Lecture 2

# A Database is a set of relations:

A relation is a set of tuples

Each tuple has values for attributes of that relation

# Key:

A key is a minimal set of attributes such that no two tuples in the relation can have the same values for the key

## Minimal: no subset of these attributes is a key

## e.g.:

Student (RIN, SSN, First Name, Last Name, Email):

Key: RIN

SSN

Email

StudentInfo (RIN, SSN, First Name, Last Name, Email, Major, Advisor):

NOTICE: each instance in the tuple must be SIMPLE (No set, list, …)

Student can have multiple majors, for a major, there is only a single advisor for a major and a student, and advisor can only advise only one major

Key: RIN + Major

Email + Major

SSN + Major

RIN + Advisor

Email + Advisor

SSN + Advisor

StudentInfo2 (RIN, SSN, First Name, Last Name, Email, Major, Advisor):

Student can have multiple majors, for a major, there is only a single advisor for a major and a student

Key: RIN + Major

Email + Major

SSN + Major

# Relational Algebra

Relational algebra consists of a set of operations that takes as input a set of tuples (or two sets of tuples) and return a new relation (a new set of tuples).

MarvelHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

DCHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

Movies (movieid, name, release\_date)

HeroInMovie (movieid, heroname, multiverseid)

## Set Operations

### Set compatibility:

two relations are set compatible, if they have the same data model:

Same attribute and attribute have the same name.

## Combination of sets:

Union:

Intersection:

Difference:

### Set Union:

= {set of all tuples that are either in R or S or both if R and S are set compatible}

### Set Intersection:

= {set of all tuples that are either in R and S if R and S are set compatible}

### Set Difference (substraction):

={the set of all tuples in R that are not in S, if R and S are set compatible}

### e.g.:

R1 = MarvelHeroes DCHeroes

All heroes in either universe

R2 = MarvelHeroes – DCHeroes

All heroes that are only in the Marvel universe

(In this case Marvel heroes)

R3 = MarvelHeroes DCHeroes

All heroes that are in both Marvel and DC Universes

(In this case )

## Projection (dicing):

Def: (Select one or several attribute from one Relation to form a new relation)

(Getting rid of unnecessary attributes)

Given a relation R:

{all tuples in R, but only the attributes in the projection}

Find name of all heroes in the Marvel Universe:

R1 = Project\_{Heroname} (MarvelHeroes)

Find all multiverse in the marvel universe

R2 = Project\_{multiverseid} (MarvelHeroes)

Find the common universe in Marvel and DC (assume the same ID indicate the same universe)

R3 = Project\_{multiverseid}(MarvelHeroes) intersect

Project\_{multiverseid}(DCHeroes)

All locations that are unique to Marvel Universe

R4 = Project\_{location}(MarvelHeroes)

R5 = Project\_{location}(DCHeroes)

R6 = R4 – R5

R7 = Project\_{heroname, multiverseid} (MarvelHeroes)

R7 = Project\_{locations, multiverseid} (MarvelHeroes)

## Selection (slicing):

Def: (Keeps only the tuples that satisfy certain conditions)

Given a relation R and a Boolean condition C over the attribute of R:

Select\_C (R) = {The set of all tuples in R that satisfy the condition C}

Find heroes who are located in NYC and in multiverse 1:

R1 = Project\_{heroname} (Select\_{ location == ‘NYC’ and multiverseid==1} (MarvelHeroes) )

R2 = Project\_{heroname} (Select\_{ location == ‘NYC’ and multiverseid==1} (DCHeroes) )

R3 = R1 union R2

Find name of all heroes from multiverse = 2 who were in a movie:

R1 = Project\_{heroname} ( Select\_{multiverse = 2} (HeroInMovie))

R2 = Project\_{heroname} ( Select\_{multiverse = 2} (MarvelHeroes))

R3 = R1 intersect R2

## Rename operator

Given a relation R with data model , rename will name every attribute in R

## Cartesian Product

Given two relations R(A1, …, An) and S(B1,…,Bn) such that R and S do not any attribute in common:

(Permutation of two relations providing all possible results)

(Row by Row)

{set of all tuples (r, s) such that r is a tuple in R, s is a tuple in S, and (r, s) has attributes (A1,…,An,B1,…,Bn) }

e.g.:

Find heroes and the multiverse they are from in the new spiderman movie

M1 (movieid1, name1, release\_date1) = Movies

HeroInMovie (movieid, heroname, multiverseid)

R1 = M1 x HeroInMovie

R2 = Select\_{movieid1 = movieid} (R1)

R3 = Select\_{name=”No way home”} (R2)

R4 = Project\_{heroname, multiverseid} (R3)

**Find all locations that the heroes in the “Avengers End Game” are from.**

Database:

MarvelHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

DCHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

Movies (movieid, name, release\_date)

HeroInMovie (movieid, heroname, multiverseid)

Movies -> movieid

HeroInMovie -> to find which hero is in movie

MarvelHeroes -> for location

R1{movieid1} = Project{movieid} ( {select\_{movieid = “Avengers End Game”} Movies }

R2 = R1 x HeroInMovie

R3 = Select\_{movieid1 = movieid}{R2} 🡪 Heroes in the movie

R4 = (heroname1, realname1, power1, location1, multiverseid1) = MarvelHeroes

R5 = R3 x R4

----- heroname1, realname1, power1, location1, multiverseid1

-----movieid1, heroname, multiverseid

R6 = Select\_{heroname = heroname1 and multiverseid = multiverseid1} (R5)

R7 = Project\_{location} (R6)

# Relational algebra operations

## Representations

Operation Text Version Standard Version

Set union R union S

Set difference R - S

Set intersection R intersect S

Projection project\_{A1,…An} (R)

Selection select\_{C} (R)

Rename

Cartesian Product R x S

Theta-Join R join\_{C} S

Natural Join R \* S R \* S

## e.g.:

Movies (movieid, name, release\_date)

HeroInMovie (movieid, heroname, multiverseid)

Find names of all heroes in Movie titles “Avengers”

T1(movieid1, name1, release\_date1) = select\_{title = “Avengers”} (Movies)

T2 = T1 x HeroInMovie

T3(heroname1) = project\_{heroname} (select\_{movieid = movieid1}(T2) )

T4 = T3 x MarvelHeroes

T5 = project\_{heroname, location} (select\_{heroname = heroname1} (T4) )

Rewrite using Join (NEEDS Verification)

T1(movieid1, name1, release\_date1) = select\_{title = “Avengers”} (Movies)

T2 = T1 join\_{movieid = movieid1} HeroInMovie

T3(heroname1, multiverseid1) = project\_{heroname} (T2)

T4 = T3 join\_{heroname = heroname1, multiverseid = multiverseid1} MarvelHeroes

T5 = project\_{heroname, location} (T4)

## Join Operation

Derived operator: Selection of a cartesian product

**R join\_{C} S = select\_{C} (R x S)** if R and S have no attribute in common and

C is a join condition (a Boolean condition containing expressions of the form A op B where A is an attribute from R and B an attribute from S)

e.g.:

Find all movies starring a hero with the power to “Stop Time”

T1 = Project\_{heroname, multiverseid} select\_{power= “stop time”} (MarvelHeroes)

T2 = Project\_{heroname, multiverseid} select\_{power= “stop time”} (DCHeroes)

T3(heroname1, multiverseid1) = T1 union T2

T4(heroname1, movieid1, heroname, multiverseid1) = T3 join\_{heroname = heroname1 and multiverseid = multiverseid1} HeroInMovie

T5 = project\_{movieid, title} (T4 join\_(movieid = movieid1) Movies)

## Natural Join

R \* S is a join of R and S for the join condition of equality of all attributes that are in common to R and S. Result does not repeat attributes with the same name

e.g.:

T1 = MarvelHeroes \* HeroInMovie (join over heroname and multiverseid)

T1 has schema: (heroname, realname, power, location, multiverseid, movieid)

T2 = T1 \* Movies

T1 has schema: (heroname, realname, power, location, multiverseid, movieid, name, release\_date)

### Exercise

MarvelHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

DCHeroes (heroname, realname, power, location, multiverseid)

Key: heroname, multiverseid

Movies (movieid, name, release\_date)

HeroInMovie (movieid, heroname, multiverseid)

Query: Return the heroname of all heroes that were in a movie in 2020 and in 2019

T1 = project{heroname} ( (select\_{release\_date = 2019 }(Movies)) \* (MarvelHeroes union DCHeroes) )

T2 = project{heroname} ( (select\_{release\_date = 2020 }(Movies)) \* (MarvelHeroes union DCHeroes) )

T3 = T1 intersect T2

which is in this case same as: T3 = T1 \* T2

which is: T3 = select\_{hero2019 = hero2020}(T1 x T2)

Hero1 movieA 2019

Hero2 movieB 2020

Hero3 movieC 2019

Hero3 movieD 2020

**A Natural Join between two Relations with nothing in common is basically a Cartesian Product of two**

**A Natural Join between two Relations with exactly same attributes is basically a Set Intersection**

Query: return heroname of all heroes who were in a movie in two consecutive years

T1 = Movies \* HeroInMovie \* (MarvelHeroes union DCHeroes)

T2 = Project\_{heroname, multiverseid, year} (T1)

T3 = Project\_{heroname1, multiverseid1, year1} (T1)

T4 = T2 x T3

T5 = select\_{heroname = heroname1, and multiverseid = multiverseid1, and year = year + 1} (T4)

T6 = project\_{heroname} (T5)

Query: return heroname of all heroes who were in a movie in two consecutive years

T7 = project{heroname} (MarvelHeroes union DCHeroes)

T8 = T7 – T6

# Normalization Theory

ShowName StartYear Creator StreanSite URL

The 2002 S HBO HBO

Babylon 5 1993 B Prime Prime

Never … 2020 C Netflix Netflix

Babylon 5 1993 B HBO

## Functional dependencies (FD):

Given a relation R is given my

A1, …, An 🡪 B1, …, Bn

Where A1,…,An, B1,…,Bn are attributes in R

It means that whenever you have two tuples in R with the same values for attributes A1,…,An, then these tuples must also have the same values for attributes B1,…,Bn

ShowName, StartYear, Creator, StremSite, URL, ShowSeasons, StreamSeasons

ShowName 🡪 StarYear, Creator, ShowSeasons

StreamSite 🡪 URL

ShowName StreamSite 🡪 StreamSeasons

ShowName 🡪 ShowName

ShowName 🡪 ShowName, StartYear

ShowName StreamSite 🡪 URL StreamSite StreamSeasons

## Functional Dependency Inference Rules

Given a relation R and a set F of functional dependencies, the following inference rules allows you to find new functional dependencies (FDS)

### Trivial

**If A is a subseteq of B, then B implies A**

e.g.:

### Transitivity

**If X🡪Y, Y🡪Z, then X🡪Z**

e.g.:

Then:

### Decomposition

Decomposition only works on the right hand side.

If X🡪 YZ, then X🡪Y and X🡪Z

e.g.:

### Combining Rule

If X🡪Y, X🡪Z, then X🡪YZ

e.g.:

Given:

Showname 🡪 StartYear

Showname 🡪 Creator

Then:

Showname 🡪 StartYear, Creator

### Augmentation Rule

If X🡪Y, then XY🡪YZ

e.g.:

AB 🡪 CD

Then:

ABE 🡪 CDE

## e.g.

Showname, Startyear, Creator, StreamSite, URL, ShowSeasons, StreamSeasons

StreamSite 🡪 URL

Showname StreamSite 🡪 StreamSeasons

URL 🡪 StreamSite

URL 🡪 URL

ShowName, URL 🡪 ShowName

That is to say:

Given a relation R and a set F of functional dependencies, X 🡪 Y is implied by F if we can obtain X 🡪 Y from F using inference rules

## Exercise:

F = {

ShowName 🡪 StarYear, Creator, ShowSeasons

StreamSite 🡪 URL

ShowName StreamSite 🡪 StreamSeasons

}

Questions:

URL 🡪 URL ?

Yes, trivial

URL 🡪 StreamSite?

Yes, subset

ShowName, URL 🡪 StreamSeasons?

URL 🡪 StreamSite

ShowName URL 🡪 ShowName StreamSite

ShowName StreamSite 🡪 StreamSeasons

Transitivity: Showname URL 🡪 StreamSeasons

Lecture 4

ShowName 🡪 StarYear, Creator, ShowSeasons

StreamSite 🡪 URL

ShowName StreamSite 🡪 StreamSeasons

# Closure of a set of attributes

## Def

Given a relation R and a set F of functional dependencies, and a set X of attributes in R, the closure of X (Denoted by X+) is the set of all attributes implied by X with respect with F.

Then:

## Algorithm to compute the closure of X:

### Algorithm

Step 1: initialize

Step 2: Do until no new attributes are added:

Find a functional dependency, such that Y is a subset of X then add Z to X+

### e.g.:

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

F= {

A🡪BCD

E🡪F

F🡪E

AE🡪G

}

A+ = {A}

A+ = {A, B, C, D} (A🡪BCD)

B+ = {B}

E+ = {E}

E+ = {E, F} (E🡪F)

AE+ = {A, E}

AE+ = {A, B, C, D, E} (A🡪BCD)

AE+ = {A, B, C, D, E, F} (E🡪F)

AE+ = {A, B, C, D, E, F, G} (AE🡪G)

That is:

AE🡪ABCDEFG

## Key Definition under FDs:

Key: Given a relation R and FDs F, a key is a set of attributes X such that:

1. X+ includes all attributes in R (Uniqueness)
2. No subset of X satisfies a) (Minimality)

### e.g.:

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

F= {

A🡪BCD,

E🡪F

F🡪D

AE🡪G

}

AE is a key for R

AF is a key for R

Examples:

R1(A, B, C, D, E, F, G)

F1= {AB🡪CD, DE🡪F, F🡪G}

(E) AB+ = {A, B, C, D} ABE is a key

(AB) DE+ = {D, E, F, G} ABDE is not a key, not minimal

F+ = {F, G}

ABE

R2(A, B, C, D, E, F, G)

F2= {AB🡪C, BCD🡪F, F🡪AG}

AB+ = {A, B, C} D+ = {A, B, C, D, F, G} ABDE

BCD+ = {B, C, D, F, A, G} BCDE

F+ = {A, G} B+ = {A, B, C, G} D+ = {A, B, C, D, F, G} FBDE

R3(A, B, C, D, E, F, G)

F3= {AB🡪CDEF, F🡪AG}

AB

BF

d)

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

F4= {AB🡪CDEF, F🡪A}

ABG

FBG

## Rules to find the key:

If something is not in the right-hand side, it MUST be in the key

If something is not in both sides, it MUST be in the key

### e.g.

Students (RIN, SSN, Name, Address, Major, Advisor, Email, Hobby)

RIN🡪SSN, Name, Address, Email

SSN🡪RIN

RIN, Major🡪Advisor

Key:

RIN, Major, Hobby

SSN, Major, Hobby

# Normal Forms

## SuperKey (superset of key)

Given a relation R and a set of FDs F, a superkey is a set of attributes X such that:

X🡪Y is trivial,

By definition: any key is a superkey

e.g.:

R (A, B, C, D)

F={A🡪BCD}

Key: A

Superkey: A, AB, AC, AD, ABC, ABD, ACD, ABCD

## Boyce-Codd Normal Form (BCNF):

Given a relation R and a set of FDs F, R is in Boyce-Codd Normal Form if and only if, every functional dependency X🡪Y in F satisfies one of following conditions:

1. X🡪Y is trivial, or
2. X is a superkey

R1(A, B, C, D) F={A🡪BCD}

Key: A

A🡪BCD, A is a superkey, R1 is in BCNF

R2(A, B, C, D) F= {AD🡪B, ABD🡪C, AB🡪A}

Key: AD

AD🡪B okay: AD is superkey

ABD🡪C okay: ABD is superkey

AB🡪A okay: trivial

R3(A, B, C, D) F= {AD🡪B, AB🡪C}

Key: AD

AD🡪B okay: AD is superkey

AB🡪C NG: AB is not a superkey,

R4(A, B, C, D) F= {AD🡪B, B🡪C}

Key: AD

B is not a superkey, not BCNF

R5(A, B, C, D) F= {AD🡪BC, B🡪A}

Key: AD, BD

AD🡪BC okay: superkey

B🡪A NG, B is not a superkey

R6(A, B, C, D) F={A🡪BC}

Key: AD

A NG: A is not a superkey

e.g.:

ShowName 🡪 StarYear, Creator, ShowSeasons

StreamSite 🡪 URL

ShowName StreamSite 🡪 StreamSeasons

Key: Showname StreamSite, Showname URL

Showname 🡪 StartYear, Creator, ShowSeasons Not okay: Showname not a superkey

StreamSite 🡪 URL Not okay: StreamSite not a superkey

URL🡪StreamSite Not okay: URL not a superkey

Showname StreamSite 🡪 StreamSeasons

Not in BCNF

If a relation is not in BCNF, then it can have

🡪Duplicated data, must be updated all for a single change

🡪Deleting something may cause a loss of data for another

(not being available in a streaming site, means we will loose data about the show)

## Basis Form

**We says that a set of functional dependencies is in a basis form if there is only one attribute on the right hand site**

(You can easily put a set in basis form by using the decomposition rule)  
{A🡪BCD} put in basis from: {A🡪B, A🡪C, A🡪D}

## Prime Attribute

**A prime attribute is an attribute in any key**

R1(A, B, C, D, E, F, G)

F1= {AB🡪CD, DE🡪F, F🡪G}

(E) AB+ = {A, B, C, D} ABE is a key

(AB) DE+ = {D, E, F, G} ABDE is not a key, not minimal

F+ = {F, G}

ABE

PRIME ATTRIBUTE: A,B,E

R2(A, B, C, D, E, F, G)

F2= {AB🡪C, BCD🡪F, F🡪AG}

AB+ = {A, B, C} D+ = {A, B, C, D, F, G} ABD

BCD+ = {B, C, D, F, A, G} BCD

F+ = {A, G} B+ = {A, B, C, G} D+ = {A, B, C, D, F, G} FBD

PRIME ATTRIBUTE: A,B,C,D,F

## Third Normal Form (3NF):

Given a relation R and a set of FDs F in basis form,

R is in Third Normal Form if and only if, every functional dependency X🡪Y in F satisfies one of following conditions:

1. X🡪Y is trivial, or
2. X is a superkey, or
3. Y is a prime attribute

IF a relation is in BCNF, then it is also in 3NF

e.g.:

R1(A, B, C, D) F={A🡪BCD}

Key: A

A🡪BCD, A is a superkey, R1 is in BCNF

BCNF🡪3NF

R2(A, B, C, D) F= {AD🡪B, ABD🡪C, AB🡪A}

Key: AD

AD🡪B okay: AD is superkey

ABD🡪C okay: ABD is superkey

AB🡪A okay: trivial

BCNF🡪3NF

R3(A, B, C, D) F= {AD🡪B, AB🡪C}

Key: AD

AD🡪B okay: AD is superkey

AB🡪C NG: AB is not a superkey, C is not a prime attribute,

Not in BCNF

Not in 3NF

R4(A, B, C, D) F= {AD🡪B, B🡪C}

Key: AD

B🡪C NG: B is not a superkey, C is not a prime attribute

Not in BCNF

Not in 3NF

R5(A, B, C, D) F= {AD🡪BC, B🡪A}

Key: AD, BD

AD🡪BC okay: superkey

B🡪A B not a superkey, A is in prime attributes

Not in BCNF

In 3NF

R6(A, B, C, D) F={A🡪BC}

Key: AD

A🡪B A is not a superkey, B is not a prime attribute

A🡪C A is not a superkey, C is not a prime attribute

Not in BCNF

Not in 3NF

# Minimal basis

A set of functional dependencies is said to be minimal if we cannot remove anything from them and still get the same meaning

Given a set F of FDs, F+ is the set of all functional dependencies I can obtain from F using all the inference rules

Two sets F1, and F2, are equivalent if and only F1+ = F2+

Alternatively:

If every functional dependency X🡪Y in F1, is implied by F2 (or X+ in F2 and check if Y is in it!)

And

If every functional dependency X🡪Y in F2, is implied by F1 (or X+ in F1 and check if Y is in it!)

Then, F1 and F2 are equivalent

e.g.:

F1 = {A🡪B, B🡪C}

F2 = {A🡪B, B🡪C, A🡪C}

A🡪C?

A+ = {A,B,C}

C is in A+

F3 = {A🡪B, A🡪C}

F4 = {A🡪B, B🡪C, A🡪C}

B🡪C?

B+ = {B}

Lecture 5

Reviews:

Normalization Theory

Key: Given a relation R and FDs F, a key is a set of attributes X such that:

1. X+ includes all attributes in R (uniqueness) [superkey condition]
2. And no subset of s satisfies a) (minimality) [key condition]

Superkey (superset of key): Given a relation R and FDs F,

F1 and F2 are equivalent

Closure = is the set of all FDs implied by

Minimal Cover Basis: A set of functional dependencies is said to be minimal if we cannot remove anything from them and still have the same closure!

If a set F of FDs is minimal, then we cannot:

* Remove a functional dependency
* Remove an attribute from the left or the right-hand side of a function dependency without altering its meaning (i.e., its closure)

# Algorithm to compute a minimal basis:

Input: a set F of FDs,

F = {A🡪B, B🡪BC, ABC🡪D, B🡪D}

1. First put the set of FDs in a basis from

F= {A🡪B, B🡪B, B🡪C, ABC🡪D, B🡪D}

1. Remove all trivial FDs

F= {A🡪B, B🡪C, ABC🡪D, B🡪D}

1. Suppose X🡪Y is in F and F’=F-{X🡪Y}

Compute X+ in F and F’, if they are the same, then we can remove X🡪Y

e.g.:

Trial 1:

F= {A🡪B, B🡪C, ABC🡪D, B🡪D}

F’= {A🡪B, ABC🡪D, B🡪D} ---removed B🡪C

In F, B+= {B, C, D}

In F’, B+= {B, D}, not the same, cannot remove B🡪C

Trial 2:

F= {A🡪B, B🡪C, ABC🡪D, B🡪D}

F’= {A🡪B, B🡪C ABC🡪D} ---removed B🡪D

In F, B+= {B, C, D}

In F’, B+= {B, D}, not the same, cannot remove B🡪D

Trial 3:

F= {A🡪B, B🡪C, ABC🡪D, B🡪D}

F’= {A🡪B, B🡪C B🡪D} ---removed ABC🡪D

In F, ABC+= {A, B, C, D}

In F’, ABC+= {A, B, C, D}, equal, can remove

1. Suppose XZ🡪Y is in F, F’=F-{XZ🡪Y} union {X🡪Y}  
   check if X+ is the same in F and F’, if so, then F’ becomes F

F = {A🡪B, B🡪C, B🡪D} ---- done, because no extra attribute on the left

1. Use combining rule to return a set of FDs

F = {A🡪B, B🡪CD}

New example

F2 = {A🡪B, B🡪C, AB🡪D}

F2’ = {A🡪B, B🡪C, A🡪D} ----- remove B from AB🡪D

In F, A+={A,B,C,D}

In F’, A+={A,B,C,D} ----- yes, can remove B

----

F2 = {A🡪B, B🡪C, AB🡪D}

F2’ = {A🡪B, B🡪C, B🡪D} ----- remove A from AB🡪D

In F, B+={B,C}

In F’, B+={B,C,D} ----- not the same, cannot remove A from AB

----

Final:

F2 = {A🡪B, B🡪C, A🡪D}

F2 = {A🡪BD, B🡪C}

# Full example

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

F = {AB🡪C, BD🡪BEF, CDF🡪AG, ABC🡪G, ABC🡪D}

Step 1:

F = { AB🡪C, BD🡪B, BD🡪E, BD🡪F, CDF🡪A, CDF🡪G, ABC🡪G, ABC🡪D}

Step 2:

F = { AB🡪C, BD🡪E, BD🡪F, CDF🡪A, CDF🡪G, ABC🡪G, ABC🡪D}

Step 3:

Try remove ABC🡪G

F+ == F’+ = {A,B,C,D,E,F,G}

Can remove ABC🡪G

F = { AB🡪C, BD🡪E, BD🡪F, CDF🡪A, CDF🡪G, ABC🡪D}

Step 4:

F = {AB🡪C, BD🡪E, BD🡪F, CDF🡪G, CDF🡪A, ABC🡪D}

F’ = {AB🡪C, BD🡪E, BD🡪F, CDF🡪G, CDF🡪A, AB🡪D}

In F, AB+={A,B,C,D,E,F,G}

In F’, AB+={A,B,C,D,E,F,G}, can remove C from AB

Step 5:

F={AB🡪CD, BD🡪EF, CDF🡪AG}

Decomposition:

Given a relation R and a set F of FDs, a decomposition is given by R1(X), R2(Y) … where X,Y are sets of all attributes of R, computed by

R1 = project\_{X} R

R2 = project\_{Y} R

Such that X union y union … make up all the attributes in R

# Lossless Decomposition

Given R and F, a decomposition R1, R2, …, Rn is lossless if it is guaranteed that

R1 \* R2 \* … \* Rn = R

i.e. the natural join is guaranteed to return the same results as the original relation.

## Algorithm to check if a decomposition is lossless

Given R and F, a decomposition R1, R2, …, Rn construct a relation R such that

For each decomposed relation, there is a tuple in R where attribute in Ri has no subscripts (known values), and the rest with a new subscript for each tuple (unknown values)

Applies function dependencies like X🡪Y, if two tuples have the same values for X, then make Y value the same (if the value for one tuple is known, make the other known, otherwise make them the same unknown value)

Continue until no rules can be applied:

If in the resulting relation, there is tuple with no subscript, the decomposition is lossless

If in the resulting relation, there is NO tuple with no subscript, the decomposition is lossy

R(A,B,C,D,E) F={AB🡪C, B🡪D, C🡪E}

R1(A,B,C) R2(A,B,E) R3(D,E)

A B C D E

a b c d1 e1  
a b c2 d2 e

a3 b3 c3 d e

Apply AB🡪C

A B C D E

a b c d1 e1  
a b c d2 e

a3 b3 c3 d e

Apply C🡪E

A B C D E

a b c d1 e  
a b c d2 e

a3 b3 c3 d e

Apply B🡪D

A B C D E

a b c d e  
a b c d e

a3 b3 c3 d e

Lossy decomposition, NO tuple with no subscripts

R(A,B,C) F={A🡪B}

R1={A,C} R2={B,C}

A B C

a b1 c

a2 b c

Can’t apply any rules, lossy

R3={A,B} R2={A,C}

A B C

a b c1

a b2 c

Apply: A🡪B

A B C

a b c1

a b c --- no subscript, lossless decomposition

projection of functional dependencies to a decomposition

Given a relation R and a set f of FDs and a decomposed relation R1. The projection of F into R1 is the set of all functional dependencies in F+ that only contain the attributes of R1.

R(A,B,C,D,E,F,G) F={AB🡪CD, BD🡪EF, CDF🡪AG}

R1(A,B,D,G) {AB🡪DG}

AB🡪ABDG (AB+={ABCDEFG}, take the only attribute appears we get ABDG)

AB🡪DG

Simplify to find minimal basis

----------

R2(A,B,C,E,F) {AB🡪CEF}

AB🡪CEF

F1 union F2 = {AB🡪CDEFG} not equivalent to F

CDF+ = {C,D,F}

F = {AB🡪CD, BD🡪EF, CDF🡪AG}

CDF+={C,D,F,A,G}

S(A,B,C) F={A🡪B, B🡪C}

S1(A,B) {A🡪B}

S2(B,C) {B🡪C}

Then F1 union F2 = {A🡪B, B🡪C} is equal to F

This decomposition is dependency preserving

## Dependency Preserving Decompositions

Given a relation R and a set F of FDs, and a decomposition R1, R2, …, Rn and suppose F1, F2,…,Fn are the projection of F onto R1, R2, …, Rn respectively

If F1 union F2 union … Fn equivalent to F, then we say that his is a dependency preserving decomposition

Lecture 6

# 3NF Decomposition

## Algorithm

Given a relation R and a set F of FDs that is minimal, the 3NF decomposition is computed as

follows:

Minimal vs Minimal Basis

{AD🡪B, AD🡪C} minimal basis

{AD🡪B, AD🡪C} minimal but not basis

Granted to be lossless and satisfy 3NF

1. For each FD X🡪Y, create a new relation with attribute X union Y
2. Remove any relation Rx if there is another relation Ry that has all the attribute in Rx
3. If there is a relation that contains all the attributes of one of the keys of R, then add an extra relation for one of the keys

## e.g.

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

F={AB🡪C, C🡪B, BD🡪E, DF🡪A}

Keys: BDF, CDF

Step 1:

R1(A,B,C) {AB🡪C, C🡪B}

R2(B,C) {C🡪B}

R3(B,D,E) {BD🡪E}

R4(A,D,F) {DF🡪A}

Step 2: (Remove R2)

R1(A,B,C) {AB🡪C, C🡪B}

R3(B,D,E) {BD🡪E}

R4(A,D,F) {DF🡪A}

Step

R1(A,B,C) {AB🡪C, C🡪B}

R3(B,D,E) {BD🡪E}

R4(A,D,F) {DF🡪A}

R5(B,D,F)

Step 3: (Remove R2)

R1(A,B,C) {AB🡪C, C🡪B} Key: AB,AC

R3(B,D,E) {BD🡪E} Key: BD

R4(A,D,F) {DF🡪A} Key: DF

R5(B,D,F) Key: BDF

Lossless, let’s prove using Chase algorithm

F={AB🡪C, C🡪B, BD🡪E, DF🡪A}

A B C D E F

a b c d1 e1 f1

a2 b c2 d e f2

a b3 c3 d e3 f

a4 b c4 d e4 f

BD🡪E

A B C D E F

a b c d1 e1 f1

a2 b c2 d e f2

a b3 c3 d e3 f

a4 b c4 d e f

DF🡪A

A B C D E F

a b c d1 e1 f1

a2 b c2 d e f2

a b3 c3 d e3 f

a b c4 d e f

AB🡪C

A B C D E F

a b c d1 e1 f1

a2 b c2 d e f2

a b3 c3 d e3 f

a b c d e f

End of Proof

R(A,B,C,D,E) F={AB🡪CD, D🡪E} Key:AB

3NF decomposition

(A,B,C,D)

(D,E)

Suppose F was in basis form

R(A,B,C,D,E) F={AB🡪CD, AB🡪D, D🡪E} Key: AB

3NF Decomposition

(A,B,C) AB🡪C

(A,B,D) AB🡪D

(D,E) D🡪E

# BCNF Decomposition

All BCNF decompositions are lossless, but not dependency preserving.

## Algorithm

Given a relation R and a set F of FDs that is minimal, the BCDF decomposition is computed as follows:

* Suppose X🡪Y is an FD in f that violates BCNF. Then, compute X+

Create two relations:

1. R1(X+): X🡪X+
2. R2 has all attributes in R, except for (X+) – (X)
   1. This means keep R-(X+)+X

for each Ri, compute functional dependency projections, check if they are in BCNF. If any of them is not in BCNF, apply BCNF decomposition recursively

## Example:

R(A,B,C,D,E,F), and F= {AB🡪C, C🡪DE, E🡪F}

Key AB:

(C🡪DE and E🡪F violate s BCNF)

Start with C🡪DE:

Step 1

C+={C,D,E,F}

R1 = {C,D,E,F} F1={C🡪DE, E🡪F} Not yet in BCNF

R2 = {A,B,C} F2={AB🡪C} Done

Step 2

E+={E,F}

R11(E,F) F1={E🡪F} Done

R12(C,D,E) F2={C🡪DE} Done

Final results:

(EF)

(CDE)

(ABC)

END of process

R(Showname, Startyear, Creator, StreamSite, URL, ShowSeasons, StreamSeasons)

F={

Showname -> StartYear, Creator, ShowSeasons (assuming a single creator)

StreamSite -> URL

Showname StreamSite -> StreamSeasons

URL -> StreamSite

URL -> URL

Showname URL -> StreamSeasons

}

Key: Showname, StreamSite

Step 1:

Take out StreamSite🡪URL

S11(StreamSite, URL) {StreamSite🡪URL, URL🡪StreamSite} BCNF

S12(Showname, StartYear, Creator, StreamSite, ShowSeasons, StreamSeasons)

Step 2:

S12(Showname, StartYear, Creator, StreamSite, ShowSeasons, StreamSeasons)

F\_S2={ Showname🡪StartYear, Creator, ShowSeasons,

Showname StreamSite 🡪 StreamSeasons}

Step 3:

Take Showname🡪StartYear, Creator, ShowSeasons

S21(ShowName, StartYear, Creator, ShowSeasons) BCNF

S22(ShowName, StreamSite, StreamSeasons) BCNF

# 4NF

DB setups:

GamesInfo (gameid, name, category, type)

Game can have multiple categories and types

{gameid 🡪 name}

Key: gameid, category, type

Games (gameid, name) {gameid🡪name} Key: gameid in BCNF

GameInfo (gameid, category, type) Key: all attribute in BCNF

Gameid Category Type

234 Adventure Strategy

234 SpaceExploration Farming

234 Adventure Farming

234 SpaceExploration Strategy

Multivalued Dependency

gameid => category

gameid => type

(for a certain gameid, there may be multiple categories and multiple types)

Example

GamesInfo (gameid, name, category, type)

Game can have multiple categories and types

{gameid 🡪 name}

Key: gameid, category, type

Games (gameid, name) {gameid🡪name} Key: gameid in BCNF

GameInfo (gameid, category, type) Key: all attribute in BCNF

Gameinfo is in BCNF, but it contains two multivalued attributes, category and type. But category and type are not coupled with each other. Hence, category and type are independent of each other.

A multivalued dependency of the form

X=>> Y

And suppose Z is the remainder of attribute in R

Says that whenever we have two tuples: (x1,y1,z1) and (x2,y2,z2)

Then we must also have tuples: (x1,y1,z2) and (x2,y2,z1)

## Check

If x=>>Y and X and Y together is not all the attributes in R, then R is not in 4NF.

(Alternatively, If X=>>Y and X , Y is all the attribute in R, then R is in 4NF)

IF a relation is not in 4NF, use a method similar to BCNF decomposition to decompose, but now using multivalued dependencies)

## Example

GameInfo (gameid, category, type) Key: all attribute in BCNF

gameid=>> category, since gameid and category together is not all the attributes, this relation is not in 4NF

Take gameid=>>category out (no closure)

Gameinfo1(gameid, category) gameid =>> category key: gameid, category 4NF

Gameinfo2(gameid, type) gameid =>> type key: gameid, type 4NF

## Example 2

R(A,B,C,D,E) F={A🡪BC, C🡪D} Key: AE

Step 1: take out C🡪D

R11(A,B,C,E) not BCNF

R12(C,D) BCNF key C

Step 2: take out A🡪BC

R21(A,B,C) BCNF key A

R22(A,E) BCNF key AE

## Example 3

(assuming we are not storing which actor appears in which location, just that movies have actors and locations)

Movies (movieid, action, location)

movieid =>>actor

Not in 4NF

Instead store:

Movies1(movieid, actor)

Movies2(movieid, locations)

Lecture 7

# ER Diagrams

## Definition

**Entity Relationship diagrams**

Object-Oriented, can be converted to relational data model.

Entities: main classes of objects we would like to store information about

An entity must have a name, and a key, and some number of attributes.

* No multivalued attributes
* Each attribute should about the entity, not describing other entity
* Each entity is in BCNF (The key for the entity implies all other attributes), if not possible, then in 3NF

Relationships connect 2 or more entities

## Example

University database

Student (RIN), fname, lname, year, email, address, SSN, pronouns, gpa

--- takes courses

Faculty/Staff (RIN), fname, lname, email, address, SSN, pronouns, position

Departments: (code), name

Majors: (code), name

Courses: (courseno, maabbr), name, credits, whenoffered

Sections: (crn), capacity,

--- who teaches it, where is located

Buildings: (abbr), name, lat, long

Rooms: (id), number

--- rooms are in buildings

Students take courses

Students have majors

Students have advisors

Courses are taught by faculty

Courses are taught in classrooms

Classrooms are in buildings

## e.g.,1

faculty/staff 🡨🡪 a department can have how many faculties who works in it? 🡨🡪 a faculty can work in how many departments

(Max questions) (Max question)

A department can have how A faculty can work in how many departments?

many faculties who work on it?

图示

描述已自动生成

should each dept have at least one faculty? Should each faculty work in 2 department?

(Min questions)

The left side of the edge implies the min/max restrictions of the right-hand side

The right side of the edge implies the min/max restrictions of the left-hand side

Converting ER diagrams to relational data model

1. Map each entity to a relation: the key of the entity becomes the key of the relation and attributes of the entity become attributes of the relation
2. For each relationships

* If the relationship has a one side (one to may or one to one), then take key for entity on the one side and include it as an attribute in the other entity
* If the relationship is many to many, create a new relation which has the key of all the entities it connects and the combination of the keys if the key of this new relation.

If the relationship has attributes, then include the attributes where you put the relationship.

图示

描述已自动生成

Faculty( (RIN), …, department\_worked\_for

Department( (code), …, chairOf,

Lecture 8

Converting an ER-Diagram to a Relational model

图示

描述已自动生成

Faculty 🡪 Department

Department🡪 Faculty

Faculty (RIN, … deptcode\_works\_for, A

Department (code, …, RIN\_chair, B

AffiliatedWith(facultyRIN, deptcode, C)

# Entity or Attributes

图示

描述已自动生成

Entity relationship Diagrams – Continued

Students(RIN, fname, lname, email, address, gpa)

Key: RIN  
Courses(CourseNo, abbr, name, credits)

Key: CourseNo, abbr

Sections(crn, semester, year, capacity, courseno)

Key: crn

Rooms(id, number)

Buildings(abbr, name, lat, long)

Key: abbr

Departments(code, name, mainoffice\_roomid)

FacultyStaff(RIN, fname, lname, address, position, email, department\_code,)

Key: RIN

Majors(code, name)

StudentMajors( RIN, code)

Key: RIN, code

StudentSections(RIN, crn, grade)

Key: RIN, crn

SectionRooms(crn, roomid)

Key: crn, roomid

SectionInstructors(crn, RIN, role)

Key: crn, RIN

FacStaffRooms( roomid, RIN, phoneno)

Key: roomid, RIN

Collaborates(RIN1, RIN2)

Key: RIN1, RIN2

In room is represented as a weak entity

Departments(code, name, mainoffice\_roomid)

SectionRooms(crn, roomid)

FacStaffRooms( roomid, RIN, phoneno)

图示

描述已自动生成

# Weak entity

图示

描述已自动生成

图示

描述已自动生成

Concatenate all hobbies of a specific student together and store it into the hobby relation

图示

描述已自动生成

图示

描述已自动生成

The relationship presented on sides are not equivalent since the decomposition on the right-hand side is a lossy decomposition

(Non-Binary relationships must take all the connected prerequisite into account)

E.g.: Faculty + Majors 0…N Students

图示

描述已自动生成

# Hierarchies

B inherits A means:

B have all the attributes that A does, and may have more attributes (own)

B, C **covers** A if B unions C represents all the elements in A

BC’s **disjoint** is the XOR of B and C

# Hierarchy

A(X,Y)  
B(X,Y,Z) is an A

C(X,Y,W) is an A

Covering hierarchy if B union C == A

Disjoint hierarchy if B intersect C == None

# Tree basic options

## Map each class to a different relation

A(X,Y) Key: X

B(X,Z) Key: X

C(X,W) Key: X

To query, you must join A and B

Pro: Simple

Con: Potentially lots of joins

## Store information about all entities together

A(X,Y) Key: X

B(X,Y,Z) Key: X

C(X,Y,W) Key: X

Pro: No joins are necessary

Con: If something that can be both B’s and C’s, then storing redundant Y values.

(i.e. if the hierarchy is not covering)

If the hierarchy is disjoint, then we can only store it as A or B or C

## Merge all entities into one

ABC (X, Y, Z, W, isB, isC)

If something is a B, then there is no W value

If something is a C, then there is no Z value

If something is an A, then there is no Z or W value

Pro: there is no joins necessary and there is no redundancy

Con: Many attributes with missing values, bad efficiency if ABC is much bigger as a result