ECE30030/ITP30010 Database Systems

Normalization

Reading: Chapter 7

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Agenda

- Motivating example

- Normal forms

- Normalization example

- *Appendix: Normalization theory*

When Data is Jumbled...

- Suppose that we combine *instructor* and *department*
  - (Below represents the join on *instructor* and *department*)

ID	name	salary	dept_name	building	budget
76766	Crick	72000.00	Biology	Watson	90000.00
10101	Srinivasan	65000.00	Comp. Sci.	Taylor	100000.00
45565	Katz	75000.00	Comp. Sci.	Taylor	100000.00
83821	Brandt	92000.00	Comp. Sci.	Taylor	100000.00
98345	Kim	80000.00	Elec. Eng.	Taylor	85000.00
12121	Wu	90000.00	Finance	Painter	120000.00
76543	Singh	80000.00	Finance	Painter	120000.00
32343	El Said	60000.00	History	Painter	50000.00
58583	Califieri	62000.00	History	Painter	50000.00
15151	Mozart	40000.00	Music	Packard	80000.00
22222	Einstein	95000.00	Physics	Watson	70000.00
33456	Gold	87000.00	Physics	Watson	70000.00

When Data is Jumbled…

- Key issues
  - Repetition of data → increases the size of database
  - *Data consistency issues*
    - Insertion anomaly: Inserting redundant data for every new record
    - Deletion anomaly: Loss of related data, when some data is deleted
    - Update anomaly: When updating certain information, every single record must be updated

ID	name	salary	dept_name	building	budget
76766	Crick	72000.00	Biology	Watson	90000.00
10101	Srinivasan	65000.00	Comp. Sci.	Taylor	100000.00
45565	Katz	75000.00	Comp. Sci.	Taylor	100000.00
83821	Brandt	92000.00	Comp. Sci.	Taylor	100000.00
98345	Kim	80000.00	Elec. Eng.	Taylor	85000.00
12121	Wu	90000.00	Finance	Painter	120000.00
76543	Singh	80000.00	Finance	Painter	120000.00
32343	El Said	60000.00	History	Painter	50000.00
58583	Califieri	62000.00	History	Painter	50000.00
15151	Mozart	40000.00	Music	Packard	80000.00

Solution: Decomposition!

- How to avoid the repetition-of-information problem?

ID	name	salary	dept_name	building	budget
76766	Crick	72000.00	Biology	Watson	90000.00
10101	Srinivasan	65000.00	Comp. Sci.	Taylor	100000.00
45565	Katz	75000.00	Comp. Sci.	Taylor	100000.00
83821	Brandt	92000.00	Comp. Sci.	Taylor	100000.00
98345	Kim	80000.00	Elec. Eng.	Taylor	85000.00
12121	Wu	90000.00	Finance	Painter	120000.00
76543	Singh	80000.00	Finance	Painter	120000.00
32343	El Said	60000.00	History	Painter	50000.00
58583	Califieri	62000.00	History	Painter	50000.00
15151	Mozart	40000.00	Music	Packard	80000.00
22222	Einstein	95000.00	Physics	Watson	70000.00
33456	Gold	87000.00	Physics	Watson	70000.00

- A: Decompose it into two schemas (as they were)
  - Normalization = decomposition of relational schemas
  - Key idea: split relational schemas such that only directly related data composes a relation

Decomposition

- Less redundancy → Uses smaller disk storage; Causes less issues associated with insertion, deletion, and update anomalies

- Not all decompositions are good
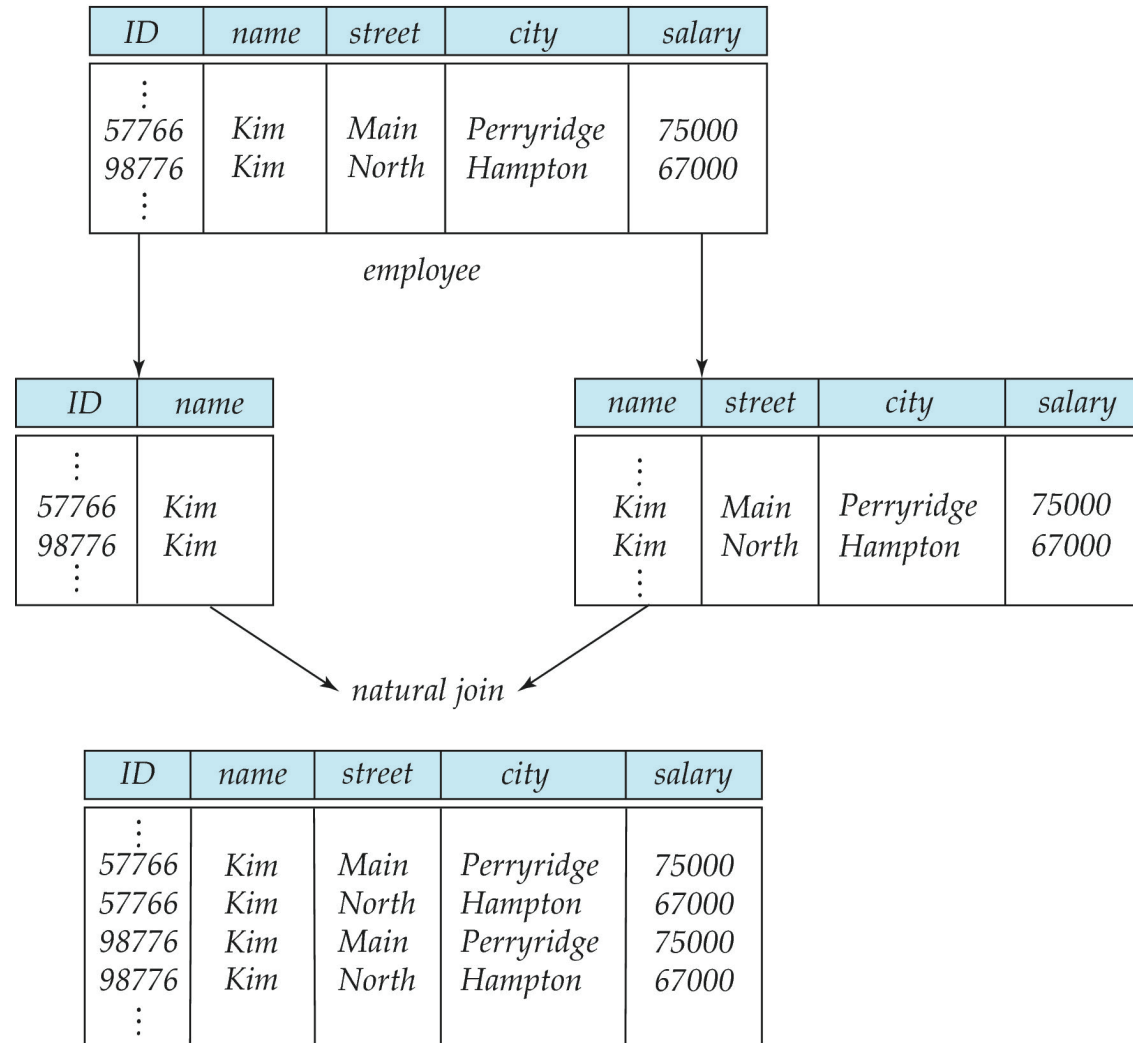 - *E.g.*, Suppose we decompose

        employee(ID, name, street, city, salary)

    into

        employee1 (*ID, name*)
        employee2 (*name, street, city, salary*)

    → Problem: *What if there are two employees with the same name?*

- Lossy decomposition: a decomposition from which the original relation cannot be reconstructed

Lossy Decomposition

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Lossless Decomposition

- Let R be a relation schema and let R_1 and R_2 form a decomposition of R; that is, $R = R_1 \cup R_2$

- A decomposition is a lossless decomposition if there is no loss of information by replacing R with the two relation schemas $R_1 \cup R_2$
  - Formally, $\prod_{R_1}(r) \bowtie \prod_{R_2}(r) = r$

    - *C.f.*, Conversely, a decomposition is lossy if when the join of the projection results is computed, a proper superset of the original relation is returned
    $$r \subset \prod_{R_1}(r) \bowtie \prod_{R_2}(r)$$

Example: Lossless Decomposition

- Decomposition of $R = (A, B, C)$ → $R_1 = (A, B)$; $R_2 = (B, C)$

$$\Pi_A (r) \bowtie \Pi_B (r)$$

Normalization

- Database normalization: Process of structuring a database to reduce data redundancy and improve data integrity
  - In accordance with a series of normal forms (next topic)
  - Through the process:
    - One can decompose relations to suppress data anomalies
    - One can make sure the decomposition is lossless

Agenda

- Motivating example
- **Normal forms**
- Normalization example

- *Appendix: Normalization theory*

Normal Forms

- Normalization process
  - Normalization is a database design technique, which is used to design a relational database table up to higher normal form
    - Procedurally separates logically independent (but related) data entities into multiple relations
    - Maintains the connections using keys
  - Progressive process
    - A higher level of database normalization cannot be achieved unless the previous levels have been satisfied
      - UNF: Unnormalized form
      - 1NF: First normal form
      - 2NF: Second normal form
      - 3NF: Third normal form
      - BCNF: Boyce-Codd normal form (3.5NF)
      - 4NF: Fourth normal form
      - …

Normal Forms

- Normal forms are backed by a set of normalization theories
    - *Functional dependencies*
        - *Partial dependencies*
        - *Transitive dependencies*
    - *Multi-valued dependencies*

- These theories decide whether a particular relation R is in "good form"
    - For a relation R is not in "good form", decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
        - Each relation is in good form
        - The decomposition is a lossless decomposition

First Normal Form (1NF)

- Requirements
  - A relation should consist of atomic values
    - Atomic value: a value that cannot be divided (≃primitive data types in JAVA)
      - Atomic - INT, FLOAT, DOUBLE, DECIMAL (NUMERIC), CHAR, VARCHAR, BLOB, TEXT
      - NOGO - Structure, List (array)
  - Attributes should have unique identifiers

- Step 1 of the normalization process
  - *"If the tables in your DB does not follow 1NF, stop using database"*

First Normal Form (1NF)

- 1NF checklist
    1. Each column should contain an atomic value
        - Entries like (x, y) violate this rule
    2. Each column should contain values that are in the same data domain
        - Do not mix different types of values in a column
    3. Each column should have a unique name
        - Duplicate names lead to confusion while accessing data
    4. The order in which data is stored does not matter
        - Using SQL, one can easily fetch data in any order
    5. There are no duplicated rows in the table
        - Primary key (PK) ensures:
            - Attributes that are part of PK are unique
            - Attributes that are part of PK are not null

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world
  - *E.g.*, some of the constraints that are expected to hold in the *university* database are:
    - *Students* and *instructors* are *uniquely identified* by their ID
    - Each *student* and *instructor* has *only one* name
    - Each *instructor* and *student* is associated with *only one* department
    - Each *department* has *only one* value for its budget, and *only one* associated *building*

  - Relations require that the value for a certain set of attributes determines uniquely the value for another set of attributes
    - A functional dependency is a generalization of the notion of a key (= trivial dependency)

Functional Dependencies

- Let R be a relation schema, α and β be its attributes ($\alpha \subseteq R$ and $\beta \subseteq R$)

- The functional dependency $\alpha \rightarrow \beta$ holds on R if and only if, for any relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β

  - That is, $t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta]$

  - *E.g.*, Consider $r(A,B)$ with the following instance of r

A	B
1	4
1	5
3	7

    - On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold (only the values of B are unique)

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using super keys

    - *E.g.*, consider the schema:
      in_dep (<u>*ID, name, salary, dept_name*</u>, *building, budget*)
        - *ID = instructor* ID
        - *dept_name* = department of the instructor

    - We expect the following functional dependencies to hold:
        - **dept_name → building**
        - **ID → building**
        - ⇒ *dept_name and ID are <u>super keys</u> = candidate keys*
            - * *ID, dept_name, {ID, dept_name}*

    - We would not expect the next to hold:
        - *dept_name → salary*

Functional Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

{

Candidate key = a1, a2, a3, {a1, a2}, {a2,a3}, {a1, a3}, {a1,a2,a3}

  - a_1 is a part of the key
    - Functional dependency: a_4 depends on its key, $a_1 a_2 a_3$
      - b_id, b_name, bd_detail_attrib1, , bd_detail_attrib2
        - Candidate keys: b_id, b_name, {b_id, b_name}

    - It should not be such that a_4 depends on a_1, and does not depend on $a_1 a_2 a_3$ $(a_1 \rightarrow a_4) \Rightarrow$ partial dependency

Trivial Functional Dependencies

- *A* functional dependency is trivial if it is satisfied by all instances of a relation
  - Example:
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Functional Dependencies

- Example: *student2*(*ID, name, tot_cred*)
  - Hold
    - *ID → name*
    - *ID → tot_cred*
  - Not hold
    - *tot_cred → name*
    - *name → tot_cred*

🔑 ID	name	tot_cred
00128	Zhang	102
12345	Shankar	32
19991	Brandt	80
23121	Chavez	110
44553	Peltier	56
45678	Levy	46
54321	Williams	54
55739	Sanchez	38
70557	Snow	0
76543	Brown	58
76653	Aoi	60
98765	Bourikas	98
98988	Tanaka	120

Functional Dependencies

- *score*(*score_id*, <u>*student_id*</u>, <u>*subject_id*</u>, *score*)
  - *{student_id, subject_id}* → *score*

score_id	🔑 student_id	🔑 subject_id	score
1	10	1	82
3	11	1	85
2	10	2	77
4	11	2	82
5	11	4	95

First Normal Form (1NF)

- Example

student_id	name	course
21800999	James Inexistente	Algorithm, OS
21800998	Mike Inexistente	Java
21800997	Matt Inexistente	Algorithm, DB

First Normal Form (1NF)

- Example

student_id	name
21800999	James Inexistente
21800998	Mike Inexistente
21800997	Matt Inexistente

student_id	course
21800999	Algorithm
21800999	OS
21800998	Java
21800997	Algorithm
21800997	DB

First Normal Form (1NF)

- Example

student_id	name
21800999	James Inexistente
21800998	Mike Inexistente
21800997	Matt Inexistente

student_id	course
21800999	Algorithm
21800999	OS
21800998	Java
21800997	Algorithm
21800997	DB

Second Normal Form (2NF)

- Requirements
  - A relation should be in 1NF (normal forms should be applied in order)

  - A relation should NOT have a non-PK that is functionally dependent on any subset of any candidate key = NO PARTIAL DEPENDENCIES!
    - Any attributes other than PK should be dependent on PK
    - It should not have partial dependencies

    - PK: Primary Key
      - An attribute or a set of attributes that uniquely identifies each tuple in a relation
      - A PK can fetch data from any specific data in a relation
      - *E.g.,* get the department name of *student_ID* = 21800999

Partial Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

  Primary key (Composite)

  - a_1 is a part of the primary key
    - Partial dependency: a_4 depends on a_1, and does not depend on $a_1 a_2 a_3$
      - Key: $a_1\ a_2\ a_3$
      - Dependency: $a_1 \rightarrow a_4$

Eliminating Partial Dependencies

- Example: *teaches2*(_ID, course_id, sec_id, semester, year_, name)
  - *ID → name*

ID	course_id	sec_id	semester	year	name
10101	CS-101	1	Fall	2,017	Srinivasan
10101	CS-315	1	Spring	2,018	Srinivasan
10101	CS-347	1	Fall	2,017	Srinivasan
12121	FIN-201	1	Spring	2,018	Wu
15151	MU-199	1	Spring	2,018	Mozart
22222	PHY-101	1	Fall	2,017	Einstein
32343	HIS-351	1	Spring	2,018	El Said
45565	CS-101	1	Spring	2,018	Katz
45565	CS-319	1	Spring	2,018	Katz
76766	BIO-101	1	Summer	2,017	Crick
76766	BIO-301	1	Summer	2,018	Crick
83821	CS-190	1	Spring	2,017	Brandt
83821	CS-190	2	Spring	2,017	Brandt
83821	CS-319	2	Spring	2,018	Brandt
98345	EE-181	1	Spring	2,017	Kim

Eliminating Partial Dependencies

- Example:
  - *teaches2a*(*ID*, *course_id*, *sec_id*, *semester*, *year*)
  - *teaches2b*(*ID*, *name*)

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2,017
10101	CS-315	1	Spring	2,018
10101	CS-347	1	Fall	2,017
12121	FIN-201	1	Spring	2,018
15151	MU-199	1	Spring	2,018
22222	PHY-101	1	Fall	2,017
32343	HIS-351	1	Spring	2,018
45565	CS-101	1	Spring	2,018
45565	CS-319	1	Spring	2,018
76766	BIO-101	1	Summer	2,017
76766	BIO-301	1	Summer	2,018
83821	CS-190	1	Spring	2,017
83821	CS-190	2	Spring	2,017
83821	CS-319	2	Spring	2,018
98345	EE-181	1	Spring	2,017

ID	name
10101	Srinivasan
10101	Srinivasan
10101	Srinivasan
12121	Wu
15151	Mozart
22222	Einstein
32343	El Said
45565	Katz
45565	Katz
76766	Crick
76766	Crick
83821	Brandt
83821	Brandt
83821	Brandt
98345	Kim

Eliminating Partial Dependencies

- Example: *score*(*score_id*, *student_id*, *subject_id*, *score*, *instructor*)
  - *subject_id* → *instructor* : *partial dependency*

score_id	student_id	subject_id	score	instructor
1	10	1	82	James Packer
3	11	1	95	James Packer
2	10	2	77	Cole Miller
4	11	2	71	Cole Miller
5	11	4	96	Adam Lee

Eliminating Partial Dependencies

- Example: *score_a*(*score_id*, *student_id*, *subject_id*, *score*)
  score_b(*subject_id*, *instructor*)

score_id	student_id	subject_id	score
1	10	1	82
3	11	1	95
2	10	2	77
4	11	2	71
5	11	4	96

subject_id	instructor
1	James Packer
2	Cole Miller
4	Adam Lee

Third Normal Form (3NF)

- Requirements
  - A relation should be in 2NF
  - A relation should NOT have transitive dependencies
    - Transitive dependency: A non-PK attribute depends on another non-PK attribute or a set of non-PK attributes

Transitive Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

  Candidate key = a1, a2, {a1,a2}

  - a_1 is the primary key in R
    - a_3 depends on a_1 ($a_1 \rightarrow a_3$) -- OK
    - a_4 depends on a_3 ($a_1 \rightarrow a_3 \rightarrow a_4$) ... non-PK $\rightarrow$ non-PK

Eliminating Transitive Dependencies

- Example: *score2*(*id, student_id, subject_id, exam_id, exam_name, exam_score*)
  - {*student_id, subject_id*} --> FINAL or MIDTERM
  - {*student_id, subject_id*} --> 77 43 … from which ?
  - {*student_id, subject_id*} → *exam_name* → *exam_score* :: transitive dependency

id	🔑 student_id	🔑 subject_id	exam_name	exam_score
2	10	1	FINAL	77
5	11	1	FINAL	68
3	10	2	FINAL	92
7	11	2	FINAL	81
9	14	2	FINAL	54
1	10	1	MIDTERM	43
6	11	1	MIDTERM	87
4	10	2	MIDTERM	65
8	11	2	MIDTERM	62
10	14	2	MIDTERM	97

Eliminating Transitive Dependencies

- Example: *score2*(*id, student_id, subject_id, exam_id, exam_name, exam_score*)
  - *student_id, subject_id → exam_name*
  - *student_id, subject_id → exam_name → exam_score*

id	🔑 student_id	🔑 subject_id	exam_name	exam_score
2	10	1	FINAL	77
5	11	1	FINAL	68
3	10	2	FINAL	92
7	11	2	FINAL	81
9	14	2	FINAL	54
1	10	1	MIDTERM	43
6	11	1	MIDTERM	87
4	10	2	MIDTERM	65
8	11	2	MIDTERM	62
10	14	2	MIDTERM	97

Eliminating Transitive Dependencies

- Example: *score2a*(*id, student_id, subject_id, exam_id, exam_name*)
  score2b(*student_id, subject_id, exam_id, exam_score*)

id	student_id	subject_id	exam_id	exam_name
2	10	1	101	FINAL
5	11	1	101	FINAL
3	10	2	101	FINAL
7	11	2	101	FINAL
9	14	2	101	FINAL
1	10	1	102	MIDTERM
6	11	1	102	MIDTERM
4	10	2	102	MIDTERM
8	11	2	102	MIDTERM
10	14	2	102	MIDTERM

student_id	subject_id	exam_id	exam_score
10	1	101	77
11	1	101	68
10	2	101	92
11	2	101	81
14	2	101	54
10	1	102	43
11	1	102	87
10	2	102	65
11	2	102	62
14	2	102	97

Boyce-Codd Normal Form (BCNF) = 3.5NF

- Requirements
  - A relation should be in 3NF
  - For any dependency A → B, A should be a super key
    - For A → B, if A is non-PK, then it is NOT in BCNF

Boyce-Codd Normal Form (BCNF)

- Example: *takes2*(*student_id*, *subject*, *instructor*)

student_id	subject	instructor
21800999	C++	Dr. Cpp
21800999	Java	Dr. Java
21800998	C++	Dr. C
21800997	Python	Dr. Python
21800996	C++	Dr. Cpp

- (student, subject) → instructor
- Instructor → subject
  - A non-PK identifies a member of PK: Not in BCNF

Boyce-Codd Normal Form (BCNF)

- Example: *takes2a*(*student_id, section_id*),
  takes2b(*section_id, subject, instructor*)

student_id	section_id
21800999	101
21800999	103
21800998	102
21800997	104
21800996	101

section_id	subject	instructor
101	C++	Dr. Cpp
103	Java	Dr. Java
102	C++	Dr. C
104	Python	Dr. Python

Boyce-Codd Normal Form (BCNF)

- Example: *takes2a*(*student_id*, *section_id*),
         takes2b(*section_id*, *subject, instructor*)

student_id	section_id
21800999	101
21800999	103
21800998	102
21800997	104
21800996	101

section_id	subject	instructor
101	C++	Dr. Cpp
103	Java	Dr. Java
102	C++	Dr. C
104	Python	Dr. Python

Fourth Normal Form (4NF)

- Requirements
    - A relation should be in BCNF
    - A relation should NOT have multi-valued dependency
        - Multi-valued dependency occurs due to a bad DB schema

        - Multi-valued dependency occurs when a relation has more than 3 attributes
        - For a relation with attributes A, B, C
            - having dependency, A → B, and
            - B and C are independent from each other
                → Then, the relation may have a multi-valued dependency

Multi-valued Dependencies

- Example
  - *student_id → course*
  - *student_id → activity*
  - *course ⊥ activity* (independent)

student_id	course	activity
21800999	Statistics	Soccer
21800999	Linear algebra	Basketball
21800999	Statistics	Basketball
21800999	Linear algebra	Soccer

Multi-valued Dependencies

- Example

student_id	course
21800999	Statistics
21800999	Linear algebra
21800998	Physics
21800998	Programming 101

student_id	activity
21800999	Soccer
21800999	Basketball
21800998	Pool
21800997	Soccer

Summary: Normal Forms

- Theories – Normal forms

Theory	Key Idea	Normal Form
Functional dependency	(PK → non-PK)	2NF
Partial dependency	Part of PK → non-PK	2NF
Transitive dependency	Non-PK → non-PK	3NF
-	Non-PK → PK	BCNF
Multi-valued dependency		4NF
Join dependency		5NF

Overall DB Design Process

- Let us assume schema *R* is given:

  (E-R Model)

  - *R* could have been generated when converting E-R diagram to a set of tables

  (Normalization)

  - *R* could have been a single relation containing *all* attributes that are of interest (called universal relation)
  - Normalization breaks *R* into smaller relations

  (Mixed)

  - *R* could have been the result of some ad-hoc design of relations, which we then test/convert to normal form

E-R Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization

- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - *E.g.,* an *employee* entity with
    - attributes
      department_name and *building*
    - functional dependency
      department_name $\rightarrow$ *building*
    - Good design would have made *department* an entity

- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- We may want to use non-normalized schema for performance
  - For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*

- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code

- Alternative 2: Use a materialized view defined a *course* ⨝ *prereq*
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Remaining Issues

- Some aspects of database design are not caught by normalization

- Example (to be avoided)
  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*)
    - Above are well normalized (in BCNF), but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Above are well normalized (in BCNF), but makes querying across years difficult and requires new attribute each year

  - This is an example of a crosstab, where values for one attribute become column names
  - → Better schema: *earnings* (*company_id, year, amount*)

Agenda

- Motivating example

- Normal forms

- **Normalization example**

- *Appendix: Normalization theory*

Example (taken from *Wikipedia*)

- Provided:

Title	Author	Author Nationality	Format	Price	Subject	Pages	Thickness	Publisher	Publisher Country	Publication Type	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Chad Russell	American	Hardcover	49.99	MySQL, Database, Design	520	Thick	Apress	USA	E-book	1	Tutorial

- Satisfying 1NF

Title	Format	Author	Author Nationality	Price	Subject 1	Subject 2	Subject 3	Pages	Thickness	Publisher	Publisher country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Hardcover	Chad Russell	American	49.99	MySQL	Database	Design	520	Thick	Apress	USA	1	Tutorial

Example (taken from *Wikipedia*)

- Satisfying 1NF (cont'd – further improvement)

Title	Format	Author	Author Nationality	Price	Pages	Thickness	Publisher	Publisher country	Genre ID	Genre Name
Beginning MySQL Database Design and Optimization	Hardcover	Chad Russell	American	49.99	520	Thick	Apress	USA	1	Tutorial

Subject

Subject ID	Subject name
1	MySQL
2	Database
3	Design

Title - Subject

Title	Subject ID
Beginning MySQL Database Design and Optimization	1
Beginning MySQL Database Design and Optimization	2
Beginning MySQL Database Design and Optimization	3

Example (taken from *Wikipedia*)

- Provided:

Book

Title	Format	Author	Author Nationality	Price	Pages	Thickness	Genre ID	Genre Name	Publisher ID
Beginning MySQL Database Design and Optimization	Hardcover	Chad Russell	American	49.99	520	Thick	1	Tutorial	1
Beginning MySQL Database Design and Optimization	E-book	Chad Russell	American	22.34	520	Thick	1	Tutorial	1
The Relational Model for Database Management: Version 2	E-book	E.F.Codd	British	13.88	538	Thick	2	Popular science	2
The Relational Model for Database Management: Version 2	Paperback	E.F.Codd	British	39.99	538	Thick	2	Popular science	2

- Compound key {Title, Format}
  - Partial dependency: Title → Author, Author Nationality, Pages, Thickness, Genre ID, Genre Name, Publisher ID

Example (taken from *Wikipedia*)

- Satisfying 2NF

Book

Title	Author	Author Nationality	Pages	Thickness	Genre ID	Genre Name	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	1	Tutorial	1
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	2	Popular science	2

Format - Prices

Title	Format	Price
Beginning MySQL Database Design and Optimization	Hardcover	49.99
Beginning MySQL Database Design and Optimization	E-book	22.34
The Relational Model for Database Management: Version 2	E-book	13.88
The Relational Model for Database Management: Version 2	Paperback	39.99

Example (taken from *Wikipedia*)

- Satisfying 3NF

Book

Title	Author	Author Nationality	Pages	Thickness	Genre ID	Genre Name	*Publisher ID*
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	1	Tutorial	*1*
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	2	Popular science	*2*

- Transitive dependency: Genre ID → Genre Name

Example (taken from *Wikipedia*)

- Satisfying 3NF

Book

Title	Author	Author Nationality	Pages	Thickness	Genre ID	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	1	1
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	2	2
Learning SQL	Alan Beaulieu	American	338	Slim	1	3
SQL Cookbook	Anthony Molinaro	American	636	Thick	1	3

Book Genres

Genre ID	Genre Name
1	Tutorial
2	Popular science

Example (taken from *Wikipedia*)

- Satisfying BCNF

Book

Title	Author	Author Nationality	Pages	Thickness	Genre ID	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	American	520	Thick	1	1
The Relational Model for Database Management: Version 2	E.F.Codd	British	538	Thick	2	2
Learning SQL	Alan Beaulieu	American	338	Slim	1	3
SQL Cookbook	Anthony Molinaro	American	636	Thick	1	3

- Non-PK → Non-PK: Author → Author Nationality

Example (taken from *Wikipedia*)

- Satisfying BCNF

Book

Title	Author	Pages	Thickness	Genre ID	Publisher ID
Beginning MySQL Database Design and Optimization	Chad Russell	520	Thick	1	1
The Relational Model for Database Management: Version 2	E.F.Codd	538	Thick	2	2
Learning SQL	Alan Beaulieu	338	Slim	1	3
SQL Cookbook	Anthony Molinaro	636	Thick	1	3

Author - Nationality

Author	Author Nationality
Chad Russell	American
E.F.Codd	British
Alan Beaulieu	American
Anthony Molinaro	American

Example (taken from *Wikipedia*)

- Provided:
  - Assume the database is owned by a book retailer franchise that has several franchisees that own shops in different locations

Franchisee - Book Location

Franchisee ID	Title	Location
1	Beginning MySQL Database Design and Optimization	California
1	Beginning MySQL Database Design and Optimization	Florida
1	Beginning MySQL Database Design and Optimization	Texas
1	The Relational Model for Database Management: Version 2	California
1	The Relational Model for Database Management: Version 2	Florida
1	The Relational Model for Database Management: Version 2	Texas
2	Beginning MySQL Database Design and Optimization	California
2	Beginning MySQL Database Design and Optimization	Florida
2	Beginning MySQL Database Design and Optimization	Texas
2	The Relational Model for Database Management: Version 2	California
2	The Relational Model for Database Management: Version 2	Florida
2	The Relational Model for Database Management: Version 2	Texas
3	Beginning MySQL Database Design and Optimization	Texas

Example (taken from *Wikipedia*)

- Satisfying 4NF
    - If we assume that all available books are offered in each area, the Title is not unambiguously bound to a certain Location
      → Does not satisfy 4NF

Franchisee - Book

Franchisee ID	Title
1	Beginning MySQL Database Design and Optimization
1	The Relational Model for Database Management: Version 2
2	Beginning MySQL Database Design and Optimization
2	The Relational Model for Database Management: Version 2
3	Beginning MySQL Database Design and Optimization

Franchisee - Location

Franchisee ID	Location
1	California
1	Florida
1	Texas
2	California
2	Florida
2	Texas
3	Texas

- Source: https://en.wikipedia.org/wiki/Database_normalization

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
No repeating groups	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells have single value)[8]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute[8]	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[9]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components[8]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Agenda

- Motivating example

- Normal forms

- Normalization example

- ***Appendix: Normalization theory***

Normalization Theory

- Decide whether a particular relation R is in "good form"
  - For a relation R is not in "good form", decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
    - Each relation is in good form
    - The decomposition is a lossless decomposition

- The normalization is based on a set of relevant theories:
  - Functional dependencies
    - Partial dependencies
    - Transitive dependencies
  - Multivalued dependencies

Functional Dependencies

- There are usually a variety of constraints (rules) on the data in the real world
  - *E.g.*, some of the constraints that are expected to hold in a university database are:
    - Students and instructors are *uniquely identified* by their ID
    - Each student and instructor has *only one* name
    - Each instructor and student is (primarily) associated with *only one* department
    - Each department has *only one* value for its budget, and *only one* associated building

  - Relations require that the value for a certain set of attributes determines uniquely the value for another set of attributes
    - A functional dependency is a generalization of the notion of a key
      (= trivial dependency)

Functional Dependencies

- Let R be a relation schema, α and β be its attributes ($\alpha \subseteq R$ and $\beta \subseteq R$)

- The functional dependency $\alpha \rightarrow \beta$ holds on R if and only if, for any relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α, they also agree on the attributes β
  - That is, $t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta]$

  - *E.g.*, Consider $r(A,B)$ with the following instance of r

A	B
1	4
1	5
3	7

  - On this instance, $B \rightarrow A$ hold; $A \rightarrow B$ does **NOT** hold (only the values of B are unique)

Keys and Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using super keys

    - *E.g.*, consider the schema:
        in_dep (*ID, name, salary, dept_name, building, budget*)
        - ID = instructor ID
        - dept_name = department of the instructor

    - We expect the following functional dependencies to hold:
        - **dept_name$\rightarrow$ building**
        - **ID $\rightarrow$ building**
        $\Rightarrow$ *dept_name and ID are <u>super keys</u> = candidate keys*
            * ID, dept_name, {ID, dept_name}

    - We would not expect the next to hold:
        - *dept_name $\rightarrow$ salary*

Functional Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

  Candidate key = a1, a2, a3, {a1, a2}, {a2,a3}, {a1, a3}, {a1,a2,a3}

  - a_1 is a part of the key
    - Functional dependency: a_4 depends on its key, $a_1a_2a_3$
      - b_id, b_name, bd_detail_attrib1, , bd_detail_attrib2
        - Candidate keys: b_id, b_name, {b_id, b_name}

    - It should not be such that a_4 depends on a_1, and does not depend on $a_1a_2a_3$ ($a_1 \rightarrow a_4$) $\Rightarrow$ partial dependency

Trivial Functional Dependencies

- *A* functional dependency is trivial if it is satisfied by all instances of a relation
    - Example:
        - *ID, name $\rightarrow$ ID*
        - *name $\rightarrow$ name*
    - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Functional Dependencies

- *student2(ID, name, tot_cred)*
  - ID → name
  - ID → tot_cred
  - tot_cred → name (X)

🔑 ID	name	tot_cred
00128	Zhang	102
12345	Shankar	32
19991	Brandt	80
23121	Chavez	110
44553	Peltier	56
45678	Levy	46
54321	Williams	54
55739	Sanchez	38
70557	Snow	0
76543	Brown	58
76653	Aoi	60
98765	Bourikas	98
98988	Tanaka	120

Functional Dependencies

- *SCORE(score_id, <u>student_id</u>, <u>subject_id</u>, score)*
  - {student_id, subject_id} → score

score_id	🔑 student_id	🔑 subject_id	score
1	10	1	82
3	11	1	85
2	10	2	77
4	11	2	82
5	11	4	95

Partial Dependencies

- For relation $R = a_1 \ a_2 \ a_3 \ a_4$

Primary key (Composite)

- a_1 is a part of the primary key
  - a_4 depends on a_1, and does not depend on $a_1 a_2 a_3$ ($a_1 \rightarrow a_4$)
    - Key: $a_1 \ a_2 \ a_3$
    - Dependency: $a_1 \rightarrow a_4$

Partial Dependencies

- *teaches2(<u>ID</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, name)*
    - ID → name
    - {course_id, sec_id, semester, year} → name

ID	course_id	sec_id	semester	year	name
10101	CS-101	1	Fall	2,017	Srinivasan
10101	CS-315	1	Spring	2,018	Srinivasan
10101	CS-347	1	Fall	2,017	Srinivasan
12121	FIN-201	1	Spring	2,018	Wu
15151	MU-199	1	Spring	2,018	Mozart
22222	PHY-101	1	Fall	2,017	Einstein
32343	HIS-351	1	Spring	2,018	El Said
45565	CS-101	1	Spring	2,018	Katz
45565	CS-319	1	Spring	2,018	Katz
76766	BIO-101	1	Summer	2,017	Crick
76766	BIO-301	1	Summer	2,018	Crick
83821	CS-190	1	Spring	2,017	Brandt
83821	CS-190	2	Spring	2,017	Brandt
83821	CS-319	2	Spring	2,018	Brandt
98345	EE-181	1	Spring	2,017	Kim

Partial Dependencies

- *SCORE(score_id, <u>student_id</u>, <u>subject_id</u>, score, instructor)*
  - subject_id → instructor (partial dependency) :: WE DON'T WANT TO HAVE THIS

score_id	🔑 student_id	🔑 subject_id	score	instructor
1	10	1	82	James Packer
3	11	1	85	James Packer
2	10	2	77	Cole Miller
4	11	2	82	Cole Miller
5	11	4	95	Adam Lee

Transitive Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

    Candidate key = a1, a2, {a1,a2}

    - a_1 is the primary key in R
        - a_3 depends on a_1 ($a_1 \rightarrow a_3$) -- OK
        - a_4 depends on a_3 ($a_1 \rightarrow a_3 \rightarrow a_4$) ... non-PK $\rightarrow$ non-PK

Transitive Dependencies

- *EXAM_SCORE*(*id, <u>student_id</u>, <u>subject_id</u>, exam_name, exam_score*)
    - {student_id, subject_id} --> FINAL or MIDTERM
    - {student_id, subject_id} --> 77   43 … from which ?
    - {student_id, subject_id} → exam_name → exam_score :: transitive dependency

id	🔑 student_id	🔑 subject_id	exam_name	exam_score
2	10	1	FINAL	77
5	11	1	FINAL	68
3	10	2	FINAL	92
7	11	2	FINAL	81
9	14	2	FINAL	54
1	10	1	MIDTERM	43
6	11	1	MIDTERM	87
4	10	2	MIDTERM	65
8	11	2	MIDTERM	62
10	14	2	MIDTERM	97

Multi-valued Dependencies (MVDs)

- Suppose we record *names* of *children*, and *phone numbers* for *instructors*
    - *inst_child*(*ID*, *child_name*)
    - *inst_phone*(*ID*, *phone_number*)

    → If we combine these schemas to obtain: (multiple orthogonal values in single table → this causes redundancy)
        - *inst_info*(*ID*, *child_name*, *phone_number*)

    → Example tuples: ID=999999 → {David, William} & {x-x-1234, x-x-4321}
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)

    - Any issues?

Multi-valued Dependencies (MVDs)

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$.

- The multivalued dependency $\alpha \rightarrow\rightarrow \beta$ holds on R
  if in any relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that
  $t_1[\alpha] = t_2[\alpha]$,
  there exist tuples t_3 and t_4 in r such that:
  $$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$
\begin{aligned}
t_3[\beta] &= t_1[\beta] \\
t_3[R - \beta] &= t_2[R - \beta] \\
t_4[\beta] &= t_2[\beta] \\
t_4[R - \beta] &= t_1[R - \beta]
\end{aligned}
$$

Multi-valued Dependencies (MVDs)

- Let R be a relation schema with a set of attributes that are partitioned into 3 non-empty subsets Y, Z, W

- We say that $Y \rightarrow\rightarrow Z$ (Y multidetermines Z)
  if and only if for all possible relations $r(R)$
  $$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$
  then
  $$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

- Note that since the behavior of Z and W are identical it follows that $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

Multi-valued Dependencies (MVDs)

- Tabular representation of $\alpha \twoheadrightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

Multi-valued Dependencies (MVDs)

- *FAV(student_id, course, activity)*
  - 21800999 - {statistics, Linear algebra} & {Soccer, basketball}
    - Favorite courses & favorite activities <= they are orthogonal

student_id	course	activity
21800999	Statistics	Soccer
21800999	Linear algebra	Basketball
21800999	Statistics	Basketball
21800999	Linear algebra	Soccer

Use of Functional Dependencies

- We use functional dependencies to:
  - Test relations to see if they are legal under a given set of functional dependencies
    - We say that *r* satisfies *F*, if a relation *r* is legal under a set *F* of functional dependencies
  - To specify constraints on the set of legal relations
    - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F*

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances
  - *E.g.*, a specific instance of *instructor* may, by chance, satisfy: *name* $\rightarrow$ *ID*

Use of Multi-valued Dependencies

- We use multivalued dependencies in two ways:

  1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies

  2. To specify constraints on the set of legal relations. We shall concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies

- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r ′* that does satisfy the multivalued dependency by adding tuples to *r*

EOF

- Coming next:
  - Advanced SQL