Agenda

- E-R diagrams
  - Mapping cardinalities
  - Primary keys in E-R models
  - Weak entity sets
  - **Reduction to relation schemas**

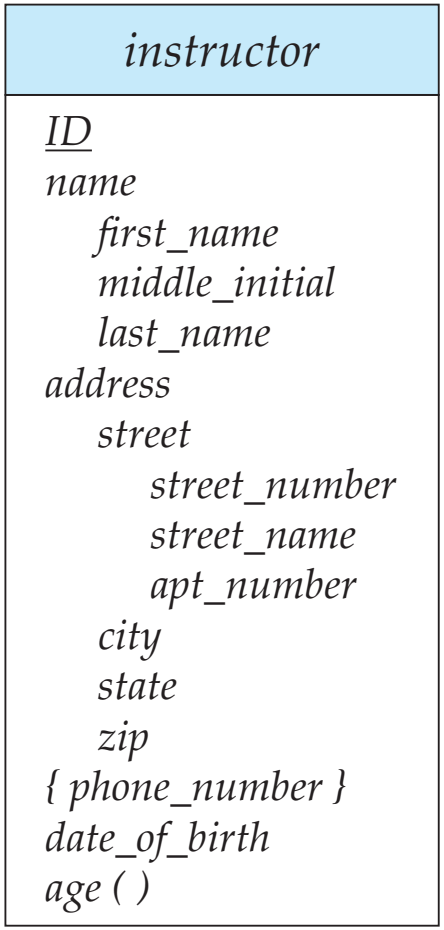
Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as relation schemas
  - For each entity set and relationship set, there is a unique schema that is assigned the name of the corresponding entity set or relationship set
  - Each schema has a number of columns (generally corresponding to attributes), which have unique names

Representing Entity Sets

- A strong entity set reduces to a schema with the same attributes
  - *E.g., student(ID, name, tot_cred)*
- A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set
  - *E.g., section (course_id, sec_id, sem, year)*

Representation of Entity Sets with Composite Attributes



- Composite attributes are flattened out by creating a separate attribute for each component attribute
  - *E.g.*, *first_name* → *name_first_name*
    last_name → *name_last_name*
  - Prefixes can be omitted if there is no ambiguity
  - *E.g.*, Ignoring multivalued attributes (*phone_number*), a corresponding instructor schema is:
    - *instructor(ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)*

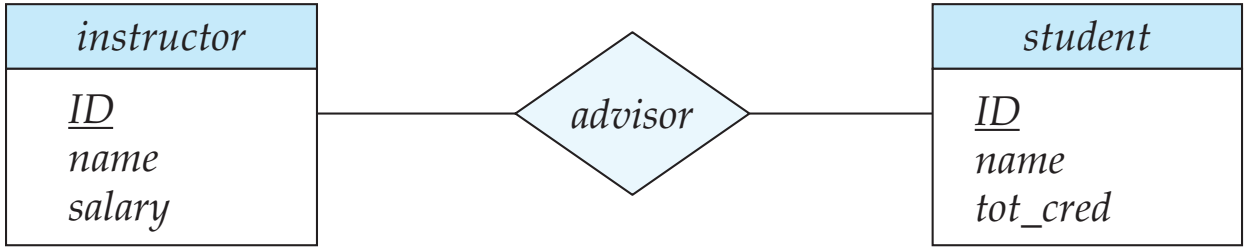
Representation of Entity Sets with Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*
  - *E.g.*, Multivalued attribute *phone_number* of *instructor*:
    - *inst_phone*(*ID*, *phone_number*)

- Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
  - *E.g.*, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567
    → maps to two tuples

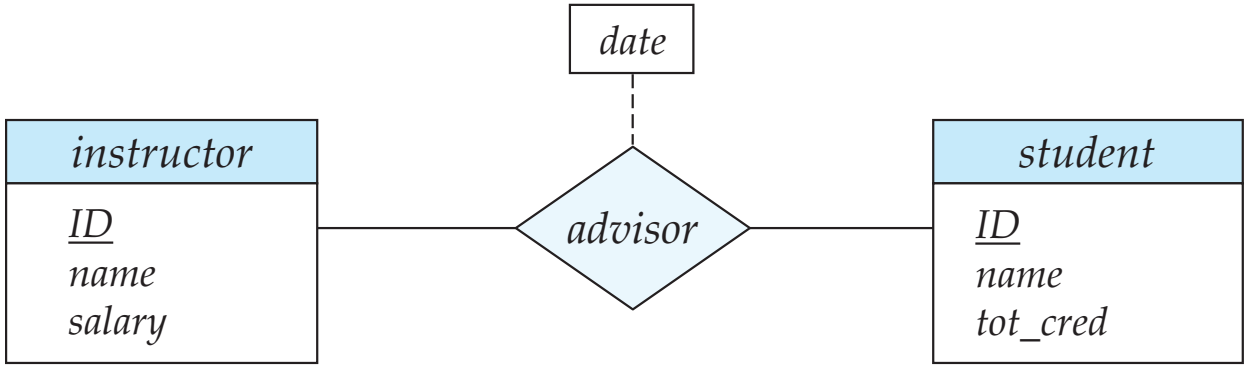
Representing Relationship Sets

- Any relationship set of strong entity sets can be represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set
  - *E.g.*, schema for relationship set *advisor*
    - *advisor* = (*s_id*, *i_id*)



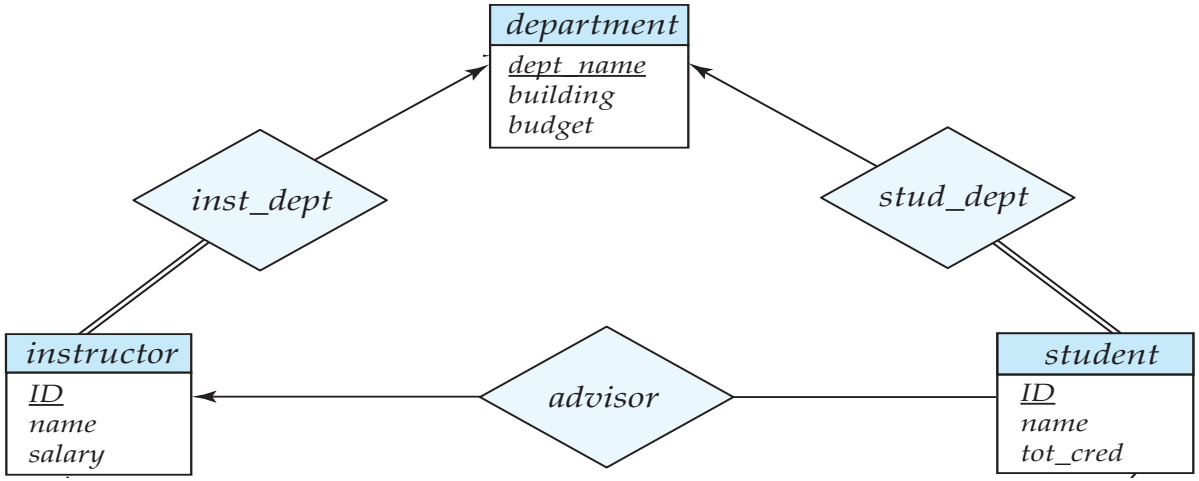
Representing Relationship Sets

- Any relationship set of strong entity sets can be represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set
  - *E.g.*, schema for relationship set *advisor*
    - *advisor* = (*s_id*, *i_id*)



Redundancy of Schemas

- Such “mapping tables” may be redundant
  - Many-to-one and one-to-many relationship sets that are total on the many-side
  - Can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side
    - *E.g.*, Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*



Redundancy of Schemas

- Such “mapping tables” may be redundant
  - Many-to-one and one-to-many relationship sets that are total on the many-side
  - Can be represented by adding an extra attribute to the “many” side, containing the primary key of the “one” side
    - *E.g.*, Instead of creating a schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema arising from entity set *instructor*
  - When participation is partial on the “many” side, replacing a schema by an extra attribute in the schema corresponding to the “many” side could result in null values

Redundancy of Schemas

- Such “mapping tables” may be redundant
  - For one-to-one relationship sets, either side can be chosen to act as the “many” side
    - An extra attribute can be added to either of the tables corresponding to the two entity sets

Example: a Record Shop

- Entity sets
  - *customer*
  - *product* (CD, vinyl)
  - *purchase*
- Relationship sets
  - *contains* (between *product* and *purchase*)
  - *buyer* (between *customer* and *purchase*)

- An entity is a specific object; *E.g.*, a newest CD of BTS
- A relationship is specific pair of related objects

Example: a Record Shop

- Attributes
  - *customer*
    - *name, address, phone number, email, etc.*
  - *product*
    - *artist, title, price, description*
  - purchase
    - *date, payment_method*

  - Artificial primary keys for all entity sets

Example: a Record Shop

- Types of Relationships
  - 1-to-1
  - 1-to-many
    - *buyer* relation (between *customer* and *purchase*)
  - Many-to-many
    - *contains* relation (between *purchase* and *product*)

E-R Diagram

Example: Flight Database

- Entity sets
  - *airport*
    - *code, name, city*
  - *flight*
    - *flight_num, STD, STA, date_offset*
  - *departure*
    - *dept_date, capacity*
  - *customer*
    - *id, name, address, milege_num*
- Relationship sets
  - *to, from* (*flight, airport*)
  - *flight_dept (flight, departure)*
  - *reserved_on* (*customer, departure*)

E-R Diagram

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)

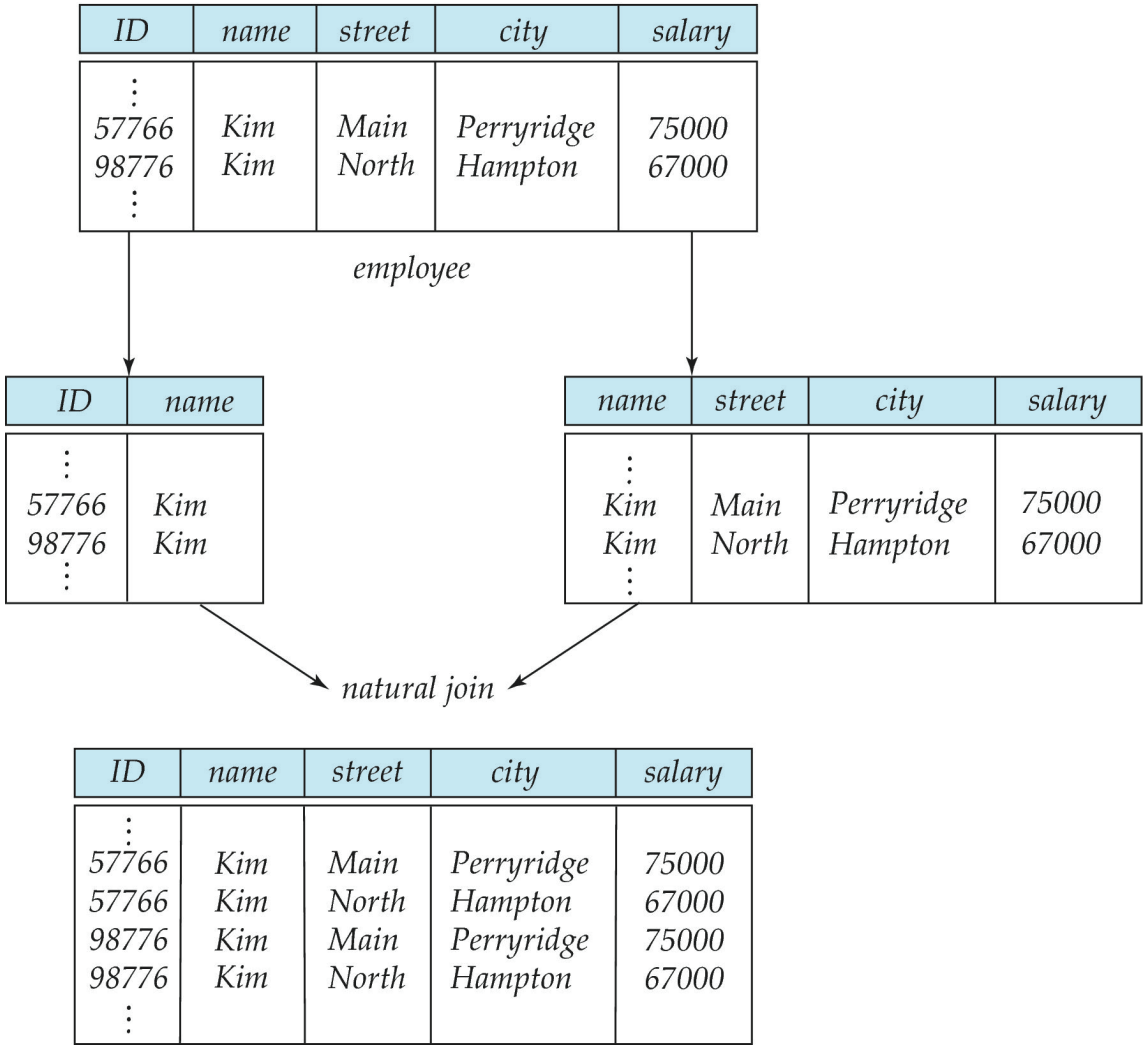
    into

    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



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 U 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 U R_2
  - Formally, $\Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) = r$
    - *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

$$r \subset \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Example: Lossless Decomposition

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

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

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

A	B	C
α	1	A
β	2	B

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

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

Examples

- (1) Which normal form does the below table violate? Justify your answer.

GROUP_ID	COURSE_NAME	TA_NAME	TA_EMAIL
1	Database System	Ms. Hwang	hwang@institue.edu
2	Algorithm	Mr. Kim	kim@institute.edu
3	Operating System	Ms. Park	park@institute.edu
4	Data Structures	Ms. Park	park@institute.edu

Examples

- (2) Which normal form does the below table violate? Justify your answer.

TA_ID	TA_NAME	TA_EMAIL	HOURLY_RATE
ta1	Ms. Hwang	hwang@institue.edu, champ122@zmail.com	15.5
ta2	Mr. Kim	kim@institute.edu, qoop@yaho.com	18.0
ta3	Ms. Park	park@institute.edu	Not specified

Examples

- (3) Which normal form is achieved by committing the following decomposition? Explain your answer.

(Initial table)

STU_ID	STU_NAME	FAC_ID	FAC_NAME
1	Olivia	4	Tim
2	Liam	3	Robert
3	Emma	2	Patricia
4	James	1	Aaliyah

➔ (After decomposition)

STU_ID	STU_NAME
1	Olivia
2	Liam
3	Emma
4	James

FAC_ID	FAC_NAME
1	Aaliyah
2	Patricia
3	Robert
4	Tim

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]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

EOF

- Coming next:
  - Advanced SQL