Database Design & Normalization: Part 1

COM 3563: Database Implementation

Avraham Leff

Yeshiva University

avraham.leff@yu.edu

COM3563: Fall 2020

Today's Lecture: Overview

1. Introduction
2. Running Example
3. Functional Dependencies

Much material derived from lectures created by Professor Zvi Kedem, NYU Courant: used with permission

Some figures from John Greiner, Rice University, used with permission

Alert

- The topic of normalization is associated with a lot of "computer science" formalism and notation
  - Example: approach taken in Textbook Chapter 7 ("Relational database design")
- My plan: two lectures
  - Today's lecture: an introduction which makes an effort to also give a more "approachable" view
  - Next lecture: more functional dependency material
  - Please give feedback after you read the Textbook presentation

Logical Database Design

- Input: a set of database tables
  - The tables may have been derived from an E-R diagram, for example
- Our task: are these tables correctly designed?
- This is not "merely" a theoretical question: we need to know whether these tables will
  - Store the information that we need so that we can retrieve & update efficiently
  - Enforce "business rules" constraints
  - Avoid anomalies (e.g., "redundant data which gets us into trouble later")

Key point: if we identify flaws in the E-R design, we must fix the problems, possible by restructuring the "input" set of tables

Big Picture

- Normalization: *decompositions ("refactoring") which fix flaws in a database design*
- Normal form: a specific "type" of decomposition (or "style") that a relation follows that embodies a specific "niceness" condition
- As we shall see, many "normal forms" exist, each with its own "degree of niceness"

- Motivation for normalization: we get objective criteria for a specific database design
- A "good" normalization produces the "good properties" listed on the previous slide

An (Initial, Very Simple) Motivating Example

R	Name	SSN	DOB	Grade	Salary
	A	121	2367	2	80
	A	132	3678	3	70
	B	101	3498	4	70
	C	106	2987	2	80

- Key insight: this relation instance has importance beyond the raw data that it contains!
- It also encodes (implicitly) the following business rule
  - "*Salary* is completely determined by *Grade*"

> - Data + the enterprise's business rules are the crown jewels of the business
> - It is only business rules that determine whether or not the relation is "factored" or normalized correctly

Unfortunately: this simple design incorporates several serious problems ☹

Design Problems: I

R	Name	SSN	DOB	Grade	Salary
	A	121	2367	2	80
	A	132	3678	3	70
	B	101	3498	4	70
	C	106	2987	2	80

The problem isn't with the data: the problem is the business rules representation

- Redundancy: the *"If Grade = 2 then Salary = 80"* rule is duplicated
  - Redundancy → update anomaly
  - In this scenario, we update one tuple (and thus change the business rule), but don't update the "cloned" tuples
- Insert anomaly: we cannot store a *Grade* → *Salary* rule for a grade that is not currently associated with an employee
  - Example: we can't store a *"If Grade = 1 then Salary = 90"* rule

Design Problems: II

R	Name	SSN	DOB	Grade	Salary
	A	121	2367	2	80
	A	132	3678	3	70
	B	101	3498	4	70
	C	106	2987	2	80

Again: the problem isn't with the data: the problem is the business rules representation

- Delete anomaly: the business rule is too tightly coupled to employee data
  - Example: what if employee with *SSN* of "132" leaves?
    - We lose the *"If Grade = 3, then Salary = 70"* rule
- Adding "fake employee data" to keep the *Grade* → *Salary* rule raises new problems
  - Example: what *SSN* should be assigned to a "fake employee"?
    - Note: can't be NULL since *SSN* is a primary key

Solution: Represent the Information Differently

Decomposition technique replaces one table with two

```
SELECT INTO S
Name, SSN, DOB,
    Grade
FROM R;

SELECT INTO T
Grade, Salary
FROM R;
```

S	Name	SSN	DOB	Grade
	A	121	2367	2
	A	132	3678	3
	B	101	3498	4
	C	106	2987	2

T	Grade	Salary
	2	80
	3	70
	4	70

Think of decomposition as a refactoring exercise in which we represent the *grade* $\rightarrow$ *salary* business rule in table *T* explicitly

Does Decomposition Solve Previous Problems?

S	Name	SSN	DOB	Grade
	A	121	2367	2
	A	132	3678	3
	B	101	3498	4
	C	106	2987	2

T	Grade	Salary
	2	80
	3	70
	4	70

- We have eliminated redundant business rules and represented them explicitly
- Now we can store the *"If Grade = 3 then Salary = 70"* rule ...
  - Even when all employees with this *Grade* have left the company
- Now we can create new rule instances even when no employees with that grade (currently) exist
  - Example: *"If Grade = 1 then Salary = 90"* rule
- Q: but perhaps decompositions cause us to lose other information?
- A: no, they don't ☺ (next slide)

Does Decomposition Lose Information?

Given *S* and *T*, we can reconstruct (the original) *R* by using a natural join

S	Name	SSN	DOB	Grade
	A	121	2367	2
	A	132	3678	3
	B	101	3498	4
	C	106	2987	2

T	Grade	Salary
	2	80
	3	70
	4	70

```sql
SELECT INTO R Name, SSN, DOB, S.Grade
AS Grade, Salary FROM T, S
WHERE T.Grade = S.Grade;
```

To produce

R	Name	SSN	DOB	Grade	Salary
	A	121	2367	2	80
	A	132	3678	3	70
	B	101	3498	4	70
	C	106	2987	2	80

Natural Join

- Given several tables: $R_1, R_2, \ldots R_n$, their natural join is computed using the following pseudo-code

```
SELECT INTO R one copy of each column name
FROM R1, R2, ... Rn
WHERE equal-named columns have to be equal
```

- Intuition: we decomposed R into $R_1, R_2, \ldots R_n$ using SELECT statements
  - We can reconstruct the original R using the corresponding SELECT statement on those "equal-valued" column
- Quick SQL review:

```
SELECT * FROM R1 NATURAL JOIN R2;
```

is a more elegant way of saying

```
SELECT ... FROM R1, R2 WHERE ...
```

- The natural join creates an implicit join clause based on the common ("same name") columns in the tables being joined

Generalizing the Problem

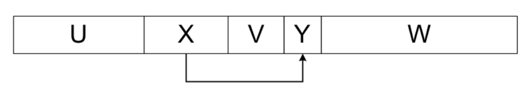
- Whenever a database table contains two columns X and Y ...
  - In previous example: X is *Grade*, Y is *Salary*
- Such that X is neither a primary key or unique key ...
  - (This condition permits database to allow multiple tuples with the same X value)
- And our business experts tell us that tuples that are equal on X must also be equal on Y ...
- Then: we have a problem ☹

> - The business rule specifying how X constrains or determines Y is "embedded implicitly" in different tuples
> - Result:
>   - The business rule is inherently going to be specified redundantly
>   - The business rule cannot be stored independently

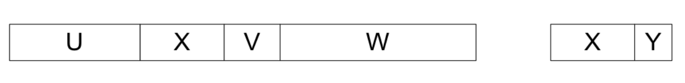
Generalizing the Solution

(Note: this is still only intuition and will need to be formalized)

- We took a single table containing columns X (*Grade*) and Y (*Salary*)



- And replaced it with two tables



We'll Take Two Approaches

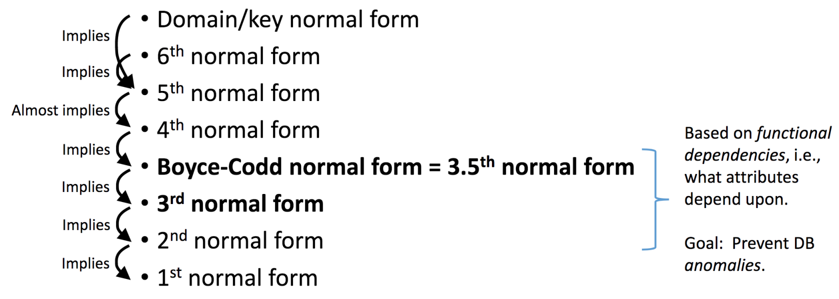
- IT practitioners do normalization (mostly) differently than the way computer scientists like to do it
  - This approach is somewhat *ad-hoc*, which has advantages and disadvantages
  - You should understand it if only to build your intuition and to be able to talk the "same language" as the IT folk
- Then: we'll focus on how computer scientists do normalization
  - Less intuitive, but algorithm based, and guarantees correct results
- We'll focus on three important issues
  - Removing redundancies
  - Lossless-join decompositions to fix those redundancies
  - Preservation of dependencies between attributes

Hierarchy of Normal Forms

(Note: not everyone will agree with these “editorial comments” ☺)

- Normal forms in order of increasing “quality” (and complexity ☹)
- First Normal Form (1NF): We have a table/relation ☺
- Second Normal Form (2NF): intermediate form in some obsolete algorithms
- Third Normal Form (3NF): very important; often the final form in logical database design
- Boyce-Codd Normal Form (BCNF): a final form that is very important in theory (but less used in practice)
- Fourth Normal Form (4NF) a final form but often its benefits can be obtained without a formal normalization process
- A plethora of additional (more exotic) normal forms exist

Normal Forms



- We'll start from the bottom of the hierarchy and work our way up
- Motivate the "higher" normalization form by showing how it addresses weaknesses in the "lower" form
- Focus on 3NF and BCNF

Running Example

(Small) University Database

	S	B	C	T	F	C	T	F
	Fang	1990	DB	Zvi	1	OS	Allan	2
	John	1980	OS	Allan	2	PL	Marsha	4
	Mary	1990	PL	Vijay	1			

The database states a collection of predicates ("truth statements") about this university

The attributes:

- S: Student
- B: Birth year of the Student
- C: Course that the Student took
- T: Teacher who taught the Course that the Student took
- F: Fee that the Student paid the Teacher for getting a good grade

Note: this example assumes that people are uniquely identified by their *first name* ☺

We Begin With A “Non-Relation”

	S	B	C	T	F	C	T	F
	Fang	1990	DB	Zvi	1	OS	Allan	2
	John	1980	OS	Allan	2	PL	Marsha	4
	Mary	1990	PL	Vijay	1			

- This representation is not even a relation (let alone an “unnormalized relation”)!
- Q: why isn’t this a relation (“seems OK to me”)?
- A: because it allows a student to take any number of courses
  - Implication: the number of attributes is not fixed
- Q: how to fix?
- A: use the technique of deconstructing multi-valued attributes into a set of single-valued attributes
  - More formally: we want to have attribute domains be atomic
  - That is: indivisible units, in contrast to e.g., sets of attributes that form a composite attribute
- OK: let’s transform this relation into 1NF ...

First Normal Form (1NF)

R	S	B	C	T	F
	Fang	1990	DB	Zvi	1
	John	1980	OS	Allan	2
	Mary	1990	PL	Vijay	1
	Fang	1990	OS	Allan	2
	John	1980	PL	Marsha	4

- 1NF: a relation with a fixed number of columns
- Alternatively: a relational schema is in first normal form if all its attribute domains are atomic
- 1NF is a minimal base-line for a relational database
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - This was considered to be "bad" ...until NOSQL & object-oriented databases were invented ☺

A Historical Note

- Always helpful to understand how terms were invented ☺
- Way, way, back when only file systems existed, data were often stored as *variable-length* records
- Relational tables must have *fixed-length tuples*, so variable-length records had to be normalized to fixed-length records

- This observation has nothing to do with how relational databases implement table storage
- Relational schema are defined at a higher, logical level
- Underlying storage management may actually use variable-length records in the implementation

Enhancing Our Example With Some Business Rules

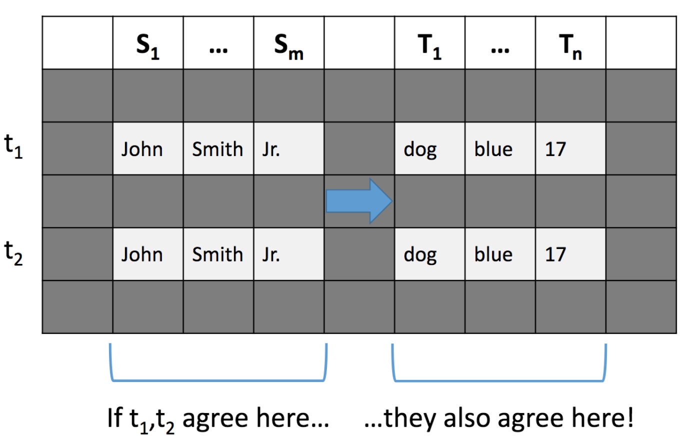
- Without further knowledge of the business domain, 1NF may be fine
- However: the constraints imposed by an enterprise's business rules often make 1NF infeasible
- Our example has the following business rules
  - Meaning: data are guaranteed to exhibit these properties
  - Some of these may seem like "d'oh", others less so, all will be important

1. A student can have only one birth year
2. A teacher has to charge the same "upgrade" fee from every student she teaches.
3. A teacher can teach only one course at a time
   - OK to teach multiple courses at different times but never teaches another course at the same time
4. A student can take a specific course from one teacher only (or not at all)

Functional Dependencies (I)

- These business rules can be formally described using functional dependencies
- (Assume for now that there are no NULL-valued attributes)
- If P and Q are sets of columns …
- P functionally determines Q, (written "$P \rightarrow Q$") *if and only if*
  - Any two tuples that are equal on (all the attributes in) P must be equal on (all the attributes in) Q
- Alternatively:
  - If a value of P is specified, it "forces" some (specific) value of Q; in other words: Q is a function of P
- (In today's initial example, we had $\mathit{Grade} \rightarrow \mathit{Salary}$)

Functional Dependencies (II)



This figure illustrates an FD in which $S \to T$

Handling NULLs

- If P and Q are sets of columns ...
- P functionally determines Q, (written "$P \to Q$") *if and only if*
  1. There are no NULL-valued attributes in P
  2. Any two rows that are equal on (all the attributes in) P must be equal on all the attributes of Q, where NULL is treated like any "real attribute"

	O	P	Q
	A	10	x
	B	20	y
	C	20	y
	D	30	NULL
	E	30	NULL

Incorporating FDs Into Our Example

The business rules:

1. A student can have only one birth year: $S \rightarrow B$
2. A teacher has to charge the same fee from every student he/she teaches: $T \rightarrow F$
3. A teacher can teach only one course (perhaps at different times, different offerings, etc, but never another course): $T \rightarrow C$
4. A student can take any specific course from one teacher only (or not at all): $SC \rightarrow T$
    - (Note: in the last rule, P consists of multiple attributes)

Determining a Primary Key: I

R	S	B	C	T	F
	Fang	1990	DB	Zvi	1
	John	1980	OS	Allan	2
	Mary	1990	PL	Vijay	1
	Fang	1990	OS	Allan	2
	John	1980	PL	Marsha	4

- Functional dependencies: $S \to B, T \to F, T \to C, SC \to T$
- ST is a candidate primary key, because given ST
  - S $\to$ B
  - T $\to$ F
  - T $\to$ C
- Any subset of ST cannot serve as a primary key
  - From S alone, we cannot determine T, C, or F
  - From T alone, we cannot determine S or B

Determining a Primary Key: II

R	S	B	C	T	F
	Fang	1990	DB	Zvi	1
	John	1980	OS	Allan	2
	Mary	1990	PL	Vijay	1
	Fang	1990	OS	Allan	2
	John	1980	PL	Marsha	4

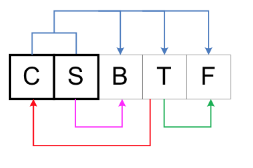
- Functional dependencies: $S \rightarrow B, T \rightarrow F, T \rightarrow C, SC \rightarrow T$
- SC is a also a candidate primary key, because given SC
  - S → B
  - SC → T
  - T → F (we can now use *SC* to determine *F* because of previous rule)
- Any subset of SC cannot serve as a primary key
  - From S alone, we cannot determine T, C, or F
  - From C alone, we cannot determine B, S, T, or F

Determining a Primary Key: III

- Functional dependencies: $S \to B, T \to F, T \to C, SC \to T$
- Note: the relation contains only SBCTF as attributes
- ST can serve as primary key because $ST \to SBCTF$
  - Can be written as "$ST \to BCF$", since $ST \to ST$
  - "Columns determine themselves"
- SC can serve as primary key because $SC \to BTF$
- Going forward, we'll (arbitrarily) pick SC as the primary key for this relation

Drawing Functional Dependencies

- Functional dependencies: $S \rightarrow B, T \rightarrow F, T \rightarrow C, SC \rightarrow T$
- A graphical representation can make it easier to understand the various types of functional dependencies
- (Note: use of color for clarity, no semantics attached)



- Each column drawn in its own box
- Box containing (primary) key (remember: can be more than one) is drawn in a box with "thick" borders
- Arrows drawn "above the boxes" represent FDs from the full key
- Arrows drawn "below the boxes" represent FDs that are not from the full key

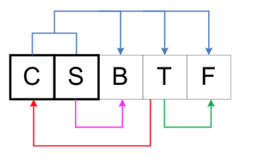
Note: this is not a "standard graphical representation", but I think it will help clarify concepts

FDs So Far

R	S	B	C	T	F
	Fang	1990	DB	Zvi	1
	John	1980	OS	Allan	2
	Mary	1990	PL	Vijay	1
	Fang	1990	OS	Allan	2
	John	1980	PL	Marsha	4

- Because $S \rightarrow B$, given a specific S, either it does not appear in the table, or wherever it does appear, it "produces" the same value of B
  - "John" is associated with "1980", wherever he appears
  - Because "Lilian" does not appear in this relation instance, she has no "B" value at all
- Because $SC \rightarrow BTF$, given a specific SC, either it does not appear in the table, or wherever it does appear, it "produces" the same value of BTF
  - "Mary, PL" is associated with "1990, Vijay, 1"
  - "Mary, OS" does not appear at all

Arrows drawn "below the boxes" represent FDs that are not from the full key: let's drill down on these "not from full key" cases

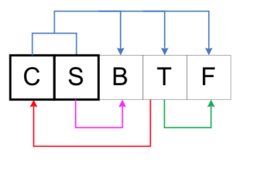


- Partial dependency: A dependency from part of the primary key to outside the key
  - Example: $S \rightarrow B$
- Transitive dependency: A dependency from outside the key to outside the key
  - Informal definition, formal definition on next slide
  - Example: $T \rightarrow F$
- "Into key" dependency: A dependency from outside the key into (all or part of) the key
  - Example: $T \rightarrow C$

More Formal Definition of "Transitive Dependency"

- We have in our relation
  - $SC \rightarrow T$
  - $T \rightarrow F$
  - $SC \rightarrow F$
- Functional dependency $SC \rightarrow F$ can be decomposed into
  - $SC \rightarrow T$
  - $T \rightarrow F$
- The $SC \rightarrow F$ is therefore termed a transitive dependency (or "transitive closure") because
  - T depends on the SC key ($SC \rightarrow T$)
    - T "is outside the key"
  - $T \rightarrow F$
  - Therefore $SC \rightarrow F$
- Alternatively: *"$SC \rightarrow F$ is a transitive dependency because it can be decomposed into $SC \rightarrow T$ and $T \rightarrow F$ so that this FD can be derived by transitivity"*

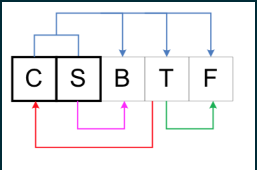
- The "not from the full key" dependencies result in a bad database design
  - Inability to store important information
  - Data redundancy



- Example: Partial dependency $S \rightarrow B$
- Scenario: a new Student appears who has not yet registered for a course
- This S has a specific B, but this cannot be stored in the table
  - We do not have a value of C yet, and the attributes of the primary key cannot be NULL
- This is referred to as an insert anomaly

Functional Dependencies & Anomalies: II

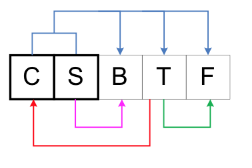
- The "not from the full key" dependencies result in bad database design
  - Inability to store important information
  - Data redundancy



- Example: Partial dependency $S \rightarrow B$
  - Variation on previous scenario
- Scenario: Mary withdraws from the only Course she has
- This S has a specific B but we are no longer able to store that B
  - Mary is no longer associated with any C, and the attributes of the primary key cannot be NULL
- This is referred to as a delete anomaly

Functional Dependencies & Anomalies: III

- The "not from the full key" dependencies result in bad database design
  - Inability to store important information
  - Data redundancy



- Example: "Into key" dependency $T \rightarrow C$
- Scenario: erase the value of C in the "Fang taught by Allan" tuple
- We can reconstruct the C value (it must be "OS")
  - Because we have another fact: *"John was taught OS by Allan"*
  - Using the rule: *"Every teacher teaches only one subject"*

- This "ability to reconstruct" proves that the relation contains redundant data
  - That's bad because redundant data implies the potential for subsequent inconsistency
  - We have an update anomaly

How To Remove Anomalies?

R	S	B	C	T	F
	Fang	1990	DB	Zvi	1
	John	1980	OS	Allan	2
	Mary	1990	PL	Vijay	1
	Fang	1990	OS	Allan	2
	John	1980	PL	Marsha	4

- The way to handle these problems is to replace a table with other, equivalent, tables that don't have these problems
  - That is: we decompose the original table
  - One anomaly at a time …

	<u>S</u>	B
	Fang	1990
	John	1980
	Mary	1990
	Fang	1990
	John	1980

	<u>S</u>	<u>C</u>	T	F
	Fang	DB	Zvi	1
	John	OS	Allan	2
	Mary	PL	Vijay	1
	Fang	OS	Allan	2
	John	PL	Marsha	4

Removing the $S \rightarrow B$ anomaly: no longer a partial dependency

Second Normal Form (2NF)

	S	B
	Fang	1990
	John	1980
	Mary	1990
	Fang	1990
	John	1980

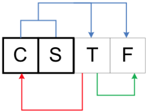
	S	C	T	F
	Fang	DB	Zvi	1
	John	OS	Allan	2
	Mary	PL	Vijay	1
	Fang	OS	Allan	2
	John	PL	Marsha	4

- Each of these tables in our database is in Second Normal Form
- 2NF means:
  - Tables are 1NF
  - No partial dependencies in any table
- The table on the left contains no anomalies!
  - As we shall see, the table on the right does contain anomalies ☹
- Note: the 1NF → 2NF decomposition was a lossless join decomposition
  - Meaning: we can use natural join operation to restore the original table

Some Anomalies Remain!

Recall: we have a transitive dependency because $SC \to T, T \to F, SC \to F$

	S	C	T	F
	Fang	DB	Zvi	1
	John	OS	Allan	2
	Mary	PL	Vijay	1
	Fang	OS	Allan	2
	John	PL	Marsha	4



- Desired property: all attributes should depend on the (entire) key, and nothing but the key
- Transitive dependency breaks that property!
  - T is not a key, so can appear in multiple tuples as a dependent attribute
  - But $T \to F$ implies that any tuple with given teacher t must have the same fee value f
- Repeating the association of a given $t \to f$ is therefore redundant information
  - Therefore vulnerable to update anomalies since you might modify one such association and not the other

Fixing Transitive Dependency Anomalies

Decomposition is the answer!

- Split the table with $X \rightarrow Y \rightarrow Z$ transitive dependency into two tables
  - One containing $X \rightarrow Y$
  - One containing $Y \rightarrow Z$
- Y is a key in the second table, and no redundant Z attributes in that table

S	C	T	F
Fang	DB	Zvi	1
John	OS	Allan	2
Mary	PL	Vijay	1
Fang	OS	Allan	2
John	PL	Marsha	4

S	C	T
Fang	DB	Zvi
John	OS	Allan
Mary	PL	Vijay
Fang	OS	Allan
John	PL	Marsha

T	F
Zvi	1
Allan	2
Vijay	1
Allan	2
Marsha	4

Third Normal Form (3NF)

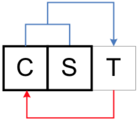
S	C	T
Fang	DB	Zvi
John	OS	Allan
Mary	PL	Vijay
Fang	OS	Allan
John	PL	Marsha

T	F
Zvi	1
Allan	2
Vijay	1
Allan	2
Marsha	4

- Each of these tables in our database is in Third Normal Form
- 3NF means:
  - Tables are 2NF
  - No transitive dependencies in any table
- Note: the 2NF → 3NF decomposition was a lossless join decomposition
  - Meaning: we can use natural join operation to restore the original table

3NF Can Still Allow Anomalies

	S	C	T
	Fang	DB	Zvi
	John	OS	Allan
	Mary	PL	Vijay
	Fang	OS	Allan
	John	PL	Marsha

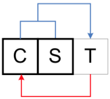


- We have an “into key” dependency in which a FD exists from a column outside the key into (all or part of) the key
- Redundancy anomaly: since $T \rightarrow C$, we have multiple tuples in which $Allan \rightarrow OS$
- Insertion anomaly: how do we represent the fact that a teacher exists if she doesn’t yet teach a course

A relation R is in 3NF *iff*

- Whenever $A \rightarrow B$ is a "non-trivial" FD in R then
  - Either: A is a superkey for R
  - Or: every element of B is part of a key

	S	C	T
	Fang	DB	Zvi
	John	OS	Allan
	Mary	PL	Vijay
	Fang	OS	Allan
	John	PL	Marsha



- In our example: $T \rightarrow C$ is a "non-trivial" FD
  - On the one hand: A is not a superkey of R (e.g., values of T do not determine values of S)
  - But: every element of B is part of a key (in this case C is part of the SC key)
- The existence of a "non-trivial" FD implies potential anomalies (and we can solve them via BCNF, "3.5NF")

Note: a trivial FD exists when $X \rightarrow Y$ and Y is a subset of X. We are interested in "non-trivial" FDs.

Together, let's take the pledge ☺

> *Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key*
>
> *so help me Codd*
>
> *Paraphrase of the original text with addendum*

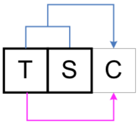
- "the key": table must be in 1NF (so every tuple has a unique key)
- (non-key attributes must depend on) "the whole key": table must be in 2NF (so no attribute has an FD on a proper subset of any candidate key)
- (non-key attributes must depend on) "nothing but the key": table must be in 3NF (so we don't have a non-key field stating a fact about another non-key field)

Segue to Boyce-Codd Normal Form (BCNF)

- BCNF is also known as 3.5NF
  - Conceptually, in the same space as 3NF
  - Yes, there is a 4NF ☺
- A relation R is in BCNF *iff* whenever $A \rightarrow B$ is a non-trivial FD in R we have
  - A is a superkey for R: meaning, "if $A \rightarrow B$, then A must comprise a key in R"
  - And R must be in 1NF
- Note: BCNF acts differently from 3NF only when the relation has multiple overlapping candidate keys
  - Any table that has only one candidate key and is in 3NF is already in BCNF
  - Therefore: no attribute is functionally dependent on anything besides that key
- In our example we can eliminate the 3NF anomalies by taking a step back
- We previously (decomposed the table to *SCT*)
  - Then "arbitrarily" selected *SC* as the relation's key
  - What if we instead select *ST* as the key?

Morphing Our Example to BCNF

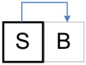
	S	C	T
	Fang	DB	Zvi
	John	OS	Allan
	Mary	PL	Vijay
	Fang	OS	Allan
	John	PL	Marsha



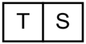
- Use *TS* as the key instead of *TC*
- We no longer have "outside of the key" dependencies
- Only issue is a "partial dependency", and we already know how to handle that ☺ (decomposition on next slide)

Our Example In BCNF Form

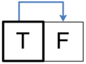
S	B
Fang	1990
John	1980
Mary	1990



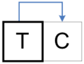
S	I
Fang	Zvi
John	Allan
Mary	Vijay
Fang	Allan
John	Marsha



I	F
Zvi	1
Allan	2
Vijay	1
Marsha	4



C	I
DB	Zvi
OS	Allan
PL	Vijay
PL	Marsha



- A relation R is in BCNF *iff* whenever $A \rightarrow B$ is a non-trivial FD in R we have
  - A is a superkey for R
- Each of these tables is in BCNF

This Lecture … & Next Lecture

- When you read the Textbook discussion, you'll note that I've approached the topic of database normalization very differently
- Textbook discussion jumps into formalism immediately
- Textbook doesn't motivate the formalism as much as I'd prefer
- I plan to cover the formalism in the next lecture
  - Feedback encouraged – after you read the textbook and after the next lecture

Today's Lecture: Wrapping it Up

Introduction

Running Example

Functional Dependencies

Readings & Homework

- Textbook covers this in Chapter 7, you may want to wait until next lecture