# Database Management System
## Schema Refinement - Functional Dependancy and Normalization

Dr. Nachiyappan S

VIT-Chennai

### Module-3

- Guidelines for Relational Schema.
- Functional dependency
- Normalization (1-NF, 2-NF and 3-NF)
- Boyce Codd Normal Form.
- Multi-valued dependency
- Fourth Normal form (4-NF).
- Join dependency and
- Fifth Normal form (5-NF).

# Text Books and References

**Text Books**

- R. Elmasri & S. B. Navathe, Fundamentals of Database Systems, Addison Wesley, 7 th Edition, 2015

- Raghu Ramakrishnan,Database Management Systems,Mcgraw-Hill,4 th edition,2015

**References**

- A. Silberschatz, H. F. Korth & S. Sudershan, Database System Concepts, McGraw Hill, 6 th Edition 2010

- Thomas Connolly, Carolyn Begg, Database Systems : A Practical Approach to Design, Implementation and Management,6 th Edition,2012

- Pramod J. Sadalage and Marin Fowler, NoSQL Distilled: A brief guide to merging world of Polyglot persistence, Addison Wesley, 2012.

- Shashank Tiwari, Professional NoSql, Wiley, 2011.

# Introduction to Schema Refinement

- E-R Modeling is used by the designer as a requirement analysis tool
- Database design using E-R diagram is a by-product.
- It may contain the amount of
  1. Inconsistency
  2. Ambiguity (Uncertainty/Unclear)
  3. Redundancy.
- Refinement is required to solve these issues and the process is referred as **Normalization**.

# Functional Dependency

- It is a relationship that exists between two attributes.
- It typically exists between the **primary key** and **non-key** attribute within a table.
- **X → Y** : where 'X' is **Determinent** and 'Y' is **Dependent**.

**Example** :

Consider the Table : **Employee** and attributes - **Emp_Id, Emp_Name, Emp_Address.**

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of Employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as: **Emp_Id → Emp_Name**
i.e Emp_Name is functionally dependent on Emp_Id.

# Functional Dependency's Classification
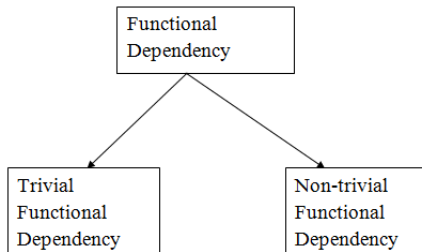


1. **Trivial FD** :
   - A → B has trivial functional dependency if B is a subset of A.
   - The following dependencies are also trivial like: A→ A, B → B
2. **Non-Trivial FD** :
   - A → B has a non-trivial functional dependency if B is not a subset of A.
   - When A intersection B is NULL, then A → B is called as complete non-trivial.

# Examples...!

Consider a table with two columns Employee_Id and Employee_Name.

**{Employee_id, Employee_Name} → Employee_Id** is a Trivial Functional Dependency as Employee_Id is a subset of {Employee_Id, Employee_Name}.

Also, **Employee_Id → Employee_Id** and **Employee_Name → Employee_Name** are trivial dependencies too.

**Employee_Id → Employee_Desig** and **Employee_Id → Employee_dob** are Non-Trivial Functional Dependency

# Building Block of Normalization

- Determinant : Attribute 'X' can be defined as determinant if it uniquely defines the attribute value 'Y' in given Relation/Entity.
- Attribute 'X' need NOT be a Key attribute.
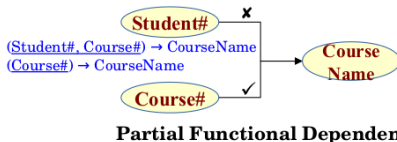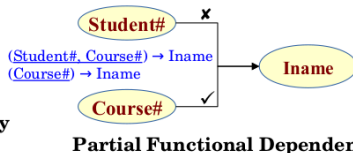- It can be represented as X → Y (X decides Y or Y is dependent on X)



- In the above Result relation, Grade is dependent on Marks, so Marks(determinant) and Grade(dependent) I.e Marks → Grade.
- Here Marks is not a Key attribute.

**Types of Functional Dependencies :**

1. **Full Functional Dependency**
2. **Partial Functional Dependency**
3. **Transitive Functional Dependency**



(Student#, Course#) → Iname
(Course#) → Iname

**Partial Functional Dependency**

(Student#, Course#) → Marks

**Full Functional Dependency**

(Student#, Course#) → CourseName
(Course#) → CourseName

**Partial Functional Dependency**

**Transitive Functional Dependency**
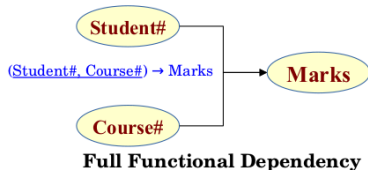
(Student#, Course#) → Marks
Marks → Grade
(Student#, Course#) → Grade

# DBMS Inference Rules

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule
2. Augmentation Rule
3. Transitive Rule
4. Union Rule
5. Decomposition Rule
6. Pseudo Transitive Rule

# DBMS Inference Rules

- **Reflexive Rule**(IR-1) : if Y is a subset of X, then X determines Y.
  i.e If **X** $\supseteq$ **Y** then X $\rightarrow$ Y
  Example : X = {a,b,c,d,e} and Y = {a,b,c}

# DBMS Inference Rules

- **Reflexive Rule**(IR-1) : if Y is a subset of X, then X determines Y.
  i.e If **X** $\supseteq$ **Y** then X $\to$ Y
  Example : X = {a,b,c,d,e} and Y = {a,b,c}

- **Augmentation Rule** : is also called as a **partial dependency**. In augmentation, if X determines Y, then XZ determines YZ for any Z (i.e if X $\to$ Y then XZ $\to$ YZ).
  Example : For **R(ABCD)**, if **A $\to$ B then AC $\to$ BC**.

# DBMS Inference Rules

- **Reflexive Rule**(IR-1) : if Y is a subset of X, then X determines Y.
  i.e If **X** $\supseteq$ **Y** then X $\rightarrow$ Y
  Example : X = {a,b,c,d,e} and Y = {a,b,c}

- **Augmentation Rule** : is also called as a **partial dependency**. In augmentation, if X determines Y, then XZ determines YZ for any Z (i.e if X $\rightarrow$ Y then XZ $\rightarrow$ YZ).
  Example : For **R(ABCD)**, if **A $\rightarrow$ B then AC $\rightarrow$ BC**.

- **Transitive Rule** : if X determines Y and Y determine Z, then X must also determine Z. (i.e, If X $\rightarrow$ Y and Y $\rightarrow$ Z then X $\rightarrow$ Z).

# DBMS Inference Rules

- **Reflexive Rule**(IR-1) : if Y is a subset of X, then X determines Y.
  i.e If **X** $\supseteq$ **Y** then X $\rightarrow$ Y
  Example : X = {a,b,c,d,e} and Y = {a,b,c}

- **Augmentation Rule** : is also called as a **partial dependency**. In augmentation, if X determines Y, then XZ determines YZ for any Z
  (i.e if X $\rightarrow$ Y then XZ $\rightarrow$ YZ).
  Example : For **R(ABCD)**, if **A $\rightarrow$ B then AC $\rightarrow$ BC**.

- **Transitive Rule** : if X determines Y and Y determine Z, then X must also determine Z. (i.e, If X $\rightarrow$ Y and Y $\rightarrow$ Z then X $\rightarrow$ Z).

- **Union Rule** : if X determines Y and X determines Z, then X must also determine Y and Z. (i.e If X $\rightarrow$ Y and X $\rightarrow$ Z then X $\rightarrow$ YZ )

# DBMS Inference Rules

- **Reflexive Rule**(IR-1) : if Y is a subset of X, then X determines Y.
  i.e If **X $\supseteq$ Y** then X $\to$ Y
  Example : X = {a,b,c,d,e} and Y = {a,b,c}

- **Augmentation Rule** : is also called as a **partial dependency**. In augmentation, if X determines Y, then XZ determines YZ for any Z (i.e if X $\to$ Y then XZ $\to$ YZ).
  Example : For **R(ABCD)**, if **A $\to$ B then AC $\to$ BC**.

- **Transitive Rule** : if X determines Y and Y determine Z, then X must also determine Z. (i.e, If X $\to$ Y and Y $\to$ Z then X $\to$ Z).

- **Union Rule** : if X determines Y and X determines Z, then X must also determine Y and Z. (i.e If X $\to$ Y and X $\to$ Z then X $\to$ YZ )

  **Proof:**
  1. X $\to$ Y (given)
  2. X $\to$ Z (given)
  3. X $\to$ XY (using IR2 on 1 by augmentation with X. Where XX = X)
  4. XY $\to$ YZ (using IR2 on 2 by augmentation with Y)

  5. X $\to$ YZ (using IR3 on 3 and 4)

- **Decomposition Rule** : is also known as project rule.
  It is the reverse of union rule.
  i.e if X determines Y and Z, then X determines Y and X determines Z
  separately. **If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$.**

  **Proof**:
  1. $X \rightarrow YZ$ (given)
  2. $YZ \rightarrow Y$ (using IR1 Rule)
  3. $X \rightarrow Y$ (using IR3 on 1 and 2)

- **Decomposition Rule** : is also known as project rule.
  It is the reverse of union rule.
  i.e if X determines Y and Z, then X determines Y and X determines Z
  separately. **If X → YZ then X → Y and X → Z**.

  **Proof**:
  1. X → YZ (given)
  2. YZ → Y (using IR1 Rule)
  3. X → Y (using IR3 on 1 and 2)

- **Pseudo transitive Rule** : if X determines Y and YZ determines W, then XZ
  determines W.
  **If X → Y and YZ → W then XZ → W**

  **Proof :**
  1. X → Y (given)
  2. WY → Z (given)
  3. WX → WY (using IR2 on 1 by augmenting with W)

  4. WX → Z (using IR3 on 3 and 2)

# Introduction to Normalization

If a table is not properly normalized and have data redundancy then it will not only eat up **extra memory space** but will also make it difficult to handle and update the database, without facing data loss.

Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized. To understand these anomalies.

Consider the below **Student Table** :

| rollno | name | branch | hod | office_tel |
|--------|------|--------|-------|-----------|
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, **hod**(Head of Department) and o**ffice_tel** is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

# Problems without Normalization

Consider the below Student Table :

| rollno | name | branch | hod | office_tel |
|--------|------|--------|------|-----------|
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

**Insertion Anomaly** : Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.

Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but Insertion anomalies.

# Problems without Normalization

Consider the below Student Table :

| rollno | name | branch | hod | office_tel |
|--------|------|--------|------|-----------|
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

## Updation Anomaly

In our Student Table, What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to **data inconsistency**.

This is Updation anomaly.

# Problems without Normalization

Consider the below Student Table :

| rollno | name | branch | hod | office_tel |
|--------|------|--------|------|-----------|
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

## Deletion Anomaly

In our Student table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information.

This is Deletion anomaly.

Hence we need to refine our design so that we make an efficient database in terms of storage, space and Inserts, Updates and Deletes Operation.
This is called as **Normalization**.

# Normalization

- Normalization of Database : is a technique of organizing the data in the database.
- It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies
- It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.
- Normalization is used for mainly two purpose,
    - Eliminating redundant(useless) data.
    - Ensuring data dependencies make sense i.e data is logically stored

## Normalization Rules

1NF First Normal Form
2NF Second Normal Form
3NF Third Normal Form
BCNF (Boyce and Codd Normal Form)
4NF and 5NF Fourth and Fifth Normal form

Every attribute should be atomic in nature

Remove Partial Dependencies

Remove Transitive Dependencies

Removing Lower Order Anomalies

Removing Higher Order Anomalies

Dependency Preservation for Lossless Join

0NF

1NF

2NF

3NF

BCNF

4NF

5NF
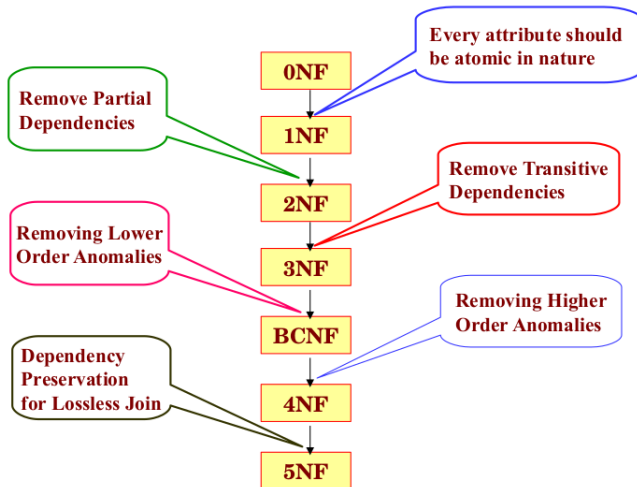
# 1NF - First Normal Form

## Important

If tables in a database are not even in the 1st Normal Form, it is considered as **bad database design**

## 1NF Rules

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Example :
Let us consider the below table:

| roll_no | name | subject |
|---------|------|---------|
| 101 | Akon | OS, CN |
| 103 | Ckon | Java |
| 102 | Bkon | C, C++ |

# 1NF - First Normal Form

Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

| roll_no | name | subject |
|---------|------|---------|
| 101 | Akon | OS |
| 101 | Akon | CN |
| 103 | Ckon | Java |
| 102 | Bkon | C |
| 102 | Bkon | C++ |

# 2NF - Second Normal Form

## 2NF Rules

1. The table should be in the First Normal Form(1NF).
2. There should be no Partial Dependency.

Let's take an example of a **Student table** with columns **student_id**, **name, reg_no**(registration number), **branch** and **address**(student's home address).

| student_id | name | reg_no | branch | address |
|------------|------|--------|--------|---------|
| 10 | Akon | 07-WY | CSE | Kerala |
| 11 | Akon | 08-WY | IT | Gujarat |

Here, **student_id** is the primary key and every other column depends on it.

# 2NF - Second Normal Forrm

Let's create another table for **Subject**, which will have **subject_id** and **subject_name** fields and **subject_id** will be the primary key.

| subject_id | subject_name |
|------------|--------------|
| 1 | Java |
| 2 | C++ |
| 3 | Php |

Let's create another table **Score**, to store the marks obtained by students in the respective subjects along with **teacher**.

| score_id | student_id | subject_id | marks | teacher |
|----------|------------|------------|-------|---------|
| 1 | 10 | 1 | 70 | Java Teacher |
| 2 | 10 | 2 | 75 | C++ Teacher |
| 3 | 11 | 1 | 80 | Java Teacher |

Both, {**student_id + subject_id**} forms a **Candidate Key** for this table, which can be the **Primary key**.

# Where is Partial Dependency?

In the Score table, column **teacher** which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

Now as we just discussed that the primary key for this table is a composition of two columns which is **student_id & subject_id** but the teacher's name only depends on subject, hence the subject_id, has nothing to do with student_id.

This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?

The simplest solution is to remove columns **teacher** from **Score** table and add it to the **Subject** table. Hence, the Subject table will become:

| subject_id | subject_name | teacher |
|------------|--------------|--------------|
| 1 | Java | Java Teacher |
| 2 | C++ | C++ Teacher |
| 3 | Php | Php Teacher |

And **Score** table is now in the second normal form, with no partial dependency.

| score_id | student_id | subject_id | marks |
|----------|------------|------------|-------|
| 1 | 10 | 1 | 70 |
| 2 | 10 | 2 | 75 |
| 3 | 11 | 1 | 80 |

# 3NF - Third Normal Form

## 3NF Rules

A table is said to be in the Third Normal Form when,

- It is in the Second Normal form.
- It doesn't have Transitive Dependency.

Student Table

| student_id | name | reg_no | branch | address |
|------------|------|--------|--------|---------|
| 10 | Akon | 07-WY | CSE | Kerala |
| 11 | Akon | 08-WY | IT | Gujarat |
| 12 | Bkon | 09-WY | IT | Rajasthan |

Subject Table

| subject_id | subject_name | teacher |
|------------|--------------|---------|
| 1 | Java | Java Teacher |
| 2 | C++ | C++ Teacher |
| 3 | Php | Php Teacher |

Score Table

| score_id | student_id | subject_id | marks |
|----------|------------|------------|-------|
| 1 | 10 | 1 | 70 |
| 2 | 10 | 2 | 75 |
| 3 | 11 | 1 | 80 |

# 3NF - Third Normal Form

In the Score table, we need to store some more information, which is the
**exam_name** and **total_marks**, as below.

| score_id | student_id | subject_id | marks | exam_name | total_marks |
|----------|------------|------------|-------|-----------|-------------|
|          |            |            |       |           |             |
|          |            |            |       |           |             |
|          |            |            |       |           |             |

For Eg: A Mechanical student will have Workshop exam but a computer science student
won't. And for some subjects you have Practical exams and for few you don't.

So **exam_name** is dependent on both **student_id** and **subject_id**.

Well, the column **total_marks** depends on **exam_name**. For example, practicals are of
less marks while theory exams are of more marks.

But, **exam_name**(not a primary key) is just another column in the score table. and
**total_marks** depends on it.

Where a non-prime attribute depends on other non-prime attributes rather than
depending upon the prime primary key.

This is **Transitive Dependency**

# How to remove Transitive Dependency?

Take out the columns **exam_name** and **total_marks** from Score table and put them in an Exam table and use the **exam_id** wherever required.

| score_id | student_id | subject_id | marks | exam_id |
|----------|------------|------------|-------|---------|
|          |            |            |       |         |
|          |            |            |       |         |
|          |            |            |       |         |

The new Exam table

| exam_id | exam_name | total_marks |
|---------|-----------|-------------|
| 1       | Workshop  | 200         |
| 2       | Mains     | 70          |
| 3       | Practicals | 30         |

Advantage of removing Transitive Dependency is :

- Amount of data duplication is reduced.
- Data integrity achieved.

# BCNF : Boyce-Codd Normal Form

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF.

A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

It is also known as 3.5 Normal Form.

For a table to be in BCNF, following conditions must be satisfied:

## BCNF Rules

1. R must be in $3^{rd}$ Normal Form
2. and, for each functional dependency ( $X \rightarrow Y$ ), X should be a super Key.

# BCNF : Boyce-Codd Normal Form

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF.

A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

It is also known as 3.5 Normal Form.

For a table to be in BCNF, following conditions must be satisfied:

## BCNF Rules

1. R must be in $3^{rd}$ Normal Form
2. and, for each functional dependency ( $X \rightarrow Y$ ), X should be a super Key.

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency $A \rightarrow B$, A cannot be a non-prime attribute, if B is a prime attribute.

# BCNF Example

Below we have a college enrolment table with columns student_id, subject and professor.

| student_id | subject | professor |
|---|---|---|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

- One student can enrol for multiple subjects. For example, student with student_id : 101, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

# BCNF Example

What do you think should be the Primary Key?

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the 2nd Normal Form as their is no Partial Dependency.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the 2nd Normal Form as their is no Partial Dependency.

And, there is no Transitive Dependency, hence the table also satisfies the 3rd Normal Form.

# BCNF Example

What do you think should be the Primary Key?

Well, in the table above **{student_id, subject}** together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the 2nd Normal Form as their is no Partial Dependency.

And, there is no Transitive Dependency, hence the table also satisfies the 3rd Normal Form.

But this table is not in Boyce-Codd Normal Form.

# BCNF Example

| student_id | subject | professor |
|------------|---------|-----------|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

Why this table is not in BCNF?

# BCNF Example

| student_id | subject | professor |
|------------|---------|-----------|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

Why this table is not in BCNF?

**{student_id, subject}** form primary key, which means subject column is a prime attribute.

# BCNF Example

| student_id | subject | professor |
|------------|---------|-----------|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

Why this table is not in BCNF?

**{student_id, subject}** form primary key, which means subject column is a prime attribute.

But, there is one more dependency, **professor → subject**.

# BCNF Example

| student_id | subject | professor |
|---|---|---|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

Why this table is not in BCNF?

**{student_id, subject}** form primary key, which means subject column is a prime attribute.

But, there is one more dependency, **professor → subject**.

And while **{subject}** is a prime attribute, professor is a non-prime attribute, which is not allowed by BCNF.

# BCNF Example

How to satisfy BCNF?

# BCNF Example

How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, student table and professor table.

## How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, student table and professor table.

Below we have the structure for both the tables.

| student_id | subject | professor |
|------------|---------|-----------|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

## How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables, student table and professor table.

Below we have the structure for both the tables.

| student_id | subject | professor |
|------------|---------|-----------|
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

And, **Professor Table**

| p_id | professor | subject |
|------|-----------|---------|
| 1 | P.Java | Java |
| 2 | P.Cpp | C++ |

# Multivalued Dependencies

- Multivalued dependencies are a consequence of first normal form (1NF) (I.e Doesn't allows an attribute to have a set of values for a tuple, and the accompanying process of converting an UNF into 1NF.

- If we have 2/more multivalued independent attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. (Redundancy)

- This constraint is specified by a MVD

# MVD Example

- For Eg-1 : Consider the relation EMP. A tuple in EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname.

- An employee may work on several projects and may have several dependents, and the employee's projects and dependents are independent of one another.

- To keep the relation state consistent, and to avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project.

**EMP**

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

The EMP relation with two MVDs:

- Ename →→ Pname and
- Ename →→ Dname.

whenever two independent 1:N relationships A → B and A → C are mixed in the same relation R(A,B,C), then **MVD** may arise.

**For Eg-2**

| Student_Id | Student_Name | Course_Name | Inst_Name |
|------------|--------------|-------------|-----------|
| 50396 | KVP | DBMS | Tulasi |
| 50396 | KVP | OS | RK Singh |
| 50396 | KVP | CN | Pradeep.K.V |
| 50396 | KVP | TOC | Prabhakar Rao |
| 50396 | KVP | CN-Lab | Murali |

In the above relation with two MVDs:

- Student_ID $\twoheadrightarrow$ Course_name and
- Student_ID $\twoheadrightarrow$ Inst_name.

# Formal Definition of Multivalued Dependency

The multivalued dependency X →→ Y holds in a relation R if whenever we have two tuples of R that agree in all the attributes of X , then we can

swap their Y components and get two new tuples that are also in R .
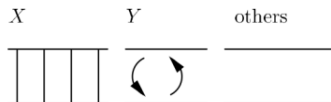


For Eg :

R = Drinkers(name, addr, phones,beerLiked) and with

MVD's : (Name →→ Phones). If Drinkers has 2 Tuples.

| Name | Address | Phones | BeerLiked |
|------|---------|--------|-----------|
| Sue | Chennai | P1 | B1 |
| Sue | Chennai | P2 | B2 |

| Name | Address | Phones | BeerLiked |
|------|---------|--------|-----------|
| Sue | Chennai | P2 | B1 |
| Sue | Chennai | P1 | B2 |

# MVD Rules

1. **Every FD is an MVD.**
   - Because if X →→ Y , then swapping Y 's between tuples that agree on X doesn't create new tuples.
   - Example, in Drinkers : name →→ addr .

2. **Complementation** :
   if X →→ Y , then X →→ Z, where Z is all attributes not in X/ Y .
   - Example: since name →→ phones holds in Drinkers,
   - so does name →→ (addr,beersLiked) .

Consider the Relation :

**Drinkers**(name, areaCode, phones, beersLiked, beerManf)

| Name | AreaCode | Phones | BeerLiked | BeerManf |
|------|----------|--------|-----------|----------|
| Sue | 650 | 555-1111 | Bud | A.B |
| Sue | 650 | 555-1111 | WickedAle | Pete's |
| Sue | 415 | 555-9999 | Bud | A.B |
| Sue | 415 | 555-9999 | WickedAle | Pete's |

Name →→ (areaCode,phones) holds, but neither
Name →→ areaCode nor Name →→ phones do

So Decompose the Above Relation into 4NF.

# Fourth Normal Form (4NF)

Eliminate redundancy due to multiplicative effect of MVD's.

- Roughly: treat MVD's as FD's for decomposition, but not for finding keys.
- Formally: R is in Fourth Normal Form if whenever MVD exists, I.e $X \rightarrow\rightarrow Y$ is non-trivial ( Y is not a subset of X , and $X \cup Y$ is not all attributes), then X is a superkey.

Remember, $X \rightarrow Y$ implies $X \rightarrow\rightarrow Y$ , so 4NF is more stringent than BCNF.

Decompose R , using 4NF violation $X \rightarrow\rightarrow Y$, into XY and $X \cup ( R - Y )$.
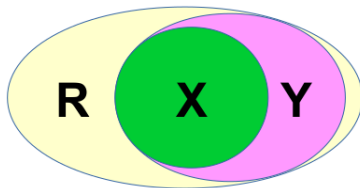
# Fourth Normal Form : Example -1

Consider the Following Example :

Drinkers(name, addr, phones, beersLiked)

   FD: name $\rightarrow$ addr

   Nontrivial MVD's: name $\rightarrow\rightarrow$ phones and name $\rightarrow\rightarrow$ beersLiked .

   Only key:   name, phones, beersLiked

All three dependencies above violate 4NF.

Successive decomposition yields 4NF relations:

- $D_1$ (name, addr)
- $D_2$ (name, phones)
- $D_3$ (name, beersLiked)

# Fourth Normal Form : Example -2

An MVD X →→ Y in R is called a trivial MVD, if
    (a) Y is a subset of X, or
    (b) X∪Y=R.

For Eg: the relation **EMP_PROJECTS** has the trivial MVD Ename→→Pname.

An MVD that satisfies neither (a) nor (b) is called anontrivial MVD.

A trivial MVD will hold in any relation state r of R; it is called trivial because it does not specify any significant or meaningful constraint on R.

**EMP**

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

**EMP_PROJECTS**

| Ename | Pname |
|-------|-------|
| Smith | X |
| Smith | Y |

**EMP_DEPENDENTS**

| Ename | Dname |
|-------|-------|
| Smith | John |
| Smith | Anna |
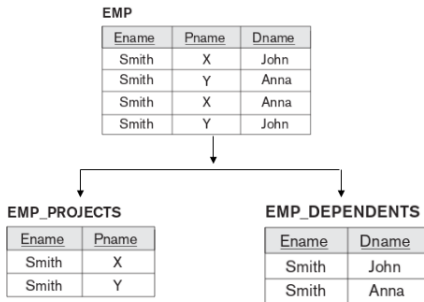
# Fourth Normal Form Definition

A relation 'R' is in fourth normal form iff it is in BCNF and contains no MVD's

Multivalued Dependency: A type of functional dependency where the determinant can determine more than one value. More formally, there are 3 criteria: There must be at least 3 attributes in the relation R(A, B, C)

- Given A, one can determine multiple values of B.
- Given A, one can determine multiple values of C.
- B and C are independent of one another.

Example-1 : Consider the Below Relation 'R'

| StudentID | Major | Activities |
|-----------|-------|------------|
| 100 | CIS | BaseBall |
| 100 | CIS | VolleyBall |
| 100 | Accounting | BaseBall |
| 100 | Accounting | VolleyBall |
| 200 | Marketing | Swimming |

**FD1: StudentID →→ Major**
**FD2: StudentID →→ Activities**

# Fourth Normal Form : Example-2

**Example-2 : Consider the Below Relation 'R'**

| Portfolio ID | Stock Fund | Bond Fund |
|---|---|---|
| 999 | Janus Fund | Municipal Bonds |
| 999 | Janus Fund | Dreyfus Short-Intermediate Municipal Bond Fund |
| 999 | Scudder Global Fund | Municipal Bonds |
| 999 | Scudder Global Fund | Dreyfus Short-Intermediate Municipal Bond Fund |
| 800 | Kaufmann Fund | T. Rowe Price Emerging Markets Bond Fund |

```
PortfolioID  ⟶  Stock Fund
PortfolioID  ⟶  Bond Fund
```

**R Decomposes into R₁ and R₂**

**R$_1$(Portfolio Id, Stock Fund)**

| Portfolio ID | Stock Fund |
|---|---|
| 999 | Janus Fund |
| 999 | Scudder Global Fund |
| 888 | Kaufmann Fund |

**R$_2$ (Portfolio Id, Bond Fund)**

| Portfolio ID | Bond Fund |
|---|---|
| 999 | Municipal Bonds |
| 999 | Dreyfus Short-Intermediate Municipal Bond Fund |
| 888 | T. Rowe Price Emerging Markets Bond Fund |

# Lossless Dependency - 5NF (Fifth Normal Form)

It is also Called as "Projection Join" Normal form.

There are certain conditions under which after decomposing a relation 'R', it cannot be reassembled back into its original form I.e Lossless Decomposition is not possible.

A Relation 'R' is said to be in 5NF if it is in 4NF and contains no redundant values/contains no join dependencies.

The process of converting the table into 5NF is as follows:

- Remove the join dependency.
- Break the database table into smaller and smaller tables to remove all data redundancy.

Definition :
Let 'R' be a relation, and 'A, B, C, .... Z' be subsets of the attributes of R.
Then, We say that R satisfies the Join Dependency – * A, B, C, ..... , Z
(pronounced as "star A,B,C,....Z" ) iff, every legal value of 'R' is equal to the join
of its projections on A,B,C,..... Z.

Consider for Eg :
R = SHIPMENTS(Supplier_Name, Part_Number, Project_Number)

| SHIPMENTS | | |
|---|---|---|
| Supplier_Number | Part_Number | Project_Number |
| S1 | P1 | J2 |
| S1 | P2 | J1 |
| S2 | P1 | J1 |
| S1 | P1 | J1 |

Now R is decomposed into $R_1 \rightarrow SP, R_2 \rightarrow PJ, R_3 \rightarrow JS$

# 5NF - Example

| SHIPMENTS | | |
|---|---|---|
| **Supplier_Number** | **Part_Number** | **Project_Number** |
| $S_1$ | $P_1$ | $J_2$ |
| $S_1$ | $P_2$ | $J_1$ |
| $S_2$ | $P_1$ | $J_1$ |
| $S_1$ | $P_1$ | $J_1$ |

**$R_1 \rightarrow$ SP**

| Supplier_ Number | Part_ Number |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S2 | P1 |

**$R_2 \rightarrow$ PJ**

| Part_ Number | Project_ Number |
|---|---|
| P1 | J2 |
| P2 | J1 |
| P1 | J1 |

**$R_3 \rightarrow$ JS**

| Project_ Number | Supplier _Number |
|---|---|
| J2 | S1 |
| J1 | S1 |
| J1 | S2 |
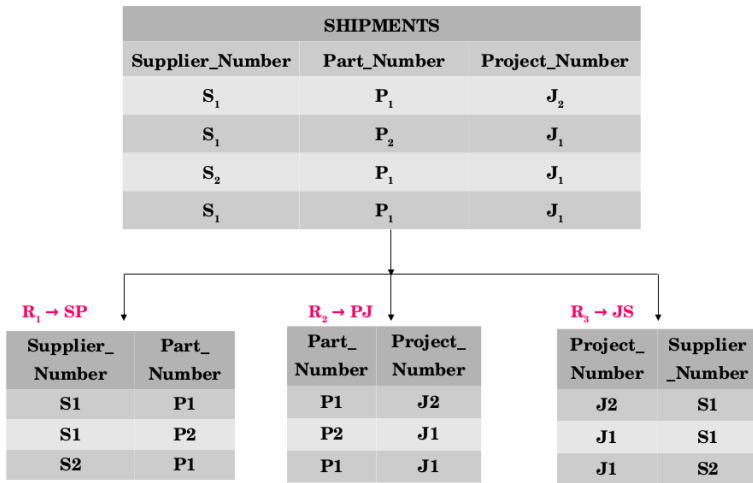
**Now the relations $R_1$, $R_2$ and $R_3$ are in 4NF, because no MVDs.**

| Supplier_ Number | Part_ Number |
|---|---|
| S1 | P1 |
| S1 | P2 |
| S2 | P1 |

⋈

| Part_ Number | Project_ Number |
|---|---|
| P1 | J2 |
| P2 | J1 |
| P1 | J1 |

Joining the
Relations $R_1$ and $R_2$
over Part_Number

| SHIPMENTS | | |
|---|---|---|
| Supplier_ Number | Part_ Number | Project_ Number |
| $S_1$ | $P_1$ | $J_2$ |
| $S_1$ | $P_2$ | $J_1$ |
| $S_2$ | $P_1$ | $J_1$ |
| $S_1$ | $P_1$ | $J_1$ |
| $S_2$ | $P_1$ | $J_2$ |

Spurious Tuple

**SHIPMENTS**

| Supplier_ Number | Part_ Number | Project_ Number |
|---|---|---|
| $S_1$ | $P_1$ | $J_2$ |
| $S_1$ | $P_2$ | $J_1$ |
| $S_2$ | $P_1$ | $J_1$ |
| $S_1$ | $P_1$ | $J_1$ |
| $S_2$ | $P_1$ | $J_2$ |

⋈

| Project_ Number | Supplier _Number |
|---|---|
| J2 | S1 |
| J1 | S1 |
| J1 | S2 |

Joining the
Relations $R_{12}$ and $R_3$
over Part_Number,
Supplier_Number

**SHIPMENTS**

| Supplier_Number | Part_Number | Project_Number |
|---|---|---|
| $S_1$ | $P_1$ | $J_2$ |
| $S_1$ | $P_2$ | $J_1$ |
| $S_2$ | $P_1$ | $J_1$ |
| $S_1$ | $P_1$ | $J_1$ |

# 3D Constraint

3D of SHIPMENTS could be more fundamental of Time-independent property. (A property satisfied by all legal values of the relation)

I.e if the relation satisfies a certain time-independent integrity constraint.

The relation SHIPMENTS is equal to the join of its **3** projections SP, PJ and JS is precisely equivalent to the following Statements

| If the pair | $(s_1 , p_1)$ | appears in SP |
| And the pair | $(p_1 , j_1)$ | appears in PJ |
| And the pair | $(j_1 , s_1)$ | appears in JS |
| The the triple | $(s_1 , p_1 , j_1)$ | appears in SHIPMENTS (Join($SP,PJ,JS$)) |

# 5NF (Join Dependency) - Example

Consider the Relation CTL (Courses_Tutor_Level)

**CTL**

| Course | Tutor | Level |
|--------|-------|-------|
| Databases | M. Ursu | Level3 |
| Databases | M. Marman | Level2 |
| Programming | M. Ursu | Level2 |
| Databases | M. Ursu | Level2 |

**CT**

| Course | Tutor |
|--------|-------|
| Databases | M. Ursu |
| Databases | M. Harman |
| Programming | M. Ursu |

**TL**

| Tutor | Level |
|-------|-------|
| M. Ursu | Level3 |
| M. Harman | Level2 |
| M. Ursu | Level2 |

**LC**

| Course | Level |
|--------|-------|
| Databases | Level3 |
| Databases | Level2 |
| Programming | Level2 |

# 5NF (Join Dependency) - (CT ⋈ TL)

The Join of CT and TL is as Follows ⋈

join(CT, TL)

| Course | Tutor | Level |
|---|---|---|
| Databases | M. Ursu | Level3 |
| Databases | M. Ursu | Level2 |
| Databases | M. Marman | Level2 |
| Programming | M. Ursu | Level3 |
| Programming | M. Ursu | Level2 |

Extra!

## Check 3D Constraint

| IF | tutor t1 teaches subject s1 |
|---|---|
| AND | level l1 studies subject s1 |
| AND | tutor t1 teaches level l1 |
| THEN | tutor t1 teaches subject s1 for level l1 |

# 5NF (Join Dependency) - (CT ⋈ TL) ⋈ LC

Now Join(Join(CT,TL), LC)

join(CT, TL)

| Course | Tutor | Level |
|---|---|---|
| Databases | M. Ursu | Level3 |
| Databases | M. Ursu | Level2 |
| Databases | M. Marman | Level2 |
| Programming | M. Ursu | Level3 |
| Programming | M. Ursu | Level2 |

Extra!

⋈

**LC**

| Course | Level |
|---|---|
| Databases | Level3 |
| Databases | Level2 |
| Programming | Level2 |

**CTL**

| Course | Tutor | Level |
|---|---|---|
| Databases | M. Ursu | Level3 |
| Databases | M. Marman | Level2 |
| Programming | M. Ursu | Level2 |
| Databases | M. Ursu | Level2 |