Chapter 5 Relational Database Design

- By
Jyoti Tryambake

Relational Database Design

In general, the goal of a relational-database design is to

- **Generate a set of relation schemas that allows us to store information without unnecessary redundancy**, yet also allows us to **retrieve information easily**.
- One approach is to design schemas that are in an appropriate **normal form**.

The Database Design Process

- At a high-level, the database design process is driven by the level of **normalization** of a relational scheme.
- If relational scheme *R* is not sufficiently normalized, **decompose** it into a set of relational schemes $\{R_1, R_2, \ldots, R_n\}$ such that:
  - Each relational scheme is sufficiently ***normalized***.
  - The decomposition has a ***lossless-join***.
  - All **functional dependencies are *preserved***.
- **So what are normalization, lossless-join, functional dependencies, and what does preserving them mean?**

Pitfalls in Relational-Database Design

Among the undesirable properties that a bad design may have are.

① **Repetition of information**

② **Inability to represent certain information**

Example:

The information concerning loans is now kept in one single relation, lending, which is defined over the relation schema.

Lending-schema=(branch-name, branch-city, assets, customer-name, loan-number, amount)

Pitfalls in Relational-Database Design

branch-name	branch-city	assets	customer-name	loan-number	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900

Lending Figure A

Pitfalls in Relational-Database Design

Problems:

① add a new loan

The loan is made by the Perryridge branch to Adams in the amount of $1500. Let the loan-number be L-31.

We add the tuple

(Perryridge, Horseneck, 1700000,Adams, L-31, 1500)

ⅰ Repeating information wastes space.

ⅱ Repeating information creates Update, deletion, and insertion anomalies.

Pitfalls in Relational-Database Design - anomalies

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- **Update Anomalies:**
  - Modify the asset value for the branch of loan L-17.
- **Insertion Anomalies:**
  - Cannot store information about a branch if no loans exist without using null values.
  - This is particularly bad since loan-number is part of the primary key.
- **Deletion Anomalies:**
  - Deleting L-17 and L-14 might result in all Downtown branch information being deleted.

Pitfalls in Relational-Database Design

② **we cannot represent directly the information concerning a branch.**(branch-name, branch-city,assets) unless there exists at least one loan at the branch. The problem is that tuples in the lending relation require values for loan-number, amount and customer-name.

Decomposition

- Decomposition is the process of breaking down in parts or elements.
- It replaces a relation with a collection of smaller relations.
- It breaks the table into multiple tables in a database.
- It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.
- Types:
  - **Lossless join decomposition**
  - **Lossy decomposition**

Decomposition

Example:

Consider an alternative design in which Lending-schema is decomposed into the following two schemas:

Branch-customer-schema=(branch-name, branch-city, assets,customer-name)

Customer-loan-schema=(customer-name,loan-number, amount)

Decomposition

branch-name	branch-city	assets	customer-name
Downtown	Brooklyn	9000000	Jones
Redwood	Palo Alto	2100000	Smith
Perryridge	Horseneck	1700000	Hayes
Downtown	Brooklyn	9000000	Jackson
Mianus	Horseneck	400000	Jones
Round Hill	Horseneck	8000000	Turner

Branch-customer B

customer-name	loan-number	amount
Jones	L-17	1000
Smith	L-23	2000
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-93	500
Turner	L-11	900

Customer-loan C

- Suppose that we wish to find all branches that have loans with amount less than $1000.
  - Original table - o/p Mianus, round hill
  - Decomposed table – o/p- Mianus, round hill, **downtown**

Decomposition

branch-name	branch-city	assets	customer-name	loan-number	amount
Downtown	Brooklyn	9000000	Jones	L-17	1000
Downtown	Brooklyn	9000000	Jones	L-93	500
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500
Mianus	Horseneck	400000	Jones	L-17	1000
Mianus	Horseneck	400000	Jones	L-93	500
Round Hill	Horseneck	8000000	Turner	L-11	900

Figure D

Additional tuples:

(Downtown, Brooklyn, 9000000,Jones, L-93, 500)

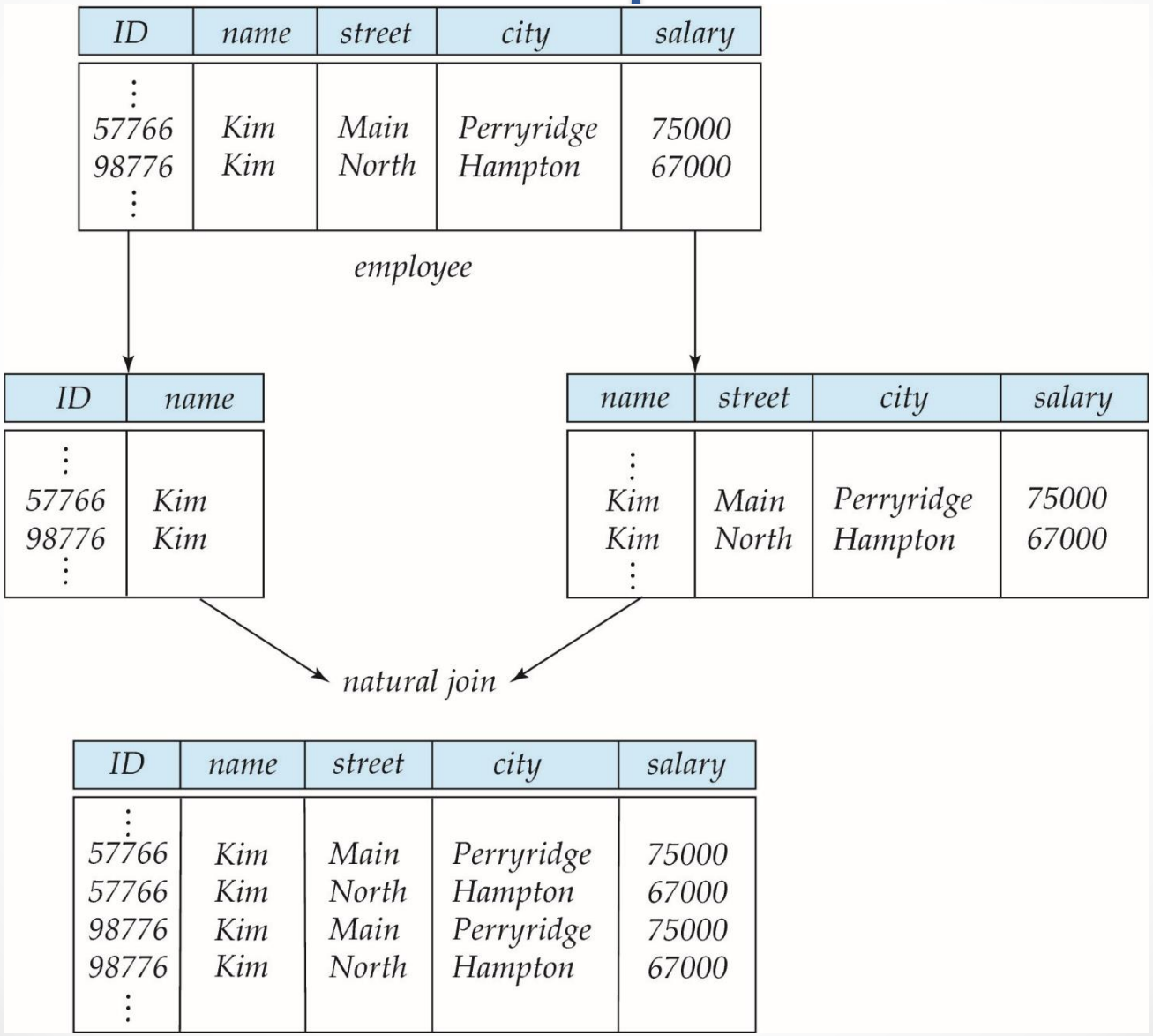
(Mianus, Horsereck, 400000,Jones, L-17, 1000)

Decomposition

Lossy decomposition:

- Example shows how we lose information
  - we **cannot reconstruct the original relation**
  - and so, this is a **lossy decomposition**.
- The decomposition of Lending-schema into Branch-customer-schema and customer-loan-schema a lossy decomposition, or a lossy-join decomposition.

A Lossy Decomposition-example



Example of Non Lossless-Join Decomposition

- **Decomposition of *R = (A, B)***
  R_2 = (A) R_2 = (B)

A	B
α	1
α	2
β	1

r

Π_A (r) ⋈ Π_B (r)

A
α
β

Π_A(r)

B
1
2

$\Pi_{B(r)}$

A	B
α	1
α	2
β	1
β	2

Decomposition

Lossless-join decomposition: All attributes of an original schema (R) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$

And For all possible relations r on schema R

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

Properties:

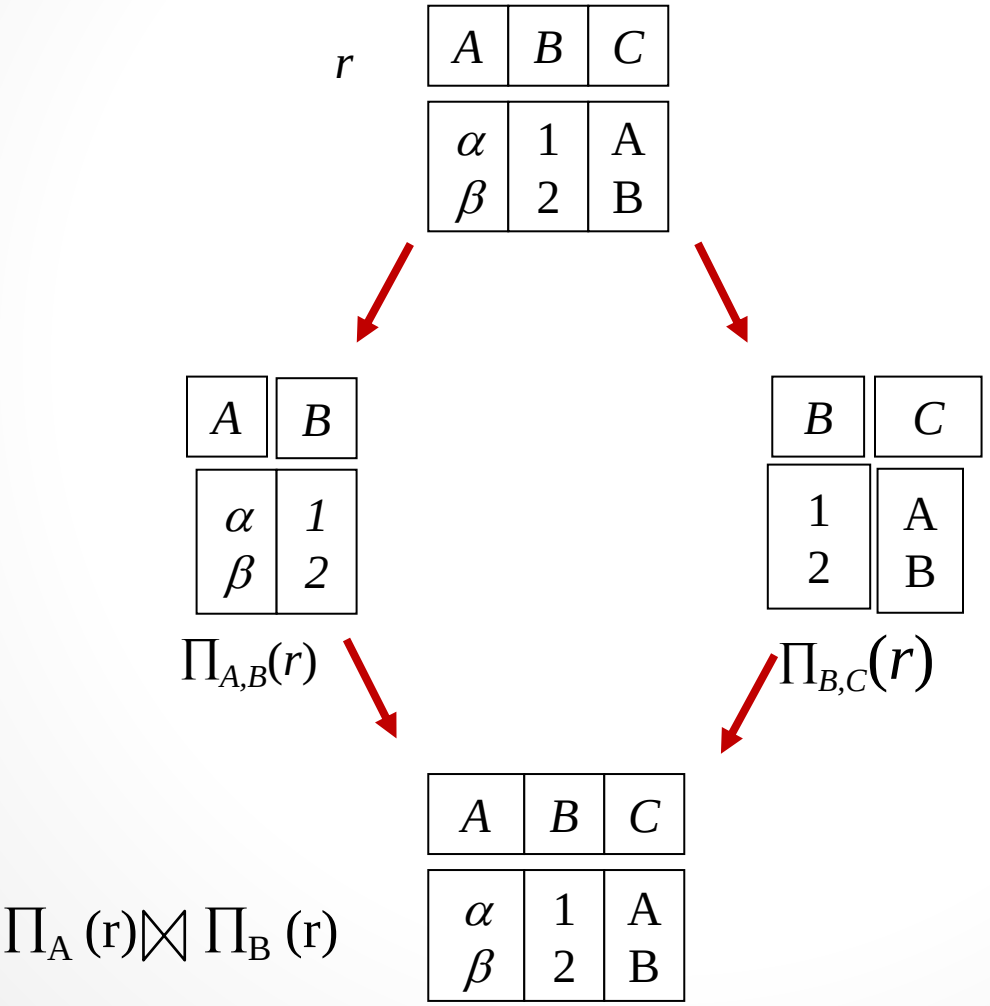
1. R1 ∪ R2 = R
2. R1 ∩ R2 should not be Φ
3. R1 ∩ R2 -> R1, that is: all attributes common to both R1 and R2 functionally determine ALL the attributes in R1 .

   R1 ∩ R2 -> R2 , that is: all attributes common to both R1 and R2 functionally determine ALL the attributes in R2

   (If R1 ∩ R2 forms a superkey of either R1 or R2 , the decomposition of R is a lossless decomposition)

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$ ➔ $R_1 = (A, B)$, $R_2 = (B, C)$



Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- In the case that 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-join** decomposition

Functional Dependencies

- Constraints on the set of legal 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***.

Functional Dependencies

- Let R be a relation schema : $\alpha \subseteq R$ *and* $\beta \subseteq R$

- The **functional dependency $\alpha \rightarrow \beta$ holds on** R if and only if for any legal 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,

  $$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider R(A,B) with the following instance r of R

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

Examples of FD constraints (1)

- Social security number determines employee name
  - SSN -> ENAME
- Project number determines project name and location
  - PNUMBER -> {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} -> HOURS

Examples of FD constraints (2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])

Inference rules = Armstrong Axioms

2.2 Inference Rules for FDs (1)

- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X -> Y
  - IR2. (**Augmentation**) If X -> Y, then XZ -> YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X -> Y and Y -> Z, then X -> Z

- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

Inference Rules for FDs (2)

- Some additional inference rules that are useful:
  - **Decomposition:** If X -> YZ, then X -> Y and X -> Z
  - **Union:** If X -> Y and X -> Z, then X -> YZ
  - **Psuedo transitivity:** If X -> Y and WY -> Z, then WX -> Z
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Inference Rules for FDs (3)

- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F
- **Closure** of a set of attributes X with respect to F is the set X^+ of all attributes that are functionally determined by X
- X^+ can be calculated by repeatedly applying IR1, IR2, IR3 using the FDs in F

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

- **Find F.D.**

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

6.3 Normalization Using Functional Dependence

Another Algorithm:

F={A→B,C→F,E→A,CE→D}

	A	B	C	D	E	F
CF	b_{11}	b_{12}	a_3	b_{14}	b_{15}	a_6
BE	b_{21}	a_2	b_{23}	b_{24}	a_5	b_{26}
ECD	~~b_{31}~~ b_{21}	~~b_{32}~~ a_2	a_3	a_4	a_5	~~b_{36}~~ a_6
AB	a_1	a_2	b_{43}	b_{44}	b_{45}	b_{46}

Lossy-join decomposition

6.3 Normalization Using Functional Dependence

F={A→B,C→F,E→A,CE→D}

	A	B	C	D	E	F
ABE	a_1	a_2	b_{13}	b_{14}	a_5	b_{16}
CDEF	~~b_{21}~~ a_1	~~b_{22}~~ a_2	a_3	a_4	a_5	a_6

Lossless-join decomposition

6.3 Normalization Using Functional Dependence

Dependency Preservation:

There is another goal in relational-database design: dependency preservation.

Let F be a set of functional dependencies on a schema R, and let $R_1, R_2, \ldots, R_n$ be a decomposition of R. The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include only attributes of R_i.

6.3 Normalization Using Functional Dependence

Let $F' = F_1 \cup F_2 \cup \ldots\ldots \cup F_n$. F' is a set of functional dependencies on schema R. if $F'^{+} = F^{+}$ is true, then every dependency in F is logically implied by F', and, if we verify that F' is satisfied, we have verified that F is satisfied. We say that a decomposition having the property $F'^{+} = F^{+}$ is a dependency-preserving decomposition.

Normalization

Normal Form:

A relation is said to be in a particular normal form if it satisfies a certain prescribed set of conditions.

Normalization

① **First normal form: A relation is in 1NF**

if and only if, in every legal value of that relation , every tuple contains exactly one value for each attribute (every attribute in that relation is singled valued attribute.)

Every cell must contain atomic value

Customer-name	City	
	Customer-city	Customer-street
Jones	Brooklyn	Ray
Hayes	Palo Alto	Heroes

Customer-name	Customer-city	Customer-street
Jones	Brooklyn	Ray
Hayes	Palo Alto	Heroes

Normalization

① First normal form: example 2

Students

FirstName	LastName	Knowledge
Thomas	Mueller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Mueller	C++, Java

Startsituation

Result after Normalisation

Students

FirstName	LastName	Knowledge
Thomas	Mueller	C++
Thomas	Mueller	PHP
Thomas	Mueller	Java
Ursula	Meier	Java
Ursula	Meier	PHP
Igor	Mueller	Java
Igor	Mueller	C++

Normalization

② **Second normal form**:

- i. A relation is in 2NF if and only if it is in 1NF
- ii. every non key attribute is fully functionally dependent on the primary key.

That means No partial dependency - i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Normalization

- **② Second normal form – example**
- **Partial dependency:** If proper subset of candidate key determines non-prime attribute, it is called partial dependency.

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

- **Candidate Key: {STUD_NO, COURSE_NO}**
- **FD set: {COURSE_NO->COURSE_NAME}**
- **Prime attributes = STUD_NO, COURSE_NO**
- **Non-Prime attributes = COURSE_NAME**

Normalization

- **② Second normal form – example**

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

- **COURSE_NO->COURSE_NAME,**
  - **COURSE_NO (proper subset of candidate key) is determining COURSE_NAME (non-prime attribute).**
  - **Hence, it is partial dependency and relation is not in second normal form.**

Normalization

- **② Second normal form – example**

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computers Network
1	C2	Computers Network

- **To convert it to second normal form, decompose the relation**

STUDENT_COURSE (STUD_NO, COURSE_NO, COURSE_NAME) as :

  - **STUDENT_COURSE (STUD_NO, COURSE_NO)**
  - **COURSE (COURSE_NO, COURSE_NAME)**

Note – This decomposition will be lossless join decomposition as well as dependency preserving.

Normalization

③ Third normal form(definition assuming only one candidate key, which we further assume is the primary key):

A relation is in 3NF if and only

- if it is in 2NF and
- every non key attribute is non-transitively dependent on the primary key.
- No transitive dependency

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

(No non-prime attribute should determine other non-prime attribute)

Normalization

③ Third normal form: Example:

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Candidate Key: {STUD_NO}

FD set:
STUD_NO -> STUD_NAME
STUD_NO -> STUD_STATE,
STUD_NO -> STUD_AGE,
STUD_STATE -> STUD_COUNTRY

Normalization

③ **Third normal form**: **Example:**

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

- STUD_NO -> STUD_STATE & STUD_STATE -> STUD_COUNTRY are true.
- So STUD_COUNTRY is transitively dependent on STUD_NO. It violates third normal form.
- To convert it in third normal form, decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
- **STUDENT (STUD_NO, STUD_NAME, STUD_STATE, STUD_AGE)**
  STATE_COUNTRY (STUD_STATE, STUD_COUNTRY)

Normalization

④ **Boyce/codd normal form(BCNF):**

A table complies with BCNF if

- it is in 3NF

- and for every functional dependency X->Y, X should be the super key of the table.

(X cannot be a **non-prime attribute**, if Y is a **prime attribute**.)

Normalization

④ Boyce/codd normal form(BCNF):

Example: Employees work in more than one department

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

- Functional dependencies in the table above:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}
- Candidate key: {emp_id, emp_dept}

Normalization

④ Boyce/codd normal form(BCNF):

Example: Employees work in more than one department

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

- Functional dependencies in the table above:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}

The table is not in BCNF as neither emp_id nor emp_dept

- alone are keys.

Normalization

④ Boyce/codd normal form(BCNF):

Example: To make the table comply with BCNF , break the table in three tables like this:

- **1. emp_nationality table:**

emp_id	emp_nationality
1001	Austrian
1002	American

- **2. emp_dept table:**

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

Normalization

④ Boyce/codd normal form(BCNF):

Example: To make the table comply with BCNF , break the table in three tables like this:

- **3. emp_dept_mapping table:**

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

- **Candidate keys**:
  For first table: emp_id
  For second table: emp_dept
  For third table: {emp_id, emp_dept}

Normalization

④ Boyce/codd normal form(BCNF):

Example 2:

Consider the following relationship : **R (A,B,C,D)**

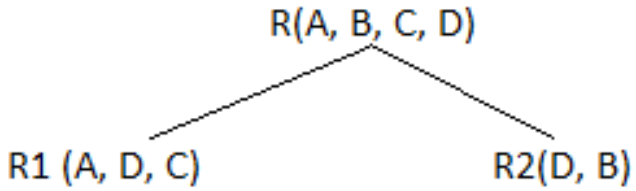
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC.**

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2.**



Breaking, table into two tables, one with A, D and C while the other with D and B.

Normalization

Fourth normal form(4NF):

- A relation will be in 4NF **if it is in Boyce Codd normal form and has no multi-valued dependency**.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example:

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

It is not in 4NF

Normalization

Fourth normal form(4NF):

Example:

- to make the above table into 4NF, we can decompose it into two tables:

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Normalization

Fifth normal form(5NF):

- A table is in the 5NF
  - if it's in 4NF and
  - if it can't have a lossy decomposition in to any number of smaller tables.
- It's also known as Project-join normal form(PJ/NF).
- Fifth normal form is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy.

Normalization

Fifth normal form(5NF):

❖ Example 1

Agent	Company	Product
Suneet	ABC	Nut
Raj	ABC	Bolt
Raj	ABC	Nut
Suneet	CDE	Bolt
Suneet	ABC	bolt

- The table is in 4NF because it contains no multi-valued dependency.
- Suppose that table is decomposed into it's three relations P1,P2 & P3.

Normalization

Fifthnormal form(5NF):

❖ P1

Agent	Company
Suneet	ABC
Suneet	CDE
Raj	ABC

❖ P2

Agent	Product
Suneet	Nut
Suneet	Bolt
Raj	Bolt
Raj	Nut

❖ P3

Company	Product
ABC	Nut
ABC	Bolt
CDE	Bolt

Example 1

Normalization

Fifth normal form(5NF):

- From above tables or relations, perform natural join between any of two above relations i.e P1⋈P2 , P2⋈P3 or P1⋈P3 then extra rows are added so this decomposition is called lossy decomposition.
- But if no extra rows are added so this decomposition is called loseless decomoposition.
- So, above three tables P1,P2 and P3 are in 5NF.

- Link for examples:
- http://www.questionsolves.com/Website-Content/Normalization.php