Database Design: Normalization

Agenda

1. Database Design
2. Normal forms & functional dependencies
3. Finding functional dependencies
4. Closures, superkeys & keys

Design Theory

- The biggest problem needed to be solved in database is ***data redundancy.***
- Why data redundancy is the problem? Because it causes:
  - Insert Anomaly
  - Update Anomaly
  - Delete Anomaly

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

- Achieved by **Data Normalization**, a process of analyzing a relation to ensure that it is well formed.

- Normalization involves **decomposing** relations with anomalies to produce smaller well structured relations.

- If a relation is normalized (or well formed), rows can be inserted, deleted and modified without creating anomalies.

Data Anomalies & Constraints

Constraints Prevent (some) Anomalies in the Data

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

Student	Course	Room
Mary	CSC261	101
Joe	CSC261	101
Sam	CSC261	101
..	..	..

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	CSC261	101
Joe	CSC261	703
Sam	CSC261	101
..	..	..

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***:

Student	Course	Room
Mary	CSC261	B01
Joe	CSC261	B01
Sam	CSC261	B01
..	..	..

...	CSC461	703

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

Constraints Prevent (some) Anomalies in the Data

Student	Course
Mary	CSC261
Joe	CSC261
Sam	CSC261
..	..

Course	Room
CSC261	101
CSC257	601

Is this form better?

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

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

Database Anomalies
Example 2

ACTIVITY Relation

StudentID	Activity	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Anomalies are problems caused by **bad database design**.

Example:

ACTIVITY(<u>StudentID</u>, <u>Activity</u>, Fee)

An **insertion anomaly** occurs when a row cannot be added to a relation, because not all data are available (or one has to invent "dummy" data)

- **Example**: we want to store that scuba diving costs $175, but have no place to put this information until a student takes up scuba-diving (unless we create a fake student)

A **deletion anomaly** occurs when data is deleted from a relation, and other critical data are unintentionally lost

- **Example**: if we delete the record with StudentID = 100, we forget that skiing costs $200

An **update anomaly** occurs when one must make many changes to reflect the modification of a single datum

- **Example**: if the cost of swimming changes, then all entries with swimming Activity must be changed too

Cause of Anomalies

Anomalies are primarily **caused** by:

1. **Data redundancy**: replication of the same field in multiple tables, other than foreign keys
2. **Functional dependencies** including:
   - Partial dependency
   - Transitive dependency
   - Multi-value dependency

Functional Dependencies

Functional Dependencies for Dummies

- A relationship between **attributes** where one attribute (or group of attributes) **determines** the value of another attribute (or group of attributes) in the **same** table.

- Example:

  SSN uniquely identify any Person

(SSN) → (First Name, Last Name)

Candidate Keys/Primary Keys and Functional Dependencies

By definition:

- A candidate key of a relation functionally determines **all** other **non-key** attributes in the row.

Implies:

- A primary key of a relation functionally determines all other non-key attributes in the row.

EmployeeID → (EmployeeName, EmpPhone)

Functional Dependency

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:

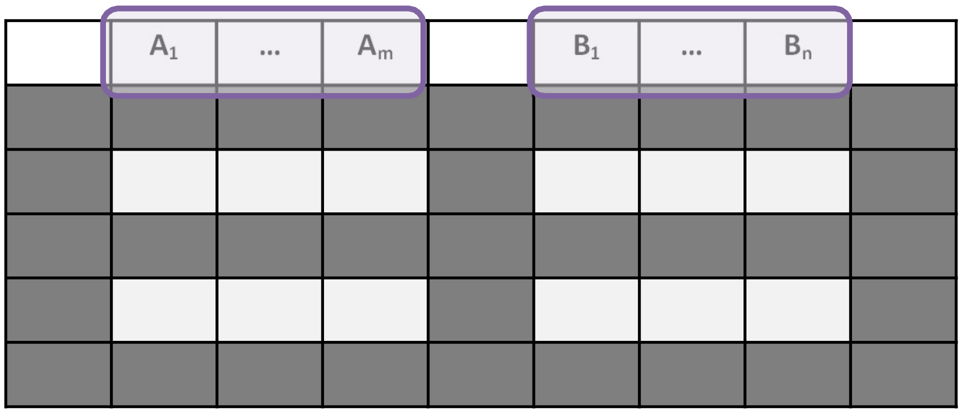
$$t_1[A] = t_2[A] \text{ implies } t_1[B] = t_2[B]$$ and we

call A → B a **functional dependency**

A → B means that
"whenever two tuples agree on A then they agree on B."

A	It is a **determinant** set.
B	It is a **dependent** attribute.
{A → B}	A **functionally determines** B. B is a functionally dependent on A.

A Picture of FDs



Defn (again):

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

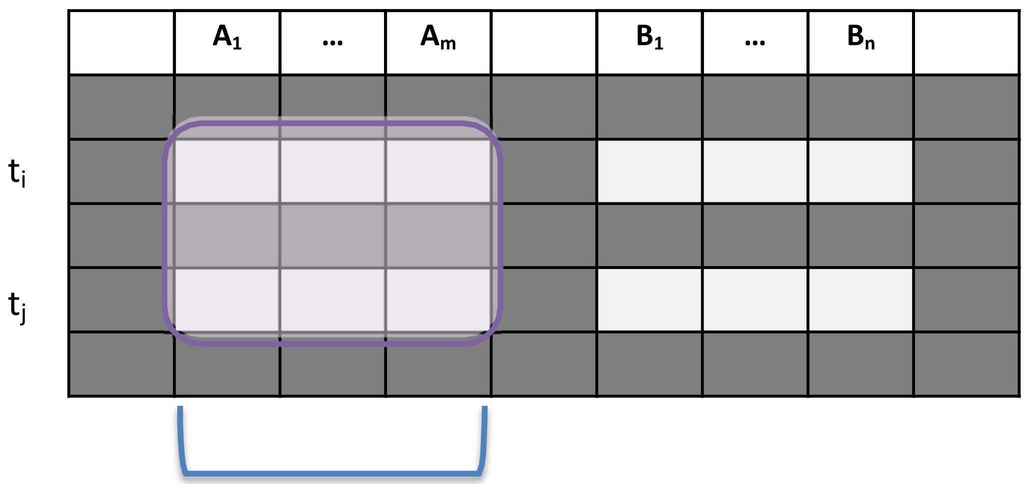
A Picture of FDs

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:

A Picture of FDs



If t1, t2 agree here..

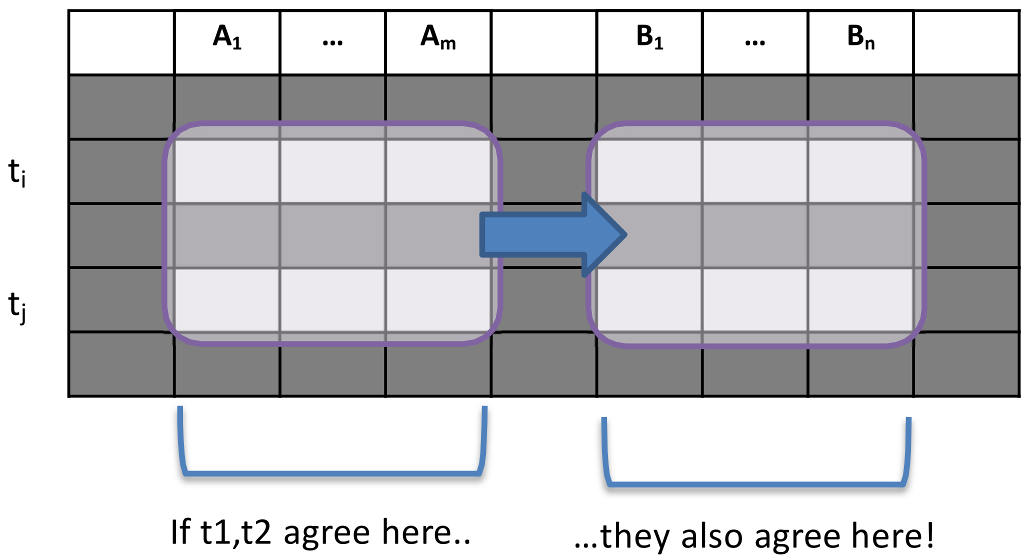
<u>Defn (again):</u>

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

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

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

A Picture of FDs



<u>Defn (again):</u>
Given attribute sets $\mathbf{A=\{A_1,\ldots,A_m\}}$ and $\mathbf{B = \{B_1,\ldots B_n\}}$ in **R,**

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

<u>if</u> $t_i[A_1] = t_j[A_1]$ AND $t_i[A_2]=t_j[A_2]$ AND … AND $t_i[A_m] = t_j[A_m]$

<u>then</u> $t_i[B_1] = t_j[B_1]$ 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?**

1. Start with some relational schema (e.g., design by ER diagram)

2. Find out its functional dependencies (FDs)

3. 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 not others.
- Part of the schema, helps define a valid *instance*.

Student	Course	Room
Mary	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

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

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

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	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

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

More Examples

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

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

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}

ACTIVITY

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 hold on this instance:

{A } → {C }
{A,B } → {C }
{E } → {D }

Armstrong inference rules

- Armstrong's Axioms is a set of rules.
- It provides a simple technique for reasoning about functional dependencies.
- It was developed by William W. Armstrong in 1974.
- It is used to infer all the functional dependencies on a relational database.

Armstrong inference rules

A. Primary Rules:

Rule 1	**Reflexivity** If A is a set of attributes and B is a subset of A, then A holds B. { A → B } (If B $\subseteq$ A, then A→B)
Rule 2	**Augmentation** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} (If A→B, then AC→BC for any C) It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A → C} A holds B {A → B} means that A functionally determines B.

Armstrong inference rules

B. Secondary Rules:

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If {A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If {A → BC}, then {A → B} and {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If {A → B} and {BC → D}, then {AC → D}

Armstrong inference rules

B. Secondary Rules:

Rule 4	**Self determination** {A → A} for any A. This follows directly from the axiom of reflexivity.
Rule 5	**Composition** If A holds B and X holds Y, then AX holds BY. If {A → B} and {X → Y}, then {AX → BY}
Rule 6	**Extensivity** The following property is a special case of augmentation when C = A If A holds C, then A holds AC. If {A → C} then {A → AC}

Armstrong inference rules

Axioms are both

Sound:

when applied to a set of functional dependencies they only produce dependency tables that belong to the transitive closure of that set

Complete:

can produce all dependency tables that belong to the transitive closure of the set

Armstrong inference rules

Three last rules can be derived from the first three (the axioms)

Let us look at the ***union rule***:

if X→Y and X→Z, the X→YZ

Using the first three axioms, we have:

if X→Y, then XX→XY same as X→XY (2nd)

if X→Z, then YX→YZ same as XY→YZ (2nd)

if X→XY and XY→YZ, then X→YZ (3rd)

Example:

Consider relation E = (P, Q, R, S, T, U) having set of Functional Dependencies (FD).

P → Q	P → R
QR → S	Q → T
QR → U	PR → U

Calculate some members of axioms are as follows:

1. P → T
2. PR → S
3. QR → SU
4. PR → SU

Axioms:

Reflexivity: if Y ⊆ X, then X→Y

Augmentation: if X→Y, then WX→WY

Transitivity: if X→Y and Y→Z, then X→Z

Derived Rules:

Union: if X→Y and X→Z, the X→YZ

Decomposition: if X→YZ, then X→Y and X→Z

Pseudo transitivity: if X→Y and WY→Z, then XW→Z

Axioms:

Reflexivity: if Y⊆ X, then X→Y

Augmentation: if X→Y, then WX→WY

Transitivity: if X→Y and Y→Z, then X→Z

Derived Rules:

Union: if X→Y and X→Z, the X→YZ

Decomposition: if X→YZ, then X→Y and X→Z

Pseudo transitivity: if X→Y and WY→Z, then XW→Z

Solution:

1. P → T

In the FD set, P → Q and Q → T

So, Using Transitive Rule: If {A → B} and {B → C}, then {A → C}

∴ If P → Q and Q → T, then **P → T.**

2. PR → S

In the above FD set, P → Q

As, QR → S

So, Using Pseudo Transitivity Rule: If{A → B} and {BC → D}, then {AC → D}

∴ If P → Q and QR → S, **then PR → S.**

3. QR → SU

In above FD set, QR → S and QR → U

So, Using Union Rule: If{A → B} and {A → C}, then {A → BC}

∴ If QR → S and QR → U, **then QR → SU.**

4. PR → SU

So, Using Pseudo Transitivity Rule: If{A → B} and {BC → D}, then {AC → D}

∴ If PR → S and PR → U, **then PR → SU.**

Trivial Functional Dependency

Trivial	If A holds B {A → B}, where B is a subset of A, then it is called a **Trivial Functional Dependency**. Trivial always holds Functional Dependency.
Non-Trivial	If A holds B {A → B}, where B is **not** a subset A, then it is called as a **Non-Trivial Functional Dependency.**

Normalization

https://www.youtube.com/watch?v=UrYLYV7WSHM
https://www.youtube.com/watch?v=l5DCnCzDb8g

Normalization

- **Normalization** is the process of removing **redundant** data from your tables to improve storage efficiency, data integrity, and scalability.
- Normalization generally involves **splitting** existing tables into multiple ones, which must be re-joined or linked each time a query is issued.
- Why normalization?
  - The relation derived from the user view or data store will most likely be **unnormalized**.
  - The problem usually happens when an existing system uses unstructured file, e.g. in MS Excel.

Unnormalized Form (table) Example

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-03	31-Aug-04	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-04	1-Sep-05	450	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-02	10-June-03	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-03	1-Dec-04	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-05	10-Aug-06	450	CO93	Tony Shaw

Normalization Example

- (Student ID) → (Student Name, DormName, DormCost)

- However, if
  - (DormName) → (DormCost)

Then, DormCost should be put into its own relation, resulting in:

(Student ID) → (Student Name, DormName)

(DormName) → (DormCost)

Normalization Example

- (AttorneyID, ClientID) → (ClientName, MeetingDate, Duration)
- However, if
  - ClientID → ClientName
- Then: ClientName should be in its own relation:
- (AttorneyID, ClientID) → (MeetingDate, Duration)
- (ClientID) → (ClientName)

Steps of Normalization

- ✓ First Normal Form (1NF)
- ✓ Second Normal Form (2NF)
- ✓ Third Normal Form (3NF)
- ✓ Boyce-Codd Normal Form (BCNF)
- ✓ Fourth Normal Form (4NF)
- ✓ Fifth Normal Form (5NF)
- ✓ Domain Key Normal Form (DKNF)

In practice, 1NF, 2NF, 3NF, and BCNF are enough for database.

Normal Forms

- 1st Normal Form (1NF) = All tables are flat
- **2nd Normal Form (2NF)**
- **3rd Normal Form (3NF)**
- **Boyce-Codd Normal Form (BCNF)**

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

- *4th and 5th Normal Forms = see text books*

First Normal Form (1NF)

The official qualifications for 1NF are:

1. Each **attribute name** must be unique.
2. Each **attribute value** must be single.
3. Each **row** must be unique.
4. There is **no repeating groups**.

Additional:

Choose a primary key.

Reminder:

A primary key is ***unique***, ***not null***, ***unchanged***. A primary key can be either an attribute or combined attributes.

1st Normal Form (1NF)

Student	Courses
Mary	{CS145, CS229}
Joe	{CS145, CS106}
…	…

Violates 1NF.

Student	Courses
Mary	CS145
Mary	CS229
Joe	CS145
Joe	CS106

In 1st NF

1NF Constraint: Types must be atomic!

First Normal Form (1NF) (Cont.)

Example of a table not in 1NF :

Group	Topic	Student	Score	
Group A	Intro MongoDB	Sok San	18	marks
		Sao Ry	17	marks
Group B	Intro MySQL	Chan Tina	19	marks
		Tith Sophea	16	marks

It violates the 1NF because:

- Attribute values are not single.
- Repeating groups exists.

First Normal Form (1NF) (Cont.)

- After eliminating:

Group	Topic	Family Name	Given Name	Score
A	Intro MongoDB	Sok	San	18
A	Intro MongoDB	Sao	Ry	17
B	Intro MySQL	Chan	Tina	19
B	Intro MySQL	Tith	Sophea	16

- Now it is in 1NF.
  However, it might still violate 2NF and so on.

Functional Dependencies

We say an attribute, B, has a *functional dependency* on another attribute, A, if for any two records, which have the same value for A, then the values for B in these two records must be the same. We illustrate this as:

A → B *(read as: A determines B or B depends on A)*

Employee_name	Project	Email_address
Joe San	POS Mart Sys	soksan@yahoo.com
Sao Ry	Univ Mgt Sys	sao@yahoo.com
Joe San	Web Redesign	soksan@yahoo.com
Chan Sokna	POS Mart Sys	chan@gmail.com
Sao Ry	DB Design	sao@yahoo.com

Employee_name → Email_address

Functional Dependencies (cont.)

EmpNum	EmpEmail	EmpFname	EmpLname
123	jdoe@abc.com	John	Doe
456	psmith@abc.com	Peter	Smith
555	alee1@abc.com	Alan	Lee
633	pdoe@abc.com	Peter	Doe
787	alee2@abc.com	Alan	Lee

If EmpNum is the PK then the FDs:

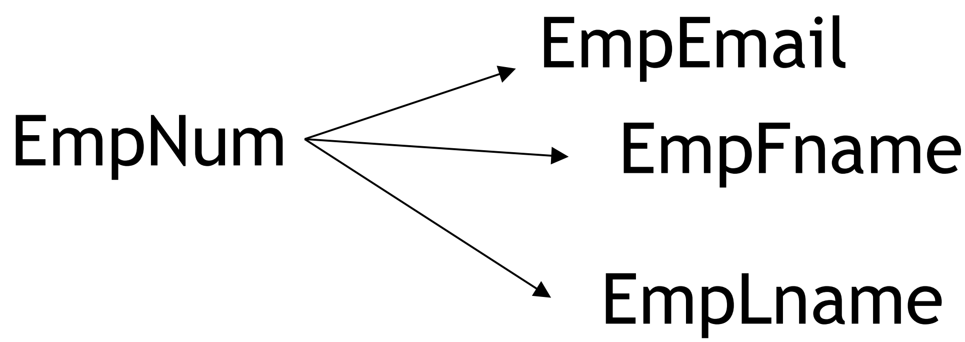
EmpNum → EmpEmail, EmpFname, EmpLname

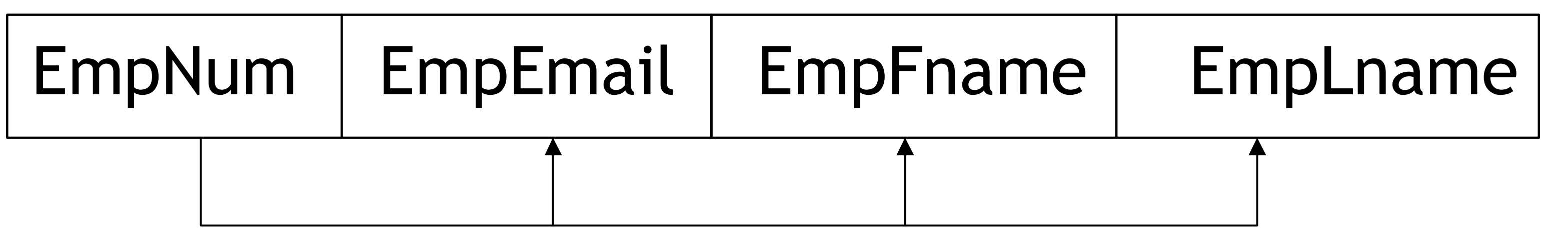
must exist.

Functional Dependencies (cont.)

EmpNum → EmpEmail, EmpFname, EmpLname

3 different ways you might see FDs depicted





Determinant

Functional Dependency

EmpNum → EmpEmail

Attribute on the left hand side is known as the ***determinant***

- EmpNum is a ***determinant*** of EmpEmail

Second Normal Form (2NF)

The official qualifications for 2NF are:

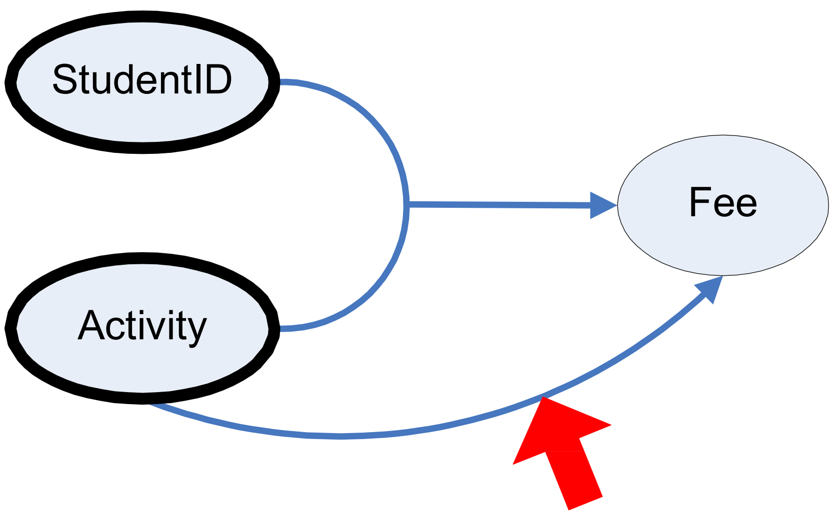
1. A table is already in 1NF.
2. All non-key attributes are fully dependent on the primary key.

All **partial dependencies** are removed to place in another table.

Partial Dependencies

- Partial dependency is a functional dependency whose determinant is part of the primary key (but not all of it)
- Example:

ACTIVITY(StudentID, Activity, Fee)



StudentID	**Activity**	**Fee**
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Example of a table not in 2NF:

CourseID	SemesterID	Num Student	Course Name
IT101	201301	25	Database
IT101	201302	25	Database
IT102	201301	30	Web Prog
IT102	201302	35	Web Prog
IT103	201401	20	Networking

Primary Key

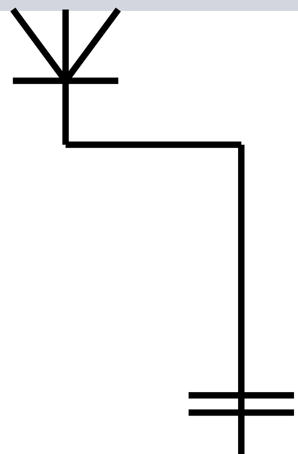
The *Course Name* depends on only *CourseID*, a part of the primary key not the whole primary {*CourseID*, *SemesterID*}. It's called **partial dependency**.

Solution:

*Remove **CourseID** and **Course Name** together to create a new table.*

CourseID	Course Name
IT101	Database
IT101	Database
IT102	Web Prog
IT102	Web Prog
IT103	Networking

CourseID	SemesterID	Num Student
IT101	201301	25
IT101	201302	25
IT102	201301	30
IT102	201302	35
IT103	201401	20



Done?
Oh no, it is still not in 1NF yet.
Remove the repeating groups too.
Finally, connect the relationship.

CourseID	Course Name
IT101	Database
IT102	Web Prog
IT103	Networking

Third Normal Form (3NF)

The official qualifications for 3NF are:

1. A table is already in 2NF.
2. Nonprimary key attributes do not depend on other nonprimary key attributes
   (i.e. no **transitive dependencies**)

   All transitive dependencies are removed to place in another table.

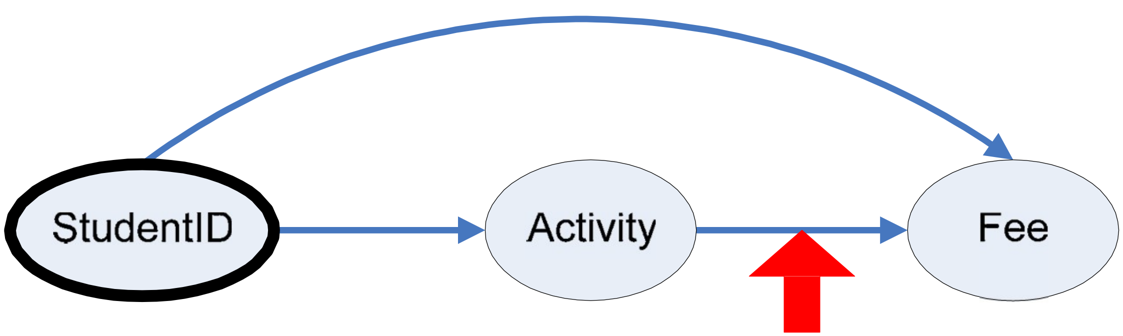
Transitive Dependencies

Transitive dependency is a functional dependency whose determinant is not the primary key, part of the primary key, or a candidate key.

Transitive functionality is a functional dependency in which a non-key attribute is determined by another non-key attribute.

Example:

ACTIVITY(StudentID, Activity, Fee)



StudentID	Activity	Fee
100	Skiing	200
100	Golf	65
175	Squash	50
175	Swimming	50
200	Swimming	50
200	Golf	65

Example of a Table not in 3NF:

StudyID	CourseName	TeacherName	TeacherTel
1	Database	Sok Piseth	012 123 456
2	Database	Sao Kanha	0977 322 111
3	Web Prog	Chan Veasna	012 412 333
4	Web Prog	Chan Veasna	012 412 333
5	Networking	Pou Sambath	077 545 221

Primary Key

The *TeacherTel* is a nonkey attribute, and the *TeacherName* is also a nonkey attribute. But *TeacherTel* depends on *TeacherName*.
It is called **transitive dependency**.

Solution:

*Remove **Teacher Name** and **TeacherTel** together to create a new table.*

Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Chan Veasna	012 412 333
Pou Sambath	077 545 221

Done?
Oh no, it is still not in 1NF yet.
Remove Repeating row.

StudyID	Course Name	T.ID
1	Database	T1
2	Database	T2
3	Web Prog	T3
4	Web Prog	T3
5	Networking	T4

Teacher Name	Teacher Tel
Sok Piseth	012 123 456
Sao Kanha	0977 322 111
Chan Veasna	012 412 333
Pou Sambath	077 545 221

ID	Teacher Name	Teacher Tel
T1	Sok Piseth	012 123 456
T2	Sao Kanha	0977 322 111
T3	Chan Veasna	012 412 333
T4	Pou Sambath	077 545 221

Note about primary key:
-In theory, you can choose *TeacherName* to be a primary key.
-But in practice, you should add *TeacherID* as the primary key.

Boyce Codd Normal Form (BCNF) – 3.5NF

The official qualifications for BCNF are:

1. A table is already in 3NF.
2. All determinants must be superkeys.

All determinants that are not superkeys are removed to place in another table.

K is a *superkey* for relation *R* if *K* functionally determines all of *R*.

K is a *(candidate)key* for *R* if *K* is a superkey, but no proper subset of *K* is a superkey.

Boyce Codd Normal Form (BCNF) (Cont.)

- Example of a table not in BCNF:

<u>Student</u>	<u>Course</u>	Teacher
Sok	DB	John
Sao	DB	William
Chan	E-Commerce	Todd
Sok	E-Commerce	Todd
Chan	DB	William

- Key: {Student, Course}
- Functional Dependency:
  - {Student, Course}→Teacher
  - Teacher → Course
- Problem: ***Teacher*** is not a superkey but determines ***Course***.

Student	Course
Sok	DB
Sao	DB
Chan	E-Commerce
Sok	E-Commerce
Chan	DB

Solution: Decouple a table contains **Teacher** and **Course** from original table **(Student, Course**). Finally, connect the new and old table to third table contains ***Course***.

Course
DB
E-Commerce

Course	Teacher
DB	John
DB	William
E-Commerce	Todd

Forth Normal Form (4NF)

The official qualifications for 4NF are:

1. A table is already in BCNF.
2. A table contains no multi-valued dependencies.

- Multi-valued dependency: MVDs occur when two or more independent multi valued facts about the same attribute occur within the same table.

  A ->-> B (B multi-valued depends on A)

Example: MVD

Customer(name, addr, phones, drinksLiked)

A drinker's phones are independent of the drinks they like.

name->->phones and name ->->drinksLiked.

Thus, each of a drinker's phones appears with each of the drinks they like in all combinations.

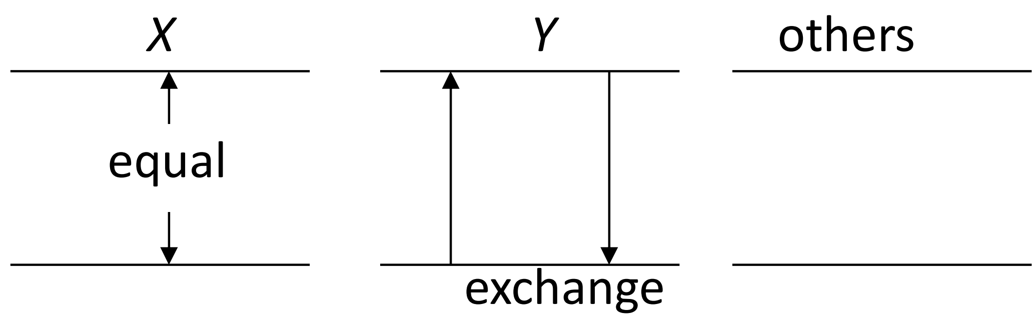
Tuples Implied by **name->->phones**

If we have tuples:

name	addr	phones	drinksLiked
sue	a	p1	d1
sue	a	p2	d2
sue	a	p2	d1
sue	a	p1	d2

Then these tuples must also be in the relation.

Picture of MVD X ->->Y



Forth Normal Form (4NF) (Cont.)

- Example of a table not in 4NF:

Student	Major	Hobby
Sok	IT	Football
Sok	IT	Volleyball
Sao	IT	Football
Sao	Med	Football
Chan	IT	NULL
Puth	NULL	Football
Tith	NULL	NULL

- Key: {Student, Major, Hobby}
- MVD: Student -->> Major, Hobby

Solution: Decouple to each table contains MVD. Finally, connect each to a third table contains ***Student***.

<u>Student</u>
Sok
Sao
Chan
Puth
Tith

<u>Student</u>	<u>Major</u>
Sok	IT
Sao	IT
Sao	Med
Chan	IT
Puth	NULL
Tith	NULL

<u>Student</u>	<u>Hobby</u>
Sok	Football
Sok	Volleyball
Sao	Football
Chan	NULL
Puth	Football
Tith	NULL

Fifth Normal Form (5NF)

The official qualifications for 5NF are:

1. A table is already in 4NF.
2. The attributes of multi-valued dependencies are not related.

Fifth Normal Form (5NF) (Cont.)

- Example of a table not in 5NF:

Seller	Company	Product
Sok	MIAF Trading	Zenya
Sao	Coca-Cola Corp	Coke
Sao	Coca-Cola Corp	Fanta
Sao	Coca-Cola Corp	Sprite
Chan	Angkor Brewery	Angkor Beer
Chan	Cambodia Brewery	Cambodia Beer

- Key: {Seller, Company, Product}
- MVD: Seller -->> Company, Product
- *Product* is related to *Company*.

Seller
Sok
Sao
Chan

1 M

Seller	Company
Sok	MIAF Trading
Sao	Coca-Cola Corp
Chan	Angkor Brewery
Chan	Cambodia Brewery

M 1

Company
MIAF Trading
Coca-Cola Corp
Angkor Brewery
Cambodia Brewery

1

M

Seller	Product
Sok	Zenya
Sao	Coke
Sao	Fanta
Sao	Sprite
Chan	Angkor Beer
Chan	Cambodia Beer

1

M

Product
Zenya
Coke
Fanta
Sprite
Angkor Beer

1

M

Company	Product
MIAF Trading	Zenya
Coca-Cola Corp	Coke
Coca-Cola Corp	Fanta
Coca-Cola Corp	Sprite
Angkor Brewery	Angkor Beer
Cambodia Brewery	Cambodia Beer

M

FINDING FUNCTIONAL DEPENDENCIES

What you will learn about in this section

1. “Good” vs. “Bad” FDs: Intuition

2. Finding FDs

3. Closures

“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”*

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	CS145	B01
Joe	CS145	B01
Sam	CS145	B01
..	..	..

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)
3. Use these to **design a better schema**
    1. 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?

Finding Functional Dependencies

<u>Example:</u>

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

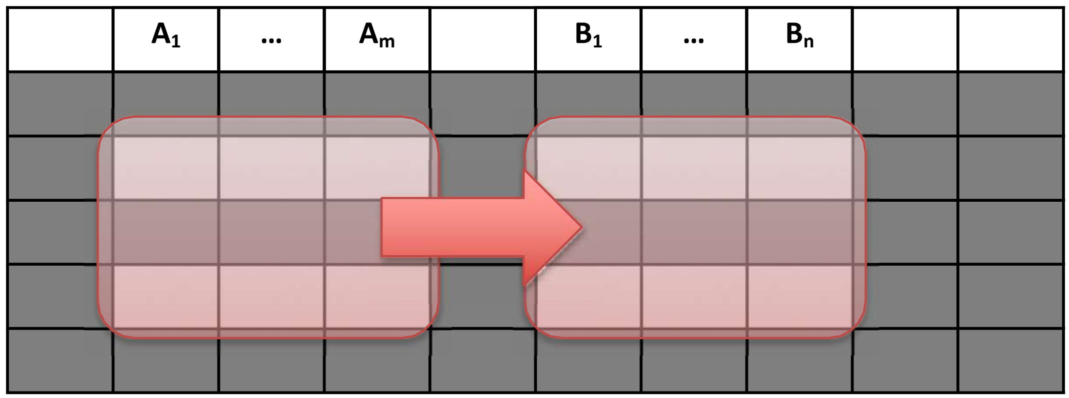
Equivalent to asking: Given a set of FDs, $F = \{f_1, \ldots f_n\}$, does an FD g hold?

Inference problem: How do we decide?

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

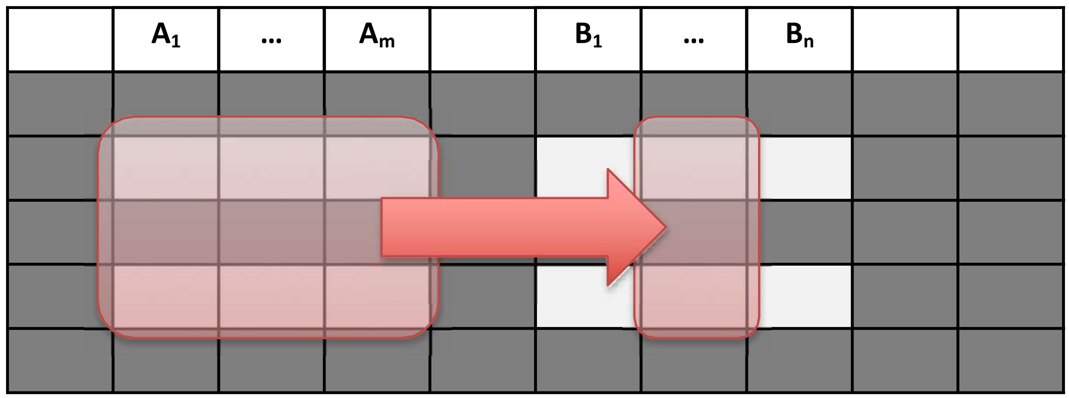
1. **Split/Combine,**
2. **Reduction, and**
3. **Transitivity…** *ideas by picture*

1. Split/Combine (Decomposition & Union Rule)



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

1. Split/Combine (Decomposition & Union Rule)

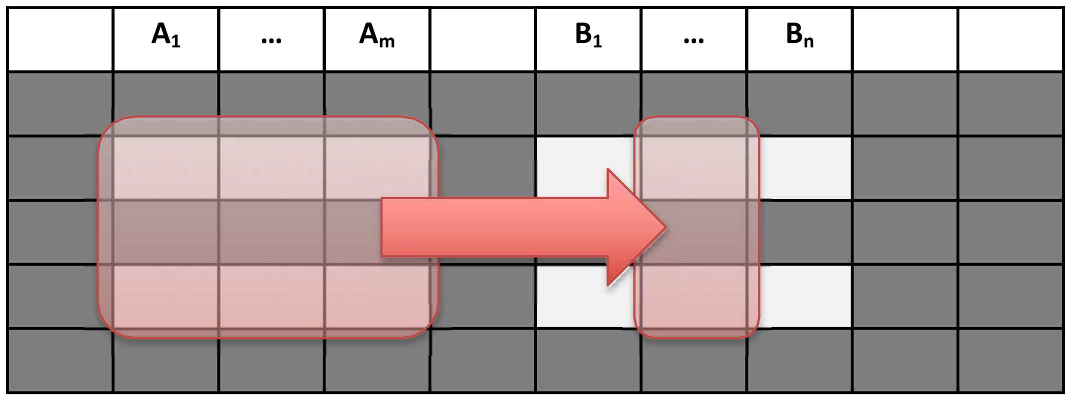


$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

… is equivalent to the following *n* FDs…

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

1. Split/Combine (Decomposition & Union Rule)

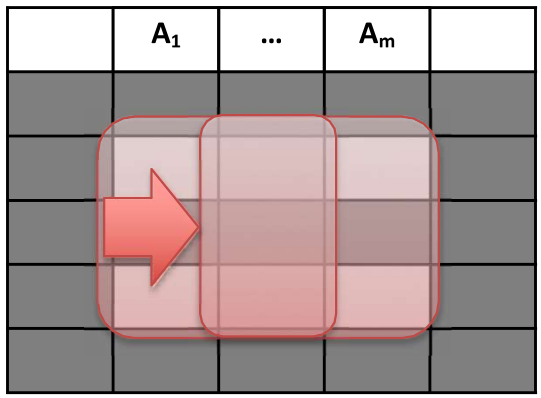


And vice-versa, $A_1,\ldots,A_m \rightarrow B_i$ for $i=1,\ldots,n$

… is equivalent to …

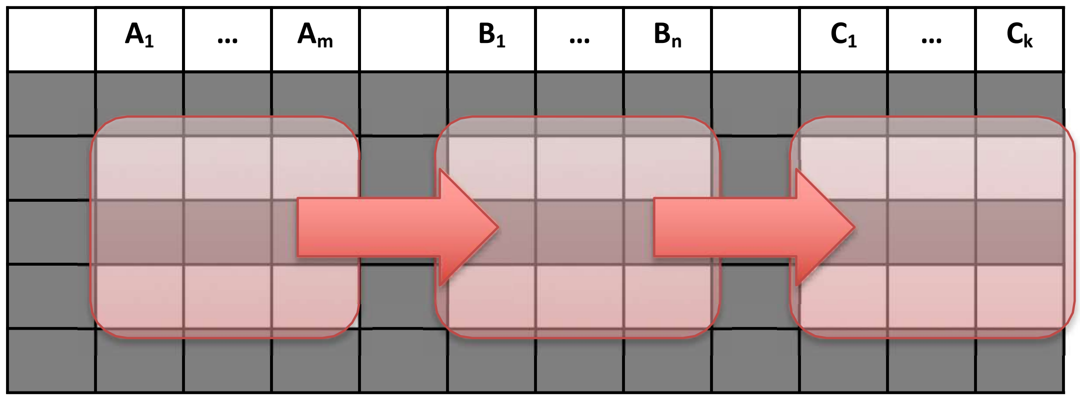
$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

2. Reduction/Trivial (Reflexive Rule)



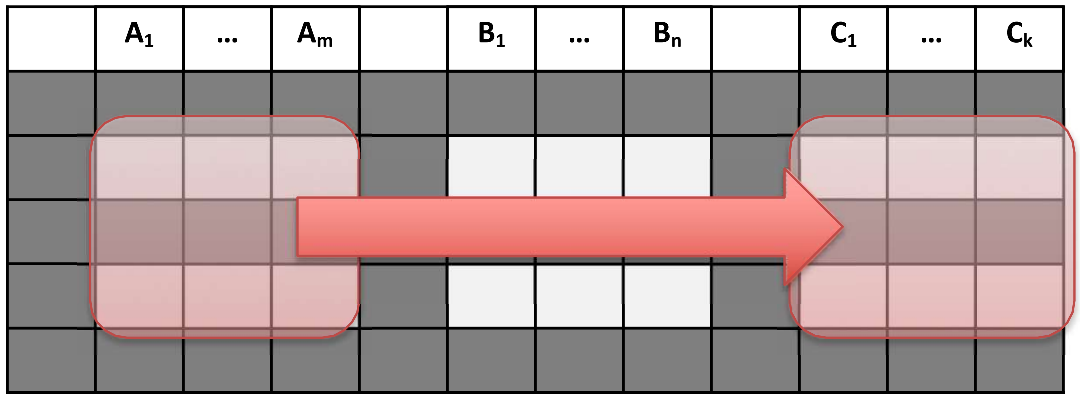
$A_1,\ldots,A_m \rightarrow A_j$ for any $j=1,\ldots,m$

3. Transitive Rule



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

3. Transitive Rule

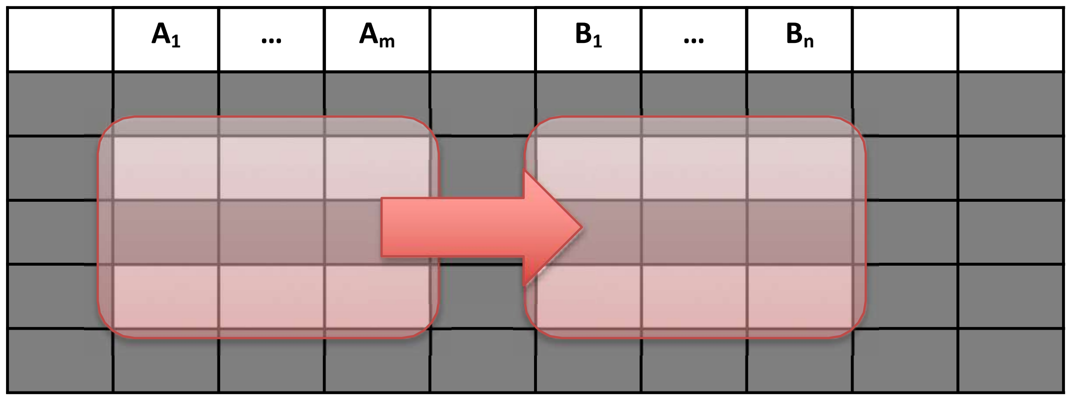


$A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$ and
$B_1,\ldots,B_n \rightarrow C_1,\ldots,C_k$

implies

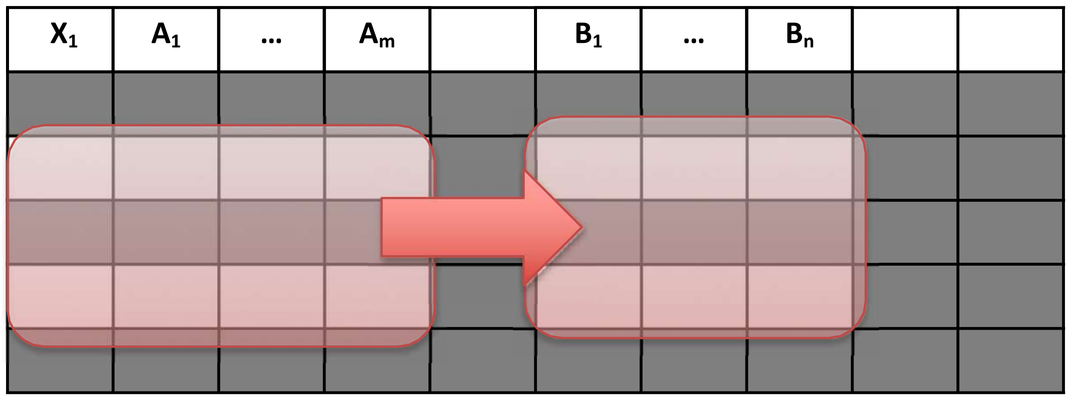
$A_1,\ldots,A_m \rightarrow C_1,\ldots,C_k$

Augmentation Rule



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

Augmentation Rule



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

implies

$X_1, A_1, \ldots, A_m \rightarrow B_1,\ldots,B_n$

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

Provided FDs:

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

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}	?

Which / how many other FDs hold?

Finding Functional Dependencies

Provided FDs:

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

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)

Can we find an algorithmic way to do this?

Yes. But we need to learn about closures before that!

Closures

Closure of a set of Attributes

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

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 $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$ **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 $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    then add C to X.
Return X as X^+

F =

```
{name} → {color}

{category} → {dept}

{color, category} → {price}
```

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

Closure Algorithm

Start with X = $\{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
  if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    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}
```

Closure Algorithm

Start with $X = \{A_1, \ldots, A_n\}$, FDs F.
Repeat until X doesn't change;
do:
if $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
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 $\{B_1, \ldots, B_n\} \rightarrow C$ is in F **and** $\{B_1, \ldots, B_n\} \subseteq X$:
    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)

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

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

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

EXAMPLE

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

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

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

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

EXAMPLE

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

$\{A,B\} \rightarrow \{C\}$
$\{A,D\} \rightarrow \{E\}$
$\{B\} \rightarrow \{D\}$
$\{A,F\} \rightarrow \{B\}$

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

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

3. CLOSURES, SUPERKEYS & KEYS

What you will learn about in this section

1. Closures
2. Superkeys & Keys

Why Do We Need the Closure?

- With closure we can find all FD's easily

- To check if $X \rightarrow A$

1. Compute X^+

2. Check if $A \in X^+$

Note here that **X** is a *set* of attributes, but **A** is a *single* attribute. Why does considering FDs of this form suffice?

Recall the **Split/combine** rule:
$X \rightarrow A_1, \ldots, X \rightarrow A_n$
implies
$X \rightarrow \{A_1, \ldots, A_n\}$

Using Closure to Infer ALL FDs

Example:

Given F =

$\{A,B\} \rightarrow C$
$\{A,D\} \rightarrow B$
$\{B\} \rightarrow D$

Step 1: Compute X^+, for every set of attributes X:

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

No need to compute these- why?

We did not include {B,C}, {B,D}, {C,D}, {B,C,D} to save some space.

Using Closure to Infer ALL FDs

Example:

Given F =

{A,B}	→	C
{A,D}	→	B
{B}	→	D

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \varnothing$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$,
$\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$,
$\{A,C,D\} \rightarrow \{B\}$

Using Closure to Infer ALL FDs

Example:
Given F =

{A,B}	→	C
{A,D}	→	B
{B}	→	D

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs $X \rightarrow Y$, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

$\{A,B\} \rightarrow \{C,D\}$, $\{A,D\} \rightarrow \{B,C\}$, $\{A,B,C\} \rightarrow \{D\}$, $\{A,B,D\} \rightarrow \{C\}$, $\{A,C,D\} \rightarrow \{B\}$

"Y is in the closure of X"

Using Closure to Infer ALL FDs

Example:

Given F =

{A,B} → C
{A,D} → B
{B} → D

Step 1: Compute X^+, for every set of attributes X:

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

Step 2: Enumerate all FDs X → Y, s.t. $Y \subseteq X^+$ and $X \cap Y = \emptyset$:

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

The FD X → Y is non-trivial

Superkeys and Keys

Keys and Superkeys

A **superkey** is a set of attributes $\mathbf{A_1, \ldots, A_n}$ s.t. for *any other* attribute **B** in R, we have $\{\mathbf{A_1, \ldots, A_n}\} \rightarrow \mathbf{B}$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

Meaning that no subset of a key is also a superkey

Finding Keys and Superkeys

- For each set of attributes X

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Do we need to check all sets of attributes?

Example of Finding Keys

```
Product(name, price, category, color)
```

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

What is a key?

Example of Keys

```
Product(name, price, category, color)
```

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

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

= the set of all attributes

⟹ this is a **superkey**

⟹ this is a **key**, since neither name nor category alone is a superkey

Acknowledgement

Some of these slides are taken from cs145 course offered by Stanford University.