Normalization

Prof. Mehdi Pirahandeh
Inha University

인하대학교

Normalization

Normalization is the process of creating a better data model. This is accomplished with adjustments to the model that reduce duplicate values. and avoid anomalies.

Duplicate data occurs when the same data items are stored at multiple locations in the database.

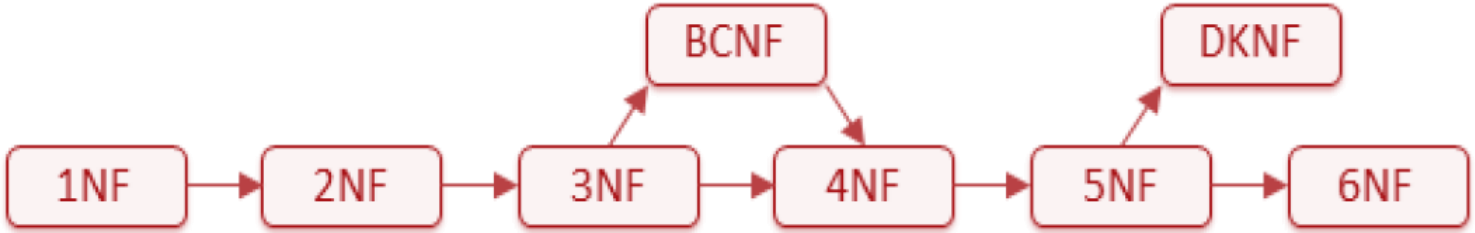
Anomalies are logical inconsistencies in the data.

The term *de-normalization* which is the opposite of normalization and this usually involves the removal of columns, tables, and relations, with the goal of making the data model more accessible and easier to manage.

Normalization Forms

There are a number of formal normalization rules that will help you in the process of creating a better model.

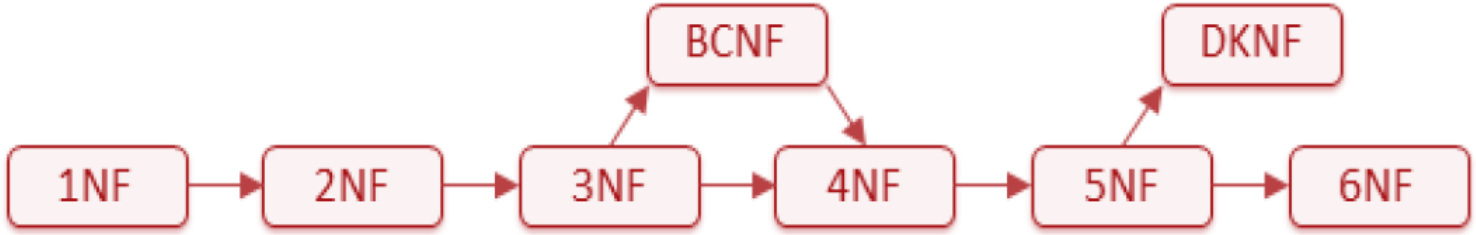
These rules are called Normal Forms and there are eight of them: 1NF, 2NF, 3NF, 4NF, 5NF, 6NF, BCNF, and DKNF. Their progression is depicted in the next figure:



Boyce Codd Normal Form (BCNF) Domain Key Normal Form (DKNF).

- A data model can be in any of the above Normal Forms.
- The higher the number the more normalized the data model and the less data redundancy there is.
- Normal forms are *additive*, meaning that when a model satisfies a normal form it also satisfies all preceding forms.
- For example, a model in 3NF is also in 1NF and 2NF.

Normalization Forms



Be aware that normal forms apply to tables and not to the database as a whole. You may hear modelers say that there database is in 3NF but what they mean to say is that all tables are in 3NF.

It is entirely possible to have some tables in 1NF and others in 5NF. In reality, it is best to aim for consistency and have most or all tables in the same normal form.

Finding the right balance between normalization and denormalization is captured nicely by this one-liner popular in data modeling circles,

"*normalize until it hurts, denormalize until it works*"

funny, but accurate at the same time.

First Normal Form (1NF)

- The aim of the 1NF is to eliminate repeating groups of data. A table is in 1NF when columns contain atomic values coming from a single domain. This means three things: a column cannot have repeated data, columns cannot be repeated, and a column cannot contain data from more than one domain.
- You may recall that we've seen violations of these before. First, repeated data in a column is a *value smell*. And, second, repeated columns in a table are a *column smell*. The 1NF formalizes the solution to these problems. Let's look at an example.
- Suppose we're starting off with the following table.

StudentId	FirstName	LastName	Company	City	Course
22	Sally	Weber	Tech Inc	New York	C# Introduction

First Normal Form (1NF)

- Each student record has a place to store the course they are taking. Clearly, this is a problem, because there are students that attend more than one course. A naïve solution would be to add courses by concatenating them to the other courses in the same Course column, like so:

StudentId	FirstName	LastName	Company	City	Course
22	Sally	Weber	Tech Inc	New York	C# Introduction; Advanced C#; Intro to LINQ

- Courses are separated by a semicolon, which makes it near-impossible for SQL to get to a course. You probably recognize the value smell.
- Another naïve solution would be to add additional course columns, like so:

StudentId	FirstName	LastName	Company	City	Course1	Course2	Course3
22	Sally	Weber	Tech Inc	New York	C# Introduction	Advanced C#	Intro to LINQ

You will probably recognize this as a repeated *column smell*. What if someone is taking more than 3 courses? Or what if you need a query that determines how many students enrolled in the Advanced C# course?

First Normal Form (1NF)

- These are all problematic.
- A better approach is to *normalize* the model and create a new Course table and store the courses attended in it:

Student

StudentId	FirstName	LastName	Company	City
22	Sally	Weber	Tech Inc	New York

These two tables have a one-to-many relationship in which a student can attend any number of courses. This exactly captures the real-world situation we are trying to model. We're beginning to see the value of normalization. Although abstract, normalization is highly relevant to solving everyday modeling challenges .

StudentCourse

StudentId	CourseId	CourseTitle
22	1	C# Introduction
22	2	Advanced C#
22	3	Intro to LINQ

- *Tables without repeating values or repeating columns are in 1NF.*

Second Normal Form (2NF)

The 2NF also eliminates redundant data but in this case it applies to rows with a composite key. We will just briefly touch on 2NF with an example.

A model satisfies 2NF when all non-key columns are fully dependent on the whole primary key. If there is a dependency on only part of the key then these elements need to be moved to their own table. Consider a situation in which we have a Student and StudentCourse table! To correct this we need to normalize the table by moving the columns that are partially dependent to their own table, in this case Course. Here is what the normalized model looks like:

Student

StudentId	FirstName	LastName	Company	City
30	Ann	Weatherford	Starbucks	Seattle

Course

CourseId	CourseTitle
4	VB Introduction

StudentCourse

StudentId	CourseId
30	4

These tables are now in 2NF. The StudentCourse table is now an intersecting table representing the fact that a student can take any course, and a course can be taken by any student. It still has a ***composite key***, StudentId, and CourseId, but there are no non-key columns that are dependent on only part of its key.

- *Create tables in which all non-key columns are fully dependent on the primary key.*
- *Tables with a surrogate primary key are automatically in 2NF.*

Third Normal Form (3NF)

The aim of the 3NF is to eliminate columns that are not functionally dependent on the primary key. 3NF is achieved when columns that don't belong in the table are moved to another table. The purpose of this rule is to make sure that columns are placed in the proper tables by examining their dependency to the primary key and the record as a whole. Consider the following table in which students and their companies are maintained:

StudentId	FirstName	LastName	Company	City
17	Craig	Hollister	IBM	New York
63	Louis	Jensen	Boeing	Seattle
15	George	McNeil	Boeing	Seattle
90	Sean	Hendrix	Boeing	Houston

StudentId is the primary key and all others are non-key columns. Of these, FirstName and LastName are dependent of the primary key as each StudentId represents a unique student. On the other hand, Company and City are not dependent on the student. What's more, they are repeated for all students coming from the same company. This increases the risks of anomalies, such as in the example below (there is no Boeing company in Houston).

Third Normal Form (3NF)

This problem is solved by moving company related columns into their own table.

Student

StudentId	FirstName	LastName	CompanyId
17	Craig	Hollister	1
63	Louis	Jensen	2
15	George	McNeil	2
90	Sean	Hendrix	2

Company

CompanyId	Company	City
1	IBM	New York
2	Boeing	Seattle

Normalization has reduced data redundancy and removed the risk of data anomalies You probably understand now why we stated that the lower normal forms are *'obvious*' because an experienced data modeler would never consider creating the pre-normalized models in the prior sections.

There is no real need to explicitly concern yourself whether a data model is in 1NF, 2NF, or 3NF because if they weren't, you would intuitively know that the model is not optimal.

o Create tables with columns that are fully dependent on the primary key.
o Tables in 3NF are coherent entities with reduced risk of data anomalies.

Other Normal Forms

BCNF

- *Create tables in which all key columns are fully dependent on the other key columns.*
- Tables with a surrogate primary key are automatically in BCNF.

Database

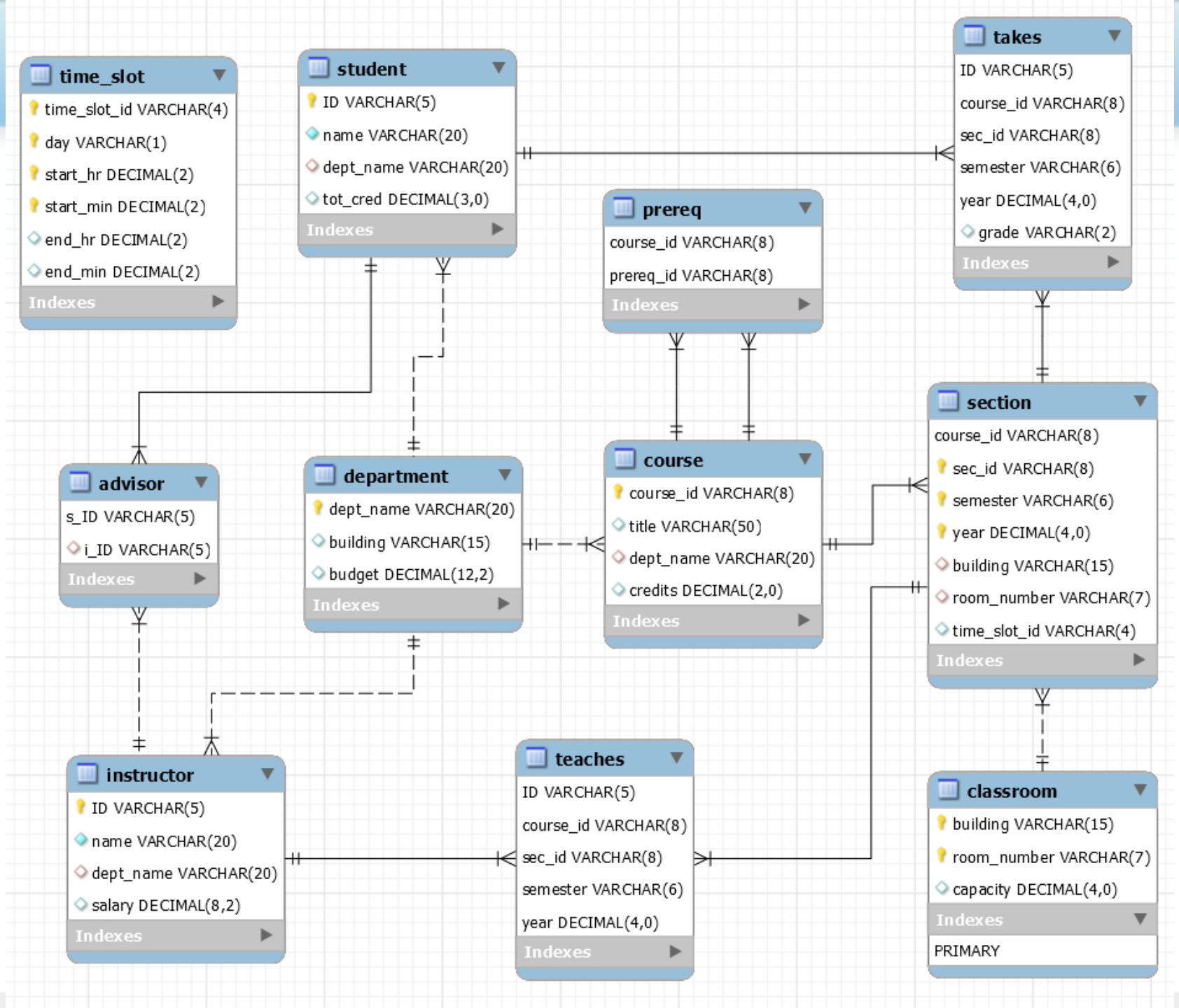
Prof. Mehdi Pirahandeh

Inha University

Email: *mehdi@inha.ac.kr*

인하대학교

ERD diagram


time_slot
time_slot_id VARCHAR(4)
day VARCHAR(1)
start_hr DECIMAL(2)
start_min DECIMAL(2)
end_hr DECIMAL(2)
end_min DECIMAL(2)
Indexes
student
ID VARCHAR(5)
name VARCHAR(20)
dept_name VARCHAR(20)
tot_cred DECIMAL(3,0)
Indexes
takes
ID VARCHAR(5)
course_id VARCHAR(8)
sec_id VARCHAR(8)
semester VARCHAR(6)
year DECIMAL(4,0)
grade VARCHAR(2)
Indexes
prereq
course_id VARCHAR(8)
prereq_id VARCHAR(8)
Indexes
section
course_id VARCHAR(8)
sec_id VARCHAR(8)
semester VARCHAR(6)
year DECIMAL(4,0)
building VARCHAR(15)
room_number VARCHAR(7)
time_slot_id VARCHAR(4)
Indexes
advisor
s_ID VARCHAR(5)
i_ID VARCHAR(5)
Indexes
department
dept_name VARCHAR(20)
building VARCHAR(15)
budget DECIMAL(12,2)
Indexes
course
course_id VARCHAR(8)
title VARCHAR(50)
dept_name VARCHAR(20)
credits DECIMAL(2,0)
Indexes
instructor
ID VARCHAR(5)
name VARCHAR(20)
dept_name VARCHAR(20)
salary DECIMAL(8,2)
Indexes
teaches
ID VARCHAR(5)
course_id VARCHAR(8)
sec_id VARCHAR(8)
semester VARCHAR(6)
year DECIMAL(4,0)
Indexes
classroom
building VARCHAR(15)
room_number VARCHAR(7)
capacity DECIMAL(4,0)
Indexes
PRIMARY

Lab 2: Intermediate SQL

- Topics you should know:
  - Join Expressions
  - Views
  - Transactions

Exercise 1: 20 minutes

1. Display a list of all instructors, showing their ID, name, and the number of sections that they have taught.
   - Make sure to show the number of sections as 0 for instructors who have not taught any section.
   - Your query should use an outerjoin, and should not use scalar subqueries.

Sample Output:

ID	name	Number of sections
14365	Lembr	2
15347	Bawa	1
16807	Yazdi	0
19368	Wieland	3
22591	DAgostino	13
25946	Liley	1
28097	Kean	2
28400	Atanassov	2
31955	Moreira	0
3199	Gustafsson	4
3335	Bourrier	2
34175	Bondi	3
35579	Soisalon-Soininen	0
36897	Morris	5
37687	Arias	0

Exercise 2: 20 minutes

1. Write the same query as above, but using a scalar subquery, without outerjoin.

Sample Output:

ID	name	Number of sections
14365	Lembr	2
15347	Bawa	1
16807	Yazdi	0
19368	Wieland	3
22591	DAgostino	13
25946	Liley	1
28097	Kean	2
28400	Atanassov	2
31955	Moreira	0
3199	Gustafsson	4
3335	Bourrier	2
34175	Bondi	3
35579	Soisalon-Soininen	0
36897	Morris	5
37687	Arias	0

Tasks

1. Display the list of all course sections offered in Spring 2010, along with the names of the instructors teaching the section.
   - If a section has more than one instructor, it should appear as many times in the result as it has instructors.
   - If it does not have any instructor, it should still appear in the result with the instructor name set to "—".
   - **TIP.** You can use **DECODE**
2. Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries.
   - Make sure to correctly handle departments with no instructors.
   - **TIP.** You can use **COUNT, GROUP BY**

THANK YOU