R2: A \twoheadrightarrow B, C, D$ doesn't violate 3NF criteria

Summary of 1NF, 2NF, 3NF

Normal Form	Test	Normalization (Remedy)
1NF	Relation should have no multi-valued attributes or nested relations	Form new relation for each multivalued attribute or nested relation
2NF	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key	Decompose and set up a new relation for each partial key with its dependent attributes using lossless decomposition
3NF	Relation should not have a nonkey attribute functionally determined by another nonkey attribute	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attributes

Boyce-Codd Normal Form (BCNF)

(Definition) A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \twoheadrightarrow A$ holds in R , then X is a superkey of R

- Difference from 3NF: 3NF allows A to be prime attribute
- Every relation in BCNF is also in 3NF
- Most relation schemas that are in 3NF are also BCNF but not all
 - Example: $R(\underline{A}, \underline{B}, C)$
 - $A, B \twoheadrightarrow C$
 - $C \twoheadrightarrow A$

Example: BCNF Violation

- TSS(Teacher, Subject, Student)
 - Student, Subject \rightarrow Teacher
 - Teacher \rightarrow Subject
- Keys in TSS
 - (Student, Subject)
 - (Student, Teacher)

Example: BCNF Violation

- Is TSS in the 3NF?
 - Student, Subject \rightarrow Teacher — superkey = okay
 - Teacher \rightarrow Subject
 - Is teacher a superkey? NO
 - Is subject a key attribute (part of key)? YES — okay

Even though TSS is 3NF — duplicate information is stored in relation (teacher, subject)

Example: BCNF Violation

- Problem arises when 2 or more composite keys are in a relation
- Is relation BCNF?
 - Student, Subject \rightarrow Teacher — superkey = okay
 - Teacher \rightarrow Subject
Teacher is not a superkey \Rightarrow BCNF violation!
- Solution: Decompose the violating FD
 - R1(Teacher, Subject)
 - R2(Teacher, Student)

Example: BCNF Normalization

- Relation $R(A, B, C, D, E, F, G, H, I, J, K, L, M)$
 - $A \rightarrow B, C, D, E$
 - $E \rightarrow F, G, H$
 - $I \rightarrow J$
 - $A, I \rightarrow K$
 - $A, L \rightarrow M$

Is Normalization Always Good?

- Example: Suppose A and B are always used together but normalization says they should be in different tables
- Decomposition might produce unacceptable performance loss (always joining tables)
- For example, data warehouses are huge historical DBs that are rarely updated after creation — joins are expensive or impractical
- Everyday DBs: aim for BCNF, settle for 3NF!

Final Note on Normalization

- Is 3NF or BCNF better?
 - 3NF can be lossless and preserves all functional dependencies
 - BCNF is guaranteed to be lossless but may not preserve all functional dependencies

Final Note on Normalization

- Ultimate goal:
 - BCNF, lossless, preserves all functional dependencies
- Next ultimate goal:
 - 3NF, lossless, preserves all functional dependencies

Database Design: Recap

- FD
- Closure algorithm to find keys
- Lossless decomposition
- 1NF, 2NF, 3NF, BCNF

