# Data Normalisation Process

Asst. Prof. Tracy Almeida e Aguiar

# Introduction to data normalization and normal forms

## Message

Message to create awareness among students before online classes.

1. **Physical distancing:** A distance of at least 6 feet should be maintained between any two individuals.

2. **Mask:** Everyone should ensure to wear a face cover/mask at all times in public places. Try to wear a mask with at least two layers or a three ply mask. Make sure your face is covered properly at all times in public places.

3. **Hand hygiene practice**: Everyone must regularly and thoroughly wash hands for 60 seconds with soap and water/liquid soap solution or clean them with a hand sanitizer (containing 60–80% alcohol).

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

# Problems Without 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 let us take an example of a **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 `office_tel` is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

## 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**.

## Updation Anomaly

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.

## 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.

# Normalization Rule

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form

# First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 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.

## 1st Normal Form (1NF)

In this Normal Form, we tackle the problem of atomicity. Here atomicity means values in the table should not be further divided. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.

| Employee ID | Employee Name | Phone Number | Salary |
|---|---|---|---|
| 1EDU001 | Alex | +91 8553206126<br>+91 9449424949 | 60,131 |
| 1EDU002 | Barry | +91 8762989672 | 48,302 |
| 1EDU003 | Clair | +91 9916255225 | 22,900 |
| 1EDU004 | David | +91 6363625811<br>+91 8762055007 | 81,538 |

In the above table, we can clearly see that the `Phone Number` column has two values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.

| Employee ID | Employee Name | Phone Number | Salary |
|---|---|---|---|
| 1EDU001 | Alex | +91 8553206126 | 60,131 |
| 1EDU001 | Alex | +91 9449424949 | 60,131 |
| 1EDU002 | Barry | +91 8762989672 | 48,302 |
| 1EDU003 | Clair | +91 9916255225 | 22,900 |
| 1EDU004 | David | +91 6363625811 | 81,538 |
| 1EDU004 | David | +91 8762055007 | 81,538 |

By this, we have achieved atomicity and also each and every column have unique values.

# What is functional dependency?

- **Functional Dependency** is a relationship that exists between multiple attributes of a relation.
- This concept is given by **E. F. Codd.**
- Functional dependency represents a formalism on the infrastructure of relation.
- It is a type of constraint existing between various attributes of a relation.
- It is used to define various normal forms.
- These dependencies are restrictions imposed on the data in database.
- If P is a relation with A and B attributes, a functional dependency between these two attributes is represented as $\{A \rightarrow B\}$. It specifies that,

| | |
|---|---|
| A | It is a determinant set. |
| B | It is a dependent attribute. |
| $\{A \rightarrow B\}$ | A functionally determines B.<br>B is a functionally dependent on A. |

- Each value of A is associated precisely with one B value. A functional dependency is trivial if B is a subset of A.
- 'A' Functionality determines 'B' $\{A \rightarrow B\}$ (Left hand side attributes determine the values of Right hand side attributes).

**For example:** <Employee> Table

| EmpId | EmpName |
| --- | --- |
| | |

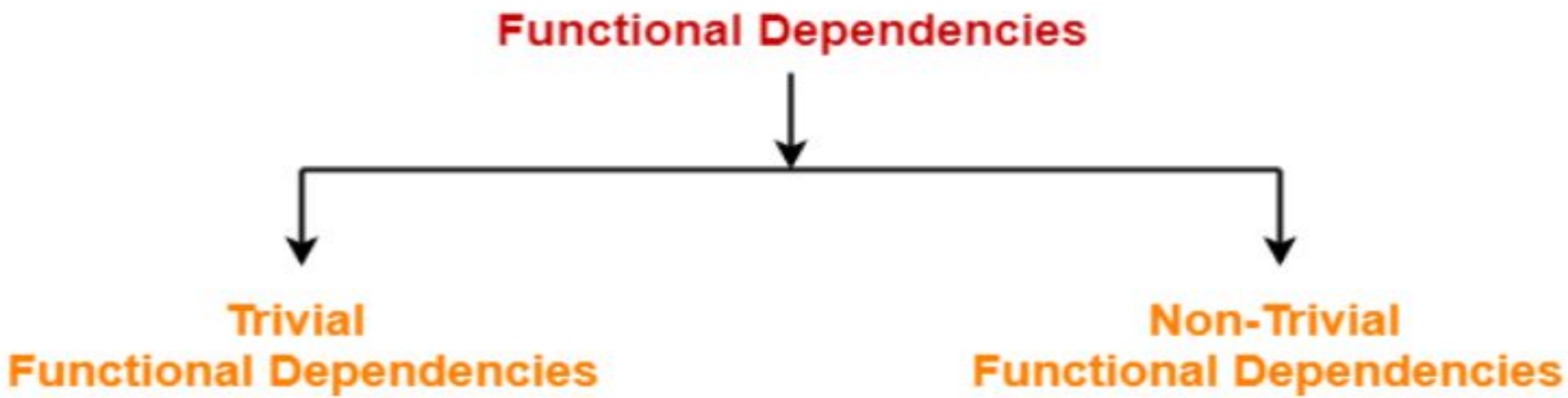
- In the above <Employee> table, EmpName (employee name) is functionally dependent on EmpId (employee id) because the EmpId is unique for individual names.
- The EmpId identifies the employee specifically, but EmpName cannot distinguish the EmpId because more than one employee could have the same name.
- The functional dependency between attributes eliminates the repetition of information.
- It is related to a candidate key, which uniquely identifies a tuple and determines the value of all other attributes in the relation.

## Advantages of Functional Dependency

- Functional Dependency avoids data redundancy where same data should not be repeated at multiple locations in same database.
- It maintains the quality of data in database.
- It allows clearly defined meanings and constraints of databases.
- It helps in identifying bad designs.
- It expresses the facts about the database design.

# Types Of Functional Dependencies-

There are two types of functional dependencies-




1. Trivial Functional Dependencies
2. Non-trivial Functional Dependencies

## 1. Trivial Functional Dependencies-

- A functional dependency $X \rightarrow Y$ is said to be trivial if and only if $Y \subseteq X$.
- Thus, if RHS of a functional dependency is a subset of LHS, then it is called as a trivial functional dependency.

### Examples-

The examples of trivial functional dependencies are-

- $AB \rightarrow A$
- $AB \rightarrow B$
- $AB \rightarrow AB$

## 2. Non-Trivial Functional Dependencies-

- A functional dependency $X \rightarrow Y$ is said to be non-trivial if and only if $Y \not\subset X$.
- Thus, if there exists at least one attribute in the RHS of a functional dependency that is not a part of LHS, then it is called as a non-trivial functional dependency.

### Examples-

The examples of non-trivial functional dependencies are-

- $AB \rightarrow BC$

# Introduction to Axioms Rules

- Armstrong's Axioms is a set of rules.
- It provides a simple technique for reasoning about functional dependencies.
- It was developed by William W. Armstrong in 1974.
- It is used to infer all the functional dependencies on a relational database.

# Various Axioms Rules

### A. Primary Rules

| | |
|---|---|
| **Rule 1** | **Reflexivity**<br>If A is a set of attributes and B is a subset of A, then A holds B. { A → B } |
| **Rule 2** | **Augmentation**<br>If A hold B and C is a set of attributes, then AC holds BC. {AC → BC}<br>It means that attribute in dependencies does not change the basic dependencies. |
| **Rule 3** | **Transitivity**<br>If A holds B and B holds C, then A holds C.<br>If {A → B} and {B → C}, then {A → C}<br>A holds B {A → B} means that A functionally determines B. |

**B. Secondary Rules**

| | |
|---|---|
| **Rule 1** | **Union**<br>If A holds B and A holds C, then A holds BC.<br>If{A → B} and {A → C}, then {A → BC} |
| **Rule 2** | **Decomposition**<br>If A holds BC and A holds B, then A holds C.<br>If{A → BC} and {A → B}, then {A → C} |
| **Rule 3** | **Pseudo Transitivity**<br>If A holds B and BC holds D, then AC holds D.<br>If{A → B} and {BC → D}, then {AC → D} |

**Sometimes Functional Dependency Sets are not able to reduce if the set has following properties,**

1. The Right-hand side set of functional dependency holds only one attribute.
2. The Left-hand side set of functional dependency cannot be reduced, it changes the entire content of the set.
3. Reducing any functional dependency may change the content of the set.

A set of functional dependencies with the above three properties are also called as **Canonical or Minimal.**

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.

The following are the types −

## Lossless Decomposition

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example −

**<EmpInfo>**

| Emp_ID | Emp_Name | Emp_Age | Emp_Location | Dept_ID | Dept_Name |
|---|---|---|---|---|---|
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Decompose the above table into two tables:

**<EmpDetails>**

| Emp_ID | Emp_Name | Emp_Age | Emp_Location |
|---|---|---|---|
| E001 | Jacob | 29 | Alabama |
| E002 | Henry | 32 | Alabama |
| E003 | Tom | 22 | Texas |

**<DeptDetails>**

| Dept_ID | Emp_ID | Dept_Name |
|---|---|---|
| Dpt1 | E001 | Operations |
| Dpt2 | E002 | HR |
| Dpt3 | E003 | Finance |

Now, Natural Join is applied on the above two tables −

The result will be −

| Emp_ID | Emp_Name | Emp_Age | Emp_Location | Dept_ID | Dept_Name |
|---|---|---|---|---|---|
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Therefore, the above relation had lossless decomposition i.e. no loss of information.

# Lossy Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

Let us see an example –

**<EmpInfo>**

| Emp_ID | Emp_Name | Emp_Age | Emp_Location | Dept_ID | Dept_Name |
|---|---|---|---|---|---|
| E001 | Jacob | 29 | Alabama | Dpt1 | Operations |
| E002 | Henry | 32 | Alabama | Dpt2 | HR |
| E003 | Tom | 22 | Texas | Dpt3 | Finance |

Decompose the above table into two tables −

**<EmpDetails>**

| Emp_ID | Emp_Name | Emp_Age | Emp_Location |
|---|---|---|---|
| E001 | Jacob | 29 | Alabama |
| E002 | Henry | 32 | Alabama |
| E003 | Tom | 22 | Texas |

**<DeptDetails>**

| Dept_ID | Dept_Name |
|---|---|
| Dpt1 | Operations |
| Dpt2 | HR |
| Dpt3 | Finance |

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.

Therefore, the above relation has lossy decomposition.

# Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

## 2. Dependency Preservation

- Dependency is an important constraint on the database.
- Every dependency must be satisfied by at least one decomposed table.
- If $\{A \rightarrow B\}$ holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

## 3. Lack of Data Redundancy

- Lack of Data Redundancy is also known as a **Repetition of Information.**
- The proper decomposition should not suffer from any data redundancy.
- The careless decomposition may cause a problem with the data.
- The lack of data redundancy property may be achieved by Normalization process.

# Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

## 2nd Normal Form (2NF)

The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate key determines a non-prime attribute. To understand in a better way lets look at the below example.

Consider the table

| Employee Id | Department Id | Office Location |
|---|---|---|
| 1EDU001 | ED-T1 | Pune |
| 1EDU002 | ED-S2 | Bengaluru |
| 1EDU003 | ED-M1 | Delhi |
| 1EDU004 | ED-T3 | Mumbai |

This table has a composite primary key `Emplyoee ID, Department ID`. The non-key attribute is `Office Location`. In this case, `Office Location` only depends on `Department ID`, which is only part of the primary key. Therefore, this table does not satisfy the second Normal Form.

To bring this table to Second Normal Form, we need to break the table into two parts. Which will give us the below tables:

| Employee Id | Department Id |
|---|---|
| 1EDU001 | ED-T1 |
| 1EDU002 | ED-S2 |
| 1EDU003 | ED-M1 |
| 1EDU004 | ED-T3 |

| Department Id | Office Location |
|---|---|
| ED-T1 | Pune |
| ED-S2 | Bengaluru |
| ED-M1 | Delhi |
| ED-T3 | Mumbai |

As you can see we have removed the partial functional dependency that we initially had. Now, in the table, the column `Office Location` is fully dependent on the primary key of that table, which is `Department ID`.

Now that we have learnt 1st and 2nd normal forms lets head to the next part of this Normalization in SQL article.

# Third Normal Form (3NF)

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

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

## 3rd Normal Form (3NF)

The same rule applies as before i.e, the table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes. That means non-prime attributes (which doesn't form a candidate key) should not be dependent on other non-prime attributes in a given table. So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z (where it is not the case that Y → X)

Let's understand this more clearly with the help of an example:

| Student Id | Student Name | Subject Id | Subject | Address |
|---|---|---|---|---|
| 1DT15ENG01 | Alex | 15CS11 | SQL | Goa |
| 1DT15ENG02 | Barry | 15CS13 | JAVA | Bengaluru |
| 1DT15ENG03 | Clair | 15CS12 | C++ | Delhi |
| 1DT15ENG04 | David | 15CS13 | JAVA | Kochi |

In the above table, `Student ID` determines `Subject ID`, and `Subject ID` determines `Subject`. Therefore, `Student ID` determines `Subject` via `Subject ID`. This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

Now in order to achieve third normal form, we need to divide the table as shown below:

| Student Id | Student Name | Subject Id | Address |
|---|---|---|---|
| 1DT15ENG01 | Alex | 15CS11 | Goa |
| 1DT15ENG02 | Barry | 15CS13 | Bengaluru |
| 1DT15ENG03 | Clair | 15CS12 | Delhi |
| 1DT15ENG04 | David | 15CS13 | Kochi |

| Subject Id | Subject |
|---|---|
| 15CS11 | SQL |
| 15CS13 | JAVA |
| 15CS12 | C++ |
| 15CS13 | JAVA |

As you can see from the above tables all the non-key attributes are now fully functional dependent only on the primary key. In the first table, columns `Student Name`, `Subject ID` and `Address` are only dependent on `Student ID`. In the second table, `Subject` is only dependent on `Subject ID`.

# Boyce and Codd Normal Form (BCNF)

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

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

**Boyce Codd Normal Form (BCNF)**

This is also known as 3.5 NF. Its the higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomalies which were not dealt with 3NF.

Before proceeding to BCNF the table has to satisfy 3rd Normal Form.

In BCNF if every functional dependency **A** $\rightarrow$ **B**, then **A** has to be the **Super Key** of that particular table.

Consider the below table:

| Student ID | Subject | Professor |
|---|---|---|
| 1DT15ENG01 | SQL | Prof. Mishra |
| 1DT15ENG02 | JAVA | Prof. Anand |
| 1DT15ENG02 | C++ | Prof. Kanthi |
| 1DT15ENG03 | JAVA | Prof. Anand |
| 1DT15ENG04 | DBMS | Prof. Lokesh |

- One student can enrol for multiple subjects.
- There can be multiple professors teaching one subject
- And, For each subject, a professor is assigned to the student

In this table, all the normal forms are satisfied except BCNF. Why?

As you can see `Student ID,` and `Subject` form the primary key, which means the `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.

Now in order to satisfy the BCNF, we will be dividing the table into two parts. One table will hold `Student ID` which already exists and newly created column `Professor ID`.

| Student ID | Professor ID |
|---|---|
| 1DT15ENG01 | 1DTPF01 |
| 1DT15ENG02 | 1DTPF02 |
| 1DT15ENG02 | 1DTPF03 |
| : | : |

And in the second table, we will have the columns `Professor ID`, `Professor` and `Subject`.

| Professor ID | Professor | Subject |
|---|---|---|
| 1DTPF01 | Prof. Mishra | SQL |
| 1DTPF02 | Prof. Anand | JAVA |
| 1DTPF03 | Prof. Kanthi | C++ |
| : | : | : |

By doing this we are satisfied the Boyce Codd Normal Form.

# Fourth Normal Form (4NF)

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

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

## What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation `R(A,B,C)`, if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Below we have a college enrolment table with columns `s_id`, `course` and `hobby`.

| s_id | course | hobby |
|---|---|---|
| 1 | Science | Cricket |
| 1 | Maths | Hockey |
| 2 | C# | Cricket |
| 2 | Php | Hockey |

As you can see in the table above, student with `s_id` **1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

You must be thinking what problem this can lead to, right?

Well the two records for student with `s_id` **1**, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

| s_id | course | hobby |
| --- | --- | --- |
| 1 | Science | Cricket |
| 1 | Maths | Hockey |
| 1 | Science | Hockey |
| 1 | Maths | Cricket |

And, in the table above, there is no relationship between the columns `course` and `hobby`. They are independent of each other.

So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

---

# How to satisfy 4th Normal Form?

To make the above relation satify the 4th normal form, we can decompose the table into 2 tables.

**CourseOpted Table**

| s_id | course |
|---|---|
| 1 | Science |
| 1 | Maths |
| 2 | C# |
| 2 | Php |

And, **Hobbies Table**,

| s_id | hobby |
|---|---|
| 1 | Cricket |
| 1 | Hockey |
| 2 | Cricket |
| 2 | Hockey |

Now this relation satisfies the fourth normal form.

A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

# Fifth Normal Form (5NF)

A database is said to be in 5NF, if and only if,

- It's in 4NF
- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

A relation is in DKNF when insertion or delete anomalies are not present in the database. Domain-Key Normal Form is the highest form of Normalization. The reason is that the insertion and updation anomalies are removed. The constraints are verified by the domain and key constraints.

A table is in Domain-Key normal form only if it is in 4NF, 3NF and other normal forms. It is based on constraints –

## Domain Constraint

Values of an attribute had some set of values, for example, EmployeeID should be four digits long –

| EmpID | EmpName | EmpAge |
|---|---|---|
| 0921 | Tom | 33 |
| 0922 | Jack | 31 |

## Key Constraint

An attribute or its combination is a candidate key

## General Constraint

Predicate on the set of all relations.

Every constraint should be a logical sequence of the domain constraints and key constraints applied to the relation. The practical utility of DKNF is less.

# Thank you