src_id	src_store_name	src_department_1	src_department_2	src_hq_location	src_loc_code	src_loc_name
1	Bameys	Mens Clothing	Footwear	Manhattan, NY	EM	Enclosed Mall
2	Macys	Housewares	Clothing	Philadelphia, PA	SM	Strip Mall
3	KitchenAide	Pots and Pans	Housewares	BentonHarbor, MI	P	Plaza
4	Apple	Computers	Phones	Cupertino, CA	SF	Store Front
5	Pottery Barn	Furniture	Lighting	SanFrancisco, CA	SF	Store Front
6	Clarke Shoes	NULL	Footwear	Waltham, MA	P	Plaza
7	Chicos	Womens Clothing	Footwear	FortMeyers, FL	EM	Enclosed Mall

PK FD M - FD M - FD C-FD FD TD - FD

Normalization

- The process of organizing existing data to
    - Improve performance of an existing DB
    - Design a new DB (from data you normalize)

Why Normalize data?

- Better questions for determining needs
- Accurate view of situation and scope of problems
- More detailed understanding of data types and granular requirements: check constraints, null, ‘lookup tables

Normalization- objective 1

- Eliminate redundancy
- Redundancy: reputation of related values in a table/data store
- Repeating values can, and do, occur in the product of SELECT statements where you are joining > 1 table, so it’s important to remember that the concept of eliminating redundancy applies to where the data is stored only, not the product of a query.

Implications of Redundant Data

Your client is managing this spreadsheet, entering each row of data ...

SUID	Name	Email	Enrolled in	Location
123456789	Arial Photo	aphoto@syr.edu	IST 359	Hinds 010
123456789	Arial Photo	aphoto@syr.edu	IST 387	Hinds 012
123456789	Arial Photo	aphoto@syr.edu	IST 429	Hinds 111
867530900	Isabelle Gunnering	igunner@syr.edu	IST 359	Hinds 010
987654321	Mike Rophone	mropho@syr.edu	IST 359	Hinds 010
987654321	Mike Rophone	mropho@syr.edu	IST 387	Hinds 012

User has to enter all information about each person every time they record an enrollment.

Redundancy → **Errors in Data Entry** → **Meaningless Data**

Normalization- objective 2

- Ensure functional dependence
- Within a table, a field must be uniquely determined by another field in the table
    - Usually its primary key, or anything that makes the record unique
- Each table should be about a specific topic and only supporting information about the topic should be included
- 1 NF- 1 PK- FD
- 2 NF- multiple PK- PD
    - Bridge tables
    - Many to many relationship

Illustrating Functional Dependence (FD)

SUID **PK**	Name **FD**	Email **FD**	Enrolled in	Location
123456789	Arial Photo	aphoto@syr.edu	IST 359	Hinds 010
123456789	Arial Photo	aphoto@syr.edu	IST 387	Hinds 012
123456789	Arial Photo	aphoto@syr.edu	IST 429	Hinds 111
867530900	Isabelle Gunnering	igunner@syr.edu	~~IST 359~~	~~Hinds 010~~
987654321	Mike Rophone	mropho@syr.edu	~~IST 359~~	~~Hinds 010~~
987654321	Mike Rophone	mropho@syr.edu	IST 387	Hinds 012
692994100	Sal Ladd	slad04@syr.edu	IST 359	Hinds 010

- **Does SUID uniquely define a student?**
  - *Yes, because when SUID changes, student name always changes.*
    - *Name is FUNCTIONALLY DEPENDENT (FD) on SUID*

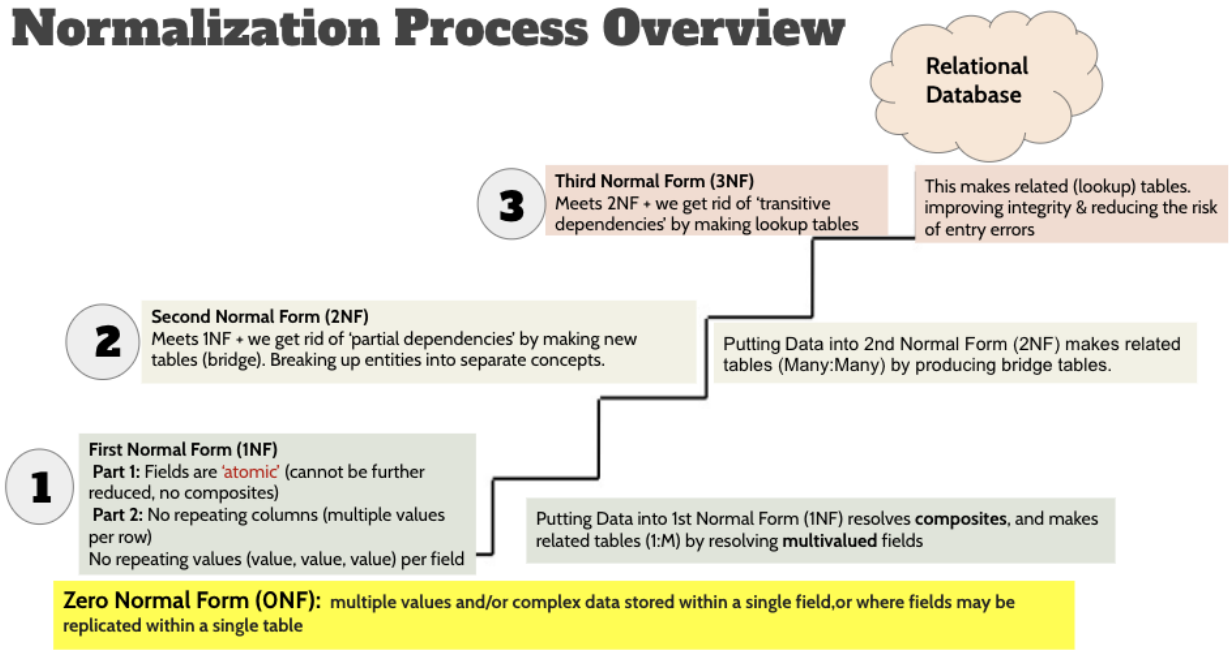
- **Does SUID uniquely define a student's email address?**
  - *Yes, because when the SUID changes, the email always changes*
    - *Email is FUNCTIONALLY DEPENDENT (FD) on SUID*

"Normal Forms"

- A state of the structure of data characterized by the amount of redundancy and unresolved functional dependencies
- Normalization is a data analysis process that progresses data through normal forms

ONF → 1NF → 2NF → 3NF

- As we reduce redundancy, and resolve dependencies
  - Increase in "normal forms"
  - This increases the number of tables in the database
  - Higher normal form= more related tables in database
- Tool used for that analysis process: dependency diagram
  - Dependency diagram- physical model (DDL)
  - Much of the time you can skip the ERD/ fact statement if you have enough data to normalize with



Eliminating atomicity

The Normalization Process: 1NF Part 1

Let's take 1st NF first: **Make your data 'atomic'**

Atomic: 1 value in each field that pertains ONLY to that field.

user_id	first_name	Last_name	email	address
bjscherr	Blythe	Scherrer	bjscherr@syr.edu	5594 East Lake Road CAZENOVIA NY 13244
dnosky	Deborah	Nosky	dlnosky@syr.edu	29311 Duwamish Lane Seattle WA 98101
mafudge	Michael	Fudge	mafudge@syr.edu	5981 Manyard Terrace Berkeley CA 65874

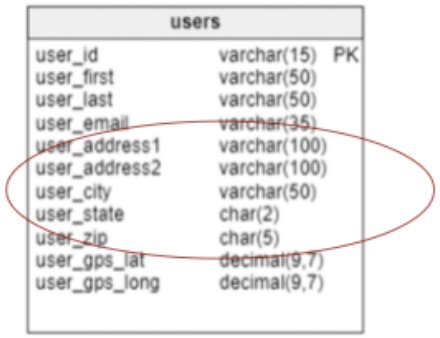
Arriving at 1NF: Field 'Atomicity'

user_id	first	Last	email	Address 1	City	State	Zip
bjscherr	Blythe	Scherrer	bjscherr@syr.edu	5594 East Lake Road	Cazenovia	NY	13035
dnosky	Deborah	Nosky	dlnosky@syr.edu	29311 Duwamish Lane	Seattle	WA	98101
mafudge	Michael	Fudge	mafudge@syr.edu	5981 Manyard Terrace	Berkeley	CA	65987

Better!
Each column is 'atomic'...

This is directly analogous to identifying and resolving COMPOSITE attributes into COLUMNS.

1 NF Part 1: Atomicity
How it looks in an actual model



When you split up composite fields into their multiple columns, you are resolving for atomicity (making your data atomic)

1

First Normal Form (1NF) Part I
1. Fields are 'atomic' (cannot be further reduced)

- diagram is when we label the columns in our spreadsheet with symbols that help us analyze its structure
    - Header (or footer) of a data set

Eliminating multiple values

- In the example below, we have the same kind of information (all emails), but it repeats. So it's not the column that's the problem, it's the data inside it that

creates a problem. It still can't be handled atomically, because splitting out the columns makes REDUNDANT columns. So you made it atomic, but then you made it redundant so it's not in 1NF yet.

1 NF Part 2: Eliminating Multiple Values

user_id	first_name	Last_name	emails
5	Arial	Photo	aphoto@fudge.com, aphoto@gmail.com
6	Mike	Rophone	mroph@fudge.com, mroph@yahoo.com
7	Michael	Fudge	mafudge@fudge.com, fudge@hotmail.com
8	Old	School	No Email

If one user can have MANY emails, what kind of table relationship are we going to make?

1 NF Part 2: Eliminating Multivalued fields w/ child / fk tables

Original table: Users

	user_id	first_name	Last_name
PK	5	Arial	Photo
	6	Mike	Rophone
	7	Michael	Fudge
	8	Old	School

New Table: User_emails

Email_id	FK_user_id	email
101	FK 5	aphoto@fudge.com
102	5	aphoto@gmail.com
103	6	mroph@fudge.com
104	6	mroph@yahoo.com
105	7	mafudge@fudge.com
106	7	fudge@hotmail.com

new pk

(1) Each email (and a PK that you create) is taken into the new **user_emails** table

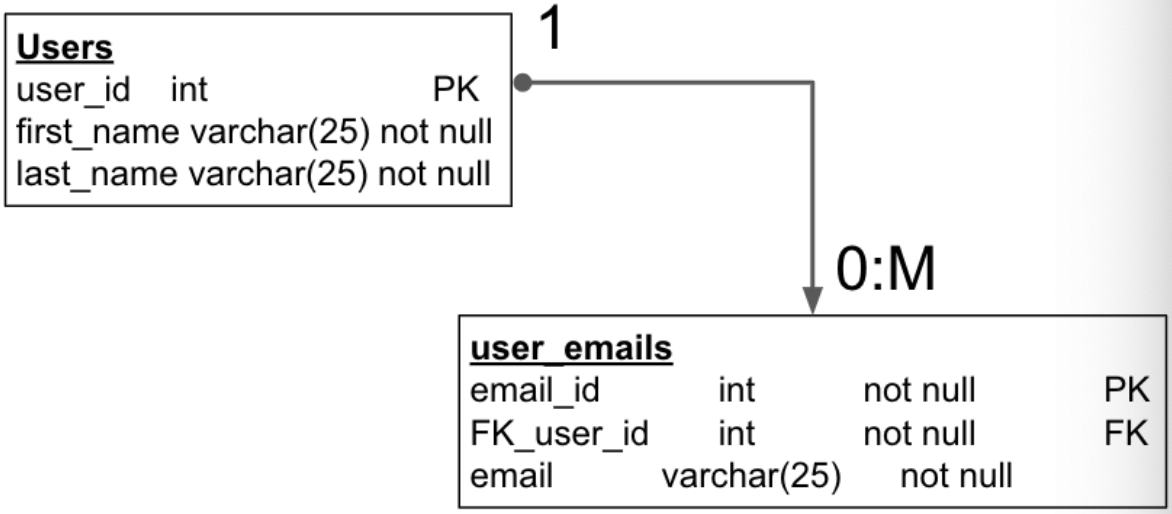
(2) The user_id field in the new Emails table becomes fk_user_id: the FK that references the PK back in the users table

(3) Translate this data structure into a physical model.

Cleaner design.
It's more efficient = more speedy
No wasted space or missing records.
You can do more with it= store info associated w/ each email

1 NF Part 2: Eliminating Multivalued Fields

How it's modeled



1NF Part 2 : Eliminating Multiple Values (Scenario 2)

M

user_id	first_name	Last_name	Email1	Email2	Email3
5	Arial	Photo	aphoto@fudge.com	aphoto@gmail.com	
6	Mike	Rophone	mroph@fudge.com	mroph@yahoo.com	mroph@Hotmail.com
7	Michael	Fudge	mafudge@fudge.com	fudge@hotmail.com	
8	Old	School			

1) Multiple blank cells (nulls) exist for every record. The database has to process these as if they contain data (time consuming)
2) SELECT statement to retrieve emails would need to call every email field, and pivot the data into a single column just to get one list of emails.
3) Which email is the one to use? With no way to know which email is for which purpose, my data isn't as powerful or informative as it could be.

Requirements for 1 NF:

1. All fields are atomic: resolve composites (addresses, names)
    a. Method: map the composite attribute so it's atomic columns
        i. Address- address line 1, line 2, city, state, zip, country
        ii. Name- first name, last name

b. Elimination of multi-valued fields: (field1, field2, field3 and/or multiple columns for same info)
    i. Method: make child table to hold multiple values pertaining to parent record (1:M relationship)
c. Resolve functional dependencies:
    i. Each table should be about a specific topic and only supporting information about that topic should be included

Arriving at 2NF

- Similar to resolving dependencies for 1 NF, but introduces a new kind of dependency: partial dependency
    - If a data source has 2 (or more) fields that makes a row unique, we identify those as 'key' fields.
    - Label them with PK1, PK2, etc….
    - Each 'non-key' is field is then tested to see which of the PKs it is dependent on
    - A field that is dependent on part of a candidate key is called a 'partial dependency'
        - Non-key field is wholly dependency, but on just one of the PKs

Step 2: Logical test against each key on non-key fields

Software Installed on IST Servers

PK1	PD1	PD1	PD1	PD1	PK2	PD2	PD2	
Serial Num	Server Name	Make	Model	Date Server Purchased	SWID	SW Title	Date Purchased	Date Installed
VNK334	www.ist	HP	Netserver LH4	8/1/2015	101	Windows 2010 Server	9/24/2015	11/8/2015
VNK334	www.ist	HP	Netserver LH4	8/1/2015	201	MS SQL Server 2016	9/24/2015	11/8/2015
VNK334	www.ist	HP	Netserver LH4	8/1/2015	302	Segate BackupExec 7	9/24/2015	11/8/2015
ASD44P	iststudents	Dell	Poweredge 255(	12/1/2016	101	Windows 2010 Server	9/24/2015	2/24/2017
ASD44P	iststudents	Dell	Poweredge 255(	12/1/2016	202	MS SQL Server 2016	9/24/2015	2/24/2017
81HLV3	istwebct	Dell	Poweredge 440(	12/1/2016	111	Red Hat Linux 7.3	9/24/2015	2/24/2017
81HLV3	istwebct	Dell	Poweredge 440(	12/1/2016	301	Webct 4.1	9/24/2015	2/24/2017

I start with **serial num**:

1. If I jump to a different **Serial Number**, does the **Server Name** change? **Make? Model? Date Purchased?**
    - YES, those are *DEPENDENT* on the SerialNum field.
2. If I jump to a different **Serial Number**, does the **SWID** change?
    - NO. There is a SWID 101 in two different servers. Same w/ SW Title,date purchased. So the SW fields are NOT DEPENDENT on Serial number.

Next, determine WHAT field(s) makes a row unique to separate out our entities:

The Serial Num combined with the SWID represents a unique instance of SW installed on a Server.

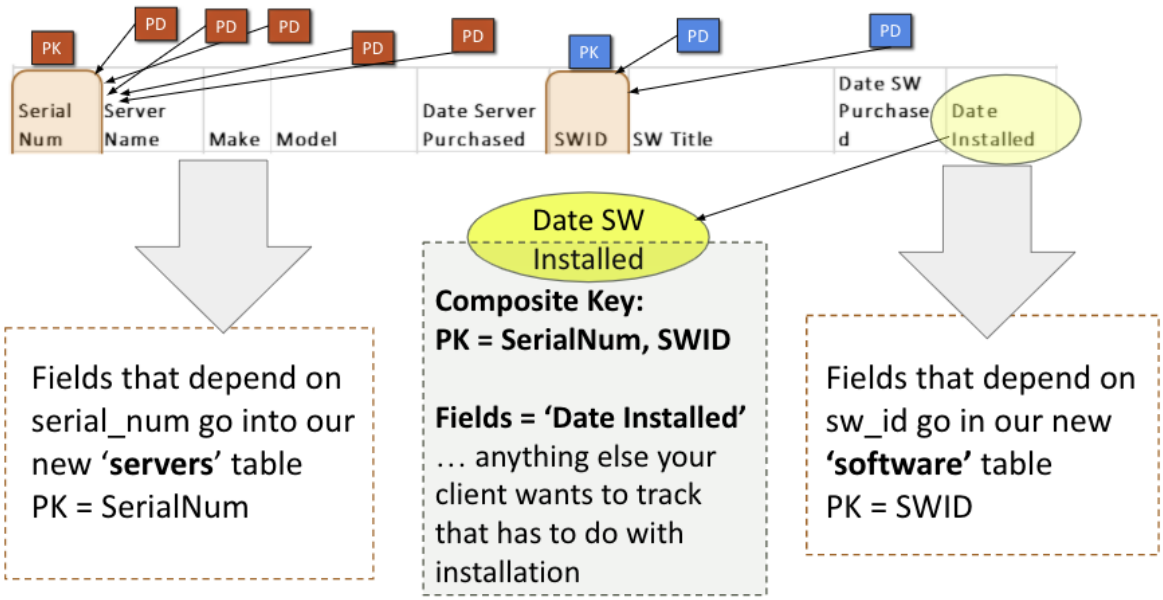
On a DEPENDENCY DIAGRAM, label both fields with a PK1 and PK2

Each field that you determined to be dependent on PK1 gets a PD1, bc it's PARTIALLY DEPENDENT on PART of the PK (PK1)

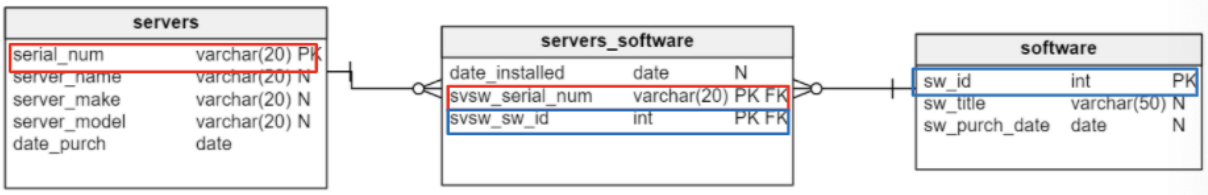
Business rule summary:

- 1 server has 1 or more SW programs installed
- 1 software program has been installed on 1 or more servers
- This is a MANY to MANY relationship

Step 4: Model those Dependencies into New Tables



Translate Results into Physical Model



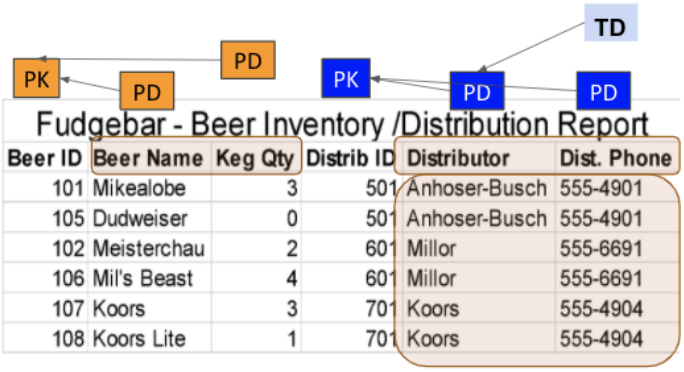
Second Normal Form (2NF)
In 1NF and we get rid of 'partial dependencies' by making new tables (bridge) tables. Breaking up entities into separate concepts.

Putting Data into 2nd Normal Form (2NF) makes related tables (Many:Many) by producing bridge tables.

A database is in second normal form (2NF) if…

- Meets INF (no Ms, No Cs_
- And
- No 'partial dependencies' (resolve PD)
- IOW: Putting Data into 2nd Normal Form (2NF) resolves M:M relationships between entities by producing 'bridge' tables

Apply a Logical Test to Non-Key Fields



Fudgebar - Beer Inventory /Distribution Report

Beer ID	Beer Name	Keg Qty	Distrib ID	Distributor	Dist. Phone
101	Mikealobe	3	501	Anhoser-Busch	555-4901
105	Dudweiser	0	501	Anhoser-Busch	555-4901
102	Meisterchau	2	601	Millor	555-6691
106	Mil's Beast	4	601	Millor	555-6691
107	Koors	3	701	Koors	555-4904
108	Koors Lite	1	701	Koors	555-4904

If I miskey Keg Qty does it mess up my Beer Name?

But if I miskey the Dist Phone #, will I reach that distributor?

Distributor Phone Number is TRANSITIVELY DEPENDENT (TD) on Distributor (which is partially dependent on Distributor ID)

- Transitive dependency: a non-key column that depends on another non-key column for it's value

Step 3: Break Transitive Fields into Lookup Tables

Keep the beers and the Distrib ID field at the end (this becomes your FK)

Beers

Beer ID	Beer Name	Keg Qty	Distrib ID
101	Mikealobe	3	501
105	Dudweiser	0	501
102	Meisterchau	2	601
106	Mil's Beast	4	601
107	Koors	3	701
108	Koors Lite	1	701

FK

PK Distributors

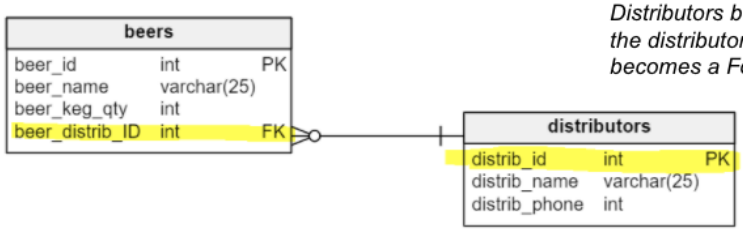
Distrib ID	Distributor	Dist. Phone
501	Anhoser-Bus	555-4901
601	Millor	555-6691
701	Koors	555-4904

Distrib ID becomes the M: side of the relationship (FK) in our Beers table (2 beers have the same distributor)

Consolidate repeating rows into new 'distributors' table

Translate Results into Logical Model

Now build your logical* model in the tool of your choice



Distributors becomes a 'lookup' table, and the distributor ID field in the beers table becomes a Foreign key

3 **Third Normal Form (3NF)**
In 2NF and we get rid of 'transitive dependencies' by making more tables

This makes related (lookup) tables. improving integrity & reducing the risk of entry errors

Third Normal Form: Example #2

PK / KEY	FD	M	FD	FD
Pet Name	**Breeds**		**Size**	**Weight**
A-Fleas Ansari	Poodle, Cairn		SM	10 - 25 lbs
A-Fleas Ansari	Poodle, Cairn		SM	10 - 25 lbs
Anderson Pooper	Chow, Lab		L	50 - 80 lbs
Obi Wan Catnobi	Domestic Shorthair		L	50 - 80 lbs
Obi Wan Catnobi	Domestic Shorthair		L	50 - 80 lbs
Bark Obama	Mastiff, Great Dane		XL	> 80 lbs
Dunkin Butterbeans	Shepherd		XL	> 80 lbs
Bilbo Waggins	Shepherd, Collie, Lab		M	26 - 49 lbs
Bilbo Waggins	Shepherd, Collie, Lab		M	26 - 49 lbs
Snarls Barkley	Pug, Yorkie		SM	10 - 25 lbs
Arf Vader	Schnauzer, Yorkie		M	26 - 49 lbs
Droolius Caesar	Schnauzer		SM	10 - 25 lbs

Size	Weight
SM	10 - 25 lbs
SM	10 - 25 lbs
L	50 - 80 lbs
L	50 - 80 lbs
L	50 - 80 lbs
XL	> 80 lbs
XL	> 80 lbs
M	26 - 49 lbs
M	26 - 49 lbs
SM	10 - 25 lbs
M	26 - 49 lbs
SM	10 - 25 lbs

If on the next line, I entered 'SM' and then entered 1-9 lbs, would that affect the integrity of the data?

SM	10 - 25 lbs
SM	**1 - 9 lbs**

Yes, because I would not know WHICH weight actually belonged to the 'SM' designation. So a value of 'SM' has no entity integrity because it is not uniquely identifiable.

How to resolve?

Sizes_lkup

Size (PK)	Weight
SM	10 - 25 lbs
M	26 - 49 lbs
L	50 - 80 lbs
XL	> 80 lbs

FK

Pet Name	Breeds	Size	Weight
A-Fleas Ansari	Poodle, Cairn	SM	10 - 25 lbs
A-Fleas Ansari	Poodle, Cairn	SM	10 - 25 lbs
Anderson Pooper	Chow, Lab	L	50 - 80 lbs
Obi Wan Catnobi	Domestic Shorthair	L	50 - 80 lbs
Obi Wan Catnobi	Domestic Shorthair	L	50 - 80 lbs
Bark Obama	Mastiff, Great Dane	XL	> 80 lbs
Dunkin Butterbeans	Shepherd	XL	> 80 lbs
Bilbo Waggins	Shepherd, Collie, Lab	M	26 - 49 lbs
Bilbo Waggins	Shepherd, Collie, Lab	M	26 - 49 lbs
Snarls Barkley	Pug, Yorkie	SM	10 - 25 lbs
Arf Vader	Schnauzer, Yorkie	M	26 - 49 lbs
Droolius Caesar	Schnauzer	SM	10 - 25 lbs

Normalization Steps

Understand the Data (ID keys, related data, talk to your client to fill in the gaps IRL)

Do Dependency Diagram (ID your PK, Multivalues (M), Dependencies FD, PD, TD)

Ensure atomicity and ...

1. **Make 1:M related tables: Remove Multi-valued Columns (M) (repeating things separated by commas) and put into new table**
2. **Resolve M:M Relationships (make Bridge tables)/ Ensure full dependency on entire PK (PD) and split out PDs into new tables**
3. **Remove any transitive dependencies (TD) and put into new (lookup) tables**

Glossary for Dependency Diagram Symbols:

1. **PK** = the column (or more likely columns) that you think represent a unique record ('what value(s) make the row unique?'
2. **M** = when there is a column that stores more than one value, or multiple columns that store the same value (multiples)
3. **FD** = occurs when you're data set has only ONE primary key column. Any and all fields that are "dependent on" that single PK column.
4. **PD** = common! Occurs when your data set has more than one primary key column. *Determine which columns depend on which of the PK columns*. Will explain this more in 2nf section.

5. **TD** = a non-pk column that is 'dependent on' another non-key column

Class example

C

FudgeCo Livery Driver Permissions Report

Driver ID #	Driver Name	Driver Chg/Hr	Driver Territories (M)	Vehicle Lic Plate	Vehicle Make	Vehicle Model	Vehicle Size	Vehicle Chg/Hr	Permission Exp. Date
101	Bill Melator	$100.00	West, North, Central	PPF673	Cadillac	Escalde	M	$ 100.00	12/31/2004
101	Bill Melator	$100.00	West, North, Central	PXK3D7T	Chevy	Tahoe	L	$ 120.00	12/31/2004
101	Bill Melator	$100.00	West, North, Central	445GH2	Lincon	Towncar	S	$ 80.00	1/30/2013
101	Bill Melator	$100.00	West, North, Central	59DLLK	Lincon	Continental	S	$ 80.00	4/30/2015
102	Willie Work	$75.00	South, East	PXK3D7T	Chevy	Tahoe	L	$ 120.00	1/15/2014
102	Willie Work	$75.00	South, East	663ETMP	Chevy	Suburban	L	$ 120.00	4/1/2015
103	Sal Ladd	$75.00	Central, East, North	667GM8	Audi	A8	M	$ 100.00	9/1/2015
103	Sal Ladd	$75.00	Central, East, North	445GH2	Lincon	Towncar	S	$ 80.00	9/1/2014
103	Sal Ladd	$75.00	Central, East, North	59DLLK	Lincon	Continental	S	$ 80.00	12/31/2012
104	Carol Ling	$100.00	Central, West	667GM8	Audi	A8	M	$ 100.00	7/31/2012
104	Carol Ling	$100.00	Central, West	PPF673	Cadillac	Escalde	M	$ 100.00	7/31/2013
104	Carol Ling	$100.00	Central, West	59DLLK	Lincon	Continental	S	$ 80.00	10/1/2014

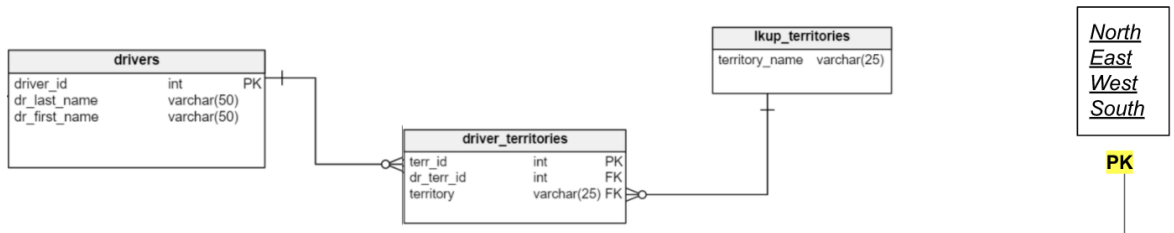
1NF: Resolving Multiples by making 'child' tables (1:M relationships)

Driver ID #	Driver Name	Driver Chg/Hr	Driver Territories
101	Bill Melator	$100.00	West, North, Central
101	Bill Melator	$100.00	West, North, Central
101	Bill Melator	$100.00	West, North, Central
101	Bill Melator	$100.00	West, North, Central
102	Willie Work	$75.00	South, East
102	Willie Work	$75.00	South, East
103	Sal Ladd	$75.00	Central, East, North
103	Sal Ladd	$75.00	Central, East, North
103	Sal Ladd	$75.00	Central, East, North

tr_id	tr_dr_id FK	Territory
1	101	West
2	101	North
3	101	Central
4	102	South
5	102	East
6	103	Central
7	103	East
8	103	North

1. Pivot the driver ID # to new rows, 1 for each value in driver territories
2. Two options for adding a PK to the 'territories' child table.
   a. New surrogate PK: *tr_id int identity (1,1)*
      i. If I do this, I should put a unique constraint on tr_dr_id and territory
   b. Composite PK: ***constraint** pk_terr **primary key** (tr_dr_id, territory)*
3. Make the new 'driver id' a FK in the 'child' table.
4. It will reference the existing 'driver id' in the main 'drivers' table.
   a. *You can (and should) be sketching out a db diagram as you go.*

How the Physical Model Looks

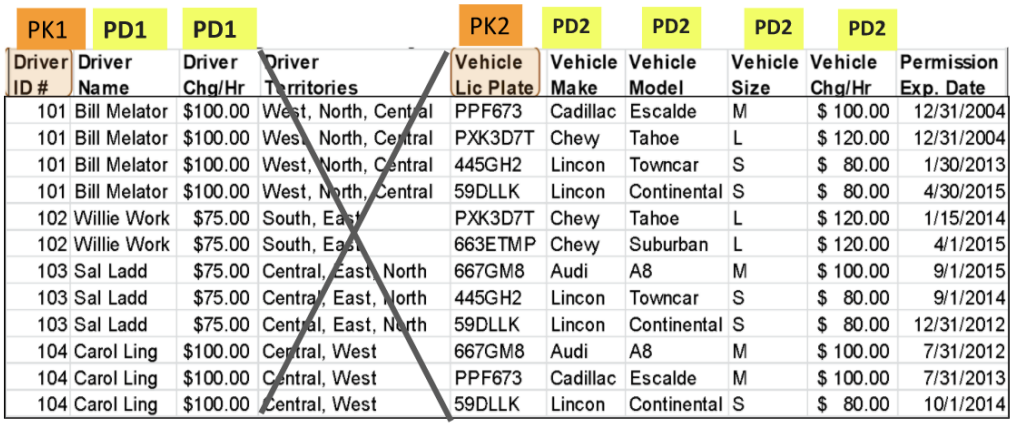


But I'm actually not done!

How do I ensure my users enter consistent values for the territories? They must enter exactly the values of *'North'*, *'West'*, *'Central'*, and *'East'* with no typos or grammatical errors. How can I enforce integrity here?

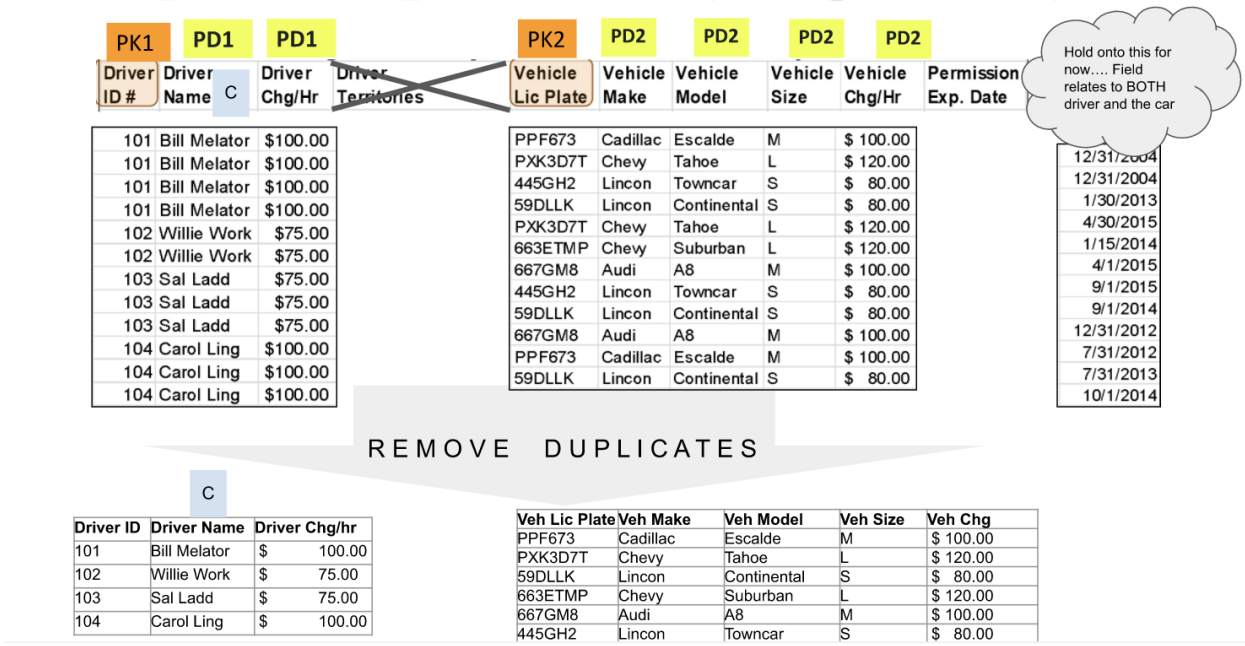
tr_id	tr_dr_id	Territory FK
1	101	West
2	101	North
3	101	Central
4	102	South
5	102	East
6	103	Central
7	103	East
8	103	North

2NF: partial dependencies (bridge tables)



PK1	PD1	PD1		PK2	PD2	PD2	PD2	PD2	
Driver ID #	Driver Name	Driver Chg/Hr	Driver Territories	Vehicle Lic Plate	Vehicle Make	Vehicle Model	Vehicle Size	Vehicle Chg/Hr	Permission Exp. Date
101	Bill Melator	$100.00	West, North, Central	PPF673	Cadillac	Escalde	M	$ 100.00	12/31/2004
101	Bill Melator	$100.00	West, North, Central	PXK3D7T	Chevy	Tahoe	L	$ 120.00	12/31/2004
101	Bill Melator	$100.00	West, North, Central	445GH2	Lincon	Towncar	S	$ 80.00	1/30/2013
101	Bill Melator	$100.00	West, North, Central	59DLLK	Lincon	Continental	S	$ 80.00	4/30/2015
102	Willie Work	$75.00	South, East	PXK3D7T	Chevy	Tahoe	L	$ 120.00	1/15/2014
102	Willie Work	$75.00	South, East	663ETMP	Chevy	Suburban	L	$ 120.00	4/1/2015
103	Sal Ladd	$75.00	Central, East, North	667GM8	Audi	A8	M	$ 100.00	9/1/2015
103	Sal Ladd	$75.00	Central, East, North	445GH2	Lincon	Towncar	S	$ 80.00	9/1/2014
103	Sal Ladd	$75.00	Central, East, North	59DLLK	Lincon	Continental	S	$ 80.00	12/31/2012
104	Carol Ling	$100.00	Central, West	667GM8	Audi	A8	M	$ 100.00	7/31/2012
104	Carol Ling	$100.00	Central, West	PPF673	Cadillac	Escalde	M	$ 100.00	7/31/2013
104	Carol Ling	$100.00	Central, West	59DLLK	Lincon	Continental	S	$ 80.00	10/1/2014

2NF: partial dependencies (bridge tables)



2NF: partial dependencies (bridge tables)

PK1	PD1	PD1		PK2	PD2	PD2	PD2	PD2	
Driver ID #	Driver Name	Driver Chg/Hr	Driver Territories	Vehicle Lic Plate	Vehicle Make	Vehicle Model	Vehicle Size	Vehicle Chg/Hr	Permission Exp. Date

PK

Driver ID	Driver Name	Driver Chg/hr
101	Bill Melator	$ 100.00
102	Willie Work	$ 75.00
103	Sal Ladd	$ 75.00
104	Carol Ling	$ 100.00

Drivers table data

FK FK

101	PPF673	12/31/2004
101	PXK3D7T	12/31/2004
101	445GH2	1/30/2013
101	59DLLK	4/30/2015
102	PXK3D7T	1/15/2014
102	663ETMP	4/1/2015
103	667GM8	9/1/2015
103	445GH2	9/1/2014
103	59DLLK	12/31/2012
104	667GM8	7/31/2012
104	PPF673	7/31/2013
104	59DLLK	10/1/2014

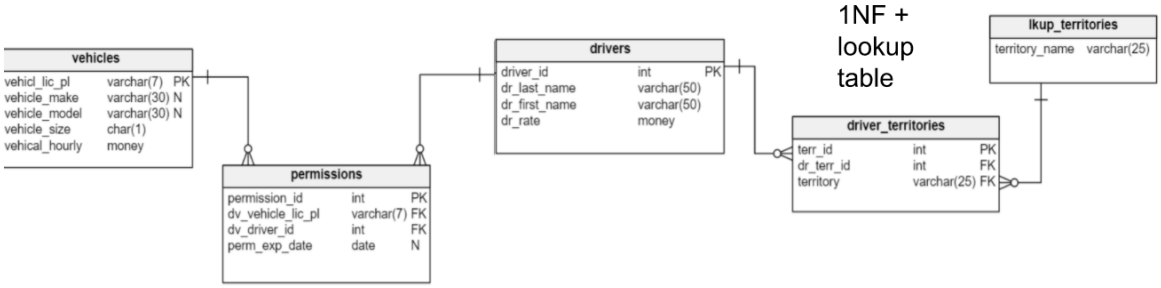
Permissions Table

PK

Veh Lic Plate	Veh Make	Veh Model	Veh Size	Veh Chg
PPF673	Cadillac	Escalde	M	$ 100.00
PXK3D7T	Chevy	Tahoe	L	$ 120.00
59DLLK	Lincon	Continental	S	$ 80.00
663ETMP	Chevy	Suburban	L	$ 120.00
667GM8	Audi	A8	M	$ 100.00
445GH2	Lincon	Towncar	S	$ 80.00

Vehicles table data

How the Physical Model Looks



2NF

3NF: Transitive dependencies

TD

Driver ID #	Driver Name	Driver Chg/Hr	Driver Territories	Vehicle Lic Plate	Vehicle Make	Vehicle Model	Vehicle Size	Vehicle Chg/Hr	Permission Exp. Date
				PPF673	Cadillac	Escalde	M	$ 100.00	
				PXK3D7T	Chevy	Tahoe	L	$ 120.00	
				445GH2	Lincon	Towncar	S	$ 80.00	
				59DLLK	Lincon	Continental	S	$ 80.00	
				PXK3D7T	Chevy	Tahoe	L	$ 120.00	
				663ETMP	Chevy	Suburban	L	$ 120.00	
				667GM8	Audi	A8	M	$ 100.00	
				445GH2	Lincon	Towncar	S	$ 80.00	
				59DLLK	Lincon	Continental	S	$ 80.00	
				667GM8	Audi	A8	M	$ 100.00	
				PPF673	Cadillac	Escalde	M	$ 100.00	
				59DLLK	Lincon	Continental	S	$ 80.00	

Does the amt charged change with the size of the vehicle?

If I mistyped the charge amount for a Large vehicle, would users not know which amount to use?

Yes, that would create confusion about which is the correct charge. That's an integrity problem we can prevent.

3NF: Transitive dependencies to lookup table

PK2 PD2 PD2 PD2 TD

Vehicle Lic Plate	Vehicle Make	Vehicle Model	Vehicle Size	Vehicle Chg/Hr	Permission Exp. Date
PPF673	Cadillac	Escalde	M	$ 100.00	12/31/2004
PXK3D7T	Chevy	Tahoe	L	$ 120.00	12/31/2004
445GH2	Lincon	Towncar	S	$ 80.00	1/30/2013
59DLLK	Lincon	Continental	S	$ 80.00	4/30/2015
PXK3D7T	Chevy	Tahoe	L	$ 120.00	1/15/2014
663ETMP	Chevy	Suburban	L	$ 120.00	4/1/2015
667GM8	Audi	A8	M	$ 100.00	9/1/2015
445GH2	Lincon	Towncar	S	$ 80.00	9/1/2014
59DLLK	Lincon	Continental	S	$ 80.00	12/31/2012
667GM8	Audi	A8	M	$ 100.00	7/31/2012
PPF673	Cadillac	Escalde	M	$ 100.00	7/31/2013
59DLLK	Lincon	Continental	S	$ 80.00	10/1/2014

When non-key attribute is dependent on another non-key attribute.

Vehicle charge/hour depends on the size of the car

Copy this as new pk, and take all vehicle size and charge columns to new table

3NF: Transitive dependencies to lookup table

FK

Vehicle Lic Plate	Vehicle Size
PPF673	M
PXK3D7T	L
445GH2	S
59DLLK	S
PXK3D7T	L
663ETMP	L
667GM8	M
445GH2	S
59DLLK	S
667GM8	M
PPF673	M
59DLLK	S

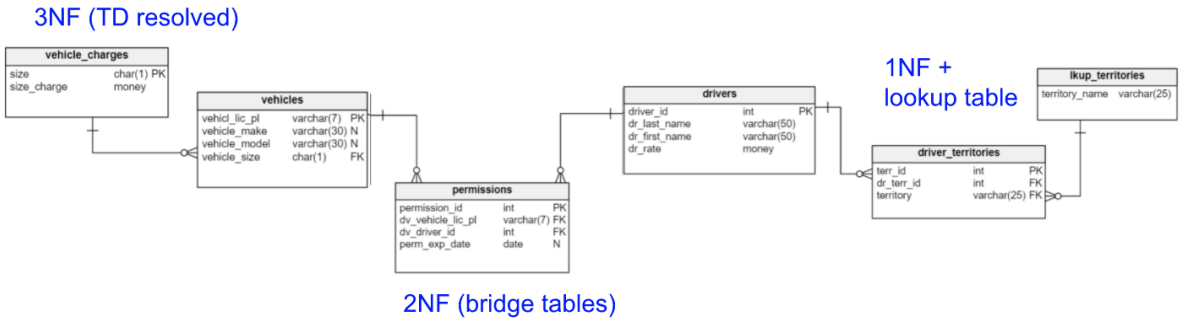
Consolidate to Unique rows

PK

Veh Size	Veh Chg/Hr
S	$ 100.00
M	$ 120.00
L	$ 140.00

Chg/hr is stored w/ vehicle size, because it's accuracy depends on the vehicle size.

How the Physical Model Looks



The purpose of a bridge table is to resolve the relationship between two tables that are in a 'many to many' relationship. A bridge table stores the 'unique combination' of the two tables' relationships.

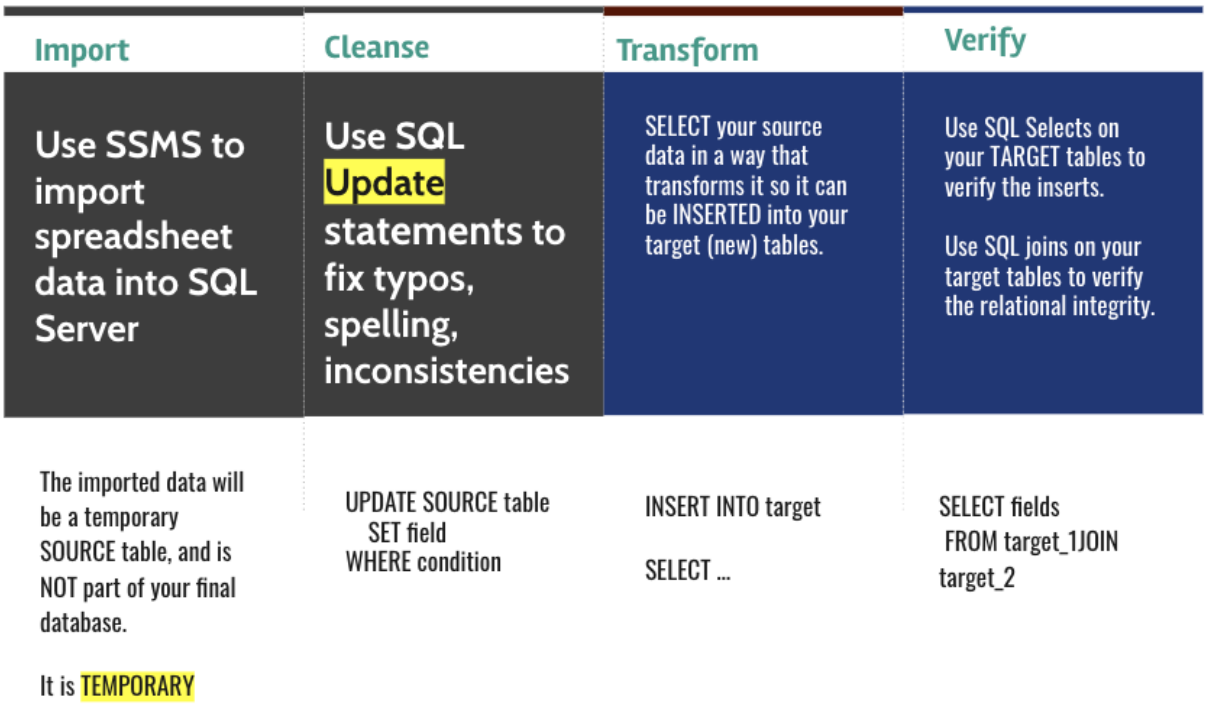
Insert Order: pk tables insert first because lookup / parent values must exist prior to inserting foreign key data: territories, vehicle_charges, vehicles, drivers, permissions

Drop order: fk tables drop first because foreign key values must delete prior to deleting the lookup / parent values

What is Data migration/conversion?

- Data migration- the act of moving data from one system to another
- Data conversion- the act of converting/ changing data from one format to another
- Typically as part of moving or upgrading systems you need to do both

Migration Process Overview



Cleanse the source data by running update statements on dat that is

- Inconsistent or missing
- Typos
- Poorly formatted

HOW??

UPDATE, SET AND WHERE

Example:

```
UPDATE source_table
    SET field_1 = 'Value you want'
    WHERE field_1= 'Value you don't want'
```

- The value you want to correct is your WHERE parameter.
- the value you want to update it to is your SET parameter

2 Cleansing Example Statement

UPDATE [dbo].['Source Data$']

Set [Drv Ter1] ='Central '

where

[Drv Ter1] = 'CENTRAL'

or

[Drv Ter1] = central

Drv Ter1	Drv Ter2	Drv Ter3
West	north	central
West	north	central
West	north	central
West	north	central
South	East	
South	East	
CENTRAL	East	North
CENTRAL	East	North
CENTRAL	East	North
central	West	
central	West	
central	West	

3 Transform the Source data into structures that can be inserted into the target tables.

HOW?

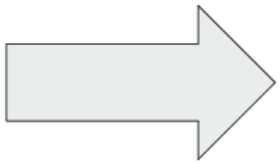
INSERT INTO target_table

SELECT STATEMENT
FROM *SOURCE TABLE*

- Run and view your SELECT statement FIRST to make sure it looks like how you want it.
  - Compare your outputs to the target (columns in right order? Same # of columns? No PK violation? No FK violations?) Check for ALL of that.
  - Once it's how you want it, THEN add the INSERT INTO ... portion of the operation.

3 Transform continued.....

Drv ID	Drv Name	Drv Chg
101	Bill Melator	100.00
101	Bill Melator	100.00
101	Bill Melator	100.00
101	Bill Melator	100.00
102	Willie Work	75.00
102	Willie Work	75.00
103	Sal Ladd	75.00
103	Sal Ladd	75.00



driver_id	driver_lastname	driver_firstname	driver_charge
101	Melator	Bill	100.00
102	Work	Willie	75.00
103	Ladd	Sal	75.00
104	Ling	Carol	100.00

1) Write a Select statement that structures the data so the target table will accept it.

```
INSERT INTO fd_drivers

SELECT DISTINCT
[Drv ID] as driver_id,

SUBSTRING([Drv Name], CHARINDEX(' ', [Drv Name]) + 1, len([Drv Name])) AS Driver_lastname,

SUBSTRING([Drv Name], 1, CHARINDEX(' ', [Drv Name]) - 1) AS Driver_firstname,

[Drv Chg] as driver_charge

FROM SourceData$
```

Verify using SQL selects on your target tables to verify the inserts

- Is the data in each field correct
- Do the tables contain the expected number of rows
- Is there referential integrity- there will be if you defined your FKs in your model before the movement of data but some people don't

Cleansing the data

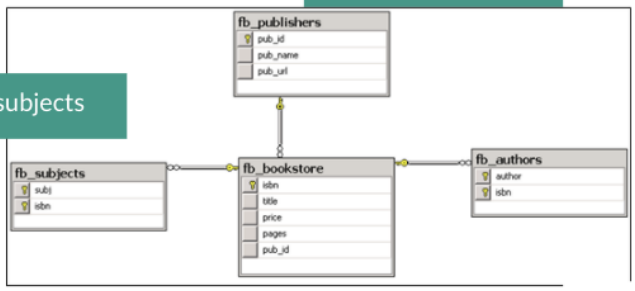
- We have to do this because it's going to be a lookup table in our new database, so we want to make sure our users can pick one of each subject

```
UPDATE fudgenbooks_import$ SET Subj1 = 'MySQL' WHERE lower(Subj1) = 'mysql';
UPDATE fudgenbooks_import$ SET Subj2 = 'MySQL' WHERE lower(Subj2) = 'mysql';
UPDATE fudgenbooks_import$ SET Subj1 = 'MSSQL' WHERE lower(Subj1) = 'mssql';
UPDATE fudgenbooks_import$ SET Subj2 = 'MSSQL' WHERE lower(Subj2) = 'mssql';
```

Order of Insertion?

It is the only table with no fk; meaning it does not depend on any other table to reference its values.

1) fb_publishers



3 or 4) fb_subjects

3 or 4) fb_authors

2) fb_bookstore

It has a FK, but since the publishers data is there, we can insert this next.

Insert into Publishers - SELECT DISTINCT

```
insert into fb_publishers
select distinct Pub_id, [Publisher Name], [Publisher URL]
from fudgenbooks_import$;
```

1 2 3

1) Source Data is where I select from; fb_publishers is where I insert.
2) Insert statement has to reflect the order, and number of columns, of the target table.
3) With ONF Source data sets, you will need to use DISTINCT or you will get an error.

Insert into Bookstores - SELECT DISTINCT

```
insert into fb_bookstore
                    1   2   3    4    5
select distinct ISBN, Title, Price, Pages, Pub_id
 from fudgenbooks_import$;
```

Hey! That's the fk to the Pub_id I just inserted into fb_publishers. It works here because I've inserted the Pub_id as a pk in fb_publishers.

Insert into Subjects - UNION

```
insert into fb_subjects

select Subj1, ISBN from fudgenbooks_import$ where
Subj1 is not null
union
select Subj2, ISBN from fudgenbooks_import$ where
Subj2 is not null
```

Subj1	Subj2
PHP	MySQL
MySQL	NULL
MSSQL	NULL
MSSQL	NULL

Insert into Authors - UNION

```
insert into fb_authors

select Author1, ISBN from fudgenbooks_import$  where Author1 is not null
   union
select Author2, ISBN  from fudgenbooks_import$ where Author2 is not null
   union
select Author3, ISBN  from fudgenbooks_import$ where Author3 is not null
```

Author1	Author2	Author3
Luke Welling	Laura Thomson	NULL
Luke Welling	Laura Thomson	NULL
Itzik Ben-Gan	Dejan Sarka	Roger Wolter
Aaron Burst	Steven Forte	NULL

Verifying Row Counts & Verifying Data

Select count (*) from fb_publishers
Select count (*) from fb_bookstore
Select count (*) from fb_subjects
Select count (*) from fb_authors

Select * from fb_publishers
Select * from fb_bookstore
Select * from fb_subjects
Select * from fb_authors

Pulling it all Together

SELECT
Fb_bookstore.isbn, title, price, pages, fb_authors.author, fb_subjects.subj,
fb_bookstore.pages, fb_publishers.pub_id, pub_name, pub_url
from
fb_bookstore
join fb_authors on fb_bookstore.isbn=fb_authors.isbn
Join fb_subjects on fb_bookstore.isbn=fb_subjects.isbn
Join fb_publishers on fb_bookstore.pub_id=fb_publishers.pub_id

```sql
select * from [dbo].['customers and projects$']

UPDATE [dbo].['customers and projects$'] Set [Customer Name] = 'Delta Airlines' where [CustomerComp#] = 2000
UPDATE [dbo].['customers and projects$'] Set [Customer Name] = 'Exelon' where [CustomerComp#] = 2200
UPDATE [dbo].['customers and projects$'] Set [Customer Name] = 'Met Life' where [CustomerComp#] = 2400
UPDATE [dbo].['customers and projects$'] Set [Customer Name] = 'United Airlines' where [CustomerComp#] = 2600

UPDATE [dbo].['customers and projects$'] Set [Contact Name] = 'Arial Photo' where [Contact #] = 1500
UPDATE [dbo].['customers and projects$'] Set [Contact Name] = 'Sal Lad' where [Contact #] = 1515
UPDATE [dbo].['customers and projects$'] Set [Contact Name] = 'Isabelle Gunnering' where [Contact #] = 1505
UPDATE [dbo].['customers and projects$'] Set [Contact Name] = 'Tuck Andrell' where [Contact #] = 1520
UPDATE [dbo].['customers and projects$'] Set [Contact Name] = 'Justin Case' where [Contact #] = 1510

UPDATE [dbo].['customers and projects$'] Set [Cust Email] = 'aphoto@united.com' where [Contact #] = 1500
UPDATE [dbo].['customers and projects$'] Set [Cust Email] = 'slad@delta.com' where [Contact #] = 1515
UPDATE [dbo].['customers and projects$'] Set [Cust Email] = 'igunner@metlife.com' where [Contact #] = 1505
UPDATE [dbo].['customers and projects$'] Set [Cust Email] = 'tandr@metlife.com' where [Contact #] = 1520
UPDATE [dbo].['customers and projects$'] Set [Cust Email] = 'jcase@exelon.com' where [Contact #] = 1510

UPDATE [dbo].['customers and projects$'] Set [Proj Name] = 'Travel Website' where [Proj#] = 7280
UPDATE [dbo].['customers and projects$'] Set [Proj Name] = 'Mileage Tracker' where [Proj#] = 7210
UPDATE [dbo].['customers and projects$'] Set [Proj Name] = 'Insurance adjuster application' where [Proj#] = 7140
UPDATE [dbo].['customers and projects$'] Set [Proj Name] = 'Claim Tracking App' where [Proj#] = 7000
UPDATE [dbo].['customers and projects$'] Set [Proj Name] = 'Financial Planning App' where [Proj#] = 7070

UPDATE [dbo].['customers and projects$'] Set [Proj type] = 'Travel' where [Proj#] = 7280
UPDATE [dbo].['customers and projects$'] Set [Proj type] = 'Travel' where [Proj#] = 7210
UPDATE [dbo].['customers and projects$'] Set [Proj type] = 'Insurance' where [Proj#] = 7140
UPDATE [dbo].['customers and projects$'] Set [Proj type] = 'Insurance' where [Proj#] = 7000
UPDATE [dbo].['customers and projects$'] Set [Proj type] = 'Finance' where [Proj#] = 7070

UPDATE [dbo].['customers and projects$'] Set [xyz proj email] = 'belotte@xyzco.com' where [Proj Mgr #] = 6000
UPDATE [dbo].['customers and projects$'] Set [xyz proj email] = 'pmoss@xyzco.com' where [Proj Mgr #] = 6060
UPDATE [dbo].['customers and projects$'] Set [xyz proj email] = 'sshores@xyzco.com' where [Proj Mgr #] = 6120
UPDATE [dbo].['customers and projects$'] Set [xyz proj email] = 'mrophone@xyzco.com' where [Proj Mgr #] = 6180
UPDATE [dbo].['customers and projects$'] Set [xyz proj email] = 'slad04@xyzco.xom' where [Proj Mgr #] = 6240

UPDATE [dbo].['customers and projects$'] Set [xyz job title] = 'Data Architect' where [Proj Mgr #] = 6000
UPDATE [dbo].['customers and projects$'] Set [xyz job title] = 'Technical Architect' where [Proj Mgr #] = 6060
UPDATE [dbo].['customers and projects$'] Set [xyz job title] = 'Application Architect' where [Proj Mgr #] = 6120
```

```sql

--transoformation--
select * from [dbo].[tasks$]
select * from [dbo].['tasks statuses$']
select * from [dbo].['customers and projects$']

INSERT INTO task_type_lookup
SELECT DISTINCT [Task type], [Task type hourly rate], [bill adjustment for task]
FROM [dbo].[tasks$]

INSERT INTO task_status
SELECT DISTINCT [Task#], [Task], [Task Status], [Action], [Date Action taken], [Proj Mgr Notified]
FROM [dbo].['tasks statuses$']

INSERT INTO Tasks
SELECT DISTINCT [ task name], [summary of task], [date assigned], [date completed], [emp# assigned to], [hours on task], [task type], [emp #
  Task Assigned], [task assigned first], [task assigned last]
FROM [dbo].[tasks$]

INSERT INTO Project
SELECT DISTINCT [Proj Name], [Proj Desc], [Proj#], [Proj type]
FROM [dbo].['customers and projects$']

INSERT INTO XYZ_employee
SELECT DISTINCT [Customer Name], [Phone], [Proj Mgr #], [XYZ Project manager], [xyz proj email], [xyz job title], [xyz proj email]
FROM [dbo].['customers and projects$']

INSERT INTO Client_contact
SELECT DISTINCT [Cust email]
FROM [dbo].['customers and projects$']

INSERT INTO Customer_company
SELECT DISTINCT [Addr line 1], [City], [State], [Zip], [Phone], [Customer Name], [Cust Email], [CustomerComp#]
FROM [dbo].['customers and projects$']
```

```sql
--verification--
select count(*) from Task_bill
select count(*) from task_type_lookup
select count(*) from task_status
select count(*) from Tasks
select count(*) from Project
select count(*) from XYZ_employee
select count(*) from Client_contact
select count(*) from Customer_company

SELECT       *
FROM            Task_bill INNER JOIN
                         Tasks ON Task_bill.task_id = Tasks.task_id INNER JOIN
                         Project ON Tasks.task_employee_id = Project.project_id INNER JOIN
                         Client_contact INNER JOIN
                         Customer_company ON Client_contact.client_customer_id = Customer_company.customer_company_id ON
                           Project.client_contact_id = Client_contact.client_id INNER JOIN
                         task_type_lookup ON Tasks.task_type = task_type_lookup.task_type INNER JOIN
                         XYZ_employee ON Tasks.task_employee_id = XYZ_employee.XYZ_employee_id INNER JOIN
                         task_status ON Tasks.task_number = task_status.task_number
```

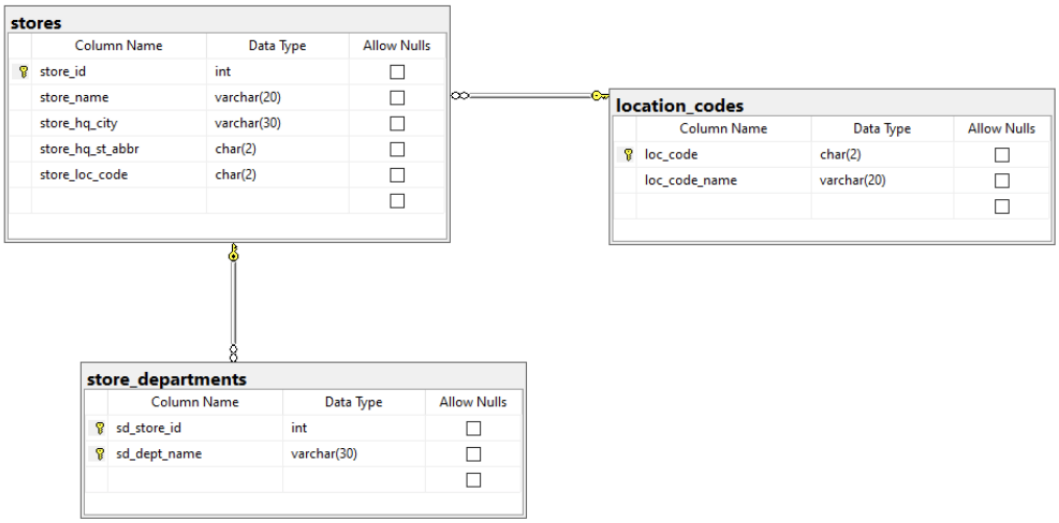
Review for exam

The data

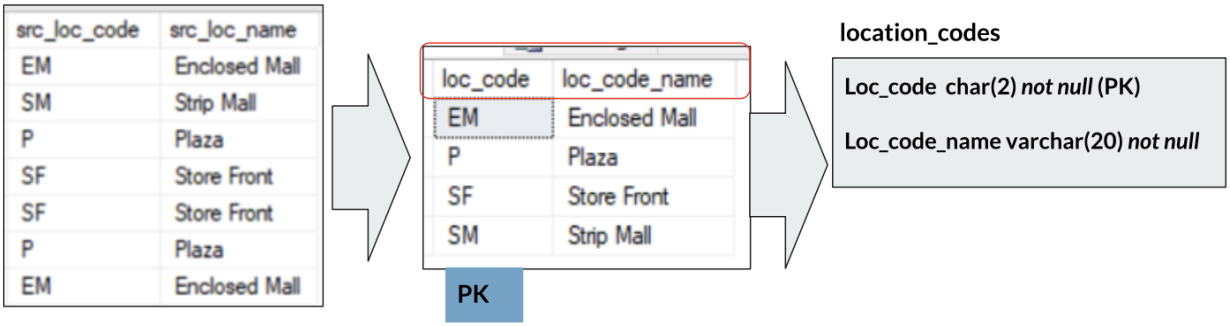
What Normal Form (NF) are we starting in? How do you know?

src_id	src_store_name	src_department_1	src_department_2	src_hq_location	src_loc_code	src_loc_name
1	Barneys	Mens Clothing	Footwear	Manhattan, NY	EM	Enclosed Mall
2	Macys	Housewares	Clothing	Philadelphia, PA	SM	Strip Mall
3	KitchenAide	Pots and Pans	Housewares	BentonHarbor, MI	P	Plaza
4	Apple	Computers	Phones	Cupertino, CA	SF	Store Front
5	Pottery Barn	Furniture	Lighting	SanFrancisco, CA	SF	Store Front
6	Clarke Shoes	NULL	Footwear	Waltham, MA	P	Plaza
7	Chicos	Womens Clothing	Footwear	FortMeyers, FL	EM	Enclosed Mall
PK		M	M	C		TD
	FD	FD	FD	FD	FD	FD

Tables in the final model



Modeling the Transitive Dependency



Cleansing Questions

'Instead of 'mens and womens' clothing, let's just have ONE 'Clothing' Category.

```
UPDATE Store_Source
  SET src_department_2 = 'Clothing'
  WHERE src_department_2 = 'Mens Clothing' Or src_department_2='Womens Clothing'
UPDATE Store_Source
  SET src_department_1 = 'Clothing'
  WHERE src_department_1 = 'Womens Clothing' Or src_department_1='Mens Clothing'
```

'Instead of 'Computers' and 'Phones', lets just have ONE 'Device' Category

```
-- or in targeted format
  Update Store_Source
set src_department_1='Devices'
where src_id =4
    Update Store_Source
set src_department_2='Devices'
where src_id =4
```

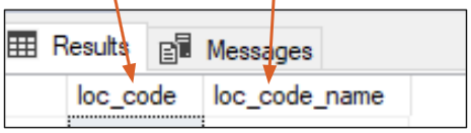
More Cleansing ...

All instances of Pots and Pans to Housewares, two techniques can be used:

```
-- POTS AND PANS TO HOUSEWARES
update Store_Source
set src_department_1='Housewares'
where src_department_1='Pots and Pans'

update Store_source
set src_department_2='Housewares'
where src_department_2='Pots and Pans'

-- Or Targeted format --
update store_source
set src_department_1='Housewares'
where src_id=3

update store_source
set src_department_2='Housewares'
where src_id=3
```

Transformation: Location Codes

```
-- insert location codes
INSERT INTO location_codes
SELECT DISTINCT src_loc_code, src_loc_name
  FROM Store_Source
```



- Don't forget the DISTINCT.
- Line up the columns you are inserting w/ the target column order,.
- Ensure you are inserting the same # of columns as are in the target table.
- Remember which is your SOURCE (goes in the From)
- And which is your TARGET (goes in the insert into…)

Transformation: Stores

Write the results that the code produces

```
-- insert stores
INSERT INTO stores
SELECT DISTINCT src_id, src_store_name,
SUBSTRING(src_hq_location,1,CHARINDEX(',',src_hq_location)-1),
SUBSTRING(src_hq_location,CHARINDEX(',',src_hq_location)+2,2),
src_loc_code
    FROM Store_Source
```

store_id	store_name	store_hq_city	store_hq_st_abbr	store_loc_code
1	Bameys	Manhattan	NY	EM
2	Macys	Philadelphia	PA	SM
3	KitchenAide	BentonHarbor	MI	P
4	Apple	Cupertino	CA	SF
5	Pottery Barn	SanFrancisco	CA	SF
6	Clarke Shoes	Waltham	MA	P
7	Chicos	FortMeyers	FL	EM

- You are inserting into the STORES table.
- Grab id, and name.
- Then interpret the substring based on the difference b/w source & result (city / state are broken up)
- Then loc code.
- Make sure you select from Store_source

Transformation: store_departments

```
--insert departments union
INSERT INTO store_departments
SELECT DISTINCT src_id, src_department_1
   FROM Store_Source
   WHERE src_department_1 is not null
   UNION
SELECT DISTINCT src_id, src_department_2
   FROM Store_Source
   WHERE src_department_2 is not null
```

- Any time you are transforming multiple columns, you are using a UNION (no exceptions).

- Follow the format here:
- Consistent column order and naming,
- just chaining together SELECT statements,
- ensuring you do NOT bring over Nulls with use of the Is Not Null in the WHERE clause.