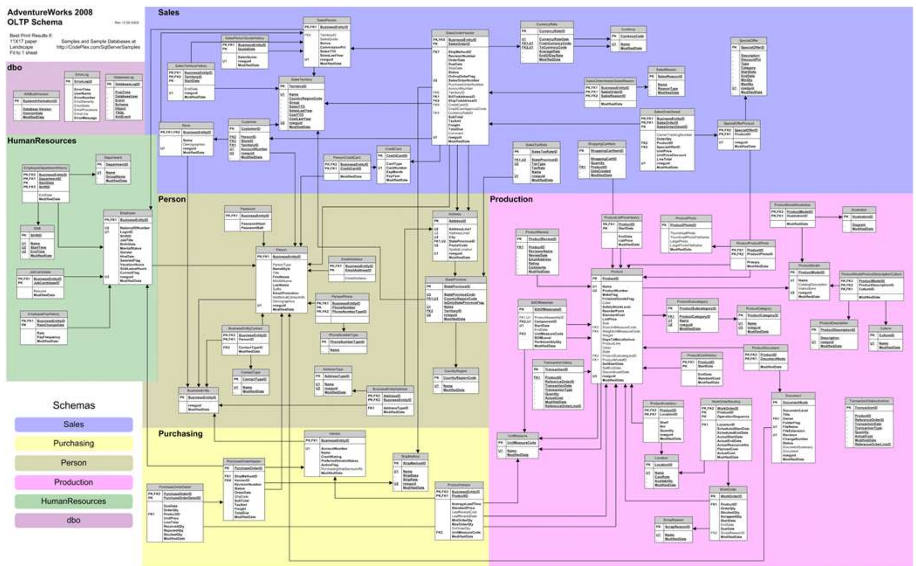
Three approaches to data storage

- Graph Databases
- Document Databases
- **Relational Databases**

Example Relational Database Documentation



This is sort of what documentation would look when you walk into typical organization as a consultant data scientist or machine learning engineer. And you'll think, why can't everything just be stored together in one big big dataframe?

What's wrong with this dataframe (table)?

StudentMajors

StudentID	**Student Name**	**Major**	**Advisor ID**	**Advisor Name**
12345678	Dustin Johnson	Statistics	5837593	John Tukey
87654321	Justin Dohnson	Statistics	1867594	Tom Bayes
55555555	Mirko Miorelli	Physics	4286549	Eric Vogt
94884848	Moe Antar	Bioinformatics	918288374	Greg Mendel
66613134	Haihan Lan	Electrical Engineering	12481632	Gordon Moore

.
.
.

What if we used this to track both students, AND advisors?

What's wrong with this table?

StudentMajors

StudentID	Student Name	Major	Advisor ID	Advisor Name
12345678	Dustin Johnson	Finance	5837593	John Tukey
87654321	Justin Dohnson	Statistics	1867594	Tom Bayes
55555555	Mirko Miorelli	Physics	4286549	Eric Vogt
94884848	Moe Antar	Bioinformatics	918288374	Greg Mendel
66613134	Haihan Lan	Electrical Engineering	12481632	Gordon Moore

.
.
.

Update Anomaly

What happens when Dustin Johnson wakes up one morning and changes his Major to Finance?
Now it looks like John Tukey is a Finance Advisor.

That's what is called an update anomaly.

What's wrong with this table?

StudentMajors

StudentID	Student Name	Major	Advisor ID	Advisor Name
12345678	Dustin Johnson	Finance	5837593	John Tukey
87654321	Justin Dohnson	Statistics	1867594	Tom Bayes
55555555	Mirko Miorelli	Physics	4286549	Eric Vogt
94884848	Moe Antar	Bioinformatics	918288374	Greg Mendel
~~66613134~~	~~Haihan Lan~~	~~Electrical Engineering~~	~~12481632~~	~~Gordon Moore~~

.
.
.

Deletion Anomaly

Let's say that someone drops out. Now Gordon Moore is apparently fired.

What’s wrong with this table?

StudentMajors

StudentID	Student Name	Major	Advisor ID	Advisor Name
12345678	Dustin Johnson	Finance	5837593	John Tukey
87654321	Justin Dohnson	Statistics	1867594	Tom Bayes
55555555	Mirko Miorelli	Physics	4286549	Eric Vogt
94884848	Moe Antar	Bioinformatics	918288374	Greg Mendel
~~66613134~~	~~Haihan Lan~~	~~Electrical Engineering~~	~~12481632~~	~~Gordon Moore~~

.
.
.

Insertion Anomaly

NEW ADVISOR: Hollis Lomax (Professor of Literature)

How do we insert a new professor in this table without creating a ‘blank’ student as a placeholder in the meantime?

It looks like for all these cases, we’re going to want to split students and advisors into separate tables, and log any advisory relationships in yet another separate table.

How about this table?

Item Name	Title	Author	Year	Pages	Keyword	Rating
0385333498	The Sirens of Titan	Kurt Vonnegut	1959	00336	Book Paperback	***** 5 stars Excellent
0802131786	Tropic of Cancer	Henry Miller	1934	00318	Book	****
1579124585	The Right Stuff	Tom Wolfe	1979	00304	Book Hardcover American	**** 4 stars
B000T9886K	In Between	Paul Van Dyk	2007		CD Trance	4 stars

Here is another example table. This time from AmazonDB. What do you notice?

One item can have multiple ratings. And it can have multiple keywords. How might this cause an issue?
Or what's a few things that could be needlessly challenging about this?

For example, what if I want to calculate the average rating of all the items?
And then what about the average ratintgs of Books versus CDs?

I could do it, it's just that I have to parse through the ratings for each case in an imperative manner, and convert to each number accordingly.

How about this table?

Item Name	Title	Author	Year	Pages	Keyword	Rating
0385333498	The Sirens of Titan	Kurt Vonnegut	1959	00336	Book Paperback	***** (revision history: ****, ***, ****) 5 stars (revision history: 2 stars, 4 stars, 5 stars) Excellent (revision history: Excellent)
0802131786	Tropic of Cancer	Henry Miller	1934	00318	Book	**** (revision history: ****)
1579124585	The Right Stuff	Tom Wolfe	1979	00304	Book Hardcover American	**** (revision history: ****) 4 stars (revision history: ****)
B000T9886K	In Between	Paul Van Dyk	2007		CD Trance	4 stars (revision history: ****)

Changes to log more information makes this more complicated without causing duplication in the item names for each additional rating's revision history by user.

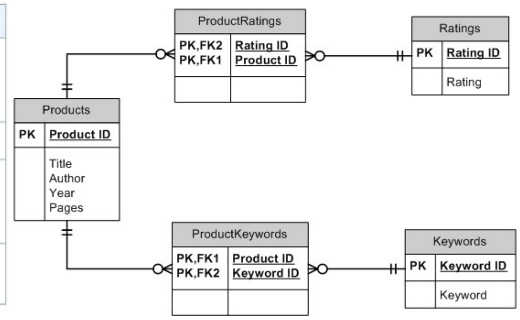
How about this table?

Item Name	Title	Author	Year	Pages	Keyword	Rating
0385333498	The Sirens of Titan	Kurt Vonnegut	1959	00336	Book Paperback	***** (revision history: **** (position: (38.23, 94.56), (39.44,93.24)), ***(position: (38.23, 94.56), (39.44,93.24)), ****(position: (38.23, 94.56), (39.44,93.24))) 5 stars (revision history: 2 stars(position: (38.23, 94.56), (39.44,93.24)), 4 stars(position: (38.23, 94.56), (39.44,93.24)), 5 stars(position: (38.23, 94.56), (39.44,93.24))) Excellent (revision history: Excellent)
0802131786	Tropic of Cancer	Henry Miller	1934	00318	Book	**** (revision history: ****)
1579124585	The Right Stuff	Tom Wolfe	1979	00304	Book Hardcover American	**** (revision history: ****(position: (38.82, 94.56), (39.44,93.24))) 4 stars (revision history: ****(position: (38.67, 94.56), (39.44,93.24)))
B000T9886K	In Between	Paul Van Dyk	2007		CD Trance	4 stars (revision history: ****(position: (38.23, 94.56), (39.44,93.24)))

And more complication that could happen.

Enter the Relational Model representation

Item Name	Title	Author	Year	Pages	Keyword	Rating
0385333498	The Sirens of Titan	Kurt Vonnegut	1959	00336	Book Paperback	***** 5 stars Excellent
0802131786	Tropic of Cancer	Henry Miller	1934	00318	Book	****
1579124585	The Right Stuff	Tom Wolfe	1979	00304	Book Hardcover American	**** 4 stars
B000T9886K	In Between	Paul Van Dyk	2007		CD Trance	4 stars



The relational model will take care of the modification anomalies and one-to-many complications we saw in the previous examples.
Here on the right is how it would be modeled with a relational model.

At most companies, they will be using this notation to describe their enterprise data warehouse. If you will be focusing on many different areas of the organization, you will be looking at many pictures like this.

This way, the data storage is perspective neutral. So we don't have to worry about accidentally dropping professors or students, or not being able to add professors until they have a student.

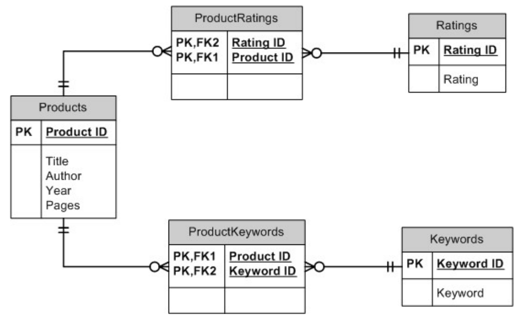
Relational Model Representation

ProductRatings

Rating ID	**Product ID**
A5	385333498
S5	385333498
D3	385333498
A4	802131786
A4	1579124585
S4	1579124585
S4	B000T9886K

Ratings

Rating ID	Rating	Value
A1	*	1
A2	**	2
A3	***	3
A4	****	4
A5	*****	5
D1	Terrible	1
D2	Ok	3
D3	Excellent	5
S1	1 star	1
S2	2 stars	2
S3	3 stars	3
S4	4 stars	4
S5	5 stars	5



Products

Product ID	Title	Author	Year	Pages
385333498	The Sirens of Titan	Kurt Vonnegut	1959	336
802131786	Tropic of Cancer	Henry Miller	1934	318
1579124585	The Right Stuff	Tom Wolfe	1979	304
B000T9886K	In Between	Paul Van Dyk	2007	NULL
	...			

ProductKeywords

Product ID	**Keyword ID**
385333498	K1
385333498	K2
802131786	K1
1579124585	K1
1579124585	K3
1579124585	K4
B000T9886K	K5
B000T9886K	K6

Keywords

Keyword ID	Keyword
K1	Book
K2	Paperback
K3	Hardcover
K4	American
K5	CD
K6	Trance
	...
K94842	Graphics Card

The bold and underlined columns are the keys. They are how we can link the relationships together between the tables.
Like an index in pandas.

PK stands for Primary Key. FK stands for Foreign Key.
The primary key is the minimal subset that uniquely identifies a row.
A foreign key references a primary key of another table.

Relational Model Representation

ProductRatings

Rating ID	Product ID
A5	385333498
S5	385333498
D3	385333498
A4	802131786
A4	1579124585
S4	1579124585
S4	B000T9886K

Ratings

Rating ID	Rating	Value
A1	*	1
A2	**	2
A3	***	3
A4	****	4
A5	*****	5
D1	Terrible	1
D2	Ok	3
D3	Excellent	5
S1	1 star	1
S2	2 stars	2
S3	3 stars	3
S4	4 stars	4
S5	5 stars	5

Products

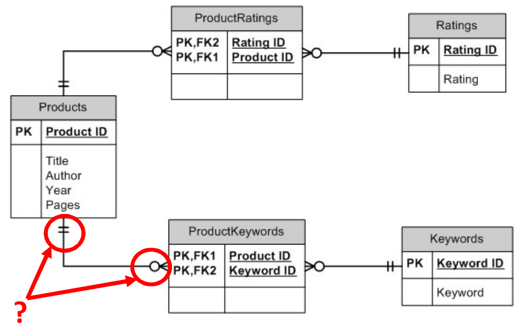
Product ID	Title	Author	Year	Pages
385333498	The Sirens of Titan	Kurt Vonnegut	1959	336
802131786	Tropic of Cancer	Henry Miller	1934	318
1579124585	The Right Stuff	Tom Wolfe	1979	304
B000T9886K	In Between	Paul Van Dyk	2007	NULL
	...			

ProductKeywords

Product ID	Keyword ID
385333498	K1
385333498	K2
802131786	K1
1579124585	K1
1579124585	K3
1579124585	K4
B000T9886K	K5
B000T9886K	K6

Keywords

Keyword ID	Keyword
K1	Book
K2	Paperback
K3	Hardcover
K4	American
K5	CD
K6	Trance
	...
K94842	Graphics Card

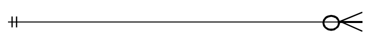


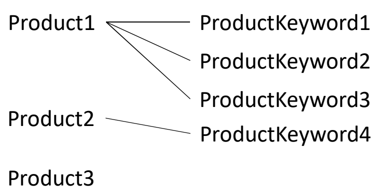
Primary Key: The minimal subset that uniquely identifies a row.
Foreign Key: References a primary key of another table,.

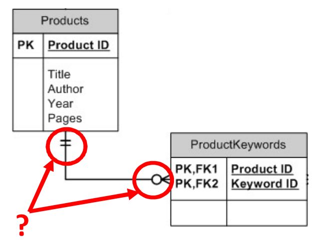
This stuff is called “crows foot notation”

Relational Model Notation

"Crows Foot"



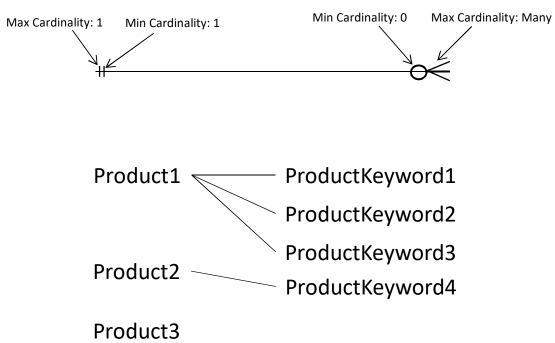


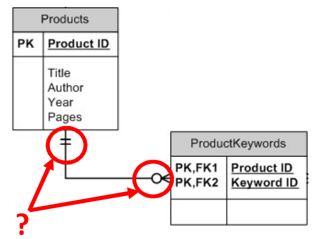


Because they look like crows feet.

Why the Relational Model?

Crows Foot Notation



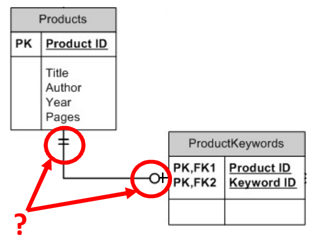


Looking from a Product, it can have at the minimum zero keywords. And at the max, many keywords.

But looking from product keyword, it must belong to one product, and only one product.

Why the Relational Model?

Crows Foot Notation



Product1 ProductKeyword1

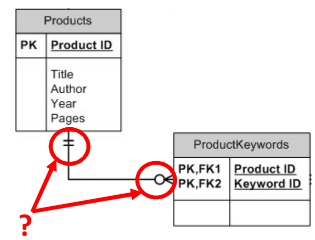
Product2 ProductKeyword2

Product3

Suppose instead we had the diagram looking like *this*

Why the Relational Model?

Crows Foot Notation

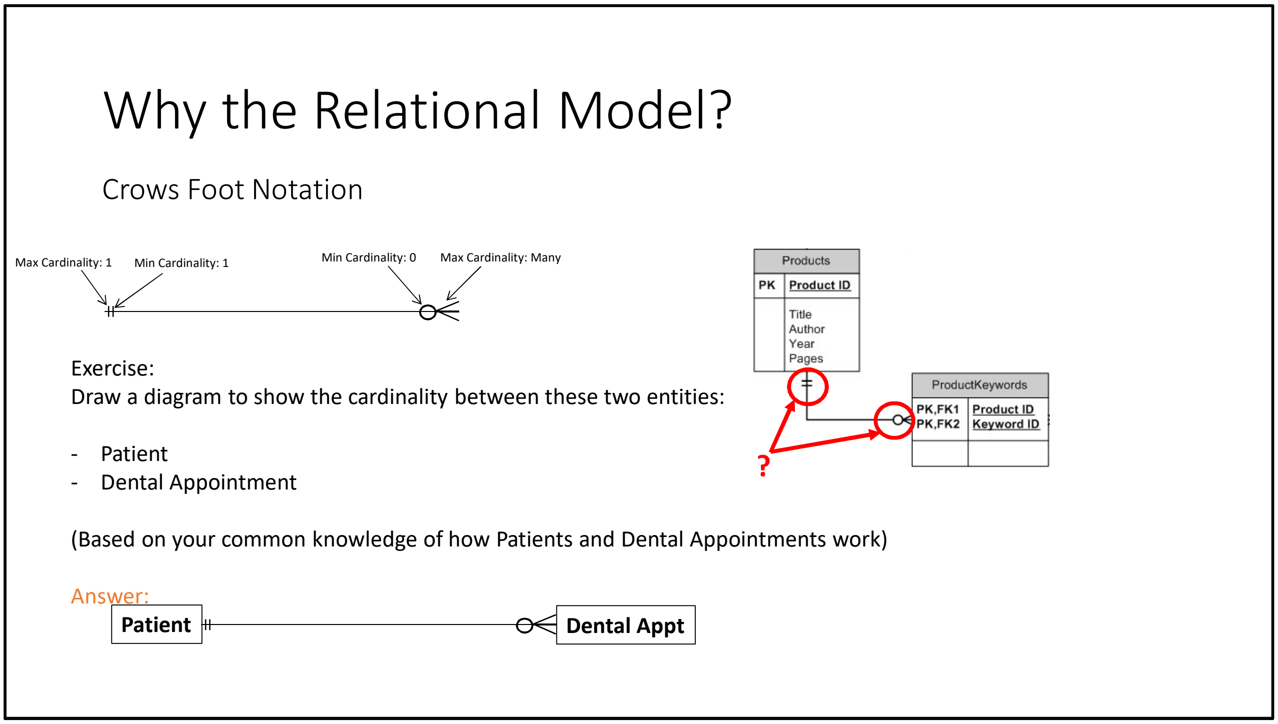


Exercise:
Draw a diagram to show the cardinality between these two entities:

- Patient
- Dental Appointment

(Based on your common knowledge of how Patients and Dental Appointments work)

Exercise time



Same as the products to ProductsKeywords relationship.

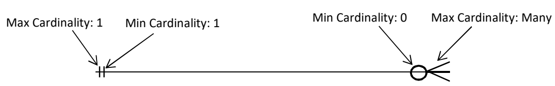
Because a patient can have at minimum zero dental visits. And one or more dental visits.
But a dental visit can have at minimum one patient, and at *most* one patient.

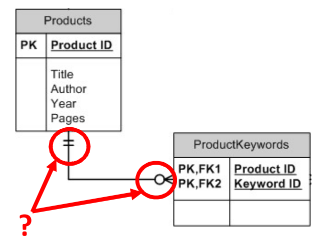
(Unless you've been to a dental clinic that deals with two patients at once)

So these are the constraints that are important to note down when you're designing how the data will flow into their data structures. And what you will read when you're building data models on top of them.

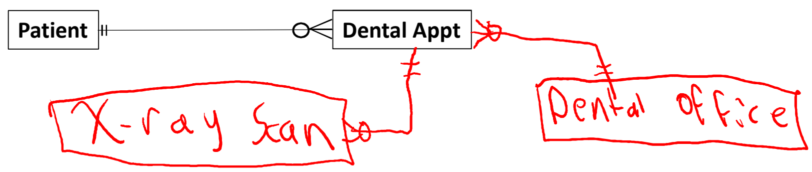
Why the Relational Model?

Crows Foot Notation

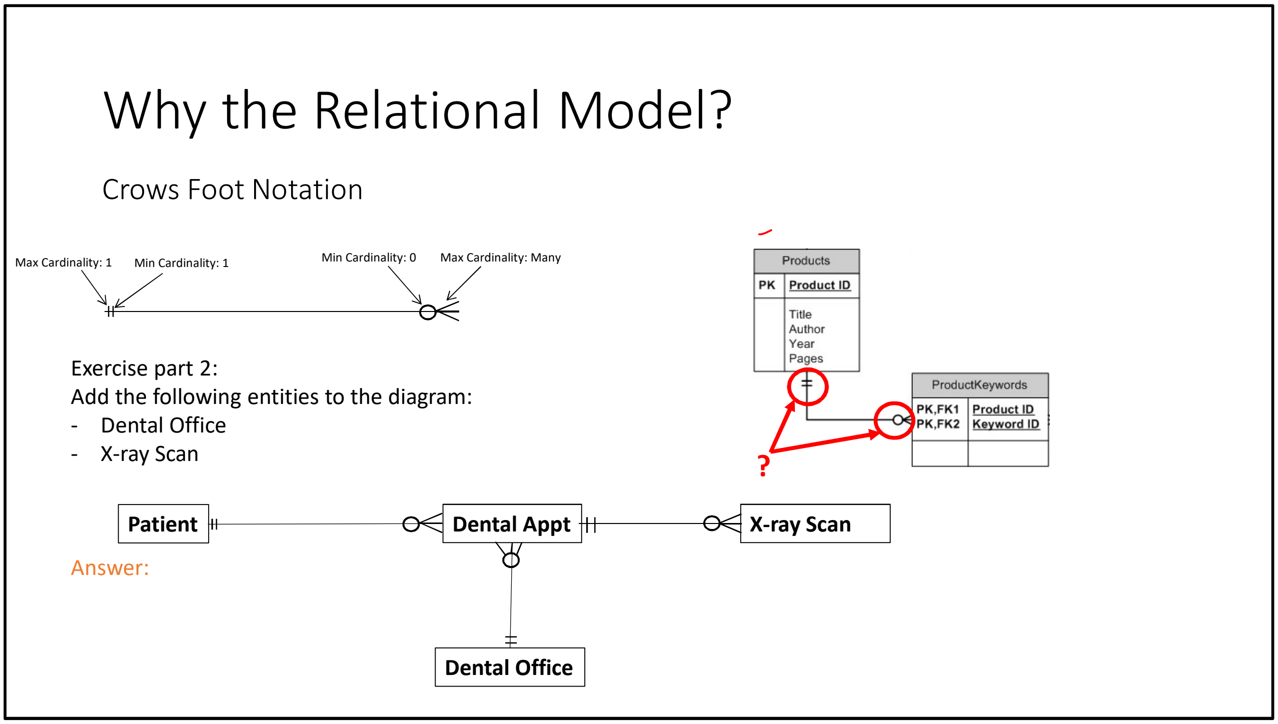




Exercise part 2:
Add the following entities to the diagram:
- Dental Office
- X-ray Scan



Exercise part 2



Personally I have never had a dental appointment that involved more than one office. If you have though, then fair enough!

I think I'd consider the number of x-ray scans to be based on the number of different pictures that get printed out. Usually in my experience they take two or three. But sometimes they don't take any.

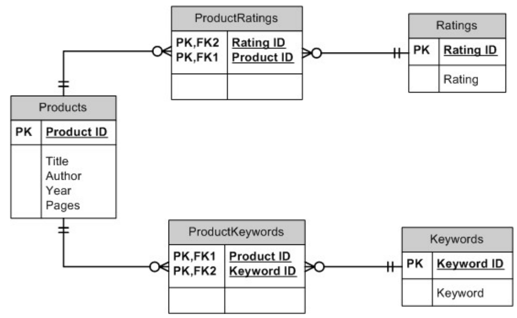
Why the Relational Model?

ProductRatings

Rating ID	Product ID
A5	385333498
S5	385333498
D3	385333498
A4	802131786
A4	1579124585
S4	1579124585
S4	B000T9886K

Ratings

Rating ID	Rating	Value
A1	*	1
A2	**	2
A3	***	3
A4	****	4
A5	*****	5
D1	Terrible	1
D2	Ok	3
D3	Excellent	5
S1	1 star	1
S2	2 stars	2
S3	3 stars	3
S4	4 stars	4
S5	5 stars	5



Products

Product ID	Title	Author	Year	Pages
385333498	The Sirens of Titan	Kurt Vonnegut	1959	336
802131786	Tropic of Cancer	Henry Miller	1934	318
1579124585	The Right Stuff	Tom Wolfe	1979	304
B000T9886K	In Between	Paul Van Dyk	2007	NULL
	...			

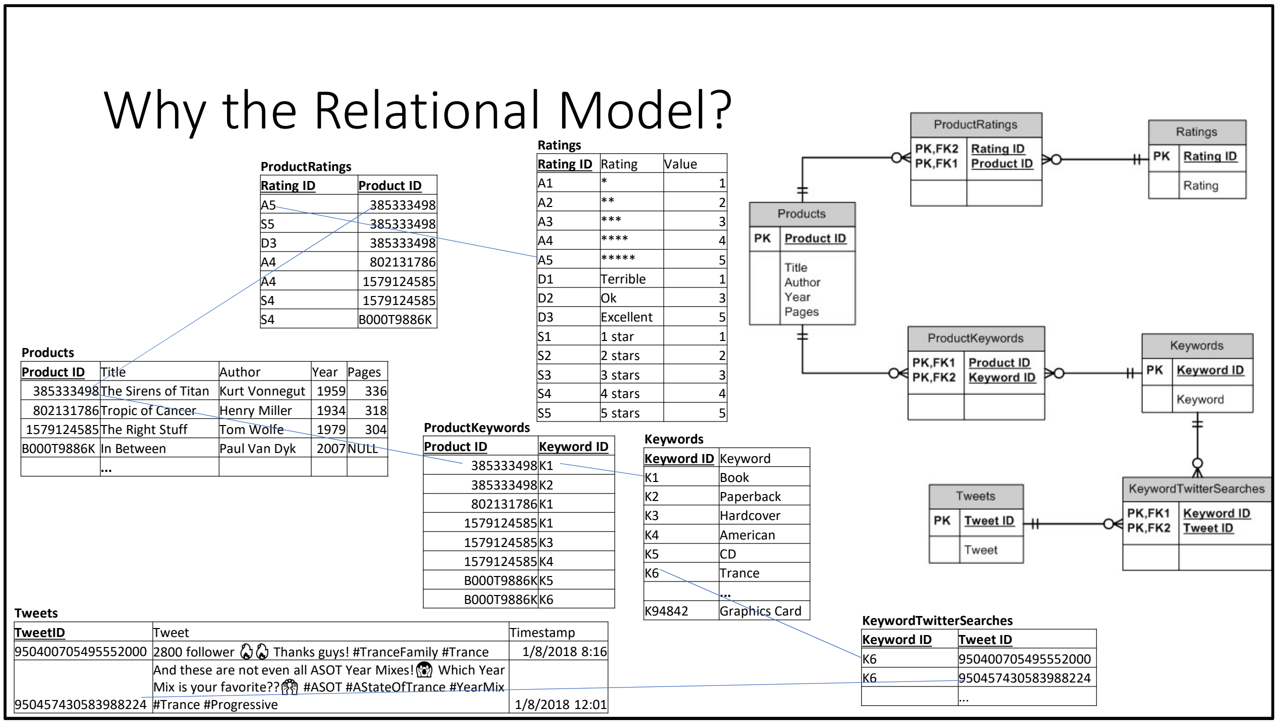
ProductKeywords

Product ID	Keyword ID
385333498	K1
385333498	K2
802131786	K1
1579124585	K1
1579124585	K3
1579124585	K4
B000T9886K	K5
B000T9886K	K6

Keywords

Keyword ID	Keyword
K1	Book
K2	Paperback
K3	Hardcover
K4	American
K5	CD
K6	Trance
	...
K94842	Graphics Card

Back to this relational model example.



Here’s why the model is so extendable. Suppose we wanted to include tweets from keyword searches.

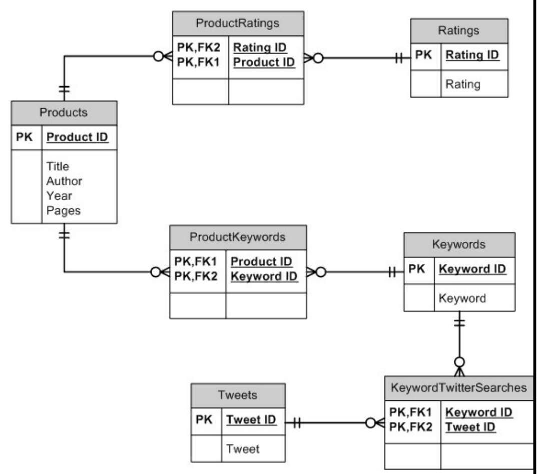
Now that we’ve added it, all the entities of this model are *still* “first class citizens”. As a product developer, I can still ask questions about, or modify products or keywords, without having to bother with tweets.

And the marketer can use the same data model, for completely different reasons than the product developer.

I don’t have to bother adding which tweets are by extension somewhat related to which products, updating the products data structure every time someone tweets anything, until I ask the query. And it figures it out very quickly.

SQL Statements

SELECT <list of column expressions>
FROM <TYPE> JOIN <…> on …
WHERE <list of logical expressions for rows>
ORDER BY <list of sorting specifications>

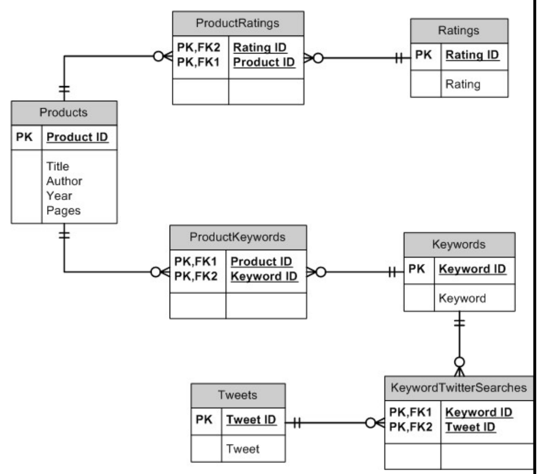


See all Authors:

```
SELECT Author
FROM Products;
```

SQL Statements

SELECT <list of column expressions>
FROM <TYPE> JOIN <…> on …
WHERE <list of logical expressions for rows>
ORDER BY <list of sorting specifications>



See Authors and their Titles:

```
SELECT Author, Title
FROM Products
ORDER BY Author, Title;
```

Note: ORDER BY will ASCEND by default.

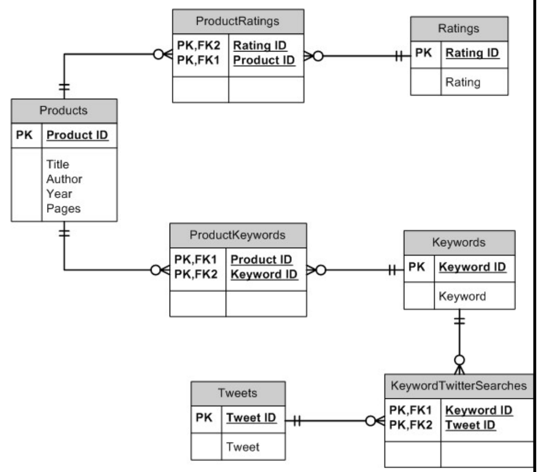
SQL Statements

SELECT <list of column expressions>

FROM <TYPE> JOIN <…> on …

WHERE <list of logical expressions for rows>

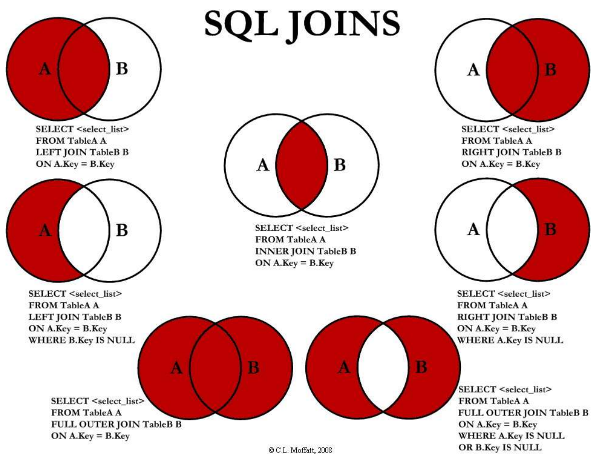
ORDER BY <list of sorting specifications>



See Authors and their Titles after 1970 and more than 700 pages long:

```
SELECT Author, Title
FROM Products
WHERE Year > 1970 AND Pages > 700
ORDER BY Author, Title;
```

Joins?



* Important thing that came up during the lecture that I'd want to be more clear about. LEFT JOIN is also known as "LEFT OUTER JOIN"

Joins?

100 students, 70 of which have lockers. You have a total of **50 lockers**, 40 of which have at least 1 student and 10 lockers have no student. (Some students share a locker)

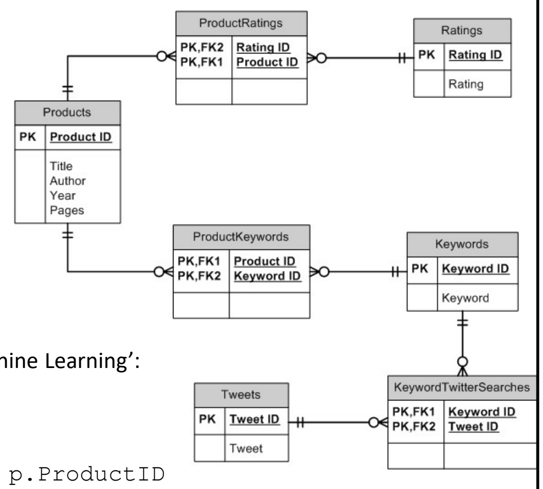
INNER JOIN is equivalent to "*show me all students with lockers*".
Any students without lockers, or any lockers without students are missing.
Returns 70 rows

LEFT JOIN would be "show me all students, with their corresponding locker if they have one".
This might be a general student list, or could be used to identify students with no locker.
Returns 100 rows

Here is a good intuition on the difference between an inner join, and a left outer join.

SQL Statements

SELECT <list of column expressions>
FROM <TYPE> JOIN <…> on …
WHERE <list of logical expressions for rows>
ORDER BY <list of sorting specifications>



See all keywords associated with the book ‘Pattern Recognition and Machine Learning’:

```
SELECT p.Title, kw.Keyword
FROM Products p
INNER JOIN ProductKeyWords pkw on pkw.ProductID = p.ProductID
INNER JOIN Keywords kw on pkw.KeywordID = kw.KeywordID
WHERE p.Title = ‘Pattern Recognition and Machine Learning’
```

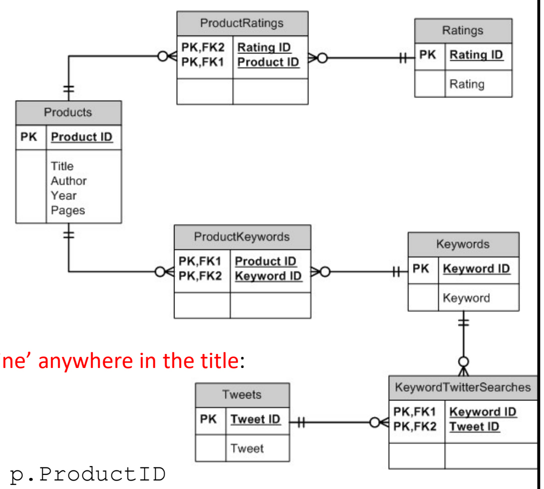
SQL Statements

SELECT <list of column expressions>

FROM <TYPE> JOIN <…> on …

WHERE <list of logical expressions for rows>

ORDER BY <list of sorting specifications>



See all keywords associated with any books that contain the word 'machine' anywhere in the title:

```
SELECT p.Title, kw.Keyword
FROM Products p
INNER JOIN ProductKeyWords pkw on pkw.ProductID = p.ProductID
INNER JOIN Keywords kw on pkw.KeywordID = kw.KeywordID
WHERE lower(p.Title) like '%machine%'
```

SQL Statements

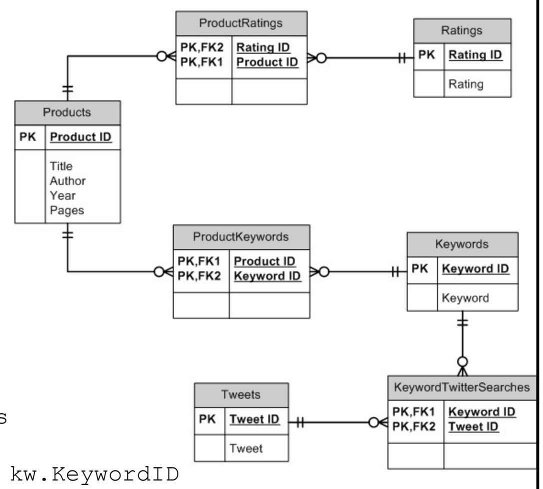
SELECT <list of column expressions>

FROM <TYPE> JOIN <…> on …

WHERE <list of logical expressions for rows>

GROUP BY <list of column expressions>

ORDER BY <list of sorting specifications>



See how many titles are associated with each keyword:

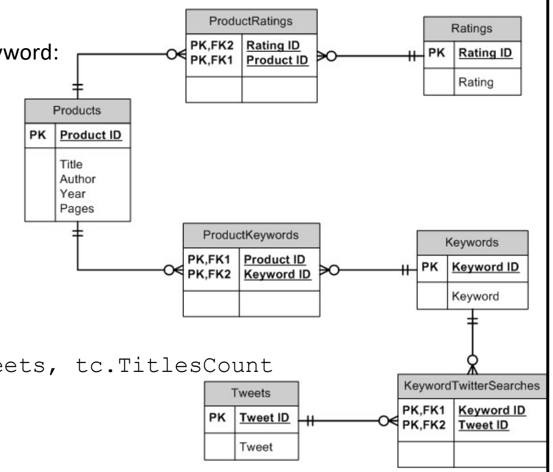
```
SELECT kw.Keyword, count(p.Title) as CountOfTitles
FROM Keywords kw
INNER JOIN ProductKeyWords pkw on pkw.KeywordID = kw.KeywordID
INNER JOIN Product p on p.ProductID = pkw.ProductID
GROUP BY kw.Keyword
```

here are group aggregations.
you want to have the group by operator to indicate which dimensions you want to aggregate the measures on.

New: Product, avg rating, countOf Tweets

SQL Statements

See how many titles and how many tweets are associated with each keyword:



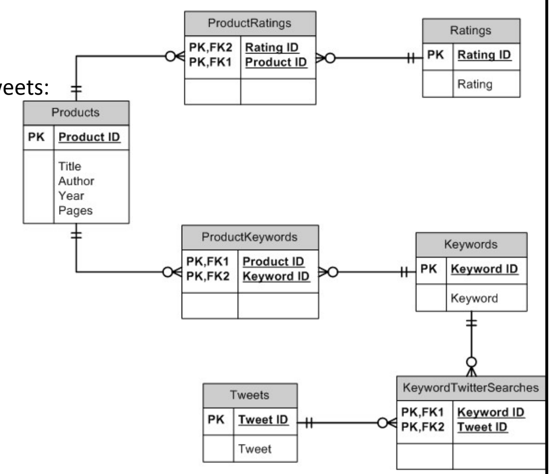

```
WITH TitlesCount as (
SELECT kw.Keyword, count(p.Title) as TitlesCount
FROM Keywords kw
LEFT JOIN ProductKeyWords pkw
      on pkw.KeywordID = kw.KeywordID
LEFT JOIN Product p
      on p.ProductID = pkw.ProductID
GROUP BY kw.Keyword
)
SELECT kw.Keyword, count(t.TweetID) as CountOfTweets, tc.TitlesCount
FROM Keywords kw
LEFT JOIN KeywordTwitterSearches kts
      on kts.KeywordID = kw.KeywordID
LEFT JOIN Tweets t
      on t.TweetID = kts.TweetID
INNER JOIN TitlesCount tc on tc.Keyword = kw.Keyword
GROUP BY kw.Keyword, tc.TitlesCount
```

here are group aggregations.
you want to have the group by operator to indicate which dimensions you want to aggregate the measures on.

This is the DRILL ACROSS trick.

SQL Statements (Exercise)

Grab a dataset to compare average product ratings with the number of tweets:



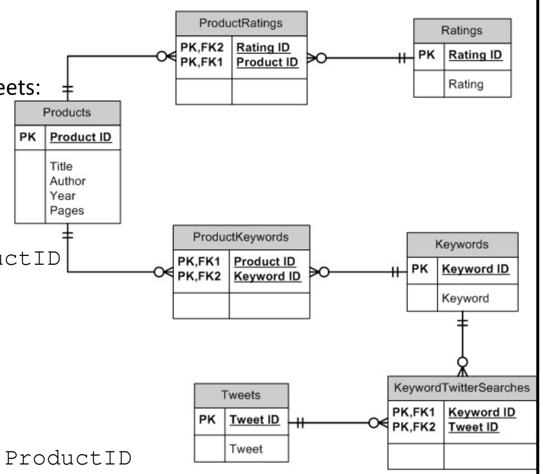
Here's an exercise that might take a slight bit of time. Give a couple minutes on it.

Look up how to do an aggregate average.

SQL Statements (Answer)

Grab a dataset to compare average product ratings with the number of tweets:

```
WITH AvgRatings as
(
SELECT p.ProductID, avg(r.Rating) as AvgRating
FROM Products P
LEFT JOIN ProductRatings pr on pr.ProductID P.ProductID
LEFT JOIN Ratings r on r.RatingID = pr.RatingID
)   group productID;
SELECT p.ProductID, p.Title
       , Tweets = count(distinct kts.TweetID)
       , AverageRating = a.AvgRating
FROM Products p
LEFT JOIN ProductKeyWords pkw on pkw.ProductID = p.ProductID
LEFT JOIN Keywords kw on pkw.KeywordID = kw.KeywordID
LEFT JOIN KeywordTwitterSearches kts on kts.KeywordID = kw.KeywordID
LEFT JOIN AvgRatings a on a.ProductID = p.ProductID
GROUP BY p.ProductID, p.Title, a.AvgRating
```



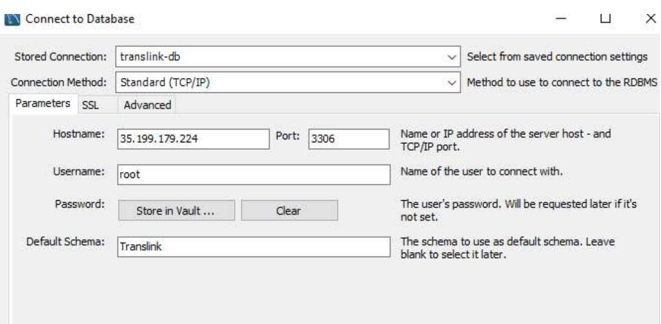
SQL Exercises / break

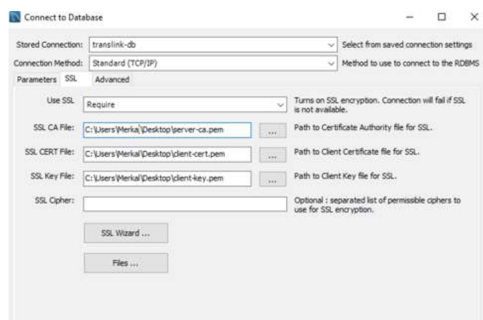
http://www.sql-ex.com/learn_exercises.php?LN=1

3:40 → 3:45

Connecting to AQM’s Database

- Hosted on Google Cloud
- Note down the username/password provided via slack
- MySQL Workbench connection manager:





Connecting from Python

Be sure to pip intall sqlalchemy and mysqlclient

```python
from sqlalchemy import create_engine
import sqlalchemy
eng = sqlalchemy.create_engine("mysql+mysqldb://AQMUser:@35.199.179.224:3306/Translink")
```

This is a view defined in MySQL by a query against a table in the Translink Schema. Now you can simply call on it here.

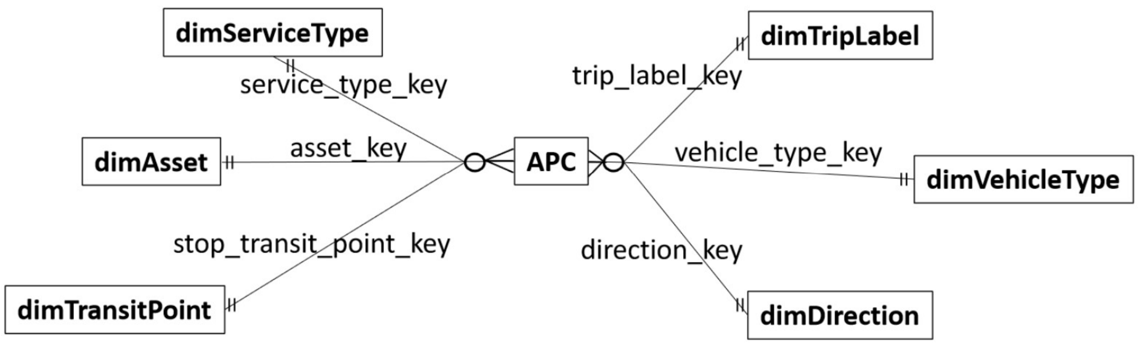
```python
dframe = pd.read_sql_table('MostPopular', eng)
```

Here is how to write back to the database creating a new table. Or adding onto it.

```python
dframe.to_sql('TestTablePleaseIgnore', eng,  schema='AQM'
              , if_exists='fail'
              , index=False
              , index_label=None
              , chunksize=None
              , dtype=None)
```

*Password redacted, but you know where to find it in the slack channel ☺
Typically this would be stored in a text file that isn't committed to a repository.

Translink Star Schema



Connecting to AQM's Database

Exercise:

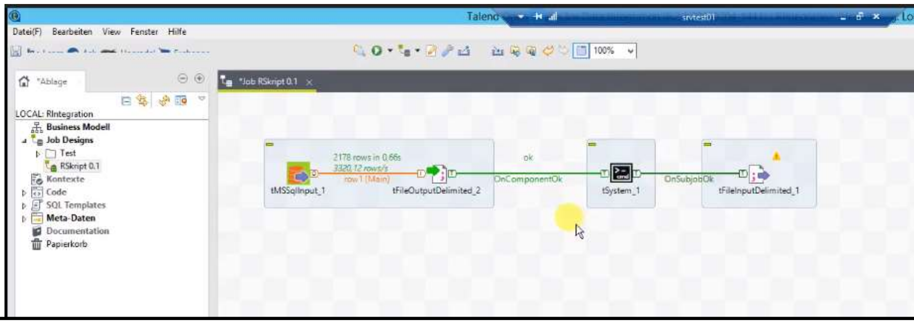
Recreate the "NextLegBunchingFlag" dataset on the week3 folder.

Select *
from Trips
where line = '010'
Limit 100

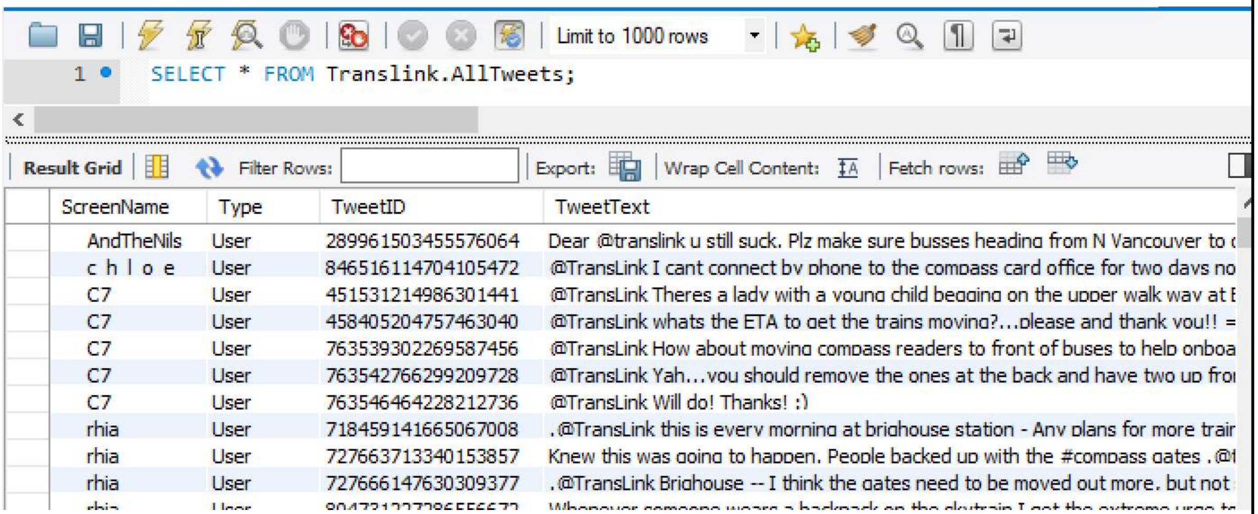
```
SELECT t1.*, t2.BusBunchingFlag as NextLegBusBunchingFlag
FROM Translink.Trips t1LEFT JOIN Translink.Trips t2 ON t1.OperationDate =
t2.OperationDate
                                                        AND t1.Trip = t2.Trip
                                             AND t1.TripLeg = t2.TripLeg-1
WHERE t1.Line = '010'
LIMIT 10;
```

Extract, Transform, Load (ETL)

- Tools
  - Spoon
  - Microsoft SSIS
  - Talend
  - A bunch of Stored Procedures



Extract, Transform, Load



Let's look at how this table is being generated.
You can't see it on here, but if you do query it on your computer (LIMIT TO 1000 ROWS), you'll see that there's the sentiment polarity associated with the message.

Extract, Transform, Load

- Daily Load example

```
CREATE DEFINER=`root`@`%` PROCEDURE `UpdateBatchLoadAllTweets`()
BEGIN

INSERT INTO AllTweets (ScreenName, Type, TweetID,TweetText,TweetDateTime
                        ,Followers,Friends,Statuses,Favourites,SentimentPolarity
                        ,Sentiment,EnsembleSentimentScore,EnsembleSentimentScoreSTDEV)
select stg.* from AllTweets_Stg stg
left join AllTweets a on a.TweetID = stg.TweetID
where a.TweetID is null;

TRUNCATE TABLE Translink.AllTweets_Stg;

CALL UpdateBatchLoadWinBacksGTID();

CALL UpdateBatchLoadFlattenedConvosGTID();

END
```

This would be called a Delta daily batch load.
This is probably the most common pattern of how you'd see your models feeding into an overall data architecture for when you want to prototype something like a fraud detection score, or overall sentiment of conversations, etc.

This is an insert statement. Fairly straightforward. You just say "insert into", name the table, and name the columns in parenthesis, and then following that write the query that represents that chunk of data that you're inserting.

TRUNCATE TABLE. That means we just want to clear the data in the table. But not ruin the structure of the table.

You'll notice that this then calls 'UpdateBatchLoadWinBacks'. And 'UpdateBatchLoadFlattenedConvos' afterwards. We call it afterwards because we only want those to run after loading the new tweets in the all tweets table. And we want to flatten the conversations after generating the winback structure.

So how does the AllTweetsStage table get its data, and what calls this stored procedure?

Extract, Transform, Load

- Daily Load example

```
SELECT * FROM Translink.WinBacks;
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

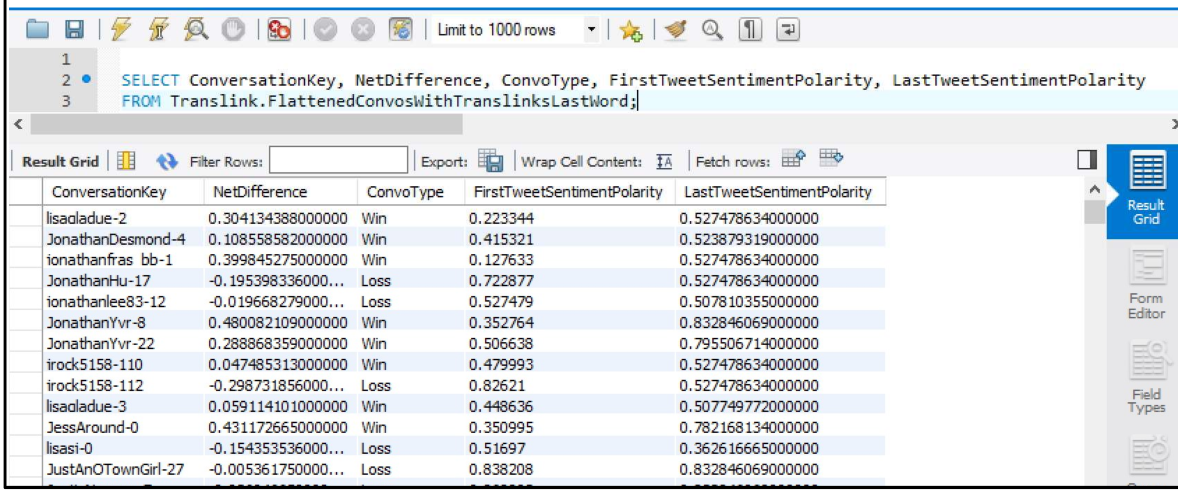
ConversationKey	ScreenName	TweetID	TweetDateTime	TweetText	SentimentPolarity
bitabitabita-2	bitabitabita	451123510669090816	2014-04-01 22:26:00	@TransLink the 25 just passed without sto...	0.34482035600000(
bitabitabita-2	Translink	451124745342562304	2014-04-01 22:31:35	@bitabitabita Sorry that happened. Please...	0.00000000000000(
bitabitabita-2	bitabitabita	451125569187373056	2014-04-01 22:34:00	@TransLink but r u hiring tho	0.52747863400000(
bitabitabita-2	Translink	451127244216619008	2014-04-01 22:41:31	@bitabitabita We do have lots of opportun...	0.00000000000000(
bitabitabita-3	bitabitabita	791037179807924224	2016-10-25 22:02:00	@TransLink lots of people waiting at main ...	0.57682523300000(
bitabitabita-3	Translink	791038249120698369	2016-10-25 22:06:30	@bitabitabita Thank you for letting us kno...	0.00000000000000(
bitabitabita-3	bitabitabita	791039355955523584	2016-10-25 22:10:00	@TransLink Yeah. A lot of elderly ppl who ...	0.58785986900000(
bitabitabita-3	Translink	791040832518447108	2016-10-25 22:16:45	@bitabitabita I have reached out to our o...	0.00000000000000(

Winbacks. Defines the conversations by the originating tweet user, and creates a key based on the 24 hour cycle. So we see here for example there was conversation 2. And it's sorted in order as the conversation took place over time. And we see the sentiment score that we can check before and afterwards.

Then we want to flatten it so the first complaint, and the last complaint.

Extract, Transform, Load

- Daily Load example



```sql
SELECT ConversationKey, NetDifference, ConvoType, FirstTweetSentimentPolarity, LastTweetSentimentPolarity
FROM Translink.FlattenedConvosWithTranslinksLastWord;
```

ConversationKey	NetDifference	ConvoType	FirstTweetSentimentPolarity	LastTweetSentimentPolarity
lisagladue-2	0.304134388000000	Win	0.223344	0.527478634000000
JonathanDesmond-4	0.108558582000000	Win	0.415321	0.523879319000000
jonathanfras_bb-1	0.399845275000000	Win	0.127633	0.527478634000000
JonathanHu-17	-0.195398336000...	Loss	0.722877	0.527478634000000
jonathanlee83-12	-0.019668279000...	Loss	0.527479	0.507810355000000
JonathanYvr-8	0.480082109000000	Win	0.352764	0.832846069000000
JonathanYvr-22	0.288868359000000	Win	0.506638	0.795506714000000
jrock5158-110	0.047485313000000	Win	0.479993	0.527478634000000
jrock5158-112	-0.298731856000...	Loss	0.82621	0.527478634000000
lisagladue-3	0.059114101000000	Win	0.448636	0.507749772000000
JessAround-0	0.431172665000000	Win	0.350995	0.782168134000000
lisasi-0	-0.154353536000...	Loss	0.51697	0.362616665000000
JustAnOTownGirl-27	-0.005361750000...	Loss	0.838208	0.832846069000000

WinbacksFlattened. Just so we can associate for each conversation quickly what the net difference is.

Extract, Transform, Load

- What calls this stored proc

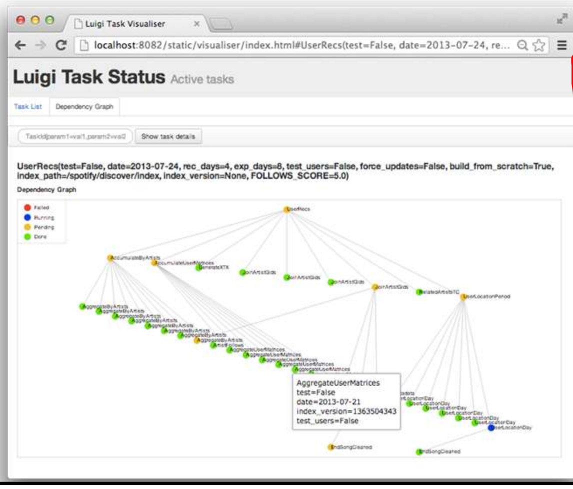
https://github.com/AQM-Repos/Translink-Dashboard/blob/master/ETL/BatchLoad.py

I used this as a link to scroll through the code in class as an example.
I'll add a stripped-down simpler version of this code to this week's folder to get the main point across.

Extract, Transform, Load (ETL or ELT)

- Dependency Graph example

https://github.com/spotify/luigi



.PY

StoredProc AT

WinBack

Flattened Con.

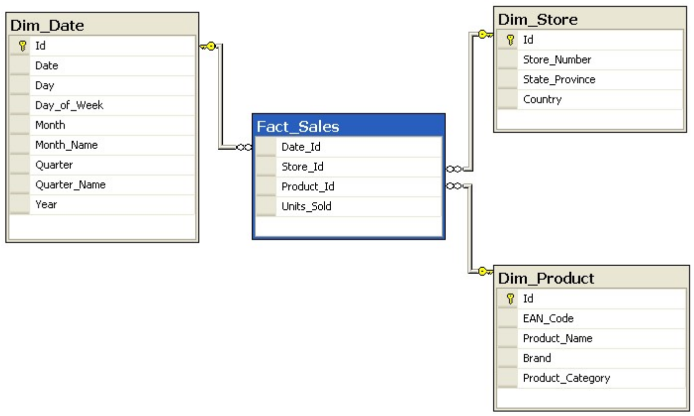
These steps can be drawn as a dependency graph. These can be quite useful. Our tiny application only really had two ETL pipelines going on. One to score winback in conversations that were initiated, and one that takes in things directed at translink regardless of if they responded.

As these dependency graphs get out of hand, and you have hundreds all talking to each other, if you're a machine learning engineer for a company like spotify or Airbnb, you have to start documenting these dependency graphs, in what you'd call a workflow orchestration tool. This way you can see if for example your new model prototype that changes a certain data structure, won't ruin the production of anything else.

This tool I'm screenshotting here is from spotify called luigi, and it's pretty good at helping you to document those task dependencies. And that can be useful too because your scheduling tool can take advantage of running the things that can run independently of each other in parallel. And the nodes that depend on something that didn't finish yet have to wait.

There's also another workflow orchestration tool called airflow by Airbnb which is also pretty good.

Schema Patterns



This is the star schema.
Sorry that it looks so 1990s. This is just something that Wikipedia used as an example.

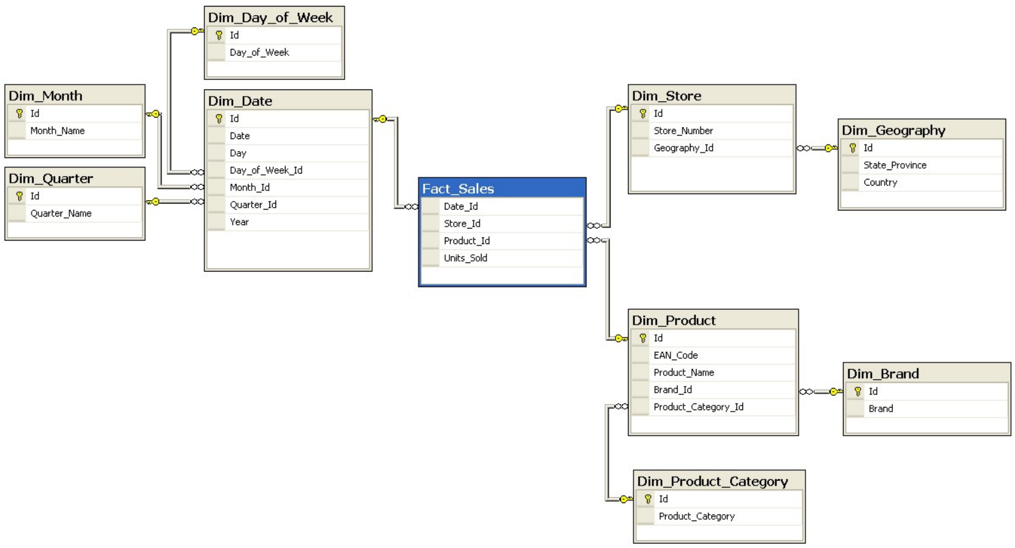
So this is a type of design pattern that you're probably going to see over and over again.

The reason is that it happens to be an easy representation and a good trade-off between normalization to maintain integrity in the system, and storing it in a way for speed in retrieval and aggregating.

This design also illustrates a point to be comfortable with: any piece of data is either a dimension or a measure.

Similar to the way in stats we say that a datapoint is either a category or a quantity.

Schema Patterns



This is the snowflake schema.
Just an extension of a star schema.

Historical Integrity for Dimension Tables

Type I Representation

Rider
CompassID
Email
Expected City
Expected Postal Code

Type II Representation

Rider
CompassID
VersionID
Email
Start Date
End Date
Expected City
Expected Postal Code
Current Ind

Type III Representation

Rider
CompassID
VersionID
Email
CurrentExpected City
CurrentPostal Code
PreviousExpected City
PreviousPostal Code

One more design consideration since we're on the topic of schemas that I think is relevant to a data scientist. Proposing how to represent your model output's historical integrity.

Let's suppose that translink wanted to store best guess estimates for compass card origin points, roughly assigning a postal code for where the customer lives in order to understand changing population commute impacts by service changes.

And they would store that in the customer table. What about when it appears that the customer has moved to rent in a new neighbourhood? If you were the mastermind with a stats or ML model that determined that likely origin point location heuristic, you could propose a few options for representing that, and representing the historical integrity of location changes over time with a few different representations.

Type 1. Just always keep the current.
Type 2. Store a version number, with a 'current version' 1/0 column, so you could see changes over time.
Type 3. Arbitrarily decide to represent it with current, previous, past columns with beginning and end dates.