



Money Transfer Application

Normalization

Indexing

Information Retrieval

Database Normalization

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

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 Rule

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



First Normal Form (1NF)

1. Rule 1: Single Valued Attributes

- a. Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

2. Rule 2: Attribute Domain should not change

- a. This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

For example: If you have a column `dob` to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

3. Rule 3: Unique name for Attributes/Columns

- a. This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data. If one or more columns have same name, then the DBMS system will be left confused.

4. Rule 4: Order doesn't matters

- a. This rule says that the order in which you store the data in your table doesn't matter.

First Normal Form (1NF)

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

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



Second Normal Form (2NF)

1. Rule 1: Table should be in the First Normal Form

2. Rule 2: There should be no Partial Dependency

- a. Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.
- b. To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.



Second Normal Form (2NF)

Partial Dependency

Subjects Table

subject_id	subject_name
1	Java
2	C++
3	Php

Scores

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

Teacher is partially dependent



Second Normal Form (2NF)

Removing Partial Dependency

Subjects		
subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

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



Third Normal Form (3NF)

1. Rule 1: It should be in the Second Normal form
2. Rule 2: And it should not have Transitive Dependency
 - a. Advantage of removing Transitive Dependency
 - i. Amount of data duplication is reduced
 - ii. Data integrity achieved



Third Normal Form (3NF)

Students 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

Subjects Table

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

Scores Table

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

Third Normal Form (3NF)

score_id	student_id	subject_id	marks	exam_name	total_marks

With **exam_name** and **total_marks** added to our Score table, it saves more data now. Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → **student_id + subject_id**.

Our new column **exam_name** depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Practical exams and for some you don't. So we can say that **exam_name** is dependent on both **student_id** and **subject_id**.

And what about our second new column **total_marks**? Does it depend on our Score table's primary key?

Well, the column **total_marks** depends on **exam_name** as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

But, **exam_name** is just another column in the score table. It is not a primary key or even a part of the primary key, and **total_marks** depends on it.

This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

Third Normal Form (3NF) Resolution

Scores Table: In 3rd Normal Form

score_id	student_id	subject_id	marks	exam_id

The new Exams table

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

Boyce-Codd Normal Form

Fourth Normal Form

Homework

<https://www.studytonight.com/dbms/database-normalization.php>



Indexing

Every time your web application runs a database query containing a WHERE statement, the database server's job is to look through all the rows in your table to find those that match your request. As the table grows, an increasing number of rows need to be inspected each time.

Indexes solve this problem in exactly the same way as the index in a reference book, by taking data from a column in your table and storing it alphabetically in a separate location called an index. The same process can be applied to all data types, for example numeric data will be stored in numeric order and dates in date order.

By doing this, the database does not need to look through every row in your table, instead it can find the data you are searching for alphabetically, then skip immediately to look at the row(s) where the data is located.



Indexing

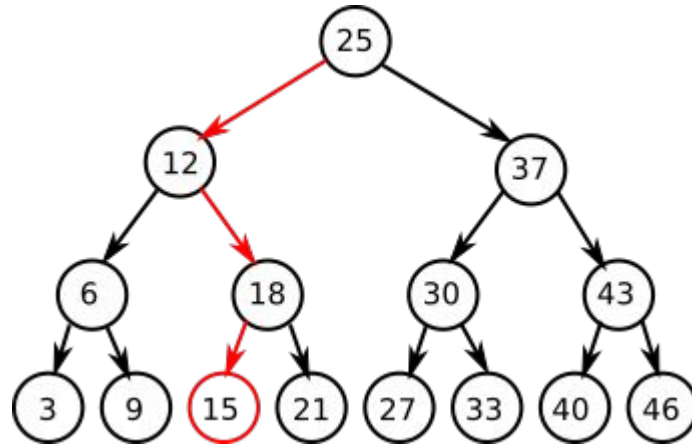
Every time your web application runs a database query containing a WHERE statement, the database server's job is to look through all the rows in your table to find those that match your request. As the table grows, an increasing number of rows need to be inspected each time.

Indexes solve this problem in exactly the same way as the index in a reference book, by taking data from a column in your table and storing it alphabetically in a separate location called an index. The same process can be applied to all data types, for example numeric data will be stored in numeric order and dates in date order.

By doing this, the database does not need to look through every row in your table, instead it can find the data you are searching for alphabetically, then skip immediately to look at the row(s) where the data is located.



B Tree Indexing



Indexing Example

Students Table


ID	First Name	Last Name	Class
1	James	Bond	6A
2	Chris	Smith	6A
3	Jane	Eyre	6B
4	Dave	Smith	6B

```
CREATE TABLE `students` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `first_name` varchar(255) DEFAULT  
NULL,  
  `last_name` varchar(255) DEFAULT NULL,  
  `class` varchar(255) DEFAULT NULL,  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB
```

Frequently Used Queries

```
1.  SELECT * FROM students WHERE id = 1
2.  SELECT * FROM students WHERE last_name = 'Smith'
3.  SELECT * FROM students WHERE class = '6A' AND last_name = 'Smith'
```

```
1.  Primary key already has a key
2.  CREATE INDEX by_last_name ON students (`last_name`);
3.  CREATE INDEX by_class_and_last_name ON students (class, last_name);
```



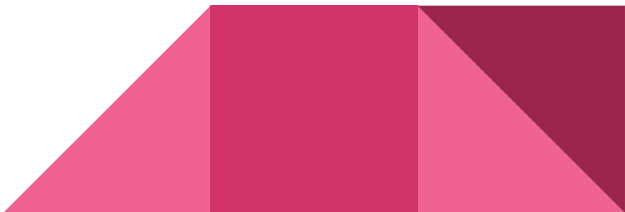
Indexing on Joins

Grades Table

ID	student_id	Timestamp	Grade
1	1	2014-01-20 15:00:00	A+
2	1	2014-02-20 15:00:00	A-

```
SELECT * FROM grades WHERE student_id = 1
```

```
CREATE INDEX by_student_id ON grades (student_id);
```



Information Retrieval

1. Selecting All Data
2. Selecting Particular Rows
3. Selecting Particular Columns
4. Sorting Rows
5. Date Calculations
6. Working with NULL Values
7. Pattern Matching
8. Counting Rows
9. Using More Than one Table



References

- <http://sql-ex.ru/>
- <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>
- <https://www.studytonight.com/dbms/database-normalization.php>
- <https://dev.mysql.com/doc/refman/8.0/en/retrieving-data.html>

