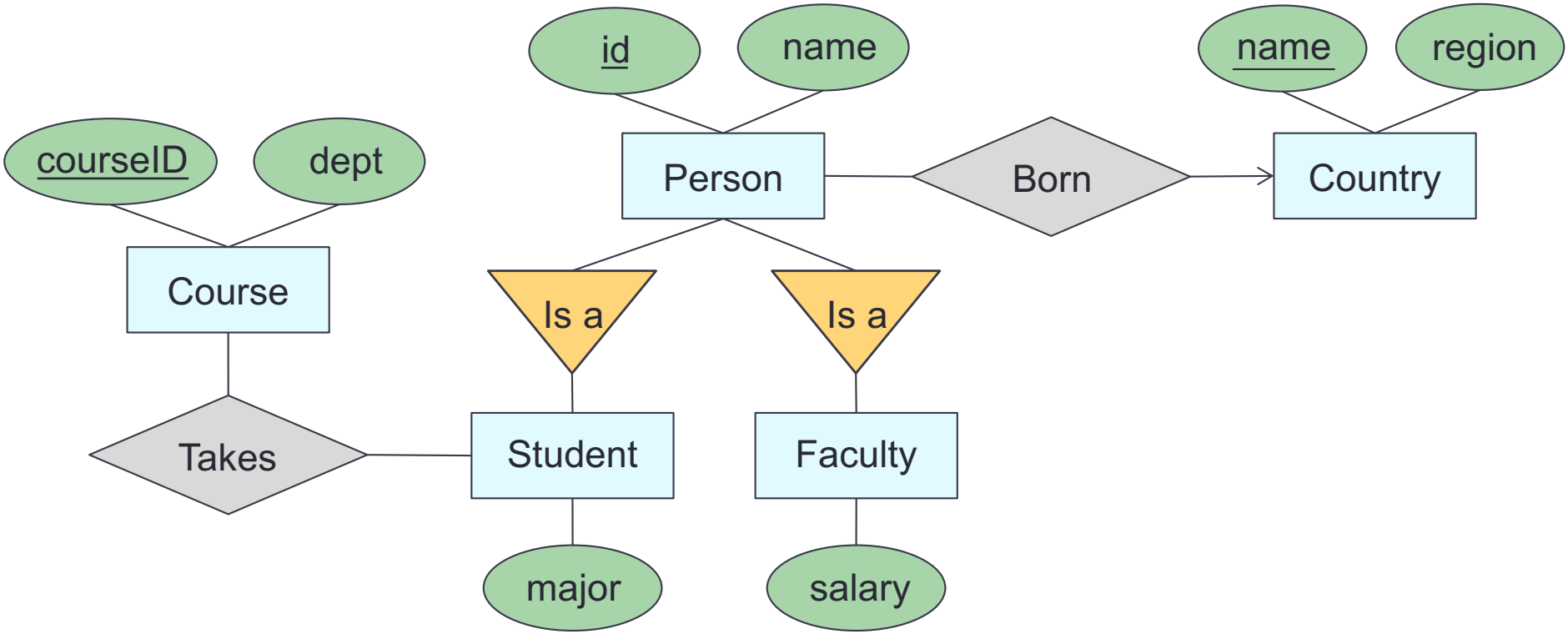
Schema Refinement
Functional Dependencies

CS 4750
Database Systems

[A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, Ch.7]
[Ricardo and Urban, Database Illuminated, Ch.6]
[https://www.w3schools.in/dbms/database-normalization/]

Recap: E-R Diagram

Convert the following ER into Schema statement

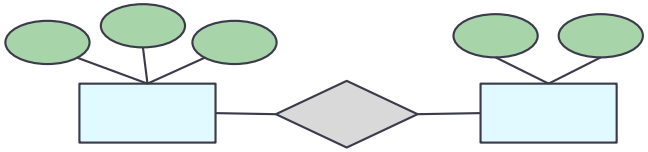


Database Design Process

revisit

Interact with users and domain experts to characterize the data

Translate requirements into conceptual model (E-R diagrams)

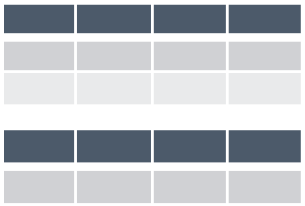


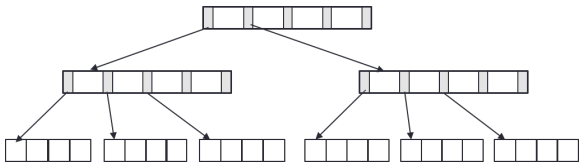
Convert the model to relational model (schema and constraints)

02/02 – 02/11

Normalize and develop conceptual (logical) schema of the database

Develop physical schema (partitioning and indexing)





Goals for 02/02 – 02/11

Figure out the fundamentals of what makes a good DB schema and being able to apply them to design a DB

- Avoid redundancy and anomalies
- Ensure data interrelationships

How:

- Recognize anomalies (things we want to avoid)
- Understand Functional Dependencies (FDs)
- Understand closures and formal definitions of keys
- Understand decomposition and normalization (3NF and BCNF)

Why Worry about Sound Structures?

If your database structure isn't sound, what could happen?

Let's share opinion / idea / experience !!

Does "Bad" database design impact any of the following?

- Data manipulation – adding, updating, deleting
- Retrieving data
- Data integrity, consistency
- Table relationships

Let's Consider: Friend Book

Make a simple friend book that can:

- Hold info about friend's name, email, phone, and city
- Associate **friends** with the **city** they live in
- Associate **friends** with any **phone numbers** they have

name	email	phone	city
Humpty	humpty@uva.edu	434-111-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-2222	Charlottesville
Mickey	mickey@uva.edu	434-333-3333	Fairfax
Minnie	minnie@uva.edu	434-555-5555	Alexandria

This instance does the job … but are there issues?

Let's Consider: Friend Book (2)

Make a simple friend book that can:

- Hold info about friend's name, email, phone, and city
- Associate **friends** with the **city** they live in
- Associate **friends** with any **phone numbers** they have

name	email	phone	city
Humpty	humpty@uva.edu	434-111-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-2222	Charlottesville
Mickey	mickey@uva.edu	434-333-3333	Fairfax
Minnie	minnie@uva.edu	434-555-5555	Alexandria

What if we want to update Dumpty's city to Norfolk

- Redundancy → slow update

part of data can be derived from other parts

Let's Consider: Friend Book (3)

Make a simple friend book that can:

- Hold info about friend's name, email, phone, and city
- Associate **friends** with the **city** they live in
- Associate **friends** with any **phone numbers** they have

name	email	phone	city
Humpty	humpty@uva.edu	434-111-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-2222	Charlottesville
Mickey	mickey@uva.edu	434-333-3333	Fairfax
Minnie	minnie@uva.edu	434-555-5555	Alexandria

How to delete Minnie's phone without deleting Minnie?

- Deletion anomalies

a consistent state becomes inconsistent after a transaction

Let's Consider: Friend Book (4)

Possible way to solve the anomalies – converting

name	email	phone	city
Humpty	humpty@uva.edu	434-111-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-1111	Charlottesville
Dumpty	dumpty@uva.edu	434-222-2222	Charlottesville
Mickey	mickey@uva.edu	434-333-3333	Fairfax
Minnie	minnie@uva.edu	434-555-5555	Alexandria

into the following

name	email	city
Humpty	humpty@uva.edu	Charlottesville
Dumpty	dumpty@uva.edu	Charlottesville
Mickey	mickey@uva.edu	Fairfax
Minnie	minnie@uva.edu	Alexandria

email	phone
humpty@uva.edu	434-111-1111
dumpty@uva.edu	434-222-1111
dumpty@uva.edu	434-222-2222
mickey@uva.edu	434-333-3333
minnie@uva.edu	434-555-5555

How can we systematically avoid redundancy and anomaly?

Another Example: TA Info Problems with "Bad" DB Design

Consider info about TAs

- Associate **year** with the **hourly_rate**

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

The hourly_rate of Humpty can be derived from the hourly_rate of Mickey (or Minnie) since they are all 4th year and we know year determines hourly_rate.

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

TA Info: Update Anomaly (2)

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	~~12~~ 13	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

- What if we attempt to update hourly_rate of Humpty?
  - Only one copy of hourly_rate has been updated
- Hourly_rate determined by year=4 appears in multiple tuples in the table
- Cannot change hourly_rate in just the 1st tuple

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

TA Info: Insertion Anomaly (3)

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10
aw6e	Awesome	2	**?? Null ?**	10

- What if we want to insert a new employee, who is in 2nd year, but we don't know the hourly rate for the 2nd year?
- Cannot insert a new employee into the table

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

TA Info: Deletion Anomaly (4)

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
~~dt2y~~	~~Dumpty~~	~~3~~	~~10~~	~~20~~
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
~~dh5h~~	~~Duhhuh~~	~~3~~	~~10~~	~~10~~

- What if we delete all employees who are in 3rd year?
- Lose information about the hourly_rate for 3rd year

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

TA Info: Potential Solution (5)

We can solve the redundancy and anomaly by converting this

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

into these

computingID	name	year	hours_worked
ht1y	Humpty	4	20
dt2y	Dumpty	3	20
md3y	Mickey	4	15
mn4e	Minnie	4	16
dh5h	Duhhuh	3	10

year	hourly_rate
4	12
3	10

How can we systematically avoid redundancy and anomaly?

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

General Design Guidelines

- Semantics of attributes should be self-evident
- Avoid redundancy – between tuples, relations
- Avoid NULL values in tuples
- If certain tuples should not exist, don't allow them

Database design = process or organizing data into a database model by considering data needs to be stored and the interrelationship of the data

Database design is about
characterizing data and the organizing data

How to describe properties we know or see in the data

How to organize data to promote ease of use and efficiency

Data Interrelationships

Rules that govern data

- Domain knowledge – things in the real world
- Pattern analysis

ATA_Emp(computingID, name, year, hourly_rate, hours_worked)

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

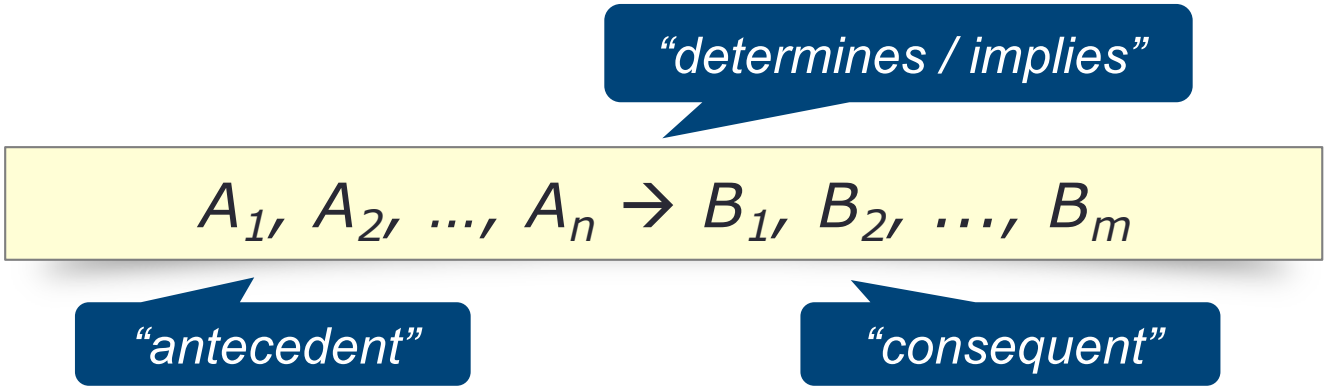
We can:

- Associate computingID with name — computing → name
- Associate year with hourly_rate — year → hourly_rate

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

Functional Dependencies (FDs)

- Describe data interrelationships
- Constraints on the set of relations
- Determine how to break/decompose a relation



- For each pair of tuples t_1 and t_2 in a relation r
  - If two tuples agree on the attributes, $A_1, A_2, \ldots, A_n$
  - Then they must also agree on the attributes $B_1, B_2, \ldots, B_m$

Functional Dependencies (FDs)

- A relation can have multiple functional dependencies
  - Example: schema of relation R(A, B, C, D)

    FDs: { A → B, B → C, C → CD }

- FD holds over a relation *R* if, for every allowable instance *r* of *R*, *r* satisfies the FD
  - Given some instance *r* of *R*, we can check if it violates some FD or not

    t1.X = t2.X but *t1.Y ≠ t2.Y*

  - We cannot tell if FD holds over R by looking at an instance (cannot prove non-existence of violation)
  - This is the same for all integrity constraints

Examples

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	12	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

year → hourly_rate holds

computingID	name	year	hourly_rate	hours_worked
ht1y	Humpty	4	13	20
dt2y	Dumpty	3	10	20
md3y	Mickey	4	12	15
mn4e	Minnie	4	12	16
dh5h	Duhhuh	3	10	10

year → hourly_rate does not hold

[Note: the hourly_rates in this example are made up. They are not associated with any organization.]

Let's Try: FDs

A	B	C
1	aa	x
1	aa	x
2	bb	y
2	cc	y
3	bb	z

Which of the following is functional dependency?

(a) A → B
(b) B → C
(c) AB → C
(d) A → C
(e) C → B

Possible
Not enough information
(cannot prove non-existence of violation)

Let's Try: FDs (2)

EmpID	Name	xPhone	Position
E1001	Mickey	6543	Clerk
E2353	Minnie	1234	Helpdesk
E4567	Daisy	9876	Salesrep
E1234	Donald	9876	Salesrep
E9372	Humpty	1234	Lawyer

List possible FD(s)

More Example

ProductName	Category	Color	Department	Price
Beyblade	Gadget	Green	Toys	12
Drone	Gadget	Green	Toys	80

Do all the following FDs hold on this instance?

ProductName → Color

Category → Department

~~Color, Category → Price~~

Possible

If we can be sure that every instance of R will be one in which a given FD is true, then we say that **R satisfies the FD**

If we say that R satisfies an FD, we are stating a constraint on R

Interesting Observation

If all these FDs are true:

ProductName → Color

Category → Department

Color, Category → Price

Then, this FD also holds:

ProductName, Category → Price

If we find out from application domain that a relation satisfies some FDs, it does not mean that we found all the FDs that it satisfies.

There may be more FDs implied by the ones we have.

Reasoning about FDs

To use FDs to break a relation, focus on a key and list all possible FDs

Given some FDs, infer additional FDs using the following rules:

Reflexivity	if b subset a, a → b
Augmentation	if a → b, then ac → bc
Transitivity	if a → b and b → c, then a → c
Union	if a → b and a → c, then a → bc
Decomposition	if a → bc, then a → b and a → c (separately)
Pseudo-transitivity	if a → b and cb → d, then ac → d

There are many rules that let us infer that one FD X → A holds in any relation instance that satisfies some other given set of FDs. To verify that X → A holds, compute the closure of X, using the given FDs to expand X until it includes A

Example: Reasoning

Supposed we know

custID → name

Can we conclude the following?

custID, hair_color → name

Yes!

Adding more attributes to the antecedent can never remove attributes in the consequent

Example: Reasoning (2)

Supposed we know

custID → name

Can we conclude the following?

custID → name, hair_color

No!

Impossible to introduce hair_color to the consequent without also introducing it to the antecedent

Attribute Closure and F+

- Attribute Closure (**α+**) = all FDs a particular attribute can imply
- Closure of F (**F+**) = a set of all FDs that are implied by F
- An FD *f* is logically implied by a set of FDs *F* if *f* holds whenever all FDs in *F* hold

- To compute the closure of a set of functional dependencies F

```
F+ = F
repeat
    for each functional dependency f in F+
        apply reflexivity and augmentation rules on f
        add the resulting functional dependencies to F+
    for each pair of functional dependencies f1 and f2 in F+
        if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F+
until F+ does not change any further
```

Example: Attribute Closure

Let's do this together

Given R(A,B,C)

FDs = {A → B, B → C }

Compute the attribute closures for all attribute and combination of attributes. Then, think about what can inferred.

	A	B	C	AB	AC	BC	ABC
A	✓	✓	✓	✓	✓	✓	✓
B							
C							
AB							
AC							
BC							
ABC							

Summary: attribute closure
A+ = ABC
B+ =
C+ =
AB+ =
AC+ =
BC+ =
ABC+ =

Example1: Computing F+

Let's do this together

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

FDs = { A → C, B → B, C → BD, D → E }

Compute F+

(1) write all LHS & remaining

A	→
B	→
C	→
D	→
E	→

(2) copy FDs as is

A	→	C
B	→	B
C	→	B D
D	→	E
E	→	

(3) apply reflexivity

A	→	A C
B	→	B
C	→	BCD
D	→	DE
E	→	E

(4) apply transitivity

A	→	ABCDE
B	→	B
C	→	BCDE
D	→	DE
E	→	E

F+ = { A → ABCDE, B → B, C → BCDE, D → DE, E → E }

Example2: Computing F+

Let's do this together

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

FDs = {A → BC, B → D, CD → E }

Compute F+

Decompose: A → A, A → B, A → C

(1) write all LHS & remaining

A	→
B	→
C	→
D	→
E	→
CD	→

(2) copy FDs as is

A	→	BC
B	→	D
C	→	
D	→	
E	→	
CD	→	E

(3) apply reflexivity

A	→	ABC
B	→	B D
C	→	C
D	→	D
E	→	E
CD	→	CDE

(4) apply transitivity

A	→	ABCD
B	→	B D
C	→	C
D	→	D
E	→	E
CD	→	CDE

Example2: Computing F+ (cont.)

Let's do this together

Decompose: A → A, A → B, A → C, A → D
Union: A → C and A → D, then A → CD

(from previous page)

(4) apply transitivity

A	→	ABCD
B	→	B D
C	→	C
D	→	D
E	→	E
CD	→	CDE

(5) apply transitivity

A	→	ABCDE
B	→	B D
C	→	C
D	→	D
E	→	E
CD	→	CDE

F+ = { A → ABCDE, B → BD, C → C, D → D, E → E, CD → CDE }

Wrap-Up

- Problems with "Bad" DB design
- Characterizing data / describe properties of data with Functional dependencies (FDs)
- Reasoning about FDs
- Attribute closure and F+ … FDs can help us fine potential keys

What's next?

- Fine-tuning database structures and normalization