**Introduction**

| | | | |
|---|---|---|---|
| **Notebook:** | CHAPTER 5 : Relational Database Design | | |
| **Created:** | 4/13/2020 9:54 AM | **Updated:** | 12/6/2023 10:56 PM |
| **Author:** | pagulearn.5334@gmail.com | | |

# Introduction

**Author:** pagulearn.5334@gmail.com

# Introduction

**Author:** pagulearn.5334@gmail.com

## Introduction

- A Relational database is a **collection of relations**(table).
- The goal of a relational database design is to **generate a set of "relation schemas" that allow us to**
  - **store information without unnecessary redundancy**
  - **retrieve information as per requirements easily and efficiently.**
- In this chapter we will discuss a way to **assess the quality of the designed database** ,and provides s**olution to convert poorly designed database into a database with good quality.**

## Pitfalls of Poor Database Design

- A poorly designed database always result in a **mal-functioning and inefficient** system.
- We will understand these pitfalls through an example.

## Example

- Taking an example of Banking System we consider two approach of designing a schema for Account and Branch table.

- In Example we will see two different approaches to design relational database.

**First Approach :**

Account:

| ano | balance | bname |
|---|---|---|
| A01 | 5000 | vvn |
| A02 | 6000 | ksad |
| A03 | 7000 | anand |
| A04 | 8000 | ksad |
| A05 | 6000 | vvn |

Branch:

| bname | baddress |
|---|---|
| vvn | Mota bazaar, VVNagar |
| ksad | Chhota bazaar, Karamsad |
| anand | Nana bazaar, Anand |

Figure 5.1: 'Account' and 'Branch' relations

- Here , two separate tables were created to store information related to Account and Branch.

**Second Approach**

Account_Branch:

| ano | bal | bname | badd |
|---|---|---|---|
| A01 | 5000 | vvn | Mota bazaar, VVNagar |
| A02 | 6000 | ksad | Chhota bazaar, Karamsad |
| A03 | 7000 | anand | Nana bazaar, Anand |
| A04 | 8000 | ksad | Chhota bazaar, Karamsad |
| A05 | 6000 | vvn | Mota bazaar, VVNagar |

Figure 5.2: 'Account_Branch' relation

- Here, instead of creating two separate tables , only single relation Account_Branch has been created.

From the above two approaches the **second approach result in poor database design**. second approach results in different **anomalies** which is described in below section.

- **Redundancy**
    - One of the main goal of the database design is to **minimize the storage space required to store data** , which means database should not contain any **redundant data.**
    - If we compare two approaches , we can see that in the **Account_Branch** table branch **addresses are store repeatedly for different account** and same branch name like (ksad,vvn).

- As there will be thousands of account available in different branches , **second approach of database design result in data redundancy and consequently -Memory Wastage.**
- Where as in the first approach **branch address for each branch is stored only once** in Branch table.which an example of **efficient utilization of memory.**

- **Update Anomalies**

- Consider that the **branch address** of one of the branch , **say "vvn" changes.**
- In such cases , **branch address in all tuples which belongs to "vvn" must need to be changed in Account_Branch relation.**
- **Failure in updating** some of the tuples will result in **inconsistent state** of the database.
- The **same** branch will have **different addresses** in **single relation.**
- Such type of problem is known as "**Update anomaly"** and which must be avoided.
- Where as , in First approach we need to make changes in **only single row of Branch table corresponding to "vvn".**
- So, first approach is easy one to apply such kind of changes.

- **Insertion Anomalies**

- Assume a situation where **bank will open a new branch** which **does not have any accounts till now?**
- In the **second approach** , if we want to **insert a record in Account_Branch** table for new branch , the only way to do this is to **place null values for attribute ano and balance** .
- But this will create problem as **ano is primary key for table we can not insert null for it.**
- So , it is **impossible to store** branch related data in **Account_Branch** table till a new account is created.
- Such type of problem is known as **"Insertion Anomalies"**.
- But In the **first approach** , **new record related to branch can be inserted in branch table without causing any problems.**

- **Deletion Anomalies**

- For **second approach** , let say we need to **delete a tuple** representing some account from Account_Branch relation, and that account is the **last account in that particular branch.**

- The information concerning that **branch** will also b**e lost** from the database.
- In our example , if tuple with **account no 'ao3' is deleted form Account_Branch, information regarding 'anand' branch will also be lost.**
- such kind of problem is known as **"Deletion Anomalies."**
- But , if we follow the **first approach** , tuple can be **deleted from account** relation , without causing any problem with Branch relation.

- **Null Values**

- When **many attributes are grouped in single relation** , they may end up in **many null values in various tuples.**
- **null** values are always **problematic for various operations such as - JOIN , and with some functions - SUM,COUNT.**
- So null values must be avoided whenever possible.

**Note :**

- **So , to convert a poorly design database into a good quality database Normalization can be used.**

## Pre-requisite of Normalization

- There some concepts which need to be understand properly before going for the Normalization.

- Those Fundamental concepts are as follows:

### Prime vs non prime Attributes

- **Prime Attribute :**

- "An attribute which is member of a primary key of the given relation is known as a prime attribute"
- "An Attribute , which forms a primary key for given relation is known as a prime attribute."

- **Example :**
  - Table : Account (ano , balance , bname)
    - here , ano is primary key so it is consider as a prime attribute

- Table : Account_Holder (cid , ano ,access_date)
  - here (cid , ano) combination is primary key .
  - both these attributes combination forms the primary key so cid and ano is prime attribute for relation Account_Holder

- **Non-prime Attribute**

  - "An attribute which is not a member of primary key of the given relation is known as a non-prime attribute."

  - **Example :**
    - Table : Account (ano , balance, bname)
      - for Account table both balance and bname is non prime attribute as it is not part of primary key of Account relation
    - Table : Account_Holder (cid , ano ,access_date)
      - access_date is non prime attribute.

## Functional Dependency

- Let **R** be a **relational** schema having 'n' **attributes A1,A2,A3.....An**
- Let **X and Y be a two subsets of attributes of relation R.**
- **"If values of X component of a tuple uniquely or functionally determines the values of the Y component , there is a functional dependency from X to Y."**
- This is denoted by **X->Y.**
- it is also said that **Y is functionally dependent on the X.**
- The abbreviation for functional dependency is **FD** or **f.d**
- Set of attributes **X** is called the **left hand side of the FD**, **Y** is called the **right hand side.**

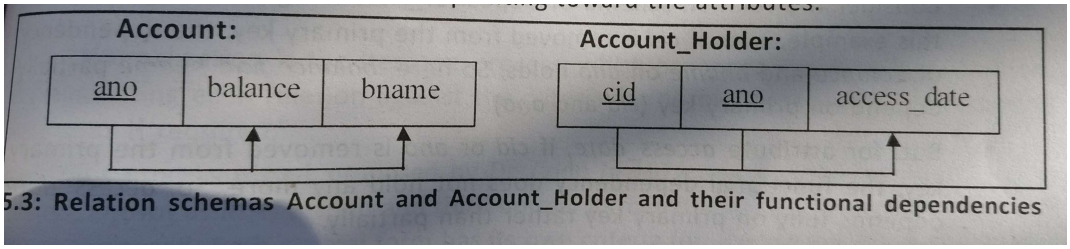
- **Example :**

  - Relation is : Account
  - Attributes are : ano , balance , bname
  - Let say,
    - X subset of attributes : X = {ano}
    - Y subset of attribute :Y ={balance , bname}
  - Now take one tuple from Account Relation { a01,5000,vvn}.
  - based on the definition values of X component of a tuple which is : X ={ano} ={a01}
  - functionally determines the values of Y component : Y = {balance,bname}={5000,vvn}

- So , we can say that {ano} uniquely determines {balance,banme}
  - In Simple words we can say that - from the account no we can know the balance and branch name
- This can be denoted as {ano}-> {balance , bname }
- Note :
  - {balance}-> {ano} it is not the correct example of function dependency because from the Balance value we can not determine ano of particular Account.
  - if you see the Account table in above figure , for balance value ={6000} we have two account =A02 ,A05 so , from balance we can not determine ano uniquely.

- **Diagrammatic Notation**

- Each FD is displayed as a horizontal line.
- Left hand side attributes of the FD are connected by vertical lines to the line representing the FD.
- The right hand side attributes are connected by arrows pointing towards the attributes



5.3: Relation schemas Account and Account_Holder and their functional dependencies

**Fully Functional Dependency**

- **"A Functional X->Y in relation R is fully functional dependency , if dependency does not hold after removal of any attribute a from X"**

- **Example 1 : Student {roll_no,sem,name,marks } [Assuming all the students of IT]**

- Functional Dependency : {roll_no,sem}--> { marks,name}
- Here X = {roll_no,sem} , Y = {marks,name}
- this functional dependency X--> Y is fully functional dependent , because removal of any attribute A = {sem} from X , then FD {roll_no}-->{marks,name} does not hold.
- So , For this example of Student Relation FD {roll_no,sem}--> {marks,name} is fully functional dependent.

- **Example 2 : Depositor_Account (cid , ano, access_date,balance, bname)**

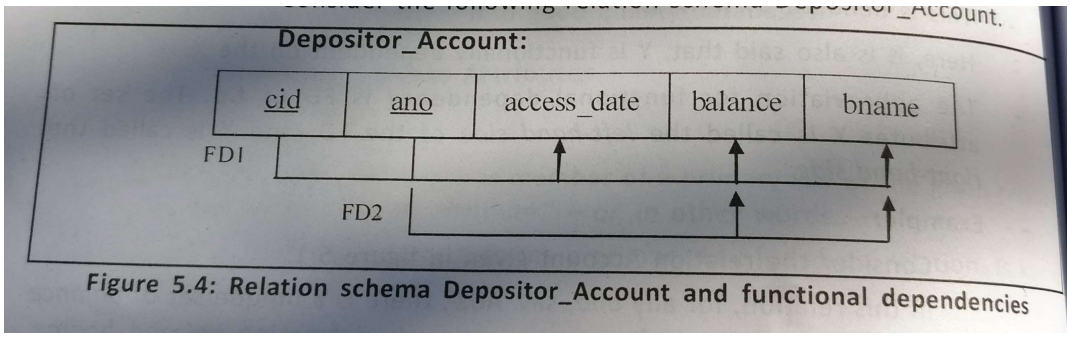


Figure 5.4: Relation schema Depositor_Account and functional dependencies

  - Primary Key : {cid ,ano}
  - Functional Dependency:
    - {cid ,ano}--> {access_date}
    - {cid,ano} -->{balance, bname}
    - {ano} -->{ balance, bname}
  - Here , in above functional dependencies, this FD{cid ,ano}--> {access_date} is fully functional because removing any attribute either cid or ano from the left side of FD, the functional dependency does not hold.
  - only {ano} can not determine access date uniquely because of join account
  - only {cid} can not determine access date uniquely because of multiple account
  - So FD : {cid ,ano}--> {access_date} is fully functionally dependent
  - But second functional dependency: FD : {cid,ano} -->{balance, bname} is not fully functional because removing cid from left side of FD . still functional dependency
    - {ano} -->{balance, bname} holds.

**Partial Functional Dependency**

- **"A functional dependency X-->Y in relation schema R is partial functional dependency , if dependency holds even after removal of any attribute A from X.**

- Example : Depositor_Account (cid , ano, access_date,balance, bname)

  - Primary Key : {cid ,ano}
  - Let's talk about functional dependency : FD : {cid,ano} --> {balance, bname}
  - It is a partial functional dependency because from the above FD after removing {cid} from left side of FD , {ano} -->{ balance,

bname} FD still holds as from ano we can uniquely determine the balance and bname.

- **Note**
  - if non prime attribute of relation is depend on prime attribute (part of primary key attribute) then partial functional dependency is present in relation.

## Transitive Dependency

- **"A functional dependency X-->Y in relation R is transitive dependency , if there is a set of attribute Z , such that Z is a non prime attribute and both dependencies X-->Z and Z-->Y hold."**

- **Example : Account_Branch(ano,balance,bname,baddress)**

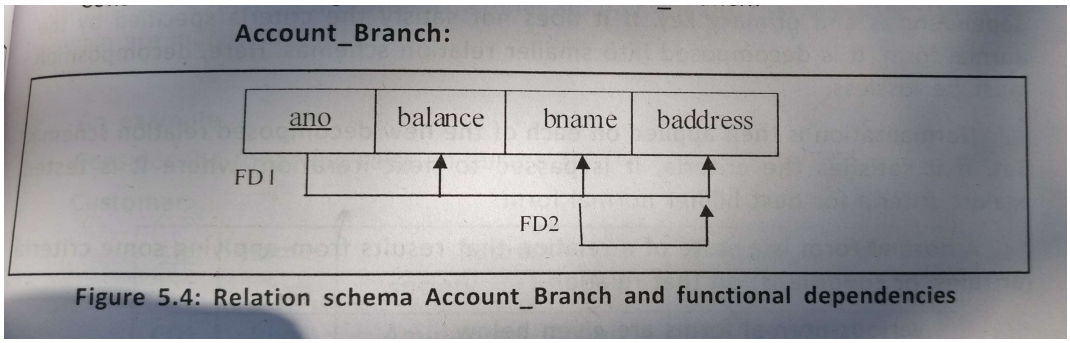


Figure 5.4: Relation schema Account_Branch and functional dependencies

- Primary Key : {ano}
- Prime attribute:{ano}
- Non prime attribute:{balance,bname,baddress}
- FD:
  - {ano}-->{balance,bname,baddress}
    - we can also re write above FD as
      - {ano}-->{balance}
      - {ano}-->{bname}
      - {ano}-->{baddress}
  - {bname}-->{baddress}

- Now from the above data we have X: {ano} , Y:{baddress} , we also have some Z :{banme}
- here Z:{bname} which is non prime attribute
- we have FD :
  - {ano}->{bname} [X-->Z]
  - {bname}-->{baddress} [Z-->Y]
- which indicates that {ano}-->{baddress} [X-->Y]

# NORMALIZATION

**Author:** pagulearn.5334@gmail.com

## NORMALIZATION

- **"Normalization is the iterative process , which proceeds in a top down fashion by evaluating each relation against the criteria for normal forms , and decomposing relations if required".**

- This process is first proposed by Dr.Codd in 1972.
- Codd proposed three normal forms : 1NF ,2NF , 3NF initially.
- A relation is said to be in a normal form , if it satisfy the criteria for that  normal form.

## Goal of Normalization

- to minimize the data redundancy
- to minimize update , insertion and deletion anomalies.

## Normalization Process

- In normalization given relation is analyzed based on its functional dependency and primary key.
- if relation does not satisfies the criteria specified by the normal form it is decomposed into smaller relation schema.
- Normalization is then applied to each of new decomposed relations.
- if it satisfy the criteria then it is passed to next iteration , where it is tested against criteria  for next higher order normal form.

## Normalization Stages

- **"A normal form is a state of a relation that results from applying some criteria on that relation."**
- Various Normal forms are given below:
    - First Normal Form(1NF)
    - Second Normal Form(2NF)
    - Third Normal Form(3NF)
    - Boyce-codd Normal Form (BCNF)
    - Forth Normal Form(4NF)
    - Fifth Normal Form(5NF)

## 1NF : First Normal Form

- Normalization process start with applying criteria of first normal form to a given relation.
- if it does not satisfy the criteria , it is decomposed / converted into relation which satisfy the criteria .
- If it satisfy the criteria then it is passed to apply second normal form.

- **Criteria**
  - **"A relational schema R is in 1NF if it does not have any composite attribute , multi-value attribute or their combination."**

- Composite and multi-value attribute create problems in storing and retrieving data.
- **1NF ensure that domains of all attributes in a relation are atomic.**
- Atomic means data values for domains are indivisible units.

- Example : **Customer (cid, name ,address, contactno)**

Customer:

| cid | name | address | | contact_no |
|---|---|---|---|---|
| | | society | city | |
| C01 | Riya | Amul Aavas, Anand | | {9876543210} |
| C02 | Jiya | Sardar Colony, Karamsad | | {232740, 25356178} |
| C03 | Piya | Marutisadan, VVNagar | | {55414,55415,55416} |
| C04 | Diya | Saral Society, Anand | | |
| C05 | Tiya | Birla Gruh, VVNagar | | {9825098250} |

Figure 5.5: 'Customer' relation with composite & multi-valued attribute

  - Here , for above relation customer contains
    - **composite attribute : address [which can be divided into society + city]**
    - **multi-value attribute : contactno**

  - **Problems with Customer relation**
    - **Information retrival**

        - if there is a need to find out all customers belonging to some particular city , it is difficult to retrieve as city names are combined with society name in address field
    - **Multiple values in single field**
        - if any customer has two or more contact no it is not possible to store multiple contact no in single field.

- **Solution for Composite Attribute**

    - Insert separate attributes in a relation for each sub attribute for a composite.

Customer:

| cid | name | society | city | contact_no |
|---|---|---|---|---|
| C01 | Riya | Amul Aavas | Anand | {9876543210} |
| C02 | Jiya | Sardar Colony | Karamsad | {232740, 25356178} |
| C03 | Piya | Marutisadan | VVNagar | {55414,55415,55416} |
| C04 | Diya | Saral Society | Anand | |
| C05 | Tiya | Birla Gruh | VVNagar | {9825098250} |

ure 5.6: 'Customer' relation with solution to composite & multi-valued attribute

    - Here , our composite attribute address can be sub divided into sub attribute society and city.
    - So , We have converted our relation in such a way that now it does not contain any composite attribute.

- **First Approach to solve Multi Value Attribute**

    - **Determine maximum allowable values for a multi value attribute , then insert separate attributes in a relation as shown in following relation schema.**
    - In Our Example we have allowed maximum two numbers are allowed to store , so insert two separate attributes to store contact numbers as shown below.

Customer:

| cid | name | society | city | contact_no1 | contact_no2 |
|---|---|---|---|---|---|
| C01 | Riya | Amul Aavas | Anand | 9876543210 | |
| C02 | Jiya | Sardar Colony | Karamsad | 232740 | 25356178 |
| C03 | Piya | Marutisadan | VVNagar | 55414 | 55415 |
| C04 | Diya | Saral Society | Anand | | |
| C05 | Tiya | Birla Gruh | VVNagar | 9825098250 | |

e 5.7: 'Customer' relation with solution to composite & multi-valued attribute

- If customer has only one contact no or no any contact no then keep related field empty for that customer
- if customer has more than two contact numbers then store two contact numbers and ignore all other numbers.

- **Second Approach to solve Multi value Attribute**

- In the second approach , **remove the multi value attribute that violates 1NF and place it in separate table (relation) along with the primary key of original relation**
- The primary key of new relation is the combination of multi value attribute and primary key of old relation.

Customer:

| cid | name | society | city |
|---|---|---|---|
| C01 | Riya | Amul Aavas | Anand |
| C02 | Jiya | Sardar Colony | Karamsad |
| C03 | Piya | Marutisadan | VVNagar |
| C04 | Diya | Saral Society | Anand |
| C05 | Tiya | Birla Gruh | VVNagar |

Customer_Contact:

| cid | contact_no |
|---|---|
| c01 | 9876543210 |
| c02 | 232740 |
| c02 | 25356178 |
| c03 | 55414 |
| c03 | 55415 |
| c03 | 55416 |
| c05 | 9825098250 |

Figure 5.8: 'Customer' relation with solution to composite & multi-valued attribute

- As you can see that we have removed multi value attribute contact_no from first relation and put it in separate relation called Customer_Contact along with the primary key :{ cid} of Customer relation.
- Primary key of new relation Customer_Contact is combination of { cid , conatct_no} .

- **Note :**

- Here , First approach is simple , but it is not possible to put restriction on maximum allowable values for multi value attributes.
- Another disadvantage of first approach is : data loss
  - it is not possible to store more than two contact no in first approach. and it introduce null values in many field.
- Second approach is superior as it does not suffer from the drawback of first approach but it is complicated one and makes data retrieval slower.
  - As data related to customer need to be retrieved by combining two tables. (Customer , Customer_Contact)

## 2NF : Second Normal Form

- If any relation satisfy the criteria for first normal form , it is tested against the criteria of 2NF.
- If it does not satisfy the criteria of 2NF then relation is decomposed into a relations that satisfy the criteria.
- if it satisfy the criteria of 2NF then it is tested against the criteria of next higher order normal form.

- **Criteria**

  - **"A relation schema R is in 2NF if**
    - **it is already in 1NF and**
    - **every non prime attribute of a relation is fully functionally dependent on primary key."**

- So , we can say that if there is a Partial Functional Dependency in Relation R then it is not in 2NF.
- **Partial Functional Dependency is relation results in data redundancy and create various anomalies.**
- **2NF ensure that relation should not have any Partial Functional Dependency.**

- **Note :**

  - A relation can violate 2NF only when it has more than one attributes in combination as a Primary key.
  - If Relation has only single attribute as Primary key , then relation is definitely in 2NF.

- Example : **Depositor_Account (cid , ano, access_date,balance, bname)**

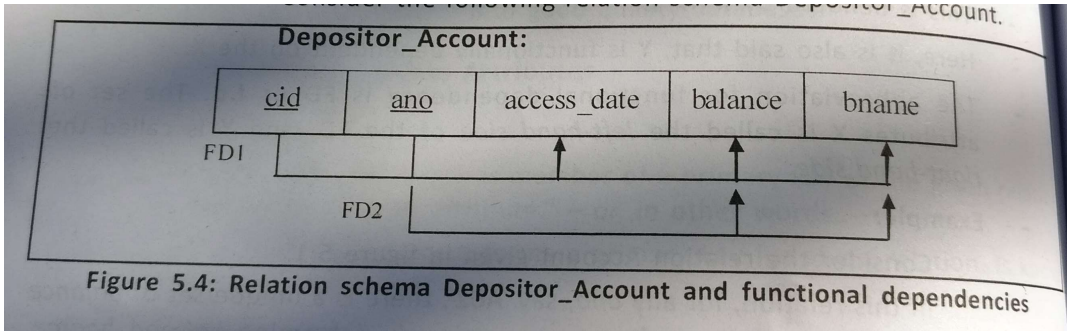


*Figure 5.4: Relation schema Depositor_Account and functional dependencies*

- Primary Key : {cid ,ano}
- Functional Dependency:
  - {cid ,ano}--> {access_date}
  - {cid,ano} -->{balance, bname}
  - {ano} -->{ balance, bname}
- Prime attributes : {cid , ano}
- Non prime attributes : {balance , bname , access_date}

- here , we can say that a non prime attribute access_date if fully functional dependent on primary key.
- but {bname, balance} non prime attribute does not fully functional dependent on primary key because {bname , balnce } can be determined from only {ano} .. [{ano} -->{ balance, bname}]
- which indicates that {bname, balance} is partially dependent.
- So , we can say that this relation Depositor_Account is not in 2NF.

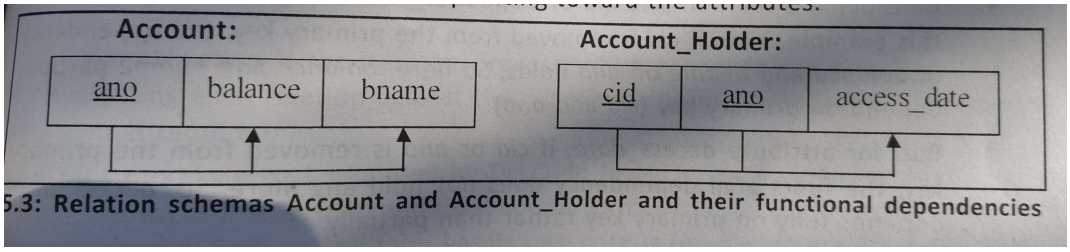
- Such kind of partial functional dependency results in Data redundancy.
  - Let say if , there are join accounts are allowed and two customer 'c01' ,'c02' is associated with 'A01' then data values will be duplicated for attribute bname and balance for two different tuples related to customer 'c01' and 'c02'.

- **Solution :**

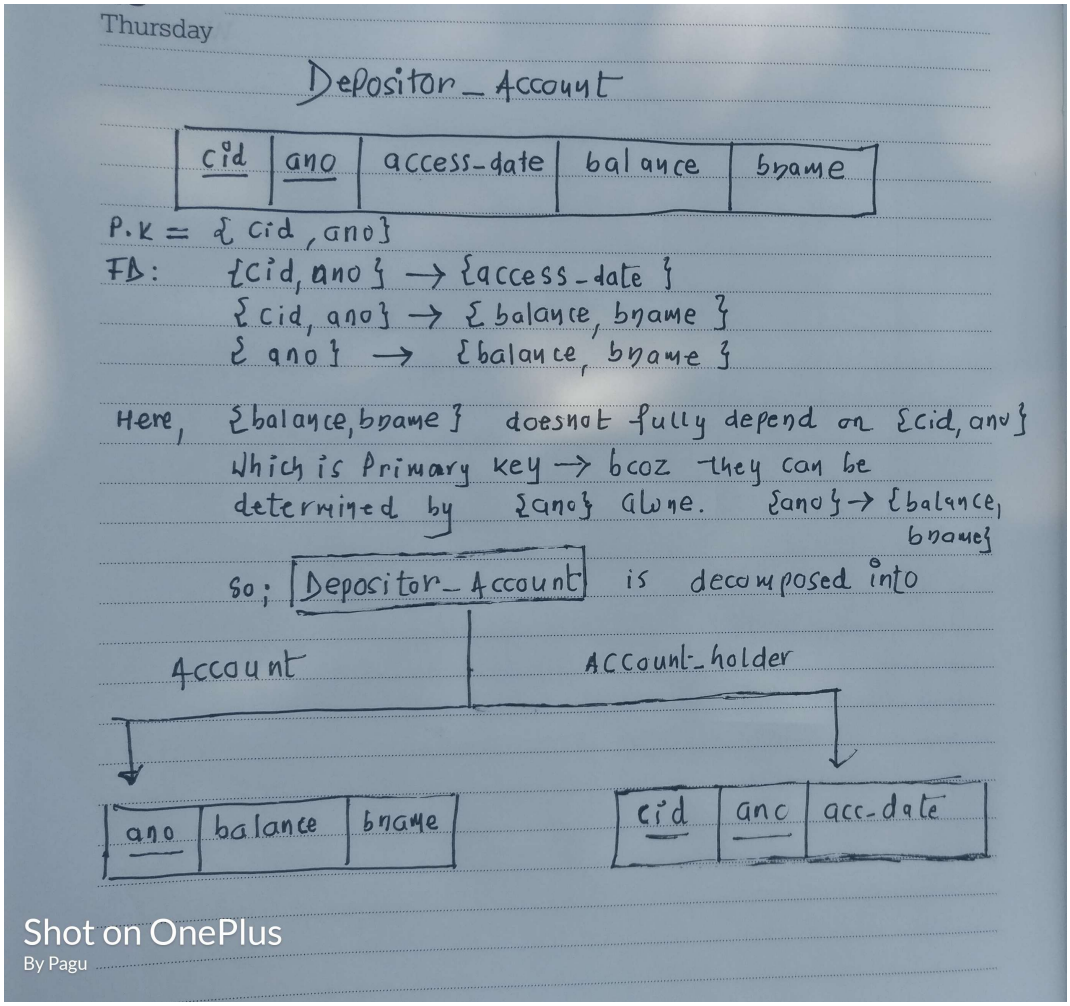
- As the relation (Depositor_Account) does not satisfy the criteria of 2NF , we need to decompose relation in such a way that it will be free from partial functional dependency.
- For this purpose following action should be taken

- remove the partial dependent non prime attributes (bname,balance) that violates 2NF from relation.
- Place them in a separate new relation along with the prime attribute (ano) on which they fully dependent.
- Primary key of new relation will be this prime attribute (ano).

- In our Example of Depositor_Account relation
  - non prime attributes {bname , balance} violates the criteria of 2NF.
  - so we will remove them from original relation Depositor_Account and put them in new relation called Account.
  - As this non prime attribute is fully depend on prime attribute {ano} so we also need to include it in new relation Account.
  - So ,new relation Account would liik like this :Account(ano,balance,bname ) and {ano} will become primary key for this relation.
  - Original relation Depositor_Account would be look like this : Depositor_Account (cid,ano,access_date) which is renamed to Account_Holder in your book.
  - Now we can say that original relation Depositor_Account violates the criteria of 2NF so it is decomposed to two sub relations Account and Account_Holder.



5.3: Relation schemas Account and Account_Holder and their functional dependencies




# NORMALIZATION -2

**Author:** pagulearn.5334@gmail.com

**3NF : Third Normal Form**

- If relational schema satisfy the criteria for 2NF , it is tested against the criteria of 3NF.
- If relation does not satisfy the criteria , it is decomposed into separate relations and converted into relation that satisfy the criteria , else it it passed for next higher order normal form.

- **Criteria :**

  - **" A Relation schema R is in 3NF if -**
    - **it is in 2NF and**

- **no any non prime attribute of a relation is transitively dependent on primary key"**

- Third normal form **ensure** that the **relation does not have any non prime attribute transitively dependent on the primary key.**

- Example : **Account_Branch(ano,balance,bname,baddress)**

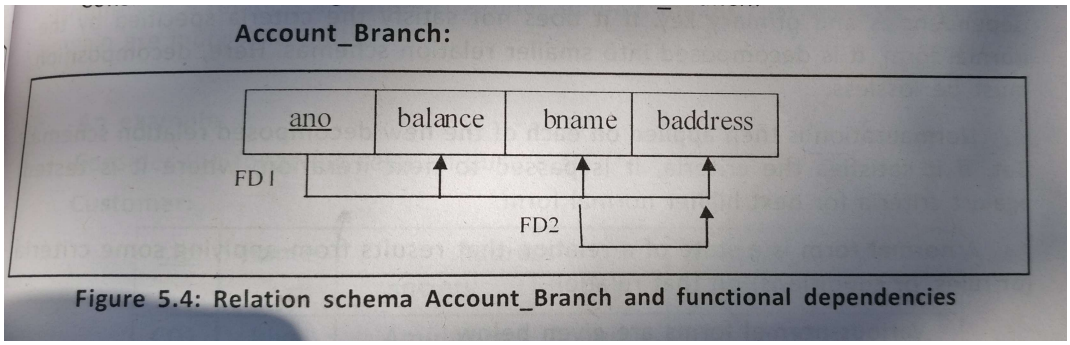


Figure 5.4: Relation schema Account_Branch and functional dependencies

  - Primary Key : {ano}
  - Prime attributes : {ano}
  - Non prime attribute : {bname , balance, baddress}
  - FD :
    - {ano}--> { bname , balance , baddress}
      - this FD can also be written like this
      - {ano}-->{bname} , {ano}-->{balance},{ano}--> {baddress}
    - {bname}-->{baddress}

- from the above functional dependencies we can see that,
  - **A non prime attribute : {baddress}** is transitively dependent on **primary key {ano}** due to following functional dependency.
    - **{ano}-->{bname} , {bname}-->{baddress} which indicates that {ano}-->{baddress}**
- So above relation has transitive functional dependency from {ano} to {baddress}
- Such kind of transitive dependency result in **data redundancy.**
  - In this relation we need to store baddress repeatedly for each of the account with same bname. which leads to wastage of memory.
- Transitive function dependency can also leads to insert , update and delete anomalies.

- **Note :**
  - in a given relation with FDs **if there is an association(f.d) among non prime attribute** than we can say that relation

contains **transitive functional dependency.**

- **Solution :**

  - **Decompose the relation** in such a way that resultant **relation do not have any non prime attribute transitively dependent on primary key.**
  - For this purpose , remove the **non-prime attributes (baddress) that violates the 3NF criteria** from relation.
  - put them in **new relation (Branch) along with the non prime attribute** (bname) due to which transitive function dependency occurred.
  - **Primary key for this new relation will be this non prime attribute (bname).**

---

  - For Our Example :
    - baddress is the non prime attribute which is transitively dependent on P.K : {ano}
    - So, we will remove baddress from Account_Branch and put it in New relation Branch.
    - this baddress is transitively dependent due a non prime attribute bname as FD : {bname}-->{baddress}
    - so, we will also put {bname} in new relation Branch along with {baddress}.
    - and {bname} will become primary key for New relation Branch.
    - Now , our original relation Account_Branch remain with {ano , balance , bname} , we rename it to Account.
    - So, now our original relation Account_Branch is decomposed into two relation Account and Branch.

---

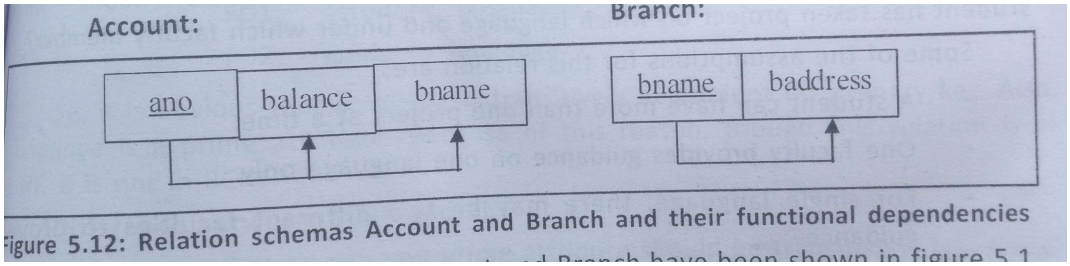


Figure 5.12: Relation schemas Account and Branch and their functional dependencies

Tuesday

ACCOUNT_BRANCH [ano, balance, bname, baddress]

Primary key : ano

Prime attributes : ano.

Non Prime attributes : balance, bname, baddress.

FD :=

1. {ano} → {balance, bname, baddress}
   → above FD can also be written as follows.

   {ano} → {balance} {ano} → {bname}
   {ano} → {baddress}.

2. {bname} → {baddress}
   in this fd. both the attributes are non prime so, we can say that there is an association between non prime attribute and this leads to trasitive f.d.

As; {ano} → {bname} . and {bname} → {baddress}
So; {ano} → {baddress}.

Solution :=

ACCOUNT_BRANCH [ano, balance, bname, baddress]

Remove baddress from it put in

New Relation BRANCH along with bname & make bname as P.K

ACCOUNT

| ano | balance | bname. |
|---|---|---|

BRANCH

| bname | baddress |
|---|---|

**BCNF - Boyce Codd Normal Form**

- Boyce Codd normal form is a slight variation of third normal form.

- it is stricter than 3NF.
- So , If relation is in BCNF, then it will also be in 3NF.
- But reverse is not true always , which means if relation is in 3NF it may or may not be in BCNF.

- **Criteria**

  - **"A relational Schema R is in BCNF if -**
    - **it is already in 3NF and**
    - **no any prime attribute of relation is transitively dependent on primary key "**

- BCNF **ensure** that the relation **does not have any prime attribute transitively dependent on primary key.**

- **Example : Project (student , language , guide)**

  - The relation stores the information about which student has taken project on which language under which faculty.
  - A student can have more than one project at a time
  - one faculty provides guidance on one language only
  - But for single language , there can be two different faculties to provide guidance.

Project:

| student | language | guide |
|---|---|---|
| Mita | JAVA | Prof. J. P. Parmar |
| Nita | Visual Basic | Prof. A. M. Panchal |
| Sita | JAVA | Prof. P. M. Tank |
| Gita | Visual Basic | Prof. H. A. Patel |
| Rita | Visual Basic | Prof. A. M. Panchal |
| Nita | JAVA | Prof. J. P. Parmar |
| Mita | Visual Basic | Prof. H. A. Patel |
| Rita | JAVA | Prof. P. M. Tank |

Figure 5.13: A relation Project that is in 3NF but not in BCNF

  - Primary Key : {student , language}

- Prime Attributes : {student , language}
- Non Prime Attributes: {guide}
- FD:
  - {student , language}-->{guide}
  - {guide}-->{language}
- From the above two functional dependency we can see that there is a **transitive function dependency from**
- **{student , language}-->{language}**
- which indicates that **{language}** which is a **prime attribute** is **transitively depends on primary key :{student , language}.**
- So it violates the criteria of BCNF , so Project relation is not in BCNF.

- **Solution:**

- Decompose relation such that **, resultant relations do not have any prime attribute transitively dependent on Primary Key.**
- So **remove the transitively dependent prime attribute that violates the BCNF criteria and place it in new relation along with the non prime attribute due to which transitive function dependency occured.**
- And **primary key** for **new relation** would be that **non prime attribute.**
- In our example ;
  - **prime attribute {language}** is **transitively dependent on primary key** due to **non prime attribute {guide}.**
  - So remove {language} and place it in separate relation called Guide_language along with non prime attribute on which it depends {guide}
  - and {guide} will become primary key for Guide_Language relation.
- So , Project relation is decomposed into two relations
  - student_Guide (student , guide)
  - Guide_language(guide , language)
- Remember that primary key for student_guide will be combination of {student,guide} both.

Student_Guide:

| student | guide |
|---|---|
| Mita | Prof. J. P. Parmar |
| Nita | Prof. A. M. Panchal |
| Sita | Prof. P. M. Tank |
| Gita | Prof. H. A. Patel |
| Rita | Prof. A. M. Panchal |
| Nita | Prof. J. P. Parmar |
| Mita | Prof. H. A. Patel |
| Rita | Prof. P. M. Tank |

Guide_Language:

| guide | language |
|---|---|
| Prof. J. P. Parmar | JAVA |
| Prof. A. M. Panchal | Visual Basic |
| Prof. P. M. Tank | JAVA |
| Prof. H. A. Patel | Visual Basic |

Figure 5.15: Relations Student_Guide and Guide_Language

**Difference Between 3NF and BCNF**

| 3NF | BCNF |
|---|---|
| • A Relation schema R is in 3NF if -<br>◦ it is in 2NF and<br>◦ no any non prime attribute of a relation is transitively dependent on primary key | • A relational Schema R is in BCNF if -<br>◦ it is already in 3NF and<br>◦ no any prime attribute of relation is transitively dependent on primary key |
| • 3NF ensures that no any non prime attribute should be transitively dependent on primary key | • BCNF ensures that no any prime attribute should be transitively dependent on Primary Key |
| • 3NF is less Stricter than BCNF | • BCNF is more stricter than 3NF |
| • If relation is in 3NF then , it may or may not be in BCNF. | • If relation is in BCNF , then it is definitely in 3NF. |
| | |

2NF Examples

| CAND_ID | SUBJECT_NO | SUBJECT_FEE |
|---|---|---|
| 111 | S1 | 1000 |
| 222 | S2 | 1500 |
| 111 | S4 | 2000 |
| 444 | S3 | 1000 |
| 444 | S1 | 1000 |
| 222 | S5 | 2000 |

TUTOR table

| TUTOR_ID | COURSE | TUTOR_AGE |
|---|---|---|
| 2115 | Java | 30 |
| 2115 | C | 30 |
| 4997 | Python | 35 |
| 8663 | C++ | 38 |
| 8663 | Go | 38 |

<StudentProject>

| StudentID | ProjectID | StudentName | ProjectName |
|---|---|---|---|
| S89 | P09 | Olivia | Geo Location |
| S76 | P07 | Jacob | Cluster Exploration |
| S56 | P03 | Ava | IoT Devices |
| S92 | P05 | Alexandra | Cloud Deployment |

3NF example

**TABLE_BOOK_DETAIL**

| Book ID | Genre ID | Genre Type | Price |
|---|---|---|---|
| 1 | 1 | Gardening | 25.99 |
| 2 | 2 | Sports | 14.99 |
| 3 | 1 | Gardening | 10.00 |
| 4 | 3 | Travel | 12.99 |
| 5 | 2 | Sports | 17.99 |

**Student_Department(Enroll_no , Name , Dept_name,Dept_HOD)**