Normalization in Relational Databases

Normalization in relational databases is a design process that minimizes data redundancy and avoids **insertion/update/deletion anomalies**. Basically, you want each piece of information to be stored exactly once; if the information changes, you only must update it in one place.

What is Normalization?

- **Normalization is the process of organizing the data in the database.**
- **Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.**
- **Normalization divides the larger table into smaller and links them using relationships.**
- **The normal form is used to reduce redundancy from the database table.**

Types of Normal Forms:

Rule	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

First Normal Form (1NF)

A relation is in **first normal form** (1NF) if (and only if):

1. Each attribute contains only one value.
2. All attribute values are atomic, which means they can’t be broken down into anything smaller.

In practice, 1NF means that you should not have lists or other composite structures as attribute values.

Example-1. Below is an example of a relation that does not satisfy 1NF criteria:

Student courses

student	courses
Jane Smith	Databases, Mathematics
John Lipinsky	English Literature, Databases
Dave Beyer	English Literature, Mathematics

This relation is not in 1NF because the *courses* attribute has multiple values. Jane Smith is assigned to two courses (Databases and Mathematics), and they are stored in one field as a comma-separated list. This list can be broken down into smaller elements (i.e. course subjects: databases as one element, mathematics as another), so it's not an atomic value.

To transform this relation to the first normal form, we should store each course subject as a single value, so that each student-course assignment is a separate tuple:

Student courses

student	course
Jane Smith	Databases
Jane Smith	Mathematics
John Lipinsky	English Literature
John Lipinsky	Databases
Dave Beyer	English Literature
Dave Beyer	Mathematics

If you're interested in reading more about the first normal form, I recommend the article What Is the Actual Definition of First Normal Form? by my colleague Konrad Zdanowski.

Second Normal Form (2NF)

A relation is in **second normal form** (2NF) if and only if:

1. It is in 1NF.
2. No non-prime attributes are functionally dependent on a subset of the candidate key(s). In other words, **any column that's not a key column is dependent on the whole information in the candidate key**.

What does this mean? If the value of attribute A is determined by the value of attribute S, then *A* is functionally dependent on S. For example, your age is functionally dependent on your date of birth.

Let's go back to the idea of candidate keys and non-prime attributes. What are they?

- A **candidate key** is a minimal set of attributes that determines the other attributes included in the relation. It's minimal in that if you removed one attribute, the remaining attributes do not form a candidate key.
- A **non-prime attribute** is an attribute that is not part of the candidate key. However, for a relation to be 2NF, the information stored by non-prime attributes must be related to the whole information in the candidate key.

Informally, the second normal form states that **all attributes must depend on the entire candidate key**.

EXAMPLE – 1 . Let's see an example of a relation that does not satisfy 2NF. The underlined attributes are the candidate key.

Bike parts warehouse

part	supplier	quantity	supplier country
Saddle	Bikeraft	10	USA
Brake lever	Tripebike	5	Italy
Top tube	UpBike	3	Canada
Saddle	Tripebike	8	Italy

- The **candidate key** is the *part* and *supplier* set, which is expressed like this *{part, supplier}*.
- The **non-prime attributes** (which are not part of the candidate key) are *quantity* and *supplier country*.

- There are **functional dependencies** between *part, supplier,* and *quantity* (expressed as *part, supplier → quantity)* and between *supplier* and *supplier country (*expressed as *supplier → supplier country)*.

Why doesn't this satisfy 2NF? The set *{part, supplier}* is the only candidate key of this relation. The value of *supplier country* is functionally dependent on *supplier*. *Supplier country* is not part of the candidate key, so it is a non-prime attribute and it is functionally dependent on **part** of the candidate key, not the entire candidate key *{part, supplier}*.

To transform this relation into 2NF, we need to split it into two relations: **Bike parts** (with the attributes *part*, *supplier*, and *quantity*) and **Suppliers** (with the attributes *supplier* and *supplier country*). This would look like as follows:

Bike parts

part	supplier	quantity
Saddle	Bikeraft	10
Brake lever	Tripebike	5
Top tube	UpBike	3
Saddle	Tripebike	8

The relation **Bike parts** is in 2NF because, as before, the *quantity* attribute depends on the pair *supplier* and *part*.

Suppliers

supplier	supplier country
Bikeraft	USA
Tripebike	Italy
UpBike	Canada

The **Suppliers** relation is in 2NF because *supplier country* is functionally dependent on *supplier*, which is the candidate key of this relation.

EXAMPLE – 2 . Let's see one more example of a non-2NF relation.

Student course fees

student	course	grade	course fee
Alison Brown	Databases	A	$100
Jason Liu	Mathematics	B	$150
Mariah Hill	Databases	B+	$100

- **Candidate key:** *{student, course}*
- **Non-prime attributes:** *grade, course fee*
- **Functional dependencies:** *student, course → grade; course → course fee*

The following relation does not satisfy 2NF. The set *{student, course}* is the relation's candidate key, but the value of *course fee* is functionally dependent on *course* alone. *Course fee* is a non-prime attribute, which is functionally dependent on only **part** of the candidate key.

Solution: To transform this into 2NF, we again split it into two relations: **Student courses** (with the attributes *student, course*, and *grade*) and **Courses** (with the attributes *course* and *course fee*). Thus, we avoid the partial dependency in the non-2NF relation above.

Student course

student	course	grade
Alison Brown	Databases	A
Jason Liu	Mathematics	B
Mariah Hill	Databases	B+

Courses

course	course fee
Databases	$100
Mathematics	$150

Note that the 2NF partial dependency rule only kicks in if your relation has a composite candidate key (i.e. one that consists of multiple attributes). **All relations that have a single-attribute key are by definition in 2NF.**

EXAMPLE – 3 . Let's see one more example of a non-2NF relation.

Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

Solution: In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF. To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

A relation is in **third normal form** (3NF) if and only if:

1. It is in second normal form (2NF).
2. All non-prime attributes are directly (non-transitively) dependent on the entire candidate key.

In other words, non-prime attributes must be functionally dependent on the key(s), but they must not depend on another non-prime attribute. **3NF non-prime attributes depend on "nothing but the key".**

EXAMPLE – 1. Let's see a non-3NF relation:

Order information

order_id	date	customer	customer email
1/2020	2020-01-15	Jason White	white@example.com
2/2020	2020-01-16	Mary Smith	msmith@mailinator.com
3/3030	2020-01-17	Jacob Albertson	jasobal@example.com
4/2020	2020-01-18	Bob Dickinson	bob@fakemail.com

- **Candidate key:** *order_id*
- **Non-prime attributes:** *date, customer, customer email*
- **Functional dependencies:** *date* depends on *order_id (order_id → date)*; *customer* depends on *order_id (order_id → customer)*, and *customer email* depends on *customer (customer → customer email)*.

This relation does not satisfy 3NF. The only candidate key in this relation is *order_id*. The value of *customer email* is functionally dependent on the *customer* attribute, which is a non-prime attribute. Thus, the relation violates 3NF.

Once again, we split this into two relations: **Orders** (with the attributes *order_id*, *date*, and *customer*) and **Customers** (with the attributes *customer* and *customer email*):

Orders

order_id	date	customer
1/2020	2020-01-15	Jason White
2/2020	2020-01-16	Mary Smith
3/3030	2020-01-17	Jacob Albertson
4/2020	2020-01-18	Bob Dickinson

Customers

customer	customer email
Jason White	white@example.com
Mary Smith	msmith@mailinator.com
Jacob Albertson	jasobal@example.com
Bob Dickinson	bob@fakemail.com

Orders is in 3NF because the *date* and *customer* attributes do not violate the rule of 3NF; their values depend on the *order_id* number. **Customers** is in 3NF because *customer email* is functionally dependent on *customer*, which is the candidate key of this relation. In both cases, **all non-prime attributes depend on the candidate key**.

EXAMPLE – 2. Let's see one more non-3NF example.

Courses

course	year	teacher	teacher date of birth
Databases	2019	Chris Cape	1974-10-12
Mathematics	2019	Daniel Parr	1985-05-17
Databases	2020	Jennifer Clock	1990-06-09

- **Candidate key:** *{course, year}*

- **Non-prime attributes:** *teacher, teacher date of birth*
- **Functional dependencies:** *teacher* depends on *course* and *year (course, year → teacher)*; *teacher date of birth* depends on *teacher (teacher → teacher date of birth)*

This relation does not satisfy 3NF. The only candidate key in this relation is *{course, year}*, but the value of *teacher date of birth* is functionally dependent on *teacher* – a non-prime attribute. This violates 3NF.

Guess how we'll transform this into 3NF? That's right; we split the relation. **Courses** gets the attributes *course, year,* and *teacher*; **Teachers** gets the attributes *teacher* and *teacher date of birth*:

Courses

course	year	teacher
Databases	2019	Chris Cape
Mathematics	2019	Daniel Parr
Databases	2020	Jennifer Clock

Teachers

teacher	teacher date of birth
Chris Cape	1974-10-12
Daniel Parr	1985-05-17
Jennifer Clock	1990-06-09

EXAMPLE – 3. Let's see one more non-3NF example.

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Database Normalization: Summary

First, second, and third normal forms are the basic normal forms in database normalization:

- The first normal form (1NF) states that each attribute in the relation is atomic.
- The second normal form (2NF) states that non-prime attributes must be functionally dependent on the entire candidate key.
- The third normal form (3NF) states that non-prime attributes must be directly (non-transitively) dependent on candidate keys.