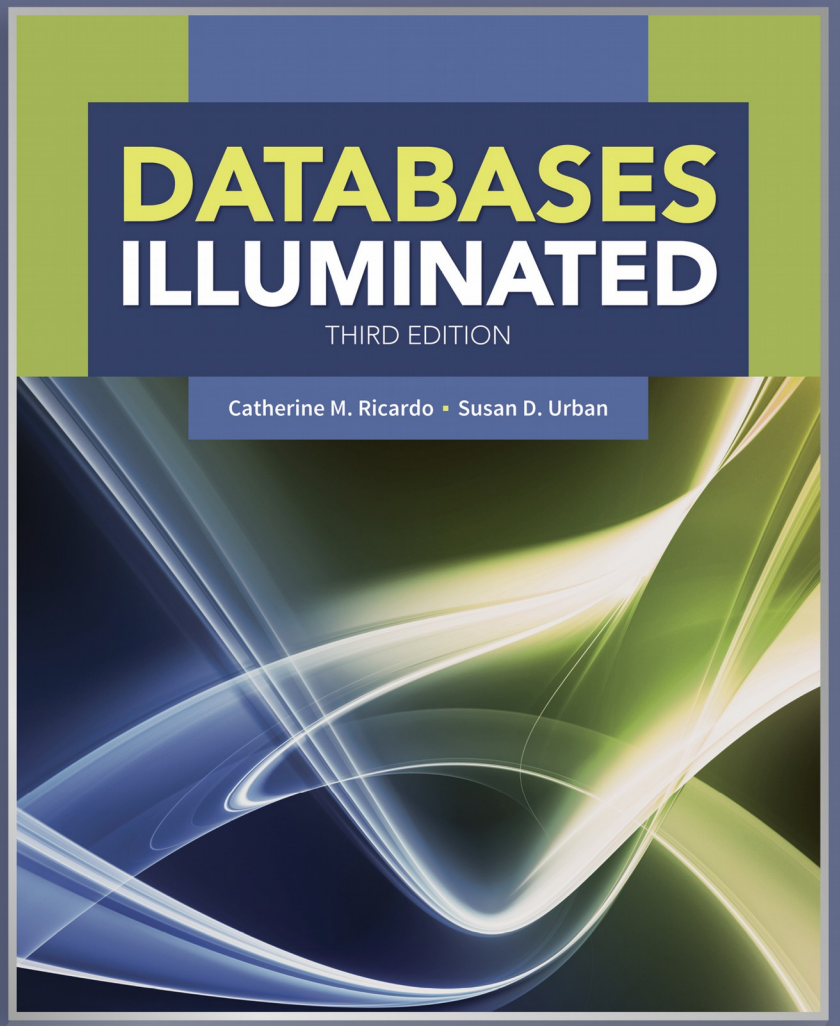
Databases Illuminated

Chapter 6

Normalization and Denormalization



Objectives of Normalization

- Develop a good description of the data, its relationships and constraints
- Produce a stable set of relations that
  - Is a faithful model of the enterprise
  - Is highly flexible
  - Reduces redundancy-saves storage space and reduces inconsistency in data
  - Is free of **update**, **insertion** and **deletion anomalies**

Characteristics of Normalized Schemas

- Each relation has a “theme”, relaying facts about a single subject
- The subject can be either an entity or a relationship
- Each cell of the table contains a single fact about that subject

Anomalies

- An anomaly is an inconsistent, incomplete, or contradictory state of the database
  - **Insertion** anomaly – user is unable to insert a new record when it should be possible to do so
  - **Deletion** anomaly – when a record is deleted, other information that is tied to it is also deleted
  - **Update** anomaly –a record is updated, but other appearances of the same items are not updated

Anomaly Examples: NewClass Table

NewClass(classNo, stuId, stuLastName, facId, schedule, room, grade)

classNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TuThF10	M110	A
CSC201A	S1006	Lee	F105	TuThF10	M110	G
HST205A	S1001	Smith	F202	MWF11	H221	

Figure 6.1 The `NewClass` Table

Anomalies

- **Update anomaly**: If schedule of ART103A is updated in first record, and not in second and third – data is inconsistent
- **Deletion anomaly**: If record of student S1001 is deleted, information about HST205A class is lost also
- **Insertion anomaly**: It is not possible to add a new class, for MTH101A , even if its teacher, schedule, and room are known, unless there is a student registered for it, because the key contains `stuId`

classNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TuThF10	M110	A
CSC201A	S1006	Lee	F105	TuThF10	M110	G
HST205A	S1001	Smith	F202	MWF11	H221	

Normal Forms

- First normal form -1NF
- Second normal form-2NF
- Third normal form-3NF
- Boyce-Codd normal form-BCNF
- Higher normal forms

- Each is contained within the previous form – each has stricter rules than the previous form
- **Normalization** means putting a relation into a higher normal form

Types of Dependencies

- Functional dependencies
- Multi-valued dependencies
- Join dependencies
- Others

All can cause problems in relational design

Functional Dependency-FD

- A **functional dependency** (FD) is a type of relationship between attributes
- If A and B are attributes or sets of attributes of relation R, B is functionally dependent on A if each A value in R has associated with it exactly one value of B in R
- Alternatively, if two tuples have the same A values, they must also have the same B values
- Write **A→B**, read **A functionally determines B**, or B functionally dependent on A
- FD is actually a many-to-one relationship between A and B

Example of FDs

FIGURE 6.2
Instance of `NewStudent` Table (Assume each student has only one major)

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

- Let R be

NewStudent(stuId, lastName, major, credits, status, socSecNo)

- FDs in R include

{stuId}→{lastName}, but not the reverse

{stuId} →{lastName, major, credits, status, socSecNo, stuId}

{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}

{credits}→{status}, but not {status}→{credits}

What FDs may exist?

- A relation *R*(A, B, C, D) with its extension.
- Which FDs *may exist* in this relation?

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Trivial Functional Dependency

- The FD X→Y is **trivial** if set {Y} is a subset of set {X}

Examples: If A and B are attributes of R,

{A}→{A}
{A,B} →{A}
{A,B} →{B}
{A,B} →{A,B}

are all trivial FDs

Keys

- **Superkey** – functionally determines all attributes in a relation
- **Candidate key** - superkey that is a minimal identifier (no extraneous attributes). An attribute that is part of any candidate key is called a **prime attribute**
- **Primary key** - candidate key actually used
- Primary key has **no-null** constraint and **uniqueness** constraint
- Should also enforce uniqueness and no-null rule for candidate keys
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key

Normalization of Relations (1)

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations

- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

Normalization of Relations (2)

- **2NF, 3NF, BCNF**
  - based on keys and FDs of a relation schema
- **4NF**
  - based on keys, multi-valued dependencies : MVDs;
- **5NF**
  - based on keys, join dependencies : JDs
- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation)

Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties

- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF and BCNF. 4NF rarely used in practice.)

- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

First Normal Form-1NF

(1NF-2NF:YouTube)

- A relation is in **1NF** if and only if every attribute is single-valued for each tuple
- Each cell of the table has only one value in it
- Domains of attributes are **atomic**: no sets, lists, repeating fields or groups allowed in domains

Counter-Example for 1NF

```
NewStu(StuId, lastName, major,
       credits, status, socSecNo)
```

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC Math	15	Freshman	088520876
S1010	Burns	Art English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

Figure 6.3A `NewStu` Table (Assume students can have double majors)

Figure 6.3A `NewStu` Table (Assume students can have double majors)

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC Math	15	Freshman	088520876
S1010	Burns	Art English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

- Assume students can have more than one major
- Assume `stuId` is the primary key
- The `major` attribute is not single-valued for each tuple. For a given `stuId`, there may be more than one value for `major`

Ensuring 1NF

- **Best solution:** For each multi-valued attribute, create a new table, in which you place the key of the original table and the multi-valued attribute. Keep the original table, with its key

Ex. `NewStu2(stuId, lastName, credits,status, socSecNo)`

`Majors(stuId, major)`

NewStu2(stuId, lastName, credits,status, socSecNo)
Majors(stuId, major)

NewStu2

stuId	lastName	credits	status	socSecNo
S1001	Smith	90	Senior	100429500
S1003	Jones	95	Senior	010124567
S1006	Lee	15	Freshman	088520876
S1010	Burns	63	Junior	099320985
S1060	Jones	25	Freshman	064624738

Majors

stuId	major
S1001	History
S1003	Math
S1006	CSC
S1006	Math
S1010	Art
S1010	English
S1060	CSC

Figure 6.3B `NewStu2` **Table and** `Majors` **Table**

Another Method for 1NF

- If the number of repeats is limited, make additional columns for multiple values

Student(stuId, lastName, major1, major2, credits, status, socSecNo)

NewStu3

stuId	lastName	major1	major2	credits	status	socSecNo
S1001	Smith	History		90	Senior	100429500
S1003	Jones	Math		95	Senior	010124567
S1006	Lee	CSC	Math	15	Freshman	088520876
S1010	Burns	Art	English	63	Junior	099320985
S1060	Jones	CSC		25	Freshman	064624738

Figure 6.3C `NewStu3` **Table with Two Attributes for** `Major`

- Complicates querying

Yet Another Method for 1NF

- "Flatten" the original table by making the multi-valued attribute part of the key

Student(<u>stuId</u>, lastName, <u>major</u>, credits, status, socSecNo

NewStu

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1006	Lee	Math	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1010	Burns	English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

Figure 6.3D `NewStu` Table Rewritten in 1NF, with {`stuId, major`} as Primary Key

- Can cause difficulties in higher normalization

Full Functional Dependency

- In relation R, set of attributes B is **fully functionally dependent** on set of attributes A of R if B is functionally dependent on A but not functionally dependent on any proper subset of A
- This means every attribute in A is needed to functionally determine B

Partial Functional Dependency Example

NewClass(courseNo, stuId, stuLastName, facId, schedule, room, grade)

classNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TuThF10	M110	A
CSC201A	S1006	Lee	F105	TuThF10	M110	G
HST205A	S1001	Smith	F202	MWF11	H221	

FDs:

{classNo,stuId} → {lastName}
{classNo,stuId} → {facId}
{classNo,stuId} → {schedule}
{classNo,stuId} → {room}
{classNo,stuId} → {grade}
classNo → facId **partial FD
classNo → schedule **partial FD
classNo → room ** partial FD
stuId → lastName ** partial FD

…plus trivial FDs that are partial…

Second Normal Form-2NF (1NF-2NF:YouTube)

- A relation is in **second normal form** (2NF) if it is in **first normal form** and all the **non-key attributes** are **fully functionally dependent** on the **key**.
- No non-key attribute is FD on just part of the key
- If key has only one attribute, and R is 1NF, R is automatically 2NF

Converting to 2NF

- Identify each partial FD
- Remove the attributes that depend on each of the determinants so identified
- Place these determinants in separate relations along with their dependent attributes
- In original relation keep the composite key and any attributes that are fully functionally dependent on all of it
- Even if the composite key has no dependent attributes, keep that relation to connect logically the others

2NF Example

NewClass(classNo, stuId, lastName, facId, schedule, room, grade)

FIGURE 6.4(A)
The NewClass Table, Not in 2NF

NewClass

classNo	stuId	lastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TuThF10	M110	A
CSC201A	S1006	Lee	F105	TuThF10	M110	C
HST205A	S1001	Smith	F202	MWF11	H221	

FDs grouped by determinant:

{classNo} → {classNo,facId, schedule, room}

{stuId} → {stuId, lastName}

{classNo,stuId} → {classNo, stuId, facId, schedule, room, lastName, grade}

Create tables grouped by determinants:

Course(classNo,facId, schedule, room)

Stu(stuId, lastName)

Keep relation with original composite key, with attributes FD on it, if any

NewStu2(classNo, stuId, grade)

FIGURE 6.4(B)
The Register, Stu, and Class2 Tables in 2NF

Register

classNo	stuId	grade
ART103A	S1001	A
ART103A	S1010	
ART103A	S1006	B
CSC201A	S1003	A
CSC201A	S1006	C

Stu

stuId	lastName
S1001	Smith
S1010	Burns
S1006	Lee
S1003	Jones

Class2

classNo	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
HST205A	F202	MWF11	H221

Transitive Dependency

- If A, B, and C are attributes of relation R, such that A → B, and B → C, then C is **transitively dependent** on A

FIGURE 6.5(A)
NewStudent Table Not in 3NF

NewStudent

stuId	lastName	major	credits	status
S1001	Smith	History	90	Senior
S1003	Jones	Math	95	Senior
S1006	Lee	CSC	15	Freshman
S1010	Burns	Art	63	Junior
S1060	Jones	CSC	25	Freshman

Example:

NewStudent(<u>stuId</u>, lastName, major, credits, status)

FD:

Credits → status (and several others)

By transitivity:

stuId → credits ∧ credits → status implies stuId → status

- ***Transitive dependencies cause update, insertion, deletion anomalies.***

Third Normal Form-3NF

- A relation is in **third normal form** (3NF) if whenever a **non-trivial functional dependency** X→A exists, then either X is a **superkey** or A is a **member** of some **candidate key**
- **To be 3NF, relation must be 2NF and have no transitive dependencies**
- No non-key attribute determines another non-key attribute. Here key includes "candidate key"
- *The essence of third normal form is that each **non-key attribute** is **functionally dependent** on the **entire key**, and on no other attribute.*

Making a relation 3NF

- For example,

NewStudent(<u>stuId</u>, lastName, major, credits, status)

with FD credits→status

- Remove the dependent attribute, status, from the relation
- Create a new table with the dependent attribute and its determinant, credits
- Keep the determinant in the original table

NewStu2(<u>stuId</u>, lastName, major, credits)

Stats(<u>credits</u>, status)

FIGURE 6.5(A)
NewStudent Table Not in 3NF

NewStudent

stuId	lastName	major	credits	status
S1001	Smith	History	90	Senior
S1003	Jones	Math	95	Senior
S1006	Lee	CSC	15	Freshman
S1010	Burns	Art	63	Junior
S1060	Jones	CSC	25	Freshman

FIGURE 6.5(B)
NewStu2 and Stats Tables

NewStu2

stuId	lastName	major	credits
S1001	Smith	History	90
S1003	Jones	Math	95
S1006	Lee	CSC	15
S1010	Burns	Art	63
S1060	Jones	CSC	25

Stats

credits	status
15	Freshman
25	Freshman
63	Junior
90	Senior
95	Senior

Boyce-Codd Normal Form-BCNF

- A relation is in Boyce/Codd Normal Form (BCNF) if whenever a **non-trivial functional dependency** X→A exists, then X is a **superkey**
- Stricter than 3NF, which allows A to be part of a candidate key
- If there is just one single candidate key, the forms are equivalent

BCNF Example

NewFac(facName, dept, office, rank, dateHired)

FDs:

office → dept

facName,dept → office, rank, dateHired

facName,office → dept, rank, dateHired

- Choose {facName,dept} as primary key
- Then NewFac is 3NF but not BCNF because office is not a superkey
- To make it BCNF, remove the dependen attributes to a new relation, with the determinant as the key
- Use projection to form two relations

  Fac1 (office, dept)

  Fac2 (facName, office, rank, dateHired)

- Note we have lost a functional dependency in Fac2 – no longer able to see that {facName, dept} is a determinant, since they are in different relations

FIGURE 6.6(A)
NewFac Table in 3NF, but not in BCNF

NewFac

facName	dept	office	rank	dateHired
Adams	Art	A101	Professor	1975
Byrne	Math	M201	Assistant	2000
Davis	Art	A101	Associate	1992
Gordon	Math	M201	Professor	1982
Hughes	Math	M203	Associate	1990
Smith	CSC	C101	Professor	1980
Smith	History	H102	Associate	1990
Tanaka	CSC	C101	Instructor	2001
Vaughn	CSC	C105	Associate	1995

FIGURE 6.6(B)
Fac1 and Fac2 in BCNF

Fac1

office	dept
A101	Art
C101	CSC
C105	CSC
H102	History
M201	Math
M203	Math

Fac2

facName	office	rank	dateHired
Adams	A101	Professor	1975
Byrne	M201	Assistant	2000
Davis	A101	Associate	1992
Gordon	M201	Professor	1982
Hughes	M203	Associate	1990
Smith	C101	Professor	1980
Smith	H102	Associate	1990
Tanaka	C101	Instructor	2001
Vaughn	C105	Associate	1995

Converting to BCNF

- Identify all determinants and verify that they are superkeys in the relation
- If not, break up the relation by projection
  - For each non-superkey determinant, create a separate relation with all the attributes it determines, also keeping it in original relation
  - Preserve the ability to recreate the original relation by joins.
- Repeat on each relation until you have a set of relations all in BCNF

Comprehensive Example of Functional Dependencies

- To summarize the various normal forms defined by functional dependencies, consider the following relation that stores information about projects in a large business:
- Work (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

FIGURE 6.7(A)
The Work Table

Work

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/15	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/15	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/15	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/14	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/14	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/14	45000	Jones	15	

Comprehensive Example of Functional Dependencies

Work (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

We make the following assumptions:

1. Each project has a **unique** name.
2. Although project names are unique, names of employees and managers are not.
3. Each project has one manager, whose name is stored in **projMgr**.
4. Many employees can be assigned to work on each project, and an employee can be assigned to more than one project. The attribute **hours** tells the number of hours per week a particular employee is assigned to work on a particular project.
5. **budget** stores the amount budgeted for a project, and **startDate** gives the starting date for a project.
6. **salary** gives the annual salary of an employee.
7. **empMgr** gives the name of the employee's manager, who might not be the same as the project manager.
8. **empDept** gives the employee's department. Department names are unique. The employee's manager is the manager of the employee's department.
9. **rating** gives the employee's rating for a particular project. The project manager assigns the rating at the end of the employee's work on that project.

- Using these assumptions, we start with the following functional dependencies:

  projName → projMgr, budget, startDate

  empId → empName, salary, empMgr, empDept

  projName, empId → hours, rating

- Since we assumed people's names were not unique, empMgr does not functionally determine empDept. (Two different managers may have the same name and manage different departments, or possibly a manager may manage several departments—see empMgr Jones in Figure 6.7(A).)
- Similarly, projMgr does not determine projName. However, since department names are unique and each department has only one manager, we need to add **empDept → empMgr**

FIGURE 6.7(A)
The Work Table

Work

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/15	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/15	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/15	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/14	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/14	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/14	45000	Jones	15	

- You may ask whether projMgr → budget.
  - Although it may be the case that the manager determines the budget, meaning the manager comes up with the figures for the budget, you should recall that functional dependency does not mean to cause or to figure out.
  - Similarly, although the project manager assigns a rating to the employee, there is no functional dependency between projMgr and rating.
- Since we see that every attribute is functionally dependent on the combination {projName, empId}, we will choose that combination as our **primary key**, and see what normal form we have. We begin by checking to see whether it is already in BCNF.
- ***BCNF:*** We look to see if there is a determinant that is not a superkey. Any one of empId, empDept, or projName is sufficient to show that the Work relation is not BCNF.
- Since we know it is not BCNF, we can begin the normalization process by checking the lower normal forms, normalizing the relation(s) as we go along. At the end, we can see if we have reached BCNF.
- ***First Normal Form:*** With our composite key, {projName, empID}, each cell would be single valued, so ***Work is in 1NF***.

Work (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

- ***Second Normal Form:*** We found partial (non-full) dependencies.

  projName → projMgr, budget, startDate

  empId → empName, salary, empMgr, empDept

- We can take care of these, transforming the relation into an equivalent set of 2NF relations by projection, resulting in

  Proj (<u>projName</u>, projMgr, budget, startDate)

  Emp (<u>empId</u>, empName, salary, empMgr, empDept)

  Work1 (<u>*projName, empId*</u>, hours, rating)

Work (<u>projName</u>, projMgr, <u>empId</u>, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

FIGURE 6.7(A)
The Work Table

Work

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/15	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/15	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/15	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/14	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/14	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/14	45000	Jones	15	

Proj(projName, projMgr, budget, startDate)
Emp(empId, empName, salary, empMgr, empDept)
Work1(*projName, empId*, hours, rating)

projName → projMgr, budget, startDate
empId → empName, salary, empMgr, empDept
projName, empId → hours, rating
empDept → empMgr

- ***Third Normal Form:*** Using the set of projections {Proj, Emp, Work1}, we test each relation to see if we have 3NF.
- Examining **Proj**, we see that no non-key attribute or combination of attributes functionally determines another non-key attribute, so Proj is 3NF.
- In **Emp**, we have a **transitive dependency**, since **empDept → empMgr**, as previously explained.
- Since empDept is not a superkey, nor is empMgr part of a candidate key, this violates 3NF. Therefore we need to rewrite Emp as

  Emp1 (empId, empName, salary, *empDept*)

  Dept (empDept, empMgr)
- **Work1** has no transitive dependency involving hours or rating, so that relation is already 3NF. Our new set of 3NF relations is therefore

  Proj (projName, projMgr, budget, startDate)

  Emp1 (empId, empName, salary, *empDept*)

  Dept (empDept, empMgr)

  Work1 (*projName, empId*, hours, rating)

- ***Boyce-Codd Normal Form revisited:*** Our new 3NF set of relations is also BCNF, since, in each relation, the only determinant is the primary key.

FIGURE 6.7(A)
The Work Table

Work

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/15	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/15	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/15	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/14	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/14	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/14	45000	Jones	15	

FIGURE 6.7(B)
The Normalized Tables Replacing Work

Proj

projName	projMgr	budget	startDate
Jupiter	Smith	100000	01/15/15
Maxima	Lee	200000	03/01/14

Dept

empDept	empMgr
10	Levine
12	Jones
15	Jones

Emp1

EmpId	empName	salary	empDept
E101	Jones	60000	10
E105	Adams	55000	12
E110	Rivera	43000	10
E120	Tanaka	45000	15

Work1

projName	empId	hours	rating
Jupiter	Jones	25	9
Jupiter	Adams	40	
Jupiter	Rivera	10	8
Maxima	Jones	15	
Maxima	Rivera	30	
Maxima	Tanaka	15	

Proj(projName, projMgr, budget, startDate)
Emp1(empId, empName, salary, *empDept*)
Dept(empDept, empMgr)
Work1(*projName, empId*, hours, rating)

Decomposition

- **Definition**: A **decomposition** of a relation R is a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that each R_i is a subset of R and the union of all of the R_i is R (i.e. each original attribute of *R* appears in at least one R_i)

- Starting with a universal relation that contains all the attributes of a schema, we can decompose into relations by projection

Desirable Properties of Decompositions

- **Attribute preservation** - every attribute is in some relation
- **Dependency preservation** – all FDs are preserved
- **Lossless decomposition** – can get back the original relation by joins

Dependency Preservation

- If R is decomposed into $\{R_1, R_2, \ldots, R_n,\}$ so that for each functional dependency $X \rightarrow Y$ all the attributes in $X \cup Y$ appear in the same relation, R_i, then all FDs are preserved

- Allows DBMS to check each FD constraint by checking just one table for each

Lossless Decomposition

- A decomposition of R into $\{R_1, R_2, ..., R_n\}$ is **lossless** if the natural join of $R_1, R_2, ..., R_n$ produces exactly the relation R

- No **spurious tuples** are created when the projections are joined.

- Always possible to find a BCNF decomposition that is lossless

Example of Lossy Decomposition

- Original **`EmpRoleProj`** table: tells what role(s) each employee plays in which project(s)

FIGURE 6.8(A)
Original Table `EmpRoleProj`

`EmpRoleProj`

empName	role	projName
Smith	designer	Nile
Smith	programmer	Amazon
Smith	designer	Amazon
Jones	designer	Amazon

FIGURE 6.8(B)
Projections of `EmpRoleProj`

Table 1

empName	role
Smith	designer
Smith	programmer
Jones	designer

Table 2

role	projName
designer	Nile
programmer	Amazon
designer	Amazon

FIGURE 6.8(C)
Join of `Table 1` and `Table 2`

empName	role	projName	
Smith	designer	Nile	
Smith	designer	Amazon	
Smith	programmer	Amazon	
Jones	designer	Nile	←spurious tuple
Jones	designer	Amazon	

Lossless Decomposition

- Lossless property guaranteed if for each pair of relations that will be joined, **the set of common attributes is a superkey of one of the relations**

- Binary decomposition of R into $\{R_1, R_2\}$ lossless if and only if one of these holds

$$\mathbf{R1 \cap R2 \rightarrow R1 - R2}$$

or

$$\mathbf{R1 \cap R2 \rightarrow R2 - R1}$$

- If projection is done by successive binary projections, can apply binary decomposition test repeatedly

Decomposition Algorithm for BCNF

- Can always find a lossless decomposition that is BCNF
- Find a FD that is a violation of BCNF and remove it by decomposition process
  - Repeat this process until all violations are removed

Given a universal relation R and a set of functional dependencies on the attributes of R:

1. D ← R
2. while there is some relation schema S in D that is not already BCNF
   {
   a. Find a functional dependency X → Y in S that violates BCNF
   b. Replace S by two relation schemas (S – Y) and (X, Y)
   }

- No need to go through 1NF, 2NF, 3NF process
- Not always possible to preserve all FDs

Normalization Methods

- Analysis
  - Decomposition method shown previously
- Synthesis
  - Begin with attributes, combine them into groups having the same determinant
  - Use functional dependencies to develop a set of normalized relations
- Mapping from ER diagram provides almost-normalized schema

De-normalization

- When to stop the normalization process
  - When applications require too many joins
  - When you cannot get a non-loss decomposition that preserves dependencies

- De-normalization means deliberately choosing a lower normal form

De-normalization

- It is always possible to find a dependency preserving lossless decomposition for 3NF.
- Even so, there may be valid reasons for choosing not to implement 3NF as well.
- For example, if we have attributes that are almost always used together in applications and they end up in different relations, then we will always have to do a join operation when we use them.
  - Assume we are storing an employee's name and address in the relation

  Emp (empId, lastName, firstName, street, city, state, zip)
  - We are assuming names are not unique. We have the functional dependency

    zip → city, state
  - which means that the relation is not 3NF. We could normalize it by decomposition;

    Emp1 (empId, name, street, zip)

    Codes (zip, city, state)
  - We have to do a join whenever we wanted a complete address for an employee.
  - We might settle for 2NF and implement the original Emp relation.
- This is purely a design decision and depends on the needs of the applications and the performance requirements for the database.

Problems with Normalized Databases

- Best used when the source data is relatively simple
- Stored data does not resemble the original documents from which it is taken, but instead is shredded into separate tables
- Usually store only the most current information, not historical data
- Useful for **OLTP**, online transaction processing
- Optimized for write operations; read operations may be slow, if joins of the tables are required

Non-normalized Databases

- **OLA**P systems
  - Used for planning and decision-making
  - Require historical data, not just current data
  - Updates are rare; optimized for reading
  - Data stored in de-normalized form
- Object-based systems
  - Needed for advanced applications
  - Objects are not normalized
- Big data systems-XML, Google's Big Table, HBase, Cassandra, and Hadoop
  - Capture data in the format of its source
  - Store data in a de-normalized, usually duplicated, form
  - Allow multiple versions of items to be stored
  - Provide efficient and scalable read access to data
  - Facilitate data transmission

Additional Material from Online Resource Follows

Multi-valued Dependency

- In R(A,B,C) if each A values has associated with it a set of B values and a set of C values such that the B and C values are independent of each other, then **A multi-determines B** and **A multi-determines C**
- Multi-valued dependencies occur in pairs
- Example:
  - `JointAppoint(facId, dept, committee)`
  - Assuming faculty member can belong to more than one department and to more than one committee
  - Table must list all combinations of values of department and committee for each `facId` to avoid implying relationships that do not exist

4NF

- A table is **4NF** if it is BCNF and has **no multi-valued dependencies**
  - Example: remove MVDs in `JointAppoint(facId, dept, committee)`

    `Appoint1(facId,dept)`

    `Appoint2(facId,committee)`

4NF

- Definition: A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency X→> Y in F^+ X is a superkey for R.
- An MVD X →> Y in R is called a trivial MVD
  - if (a) Y is a subset of X,
  - or (b) X ∪ Y = R
- We can state the following points:
  - An all-key relation is always in BCNF since it has no FDs.
  - An all-key relation such as the EMP relation which has no FDs but has the MVD Ename →>Pname | Dname is not in 4NF.
  - A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF.
  - The decomposition removes the redundancy caused by the MVD.

Fourth Normal Form

(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Fourth normal form. (a) The EMP relation with two MVDs: Ename –>> Pname and Ename –>> Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF.

1NF-2NF-3NF-4NF Example

YouTube

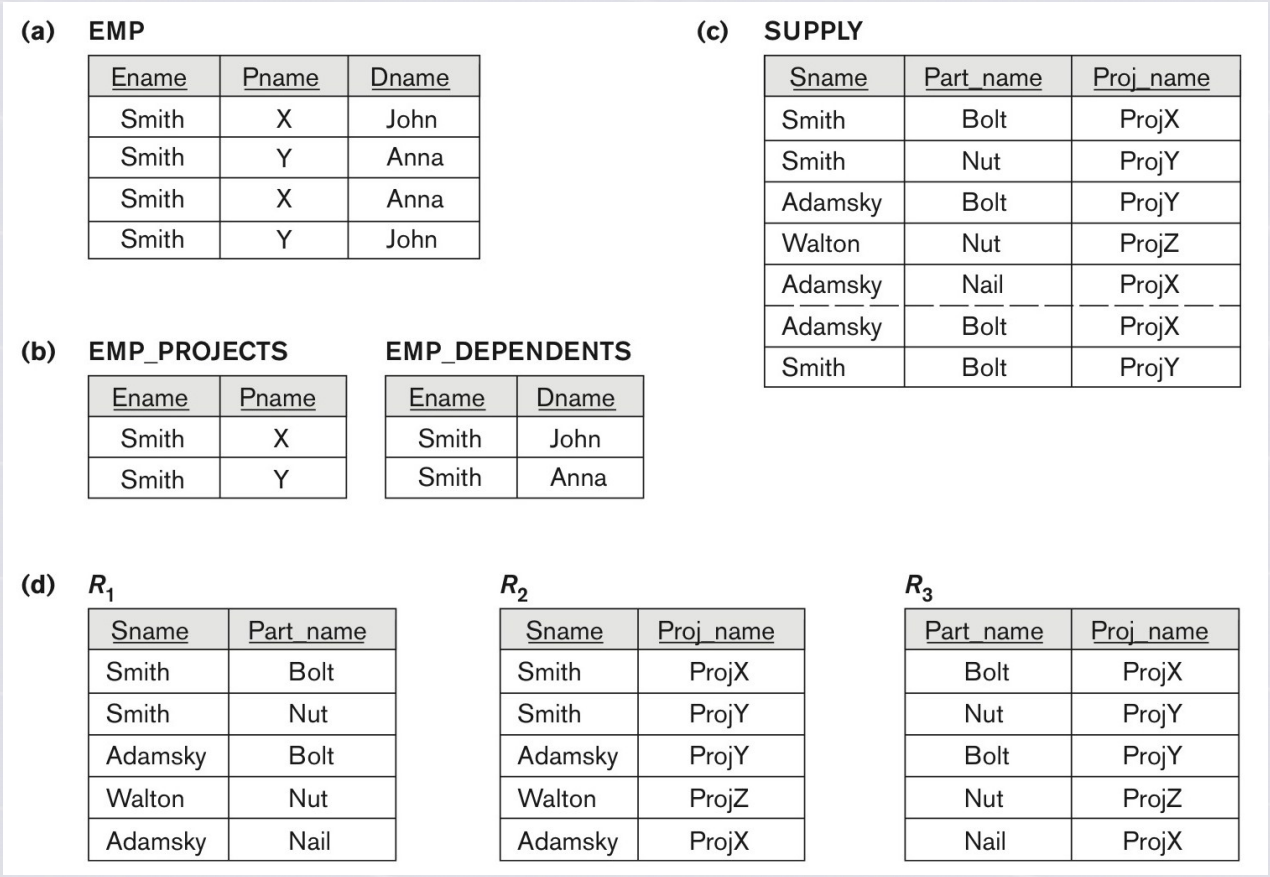
5NF (Project-Join Normal Form)

- A relation is **5NF** if there are no remaining non-trivial lossless projections

- In some cases there may be no nonadditive join decomposition of a relation R into two relation schemas, but there may be a nonadditive join decomposition into more than two relation schemas.

- It is very difficult to detect in practice; therefore, normalization into 5NF is very rarely done in practice.

Fifth Normal Form



(a) EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

(c) SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Fifth normal form. (a) The EMP relation with two MVDs: Ename –>> Pname and Ename –>> Dname. (b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS. (c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R1, R2, R3). (d) Decomposing the relation SUPPLY into the 5NF relations R1, R2, R3.

Inference Rules for FDs

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- Armstrong's Axioms (YouTube)
  - **Reflexivity** If B is a subset of A, then A → B.
  - **Augmentation** If A → B, then AC → BC.
  - **Transitivity** If A → B and B → C, then A → C
- Additional rules that follow:
  - **Additivity** (**Union**) If A → B and A → C, then A → BC
  - **Projectivity (Decomposition)**
    - If A → BC, then A → B and A → C
  - **Pseudotransitivity** If A → B and CB → D, then AC → D

Closure of Set of FDs

- If F is a set of functional dependencies for a relation R, then the set of all functional dependencies that can be derived from F, F^+, is called the **closure of F**

- Could compute closure by applying Armstrong's Axioms repeatedly

Closure of an Attribute (YouTube)

- If A is an attribute or set of attributes of relation R, all the attributes in R that are functionally dependent on A in R form the **closure of A,** A^+

- Closure Algorithm for A

result ← A;
while (result changes) do
for each functional dependency B → C in F
if B is contained in result
then result ← result ∪ C;
end;
end;
A^+ ***← result;***

Uses of Attribute Closure

- Can determine if A is a superkey-if every attribute in R functionally dependent on A (YouTube)

- Can determine whether a given FD $X \rightarrow Y$ is in the closure of the set of FDs. (Find X^+, see if it includes Y)