Normalization

Normalization is a process in which we refine the quality of the logical design. Some call it a form of evaluation or validation. Codd said normalization is a process of elimination, through which we represent the table in a preferred format. Normalization is also called “non-loss decomposition” because a table is broken into smaller components but **no information is lost**.

Are We Normal?

If all values in a table are single atomic values (this is first normal form, or 1NF) then our database is technically in “normal form” according to Codd—the “math” of set theory and predicate logic will work. However, we usually want to do more to get our DB into a preferred format. We will take our databases to at least 3rd normal form.

Why Do We Normalize?

1. We want the design to be easy to understand, with clear meaning and attributes in logical groups.

2. We want to avoid anomalies (data errors created on insertion, deletion, or modification)

    Suppose we have the table Nurse (SSN, Name, Unit, Unit Phone)

    Insertion anomaly: You need to insert the unit phone number every time you enter a nurse. What if you have a new unit that does not have any assigned staff yet? You can’t create a record with a blank primary key, so you can’t just leave nurse SSN blank. You would have no place to record the unit phone number until after you hire at least 1 nurse.

    Deletion anomaly: If you delete the last nurse on the unit you no longer have a record of the unit phone number.

    Modification anomaly: if the unit changes phone number you must change this number for every nurse! Lots of work, you must change them all and type correctly each time. If you miss even one – then you would have two different phone numbers for the same unit, and no way to know which is the correct number.

3. Avoid attributes that are frequently null (not as big an issue as in the past, as space is inexpensive, but good practice).

Background Concept: Functional Dependency

If a and b are attributes in table R, then a → b (b is functionally dependent on a) means that all rows with same value for a will ALWAYS also have same value for b. This is NOT necessarily reversible

SSN → Name
If we know SSN then we know Name (if we had a lookup table, we could look up name based on the SSN; each SSN would give only 1 name)

If we know Name we don’t know SSN (same name could be different SSN; e.g., all of George Foreman’s sons are also named George Foreman)

Composite Functional Dependency example

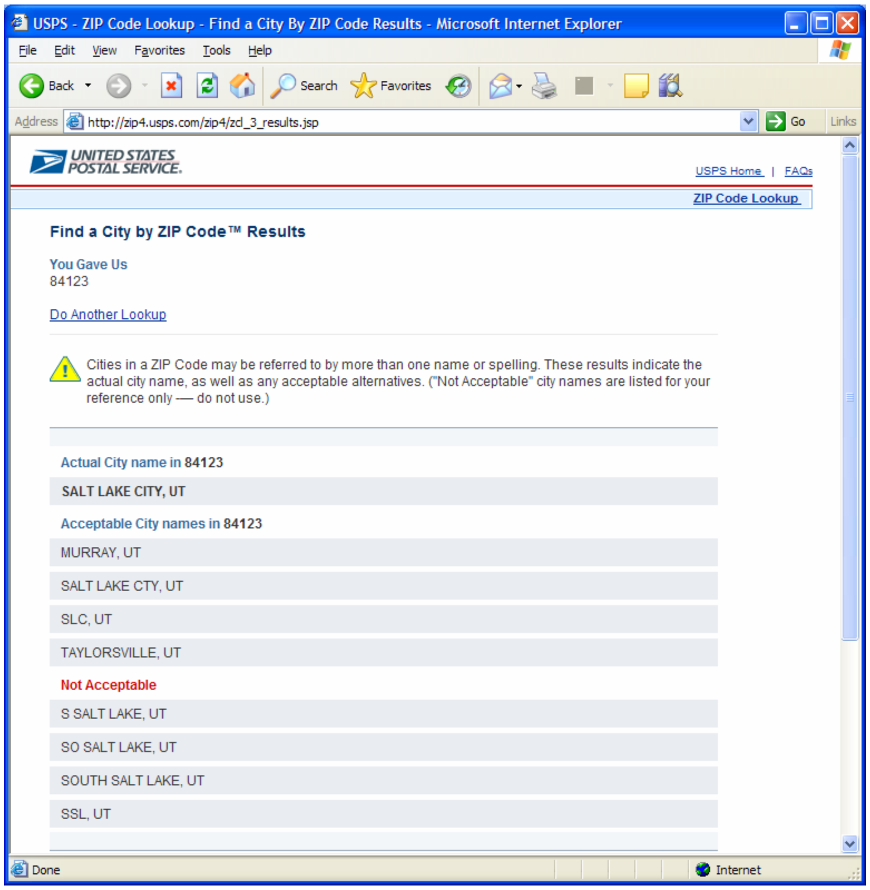
Zip → City, State

We will assume **for this course**, unless otherwise specified:

* Assumption: If an address is entered we will always know the zip code for the address

* Assumption: We only use the “default” post office city

In real situation these assumptions may not be true! A person may know their city but not their zip code, or they may use one of the “alternate city names” in their address. But for this course it’s a useful way to practice with composite functional dependency.


USPS - ZIP Code Lookup - Find a City By ZIP Code Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://zip4.usps.com/zip4/zcl_3_results.jsp Go Links
UNITED STATES POSTAL SERVICE.
USPS Home | FAQs
ZIP Code Lookup
Done
Internet

Find a City by ZIP Code™ Results

You Gave Us
84123

Do Another Lookup

Cities in a ZIP Code may be referred to by more than one name or spelling. These results indicate the actual city name, as well as any acceptable alternatives. ("Not Acceptable" city names are listed for your reference only — do not use.)

Actual City name in 84123

SALT LAKE CITY, UT

Acceptable City names in 84123

MURRAY, UT

SALT LAKE CTY, UT

SLC, UT

TAYLORSVILLE, UT

Not Acceptable

S SALT LAKE, UT

SO SALT LAKE, UT

SOUTH SALT LAKE, UT

SSL, UT

In this example, I looked up zip code 84123

The default (“actual”) city and state is
Salt Lake City, UT

We will assume for class exercises that this is the only choice. In reality there are a number of “acceptable” city names for this same zip code.

<u>The Process of Normalization</u>

Normalization is a formal technique for analyzing a table based on its primary key and functional dependencies between its attributes.

It is executed as a series of steps. Each step corresponds to a specific normal form, which has known properties. As normalization proceeds, relations become progressively more restricted in format (stronger and less vulnerable to update anomalies).

Normal Forms are rules or standards to which tables should conform. (Date says that normalization “formalizes common sense”). Other features of the higher normal forms are to minimize duplication and reduce risk of anomalies.

There are actually many standards for normalization. We will learn the most widely accepted rules. We will assume that most tables have only one candidate key, which is the primary key.

A table is in a specified normal form if it meets ALL the criteria of that normal form. The tables must meet criteria of "earlier" normal forms (can't be in 2NF unless it is also in 1NF…). **As set of tables is in a specified normal form only if ALL tables meet the criteria of that normal form.**

The normalization rules depend on relationships between "key" and "non-key" attributes. Primary key, alternate key (candidate keys), and foreign keys are "key" attributes. The rest are "non-key" attributes.

<u>Overview</u>
First Normal Form: single atomic values

Second Normal Form: when there is a composite PK, all non-key depend on the <u>entire</u> PK

Third Normal Form: all non-key attributes depend ONLY on the PK (no non-key/non-key dependency)

Fourth Normal Form: looks at multivalued attributes that are not related to eachother – basically asks if all attributes really belong in the same table

Fifth Normal Form: Only the PK plus one other attribute.

We will normalize our DB to 3rd NF (but may address grossly bad 4NF violations).

Now, what does all that really mean?

<u>**First Normal Form** (1NF, step 5)</u>
A table is in 1NF if each row and column intersection (cell) has only 1 value and that value is ATOMIC (It can't be further broken down). The domains (allowable values for the cells) consist of **single, atomic values**.

In checking 1NF, watch for two things
(1) Compound attributes: attributes such as name or address that need to be broken into smaller components.

> What to do: list the individual components as separate attributes.
>
> Name may or may not be considered atomic, it depends on your use. If you'll ever want just part of the name (e.g., sort by last name), then consider it at composite attribute and break it into smaller components, such as first name and last name. So in our SNDB, the demog table <u>should have</u> broken the name column into two columns – one for last name and one for first name. It would have allowed us to more easily search for specific people, without needing to use "like" and wildcards.
>
> For this course, please break name into at least first name and last name.

(2) Plural attributes: columns that may have more than one value. For example, a person's allergies.

> What to do: Create a new table that contains the PK from the original table, and the plural attribute. PK of the new table is BOTH the old PK plus the attribute.
>
> Example: Child (<u>ChildID</u>, Name, DOB, Allergies)   Allergies is plural. This becomes
>
> Child (<u>ChildID</u>, FirstName, LastName, DOB)
> ChildAllergy (<u>Child ID</u>, <u>Allergy</u>)

Try it. Put the list of attributes we created in step 4 into 1NF.

Customer (CustomerID, CompanyName, ContactName, ContactTitle, Address, Phone, Fax)
Employee (EmployeeID, Name, SSN, Title, BirthDate, Address, HomePhone, OfficeExtension, Note, *ManagerID*)
Order (OrderID, OrderDate, RequiredDate, ShippedDate, ShippedVia, *EmployeeID*, *CustomerID*)
Product (ProductID, ProductName, CategoryName, CategoryDescription, ReorderLevel, QuantityPerUnit)
ProductOrder (ProductID, OrderID, UnitPrice, Quantity)
SalaryHistory (EmployeeID, DateOfChange, HourlyRate, WithholdingStatus, EmplHireDate)

You should get something like what's on the next page. I will use bold and strike through to show my changes.

1NF (step 5) example
Assumption: Customer.ContactName is last name only

Customer (CustomerID, CompanyName, ContactName, ContactTitle, ~~Address,~~ **Street, City, State, Zip,** Phone, Fax)
Employee (EmployeeID, ~~Name~~, **LastName**, **FirstName**, SSN, Title, BirthDate, ~~Address~~, **Street, City, State, Zip,** HomePhone, OfficeExtension, Note, *Manager*)
Order (OrderID, OrderDate, RequiredDate, ShippedDate, ShippedVia, *EmployeeID*, *CustomerID*)
Product (ProductID, ProductName, CategoryName, CategoryDescription, ReorderLevel, QuantityPerUnit)
ProductOrder (ProductID, OrderID, UnitPrice, Quantity)
SalaryHistory (EmployeeID, DateOfChange, HourlyRate, ~~WithholdingStatus~~, **W4Marital, W4NumDep**, EmplHireDate)

Note that I broke up the name field in Employee, but not in Customer. That is based on the "Assumption" that customer.ContactName is last name only. The address in both tables was broken up into street address (like 10 S 2000 E), city, state, and zip. Withholding status is like "Married with 3 dependents" or "Single with no dependents", so I split this into the marital status and the number of dependents.

Second Normal Form (2NF, step 6)
Must be in 1NF AND
Every non-key column must depend on the entire primary key.

To test 2NF, we only need to examine tables with a composite primary key (two or more attributes in the PK). In our sample, we only care about the ProductOrder and SalaryHistory tables.

For example, consider the table below

Book (Author ID, Title, AuthorName, AuthorCity, BookGenre, #Pages, Publisher)

We need both attributes in the PK because it's possible that books by the different authors may have the same title. For example, there is a book named "Second chance" by Jane Green, and one by James Patterson.

Author name is a problem – the author isn't going to change their name every time they write a new book. The name is an attribute of the author, but not dependent on the book title

Author city is a problem. You would need to enter the author's city for every book title. Most authors do not move to a new city every time they write a book. AuthorCity is an attribute of the author, but not dependent on the book title

So we see that some of the attributes belong to the book, and depend on the full primary key of AuthorID and Title. Other attributes belong to the author regardless of what book.

To put into 2nd normal form, pull out the attributes that only depend on part of the original PK:
Book (Author ID, Title, #Pages, Publisher)
Author (Author ID, Author Name, Author City)

Hint: you might already have a table that uses the same primary key as what you are pulling out. See if the attributes make sense in that table instead of creating a new table.

Try it. Put the list of attributes we created in step 5 into 2NF.

2NF (step 6)   example

I copied the tables from 1NF and accepted all the changes. Then I made the changes for 2NF.
We only need to check the ProductOrder and SalaryHistory tables – these are the only ones with composite PKs.

Customer (CustomerID, CompanyName, ContactName, ContactTitle, Street, City, State, Zip, Phone, Fax)
Employee (EmployeeID, LastName, FirstName, SSN, Title, BirthDate, Street, City, State, Zip, HomePhone, OfficeExtension, **EmplHireDate**, Note, Manager)
~~EmployeeHire (EmployeeID, EmplHireDate)~~
Order (OrderID, OrderDate, RequiredDate, ShippedDate, ShippedVia, EmployeeID, CustomerID)
Product (ProductID, ProductName, CategoryName, CategoryDescription, ReorderLevel, QuantityPerUnit)
ProductOrder (ProductID, OrderID, UnitPrice, Quantity)
SalaryHistory (EmployeeID, DateOfChange, HourlyRate, W4Marital, W4NumDep, ~~EmplHireDate~~)

Note: UnitPrice is not a 2NF violation because the user requirements (executive summary) told us that it is negotiated with each order – so it needs the entire PK - both the order and the product.

I pulled out EmplHireDate as 2NF violation. According to the rules, I created a table specifically for this:
EmployeeHire (EmployeeID, EmplHireDate)

But I noticed that I already had a table that used EmployeeID as the primary key (Employee table). EmplHireDate made sense in that table, and it keeps my set of tables as simple as possible. So I put EmplHireDate in the Employee table instead.

Third Normal Form (3NF, step 7)
3NF - A relation that is in 1NF and 2NF and in which no non-key attribute is transitively dependent on the primary key (assumes PK is the only candidate key). *WHAT??????*

3NF is based on the concept of transitive dependency:
A, B and C are attributes of a relation such that if A $\rightarrow$ B and B $\rightarrow$ C,
then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C). The 3NF rule tells us that all non-keys must be dependent ONLY on the Primary Key.

Again, you may ask…*WHAT??????*

Basically, the 3NF rule means
No non-key attribute may depend on another non-key attribute

Third Normal Form Example
Book (Book Title, Type, Editor Name, Editor Phone)

Editor phone (non-key) depends on editor name (non-key). This causes all three types of anomaly
Insertion Anomaly: can't record the editor's phone until you have a book
Deletion Anomaly: Delete book - delete editor’s phone
Modification Anomaly: Change editor’s phone - have to make this change for all books by that editor

How I go about it.
You have to ignore all the key attributes (PK, alternate/candidate keys, and FK). Just look at the non-key attributes. **3NF only cares about non-key attributes**.

Take the first non-key attribute in the table. Ask yourself – if I know this, do I know any other non-key attribute? Then take the next non-key attribute in that table. “If I know this do I know any other non-key attribute?”.

Or you might picture the table filled with data, and ask – every time I have this non-key attribute, will I also have the same value in another non-key attribute?

(Think about our post office class assumption, discussed at the beginning of this document. City, state, and zip code are non-key items in an address record.

If I have “Salt Lake City” as the city, will I always have the same zip code? (NO, SLC has many zip codes).

If I have “Salt Lake City” as the city, will I always have the same state (OK, yes for Salt Lake City, but for other city names the same city name is in multiple states. So the answer to the generic question – if I know the city do I know the state – is NO. **Normalization rules are “all or none” – the rule must apply to all rows in the table**).

However, if I have 84123 for the zip code, I will always have “Salt Lake City” as the city and will always have “UT” as the state). According to our class assumptions, the zip code will not be null (we always know the zip code), and if we know the zip code then there is only 1 choice for city and state. For any given zip code, we would always have the same values for city and state.

So, how do you fix the redundancy and put the table in 3NF? Create a new table with the non-key attributes. The one that is **depended on** is the primary key of the new table. That attribute is also left as a foreign key in the original table. Delete the other attribute from the original table

So our book table

Book (Book Title, Type, Editor Name, Editor Phone)

becomes

Book (Book Title, Type, *Editor Name*)
Editors (Editor Name, Editor Phone)

What about times when you have alternate keys? For example, suppose you have

Patient (MRN, SSN, First, Last)

SSN determines First and Last. Is this a 3NF violation. No! Why? SSN and MRN are both *Candidate Keys* (either one could have been chosen as the PK). When we chose MRN as the PK, SSN becomes designated as an *alternate key*. **3NF only cares about non-key attributes** so it is NOT a 3NF violation!

The resolution to 3NF (creating a new table with the dependent attributes) creates a sort of "look up table". You will recognize a 3NF violation if you look at data, because you'll see redundant data in non-key columns. You'll start asking yourself – do I really want to re-type this every time when I could just look it up?

Try it. Put the list of attributes we created in step 6 into 3NF.

I copied the table schemas from 2NF and accepted all the changes. Then I made the changes for 3NF. We need to check the NON-KEY attributes. We can ignore PK, Alternate keys, and FK.

3NF (step 7) example
Assumptions: If an address is entered the user knows the zip code. Only "default city name" is allowed.

Customer (CustomerID, CompanyName, ContactName, ContactTitle, Street, ~~City, State~~, *Zip*, Phone, Fax)
Employee (EmployeeID, LastName, FirstName, SSN, Title, BirthDate, Street, ~~City, State~~, *Zip*, HomePhone, OfficeExtension, EmplHireDate, Note, *Manager*)
ZipCodes (Zip, City, State)
Order (OrderID, OrderDate, RequiredDate, ShippedDate, ShippedVia, EmployeeID, CustomerID)
Product (ProductID, ProductName, CategoryName, ~~CategoryDescription~~, ReorderLevel, QuantityPerUnit)
ProductCategory (CategoryName, CategoryDescription)
ProductOrder (ProductID, OrderID, UnitPrice, Quantity)
SalaryHistory (EmployeeID, DateOfChange, HourlyRate, W4Marital, W4NumDep)

Note: Employee.SSN is a candidate key so not a 3NF violation in Employee.

Notice that I created a zip codes table based on the assumptions above. If I know the zip code, I know the city and state – 84123 is always "Salt Lake City, UT". I only created ONE zipcodes table, even though I have two addresses. Both of them can use the same zip codes look up table.

ProductCategory makes sense as a lookup table as well. You might have a short category name and a lengthy description of the category. For every category name, you will have the same category description. You wouldn't want to retype the category description for every product in that category.

Boyce-Codd Normal Form (BCNF)

You can have difficulties with 3NF if you have ALL of the following:

- 2 or more candidate keys
- Candidate keys were composite
- Candidate keys share at least one attribute

Suppose you have two CK: (X, Y) and (Y, Z).

A table is in BCNF "If and only if the only determinants are candidate keys" (Date, 2000, p. 367). Candidate keys are always determinants (by definition). In BCNF there are no other determinants.

Difference between 3NF and BCNF is that for a functional dependency $A \rightarrow Y$, if A is a non-key attribute and Y is a primary-key attribute, 3NF allows this dependency (the rule only addresses non-key to non-key dependency, not non-key to key). Whereas, BCNF insists that for this dependency to remain in a table, A must be part of a candidate key.

Every table that is in BCNF is also in 3NF. However, a table may be in 3NF and still may not be in BCNF.

BCNF example
SupplierProduct (SupplierNum, SupplierName, ProductNum, Quantity)

Candidate Keys
(SupplierNum, ProductNum)
(SupplierName, ProductNum)

SupplierNum → SupplierName

To fix this:
SupplierProduct (<u>SupplierNum</u>, <u>ProductNum</u>, Quantity)
Supplier SupplierProduct (<u>SupplierNum</u>, SupplierName)

You will generally not have to worry about BCNF in this class. But it's in your readings. And if there's a big problem with it, I might have you fix it.

<u>Fourth Normal Form (4NF, step 8)</u>
In 3NF (or BCNF) and no **nontrivial** multi-valued dependencies.

Multi-valued dependencies are when you have three attributes (A, B, C)
A → B (B is multi-valued, so you need both A & B in the PK). Like a person's list of allergies…you need a table with both the personID and the allergy as PKs.

A → C (C is multi-valued so you need A & C in the PK). Like person and adm_dx; you need both the personID and the diagnosis.

B and C are independent of each other. Allergies and admitting diagnosis are unrelated (usually).

You would NOT want

AdmitInfo (<u>PersonID</u>, <u>allergy</u>, <u>admit_diagnoses</u>)

Because all attributes are part of the Primary Key, the above doesn't violate previous normal form rules. However, if you add a second allergy, you don't want to repeat the admitting diagnosis.

To fix: split into two tables

Allergies (<u>PersonID, allergy</u>)
Adm_dx (<u>PersonID, admit_diagnosis</u>)

4NF violations are easy to spot because the attributes are independent of each other. Usually as soon as you start to look at the data you'll think – "These things do not belong together in the same table".

4NF (step 8) example

Customer (<u>CustomerID</u>, CompanyName, ContactName, ContactTitle, Street, *Zip*, Phone, Fax)
Employee (<u>EmployeeID</u>, LastName, FirstName, SSN, Title, BirthDate, Street, *Zip*, HomePhone, OfficeExtension, EmplHireDate, Note, *Manager*)
ZipCodes (<u>Zip</u>, City, State)
Order (<u>OrderID</u>, OrderDate, RequiredDate, ShippedDate, ShippedVia, *EmployeeID*, *CustomerID*)
Product (<u>ProductID</u>, ProductName, CategoryName, ReorderLevel, QuantityPerUnit)
ProductCategory (<u>CategoryName</u>, CategoryDescription)
ProductOrder (<u>ProductID</u>, <u>OrderID</u>, UnitPrice, Quantity)
SalaryHistory (<u>EmployeeID</u>, <u>DateOfChange</u>, HourlyRate, ~~W4Marital, W4NumDep~~)
WithholdingHistory (<u>EmployeeID, DateOfChange</u>, W4Marital, W4NumDep)

Changes to withholding and changes to salary don't necessarily belong together. Your salary doesn't change if you add a new dependent (but don't we wish it would?)

Fifth Normal Form

Created to deal with complex, rare problems. Trade-offs: eliminates all redundancy of non-key attributes, but increases redundancy in keys.

The rule: break tables into smallest possible pieces – just the PK plus one other attribute.

Patient (MRN, Last, First, DOB) becomes

Patient Last Name (MRN, Last)
Patient First Name (MRN, First)
Patient Birth (MRN, DOB)

Almost never used in real life!

Denormalization

Sometimes, you intentionally violate a normalization rule (or return to a lower normal form). Often for performance reasons, sometimes because assumptions are found to be incorrect or only apply "sometimes".

Document the rationale for denormalizing.

Be aware that this can introduce redundancy and update problems.

Normalization Review – I remember the normal forms this way.

(sort of like "the truth, the whole truth, and nothing but the truth…")

The Key,	(1NF) – single atomic values
The Whole Key	(2NF) – all nonkey depend on the entire PK
And Nothing but the Key	(3NF) – no non-key/non-key dependency.

4NF: **Does it all belong together**? Is the PK correct?

5NF: **How small can I slice it**?