Dimensional Data Modeling

Dimensional Data Modeling Day 2 Lecture
Data Modeling - Slowly Changing Dimensions and Idempotency

Transcript:

0:00
[Music] welcome to dimensional data modeling day
0:07
two lecture today we're going to be talking about slowly changing Dimensions so a slowly changing Dimension is an
0:13
attribute that drifts over time for example you can think of like your favorite food you know back in the early 2000s my favorite food was lasagna but
0:20
then I kind of became more lactose intolerant so I did not enjoy eating lasagna nearly as much and then after I
0:26
moved to the Bay Area it became uh Curry so you can see like that back in the day it was lasagna and then it changed to
0:31
Curry so there are attributes and dimensions that change over time like that and we need to track the values so
0:37
it was like from 2000 to 2008 uh it was lasagna and then it's been Curry since then or whatever you have like
0:43
Dimensions have a time frame with them slowly changing Dimensions do other dimensions don't change right like your
0:48
birthday your birthday is it's the same it's not something that you can really change about your life right it's it is
0:54
what it is and so some Dimensions change some Dimensions don't and you need to model slowly changing Dimensions the
0:59
right way because if you don't they impact this thing called item potency and item potency is the ability for your
1:06
data pipelines to produce the same results whether it's running in production or it's running in backfill
1:11

which is a very very important property of pipelines in order to enforce data quality and you're going to hear me say
1:17
the word item potent a million times in this presentation and definitely do more research on it because if you can build
1:23
item potent pipelines you will be a much much better data engineer and I promise that is going to get you a lot more
1:28
money I hope you enjoy the the lecture and if you want to find ways to build more awesome item potent pipelines in
1:35
the cloud definitely check out the data expert Academy in the description below okay
1:43
so item potent pipelines are critical if you have a data pipeline that isn't item
1:48
potent it's going to cause you a lot of pain and suffering a lot of pain and suffering so uh what does that mean what
1:55
does it mean to be um what does it mean to uh oh go back on
2:01
what does it mean to be item potent uh item potent pipelines okay this is this is the literal definition of item
2:07
potency right here is denoting an element of a set which is unchanged in value when multiplied or otherwise
2:14
operated on by itself and that's probably like hey uh
2:19
can can all y'all mute yourselves sorry um and so what does that mean that
2:25
Pro that definition is probably pretty terrible like at least from and like how does that even relate to data engineering right so uh let let's let's
2:32
dive deeper into this because this concept is very very important uh in
2:38
data data engineering so pipelines should produce
2:43
the same results regardless of the day you run it regardless of how many times
2:49
you run it regardless of the hour that you run it if you have all of your
2:55

inputs that are available and all of your signals are ready to go it
3:01
shouldn't matter if you run the pipeline today or if you run the pipeline in a
3:07
year or if you run the pipeline in 10 years it should still produce the same data because that's like the essence of
3:14
item potency is that like it's a function it's a mathematical function where you have these inputs and you have
3:21
these outputs and those things are very very important and data
3:28
engineers make mistakes takes a lot with these three these three
3:34
ways of doing things and we're going to we're going to dive really deep into each one of these categories and you can
3:40
save yourself a lot of pain and suffering if you just follow a couple best practices when you are building
3:46
your pipelines so that you get these results because imagine imagine this
3:51
right if you have a pipeline that's not item potent say you run it today and then you wait a week and then you
3:57
backfill it and what would happen is you'll you'll actually get different data like you'll have different results
4:03
even if you don't change the code you'll have different data and that can be a very painful thing that can happen and
4:09
that can cause so much pain and suffering for your analytics teams and so much like data discrepancy issues and
4:16
people are like why does this number not match this number and all there's a lot of pain and suffering that can happen if
4:22
you don't follow these rules so let's go a little bit deeper into each one of these
4:27
buckets okay so why is why is it hard to troubleshoot
4:34
first so if you if you have a pipeline that's not item potent uh it doesn't fail it actually

4:42
doesn't fail it just produces different data it just produces incorrect data or
4:47
different data and or not it's the I don't like to use the words incorrect or
4:52
different I like to use nonreproducible that's the more correct way of defining this is you have a a
5:00
pipeline that it it doesn't produce the same data depending on when you run it so that can be a very big problem and
5:08
and the way that this shows up for the most part is when a data analyst will
5:15
come back to you and be like why do why does why does the data look weird like what is going on like why do these
5:21
numbers not match why when I look at this table in this table these numbers don't match and that could be because
5:26
you backfilled it recently and it's not item potent or like it can all it can also be the other way around where you
5:33
are depending on a non- item potent pipeline or a non- item potent data
5:38
table and then that puts you at the mercy of them so that's the other thing about this is that like if you have
5:44
Downstream data Engineers that are depending on your data and your data isn't item potent then their data will
5:50
also not be item potent It's like because it follows the transitive property like that so you can cause
5:57
inconsistencies to bleed throughout your entire Warehouse if you don't follow these best practices so and and you'll
6:04
also kind of piss off your analytics teams and you'll cause them to lose trust in the data sets because they're like what's going on here like why why
6:10
don't why can't these numbers match so we're going to go into a little bit more details on like what happens or what
6:17

causes this this uh property of pipelines to happen remember the one of the ones I said was a pipeline should
6:23
produce the same results regardless of how many times it's ran this is a big one this first one
6:31
here uh this insert into without truncate so what that means is like say
6:37
you're running the data for a day and you have insert into and you run you run the pipeline once great you run it twice
6:45
now there's twice as much data so like you notice how uh like the if you don't
6:51
have a truncate statement that clears out the data that you're about to insert
6:57
then you're just going to keep duplicating the data over and over and over again and then uh your pipeline is
7:02
no longer item potent because it doesn't produce the same data regardless of how many times it's ran that rule from the
7:10
slide before so insert into is uh kind
7:15
of a bad one right don't my my my rule as a data engineer is insert into should
7:21
never ever ever ever ever be used like unless you're using truncate 2 generally
7:26
speaking I don't like insert into at all like I'm all about mge merge and insert overwrite merge and insert overwrite are
7:32
going to be your way better ways to go so merge what it does is it takes your
7:38
new data and it merges it in with the old data but then if you run it again
7:43
you you you you don't get any duplicates right because then it just it it notices that everything matches and nothing
7:49
happens and the second time you run the pipeline you get you have the same data set and you can run the pipeline as many
7:55
times as you want and you'll still have the same data set you won't have like um you know n number of rows where n or or
8:02

n times the number of rows where n is the number of times you ran the pipeline you won't have that right that's where
8:07
merge is very very powerful uh insert overwrite is also very powerful uh with
8:13
insert o insert overwrite is sometimes even better where like what you do is instead of matching all the records you
8:20
just overwrite that partition so whatever data was in there you just overwrite it with the new data because
8:26
then every time you you overwrite you just keep overwriting the data and overriding and overriding this Essence
8:32
like this insert overwrite Essence like if y'all know about Hive metastore and
8:37
uh like all the file formats from back in the day with like Hadoop And Hive metastore this this property is why
8:45
everything is insert overwrite because you really want things to be item potent in that way so let's talk about the
8:53
second bullet you see how it says using start date greater than without a corresponding end date less than so what
8:59
that that means is like say you have a pipeline with a wear clause and you say where date is greater than today okay if you run it um or or or
9:08
let's say where date is greater than yesterday that's a probably a better example if you run it today right then
9:13
that means you get one day of data but if you run it tomorrow you'll get two days of data and if you run it in the
9:19
next day you'll get three days of data and every time you run for that previous day you're going to get one more day of
9:24
data and like and that is a problem because again it's not item potent you
9:30
your pipeline is not producing the same results regardless of when it is Ran So now so now we're talking about when
9:37

instead of how many when it is ran this is a problem right because you really
9:42
need to be looking at a window right so that's why you have to have the great
9:47
the end date less than so you have like a window a chunk where you say I want to process seven days of data or I want to
9:53
process six days of data not I want to process an unbounded number of days of data because that is going to cause all
10:00
sorts of pain and suffering and don't do it and it it has a different problem this one here has a different problem
10:07
which is you can also end up creating um out of memory exceptions uh when you
10:12
backfill so definitely remember to always put an end date on your pipelines
10:18
as well as so that you have like a a window uh another big one is not using a
10:24
full set of partition sensors so what can happen here is
10:30
your pipeline is going to run and maybe it's going to run with an incomplete set
10:35
of inputs where it you aren't checking for the full set of all of the inputs
10:41
that you need for your pipeline so it runs and it actually runs too early it runs before all the inputs are ready and
10:48
then it and then that causes an a problem because then when you back fill it right the the data is probably there
10:55
and uh it will run with the full data set then but then in production when it's running it it it fires too early
11:02
and it's missing one of the input data sets and that can cause another set of problems where uh your backfill and
11:09
production runs of the data pipeline do not produce the same results and that can be another painful painful
11:16
problem and this last problem uh is oh it's called depends on past uh depends
11:22

on past is have what is called an airflow uh it can also another term for
11:27
it is a sequential processing so if you have a cumulative pipeline from uh from
11:34
yesterday you know how we talked about uh today and yesterday and how we need a full outer join those tables and then
11:41
that produces the new data set so if you have a pipeline that depends on yesterday's data that means that the
11:47
pipeline can't run in parallel it has to run like yesterday then today then
11:52
tomorrow then the next day where most pipelines aren't like that most pipelines you can run all the days in
11:57
parallel when you backfill and it can run a lot faster but you can't do that with cumulative pipelines and that cause
12:04
and if you don't if you don't put this sequential processing in your cumulative pipeline uh one of the things that can
12:10
happen is you can end up getting uh it can end up reading in yesterday's data
12:17
and yesterday's data hasn't been created yet and because you're processing things
12:23
in parallel and since yesterday's data hasn't been created yet you're going to remove you're going to essentially start
12:28
over on your cumulation and you'll just have today's data and so that can be another very important thing that you
12:35
need to put in your pipeline to make it item potent so that the production or the production Behavior and the backfill
12:42
behavior of the pipeline is the same if it's uh that's the big thing remember
12:48
when I use this word item potent production and backfill are the same that's the beauty of item potent
12:57
pipelines okay so we have some more things that uh can make a pipeline non item potent so this one is going to be
13:05

great uh we're going to um this is where I'm going to talk a little bit about uh one of the very last problems I worked
13:11
on at Facebook before I I quit this this problem actually was so painful that
13:16
like it was definitely one of the big things that um made me really frustrated uh so one of the last things I worked on
13:23
at Facebook was I built this uh data model that tracked fake accounts because
13:28
an account can like be labeled fake and then it can be unlabeled fake because they passed the challenge because it can
13:34
be like there can be a false positive but then they could be labeled fake again because they start doing sketchy
13:39
stuff again and like they start acting like a fake account even after they pass the challenge so they can be relabeled fake and there's all sorts of different
13:45
states of a fake account where it can be like a new fake account a reclassified fake account there's all these different like uh States for a fake account and I
13:53
was building out this this this this chart that like tracked the inflows and outflows of fake account
13:59
and uh one of the things about that was there was a table called Dim All fake accounts and that table was not item
14:06
potent so what that table did was instead of relying on today's data what
14:13
it did was it relied on the latest data from the pipeline and uh and the reason
14:19
why they did that this is because they wanted the data to arrive as quickly as possible they wanted the the fake
14:24
account data to be ready as quickly as possible which is one of the nastiest
14:31
shortcuts that you can take as a data engineer uh is is like if you like if if
14:36
you have a a a problem where you really are getting pressured to be like I need
14:41

this data as quickly as possible every day and I need to reduce the landing time every day don't do this just please
14:48
don't do this like this is like not the way to go this is going to be uh this causes all sorts of problems so let me
14:54
explain what happened in this case so you have a dim all fake accounts which was relying on the latest data so uh of
15:01
of this other table called Dim All users which is a table that I talked about in the last um class uh which was the table
15:08
that just had all the users in it and their growth metrics and stuff like that so some days demm all users would be
15:14
ready and then demm all fake accounts would pull in the data for today and it would be right some days demm all fake
15:20
accounts instead of pulling in the data from demm all users uh from today it would pull in yesterday's demm all users
15:26
because demm all users was behind for what whatever reason so that means that dimol fic accounts sometimes used
15:32
today's data and sometimes used yesterday's data which which I don't
15:37
know like for some purposes that's okay that's totally okay for analytical purposes that's usually not okay because
15:44
you need to you want to track the inflows and outflows of these accounts and you need to not be looking at this
15:49
stuff in such a way that uh you you're caring about data latency too much that
15:54
you sacrifice data quality so anyways I built out all these charts and all these State Transitions and then the analytics
16:01
team comes back to me and as like why why why are your numbers not reproducible Zach why don't they match
16:07
with demm all users and I'm like and like I'm looking at all my code and all
16:12

my SQL queries and I'm like I have no idea like I I was looking at this for
16:17
like over a month to like just dig through and try to figure out like why my numbers didn't line up cuz it made me
16:23
feel like such a bad engineer because I was like I felt like I was doing everything right where I wrote All my
16:29
code the right way and like that and like I was like there's no way my sequel is wrong and then it took me a long time
16:34
to figure out like oh it's actually no it's my Upstream team that actually did this right and I got very frustrated from this I got very very frustrated
16:40
from this I ended up quitting Facebook about uh six seven weeks later because this was just like such a such a painful
16:45
problem for me to to look at so also that's another thing that can happen if you have a non item potent pipelines
16:53
that cause data quality bugs uh cause your data Engineers to quit so um
17:00
there is one exception here there is one exception here for relying on the latest
17:05
partitions and that's is if you are backfilling so if you're backfilling
17:11
data and you have a properly modeled SCD table then what SCD standing for slowly
17:17
changing Dimension if you have a properly modeled slowly changing Dimension table and you're back filling
17:24
not in production but and you're back filling then you can rely on the latest data that's the only exception to this
17:30
rule every other data set should never do this so we're going to be talking
17:35
more about this in the lab and all the stuff as well so if uh I'm I'm covering a lot right now that that that's how it
17:42
is um so that that's going to be the big thing that happens right so one of the
17:47

last things I want to talk about in on this slide is if you have these cumulations right so say you have a non
17:52
item potent pipeline that introduces a bug because it it it gives the wrong data or it gives the um the the
17:59
incorrect data based on whether it was backfield or in production it just gives the a inconsistent data and then you
18:05
have a cumulative pipeline that depends on that data then what ends up happening is that cumulative pipeline just carries
18:12
those bugs forward every day every day every day and then you have to start over with a cumulative pipeline if your
18:18
cumulative pipeline depends on a non- item potent pipeline then it it's going to be very painful so that's another big
18:25
thing that I learned about um data engineering uh when I was near the end of my time at
18:32
Facebook okay so I'm sure you've noticed like there's been item potent on almost
18:38
every single slide so far so these are kind of a couple of these things I've
18:43
mentioned we're going to be able to go over a couple more of these right so remember backfilling right if if your
18:49
pipeline's not item potent backfill and production are not going to create the same data you're going to you're going
18:54
to essentially overwrite the data and it will be new and it will be different and that's like very painful very very
19:01
painful and that's like it prod produces hard to troubleshoot bugs the only way that you end up seeing them is if an
19:07
analytics team really Dives deep into the sequel and really figures out like oh wow this is a
19:12
problem um then you have unit testing can't replicate the production Behavior so a lot of times like data Engineers
19:18

are like well it's fine because like I can just unit test my pipelines and it's like no because it's like if you write
19:23
unit tests and your pipeline isn't item potent your unit tests will still pass
19:29
and you can't fix that problem but if your pipeline is written in an item
19:35
potent Way Unit tests are better because then they they can essentially guarantee the quality of your pipeline and that
19:41
your your pipeline is going to be producing the correct results and the last point which I talked about in an
19:46
earlier slide silent failures and you have failures upon backfill failures upon restatement all of those things
19:54
like they are the most one of the most timeconsuming and painful parts of data engineering so definitely remember that
20:02
that is the case here and that like this problem like is is is it's interesting
20:07
because the the things that you have to do here are very small but they they
20:13
matter a lot so yeah let's go to the next slide okay so we're shifting gears here
20:20
a little bit um we're going to talk about uh slowly changing Dimensions here
20:27
right so first off I'm going to just give an overview of like what a slowly changing Dimension even is and then we
20:34
can kind of go on from there so a slowly changing Dimension is a dimension that
20:40
changes over time so one the the most obvious one here is like age you can
20:47
imagine like my age is a dimension I'm 29 now I'll be 30 next year right all
20:52
sorts of like ages that's a that's the very like that's like the poster child of slowly changing cuz that one changes
20:58
it very like regular Cadence but you have other ones too like right now I'm an iPhone user but back in the day I was
21:04

an Android User it's another one you can kind of switch right uh even country
21:10
right I mean some people like I know they used to live in like the Dominican Republic and then they move to the US
21:15
they move to New York right you have all sorts of different like movements of people and you can have like all sorts
21:21
of different paths that people can go on and like so country can also be a slowly changing Dimension uh there's all sorts
21:28
of different dimensions like that like some Dimensions aren't though right some Dimensions just straight aren't like my birthday is not that's set in stone
21:36
right my eye color for the most part is set in stone I might be able to like bust out like a surgery or something like that but like for the most part
21:42
it's probably set in stone right so but generally speaking most dimensions are
21:48
slowly changing um there's also like this concept of like a rapidly changing Dimension which is something to think
21:54
about like for example a rapidly changing Dimension might be like heart rate like minute to minute because it's
22:02
like that's going to change over time and that's like a that that you could consider that both like a metric and a
22:07
dimension it's kind of a one that's in the middle but that is where how you
22:12
model these things and how quickly they change like how like because like when you say slowly changing you can always
22:18
be like well how slow are we talking right and like the key thing here is if
22:24
you want to model it as a slowly changing Dimension the slower changing
22:29
it is the better the and that's where you get the most efficiency gains is if
22:35
you have this the the most slowly changing it can be it's a a beautiful thing to keep in mind um so uh like I
22:45

want to talk now about what's actually on this slide so I have a good friend Max Max and I had a little bit of overlap at Facebook he also created
22:52
Apache air flow and uh he uh wrote this whole like Manifesto about how slowly in
22:58
dimensions are terrible and that you should never model them this way and uh his whole point is that slowly changing
23:04
dimensions are inherently non item potent which in some in some way he's he's right because he's all about like
23:11
the this idea of functional data engineering which is a lot of what I've been talking about here is like you have
23:18
you treat your pipelines as functions you have your inputs and your outputs and it just works so in Max's opinion
23:24
scds shouldn't really exist and you should just have every day you have whatever your dimension value is right
23:30
like it's like today I'm this age tomorrow I'm this age the next day I'm this age right you just you it's kind of
23:37
duplicated because it's always the same value but like his whole his whole opinion is well storage cost is so cheap
23:44
but the the cost of fixing these data quality issues and introducing non- item
23:49
potent pipeline stuff into this realm can be uh the cost there is so high that
23:57
like you might as well just stick with daily dimensions and just go with that so let's talk about the the the three
24:04
different ways that things can happen right like there's of like how you can model your Dimensions so you have uh you
24:10
can think of them as like you have your latest snapshot so instead of modeling it day by day you just have like
24:15
whatever the current value is and then that's what you use the problem with that one is if you have a slowly
24:21

changing Dimension and you only hold on to the latest value then that means that like the pipeline is non item potent
24:28
because if you backfill then like my like say I'm 29 right now and we
24:34
backfilled my Facebook data in 2012 when I was like you know 18 and like then
24:40
that data would then pick up that 29 Dimension and then it would be like Zach made this really really terrible post on
24:46
Facebook when he was 29 like he should have known better but actually I was a teenager right and so like this is where
24:51
uh latest snapshot can be a problem it can really put you in some some hot water and might even get me canceled so
24:58
yeah be careful with uh latest snapshot then obviously you have like daily snapshot this is Max's Max is all about
25:05
daily snapshot because then in that case instead of uh the latest snapshot you
25:10
use daily so then when we backfill 2012 we pick that daily Snapshot from 2012
25:17
and then that will pull in the dimension 18 and then I I don't get canceled because then they'll be like Zack was
25:22
just a teenager so it's whatever and then the last the last way to model is
25:27
slowly changing Dimension we slowly changing Dimension is essentially a way of collapsing those daily snapshots
25:35
based on whether or not the data changed day over day and you essentially say like okay like for the entire year right
25:43
instead of having 365 rows where it says I'm 18 we say we have one row where it
25:49
says I'm 18 from January 30th you know um of one year to January 30th of
25:56
another year and so you have one row instead of 365 rows and it collapses it
26:02
down and that's where like that's why like if if a mo if a dimension is really
26:08

slowly changing that's when you really get that compression that I'm talking about here where um you and if it's even
26:17
more slowly changing then like you get even more compression but like if it's more rapidly changing you might like at
26:23
some point you might as well just do daily because it's like if if it changes once a week then like you're only going
26:29
to collapse down like you know a couple rows every day as opposed to like um
26:34
collapsing down um like 300 or 400 rows so that's a key thing to remember when
26:41
asking the question should I model this as a slowly changing Dimension and if it's not changing very quickly then uh I
26:50
I highly recommend that could work out this is where me and Max actually see the world a little differently because I
26:55
am a fan of slowly changing Dimensions cuz I really like that compression that you see when using these things so yeah
27:02
let's go to the next slide okay so why do Dimensions change
27:08
uh all sorts of different things right uh our preferences change we change countries we change phones right we
27:14
change all sorts of stuff like that there's all sorts of different things like for example like when I worked at
27:19
Netflix uh almost everyone at Netflix had a sticker on their laptop where they were either um it was either a catflix
27:27
sticker or a dog flicks sticker and like it's crazy because like back then I
27:32
actually was I had I was on team cat flicks dude like I've changed I've changed as a human like I'm I'm I I now
27:39
looking back on that I'm like I am I'm emphatically a team dog flicks now and I'm like when that's just a another
27:45

great example of a slowly changing Dimension that happened for me right you can kind of you know alternate between
27:52
like all sorts of different things and I could you know maybe maybe in the future I'll be a cat guy again you know doesn't
27:57
mean that I I I'm I'm married to team dog now like it's like it can it kind of
28:02
flows all the time right so remember that like Dimensions can change for all sorts of different types of
28:10
reasons okay so how can you model Dimensions that change so there's really
28:17
three ways to model them uh like or yeah three ways and like one of them has
28:23
three Subways so remember I was talking about the latest snapshot and that problem where like it would if you
28:30
backfilled the data uh like and you only had the latest value then all the all
28:35
the dimensional values of your old data would pull in that latest value which might not be correct for the older data
28:42
because you might need the the old version of that Dimension not the new version so for the most part like never
28:50
do this uh never ever do this like you see how like that's in all caps uh
28:55
you'll make Zach really really sad if you back fill with latest uh only the
29:00
latest snapshot of Dimensions like that's a very like that's that's a nasty nasty data engineering no no um then you
29:07
have the daily partition snapshots that's um Max's way to go right um which
29:13
can be a very powerful way to uh build out your pipeline it's very simple very basic very like easy to understand where
29:20
it's like every day I have like every day I have a value for my Dimension and we just use that for that day and very
29:28

straightforward and then uh the last one here is you have um the slowly changing Dimension modeling and there's three
29:36
types you have type one type two and type three slowly changing dimensions and we're going to go into more detail
29:43
on each one of these different types of Dimension slowly changing Dimension
29:48
modeling okay there's actually technically uh SCD type zero as well there one 0 1 two and three and uh type
29:57
zero are actually not slowly changing Dimensions so if you are sure that your
30:03
dimension is not going to change or it's like kind of fixed in stone then you can model it as type zero and you can do
30:09
this because it's just like that's fine and like you won't you won't get the wrath of Zach like if you have that if
30:15
it's generally if it's genuinely a dimension that will not change then doing this is totally okay and this is
30:21
where like you essentially have a table that has like the identifier of the entity and then then the dimension value
30:28
and that's it like that's the whole table like there's no uh temporal component to the table cuz you don't
30:33
need it because the value never changes then you have type one which is
30:40
the one I hate the most I hate type one the most this is the one where you you only you just don't care like you just
30:47
say okay the value changes but we only care about the latest value uh don't use this like this is where like uh data
30:54
modeling actually has like uh an interesting debate right where um for like online transactional
31:01
processing like oltp which is um what I talked about in the last class uh where
31:07
you're doing the data modeling for the online apps then using the latest value
31:13

is probably going to be fine because you never really need to do look at the past
31:19
because online apps really only care about like the current value in the current state of things so in the online
31:26
World type one is probably okay but as data Engineers who care about analytics
31:32
which is almost everyone in this meeting uh you should not do this like because
31:37
it makes your pipeline not item potent anymore and that is uh a lot of pain and suffering that you will end up dealing
31:43
with so yeah's uh let's go to the next slide all right this is my favorite type
31:50
two UHD type 2 is um what um Airbnb
31:55
calls the gold standard of slowly changing Dimensions so how this works is
32:02
instead of modeling your data as just the latest value what you do is you have
32:09
a start date and an end date right so imagine like for me my favorite food from like 2000 to 2010 was lasagna and
32:17
so like there would be a record that's like lasagna 2010 or 2 to 2010 and then
32:23
it changed to a curry and then you have 2010 to 2015 and then then you have two
32:28
rows that show my values at various points in time and so these are great uh
32:35
SCD type 2 uh dimensions are really awesome they have a lot of really good
32:41
value to them and they really show really great um they they don't lo you don't lose any
32:49
uh Clarity in the dimension and they are item potent because of the fact that you can go back and then you essentially
32:56
like if you go back in the the past right so back to 2012 what you would do is you say Okay 2012 is between this
33:04
start date and this end date so that means it's this dimensional value and we pull out this the the value in between
33:10

start date and end date and that is great so one of the things that is um uh
33:16
interesting about this is for the the current value uh what Airbnb does is you have the current value has a start date
33:22
and an end date and usually how it works is the start date is whenever it changed and then the end date is just really far
33:28
into the future like it's like the year 9000 like they they like the 999 1231
33:34
like December 31st of the Year 9,999 uh that's the value that they put
33:39
some uh there's a lot of debate on this like I've posted about this on LinkedIn and people say some people like to use
33:44
null for the end date instead uh a lot of times with type two there's another
33:50
column as well which is is current which is like a Boolean that is like the is this the current value for this table so
33:58
this is uh one of my favorites and we are going to be working with this in the lab today we're going to be creating a
34:04
type to slowly changing dimension in the lab today so that should be fun so let's go over a couple more uh slowly changing
34:11
Dimension options right so that like you can kind of understand that like there are even more ways to do this um okay so
34:19
type three is uh where instead of uh holding on to all
34:26
of history with like a start dat and an end date you only hold on to two values you hold on to the original value and
34:32
the current value and that's all you hold on to and it's uh like it's all
34:38
right I feel like this is like a gross Middle Ground a little bit because if your dimension changes more than once
34:45
then you are done right you don't like you don't know a and another important
34:50
thing is like this usually doesn't store when the things change sometimes they

34:55
have like four columns you have like the original date and then you have like the current date the current change date so
35:01
that you still have one row so that's the benefit here right is you still have one row per Dimension so you don't have
35:06
to do any filtering like in SCD type 2 you have to filter on that start date and end date and pick the depending on
35:13
what date you have in mind you have to pick a pick a value but in this case
35:18
like you don't because you have one row but then you have to know if it's the original or current and that's where
35:24
this uh S Type 3 fails the item potent test because of the fact that if
35:30
something changes more than once then uh you lose that history and then back
35:35
filling will produce incorrect results and you because you lose that Clarity so that's going to be another big thing to
35:42
think about as you kind of go through this uh Journey here um okay so to recap
35:49
here uh type zero and type two are item potent type zero is only item potent if
35:57
the the dimension is unchanging right it's a dimension that cannot change then it is TR if it's a truly unchanging
36:03
Dimension it's item potent type two is item potent but you got to be careful you got to be careful about the start
36:09
date and the end date type one not item potent latest Dimension doesn't work
36:15
right because you'll pull in all those like the those latest values into the old data when you backfill terrible type
36:21
three is kind of like in the middle like and it still does not quite give you what you're looking for so
36:28
what I'm mainly trying to say about uh slowly changing Dimensions here is you should only care about type zero and

36:36
type two so and there's actually other types there's like type four and five and six
36:41
and there's like there's a lot of other types as well but like they're very very rarely used in any like data engineering
36:47
context for the most part so I would pick between these mainly type zero and type two are the ones that you should
36:54
really be looking at and really uh like honing your skills in we're going to be shifting gears here a
37:00
little bit about all those types all those things that we were just covering and we're just going to talk about
37:07
um uh slowly changing Dimension type two
37:12
so there's two ways that you can load these tables right one way that you can load the tables is through one giant
37:21
query that crunches all of the data all of that daily data and it crunches it down uh that's one way to do it uh the
37:29
other way to do it is you do it in a cumulative way where like you have the data from the previous date and then you
37:36
bring in the new data from uh the current date and then like so that you only process like one new day at a time
37:44
and so generally speaking you want to have your production run be the ladder
37:49
where you do it incrementally because then you don't have to like process all of history all the time um it's
37:55
interesting though because like these two options are not necessarily like
38:00
better or worse than each other especially like if if your data set is small then like these two options can be
38:06
essentially the same whether you uh generate the SCD every day or you load
38:12
it incrementally they can be about the same I know for me that was a big thing I noticed when I worked at Airbnb was uh

38:19
I had this pipeline uh unit economics where we had uh it was an SCD table because like imagine you have like uh
38:25
someone pays right and they have their fees but then they get a refund so then the value of that changes over time so
38:33
one day they they have their pay and we have we count that as profit but then they get a refund later so then we have
38:38
like the the the start date and end date of that value for their line items and
38:44
uh I always had this dream where I was like I'm going to make it so unit economics loads incrementally but I
38:50
never got there because like it didn't matter that much right and I already had everything working with the load entire
38:56
history I always felt like a little bit of pain there cuz I was like wow like every day we're just processing all of
39:01
the transactions back to 2016 like wow like this is a lot like this is painful
39:07
but at the same time it's like it is what it is right and like um is because for me as an engineer it's like is that
39:13
the most important thing to work on is to make that pipeline a little bit more efficient or should I be spending my
39:18
time working on other things for the business that could be more valuable and that's a big thing that to remember as a
39:24
data engineer is you don't want to get caught up with the idea that every pipeline that you build has to be a
39:30
Ferrari and it has to be like perfect right it has to have like all the efficiencies and be as efficient and and
39:37
as beautiful as possible because what that means is you're wasting time on like marginal value when you could be
39:45

looking at new stuff and that's a big thing to remember throughout your career as you're building stuff and creating
39:51
things is like yeah you might be able to get the data as small as possible but like in doing so like what was the
39:57
opportunity cost of doing that and that's a big thing to remember as you go through your career and just like in
40:03
that Airbnb unit economics example like I worked on other things I worked on like pricing and availability and these
40:08
other really important parts of the business instead of working on this incremental thing and like I definitely
40:14
found more value there so that's just a you know side you know little tangent there on like career and prioritization
40:21
of things [Music]

English (auto-generated)

All

From the series

From Data with Zach

Related

Watched

Learning