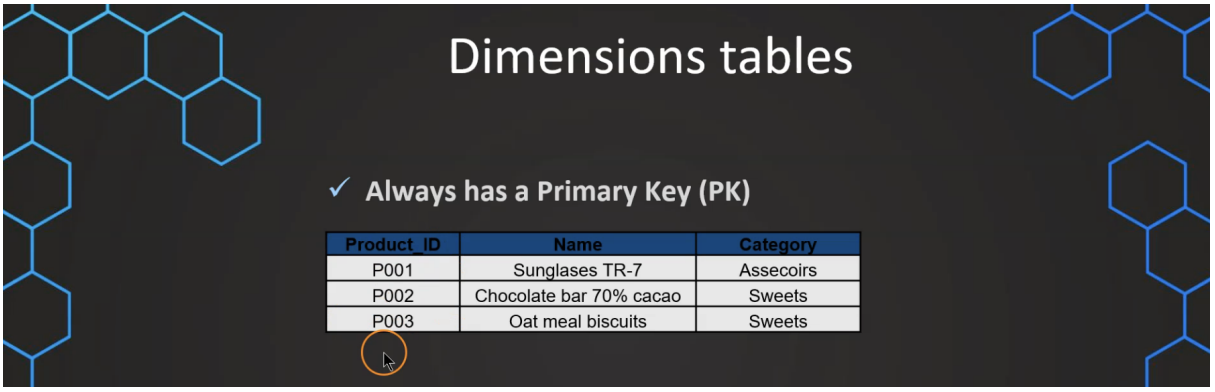
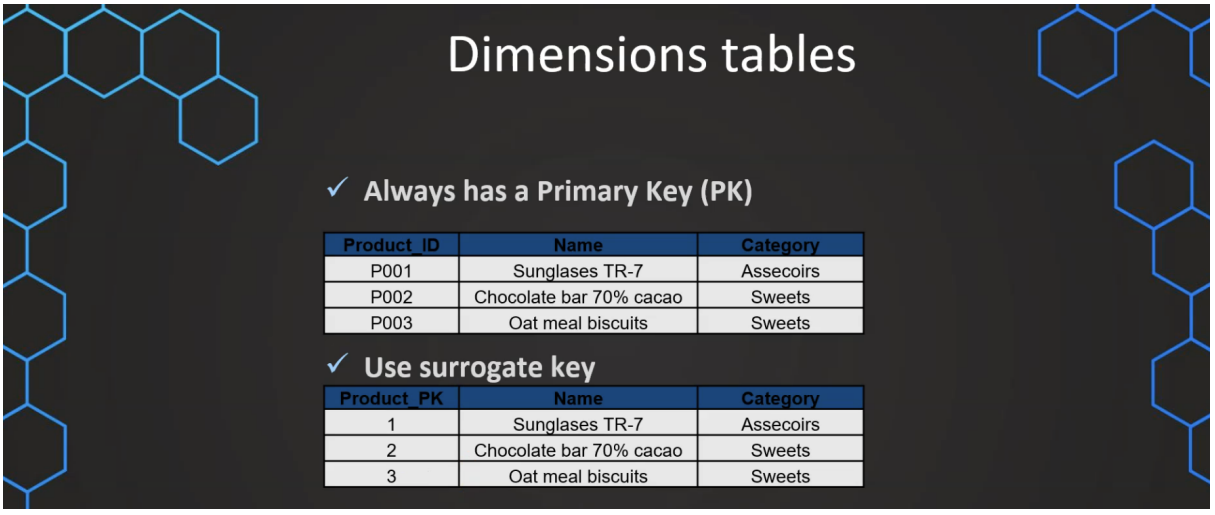
07. Dimensions

- Dimension table is used to slice and dice our data.
- We have learned that a dimension table always needs a primary key. And in below case we can see that there is a natural key which is coming straight out of the source system.

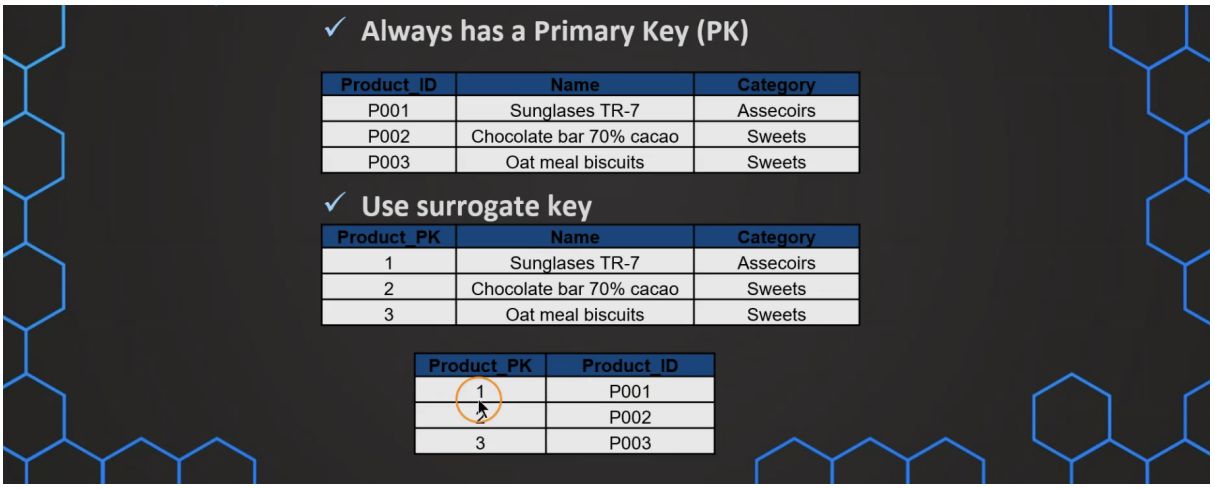


- But this is not the best way of a primary key. We should rather replace those primary keys and use surrogate keys.
- Surrogate key is usually a integer number that is just increasing one by one. And this has, as we have learned more benefits and therefore we should rather use these surrogate keys.

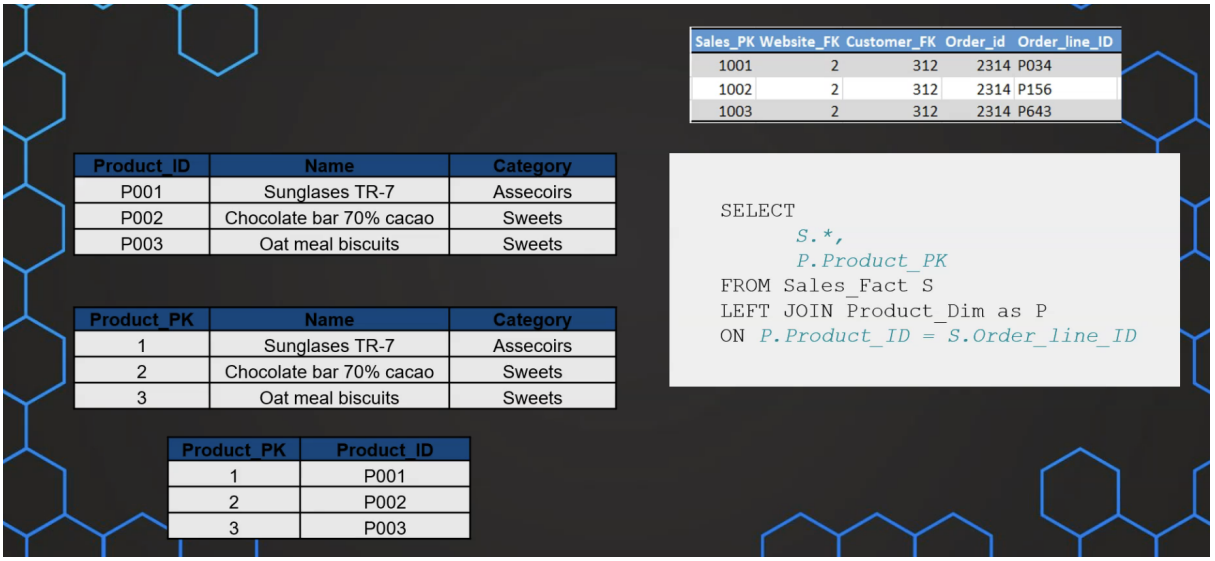


- And then also the question is do we need to keep those original natural keys? Yes, we can, but oftentimes it is also not necessary.

- But what we should do is we want to have usually a lookup table. So this is just giving the reference of our key so our created surrogate key and the natural key.



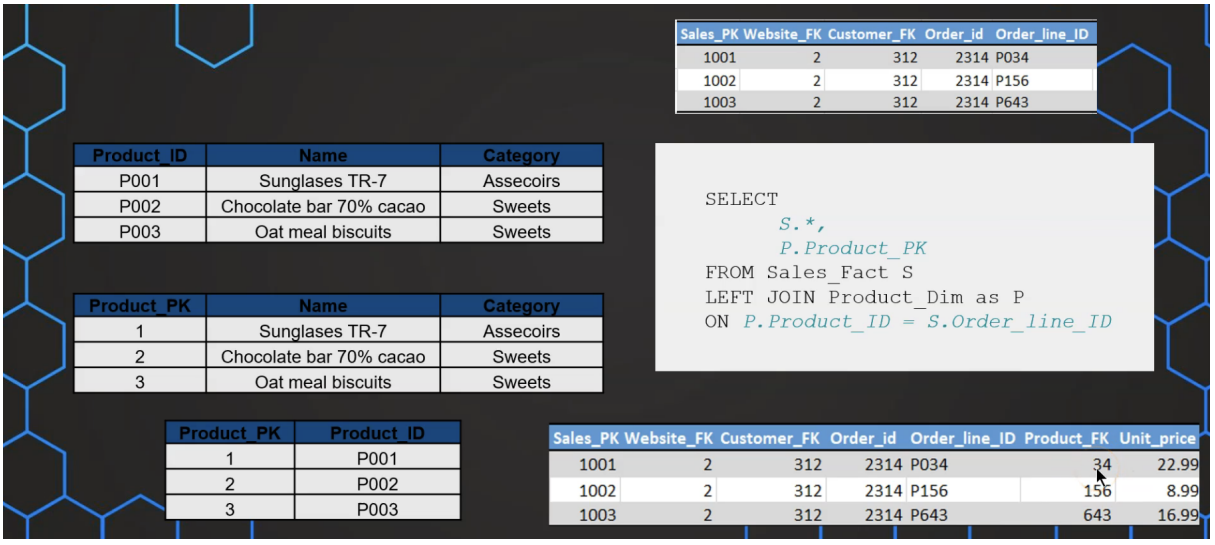
- So this is done very easily. We can just query in SQL the distinct values of this product ID, the natural key and then just populate a sequence next to that.
- But now the question that you might have is, how can we now create the correct reference from our fact table to this dimension table if we use this surrogate keys?



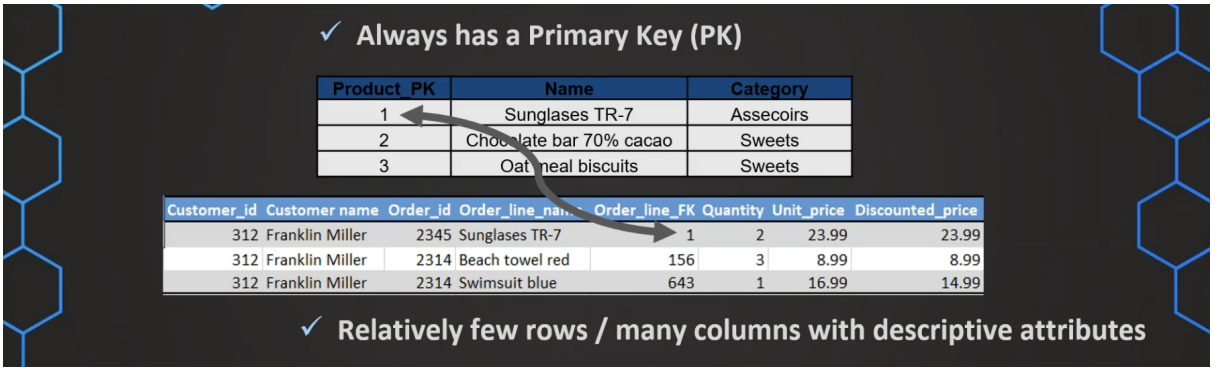
- For example, in here in the fact table which is originally coming out of the source system we see we have still this natural key but we can now use this

lookup table for example, or if we have just still both the surrogate and the natural key in our table then we can just create a join.

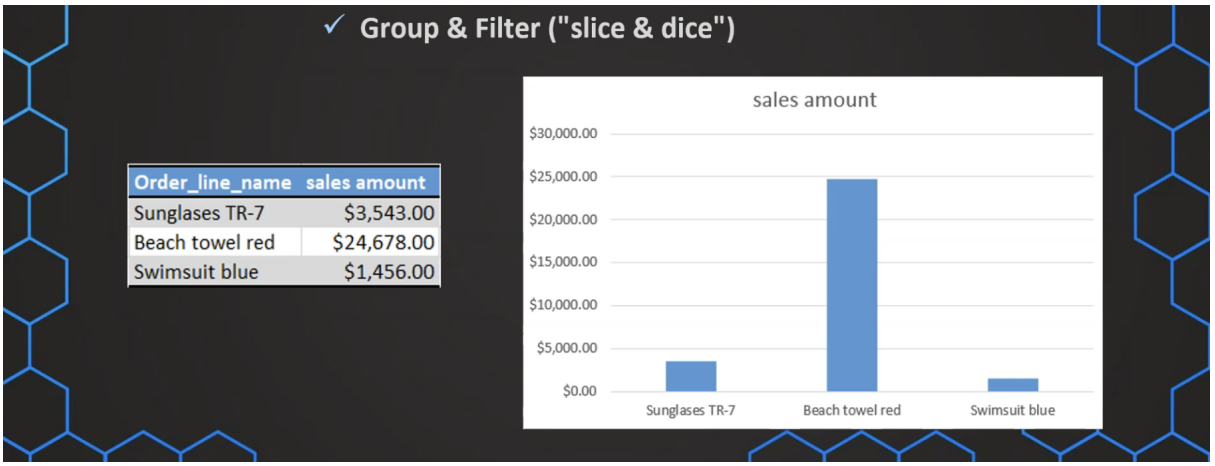
- So we want to start with the original fact table. So all of the attributes that we have already created and transformed. So this replacement of this natural key in our fact table is usually the last step after we have done all of the transformations in our fact table.
- And now we can just use all of that columns that we have created and then we can just add the value from this product table. So we use a joint for that.



- In our case, we can use a left join from this sales fact table to this product dimension. And we have now all of these correct references to our dimension table.
- And then with that dimension table, we can create now the correct reference from this product primary key that is now referenced in our fact table. And of course, needs to refer to the correct values. So in this case we have sunglasses TR7.

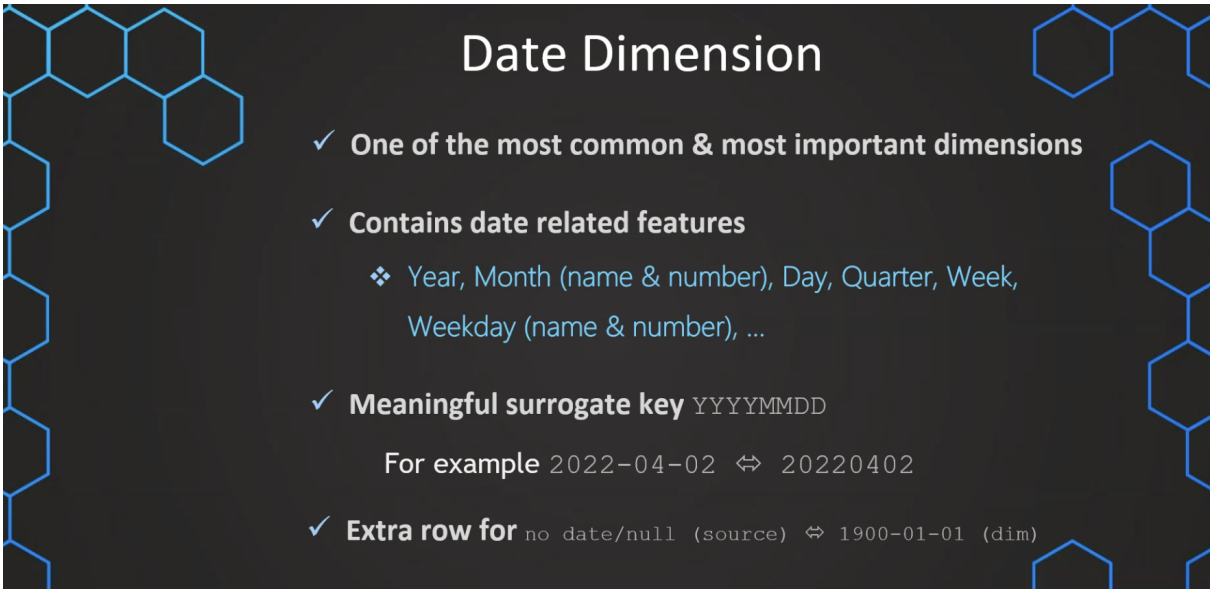


- So this is the order line one, and this is referencing to the correct product name and also the related attributes such as category, subcategory and so on.
- So like this, we don't need to keep the sunglasses name and all of the other attributes, but we can just remove it from the fact table and just use all of the attributes in our dimension table.
- And with that structure, the dimension tables usually have not so many rows, but usually it is a very wide table. So we have many columns with all of the different descriptive attributes.



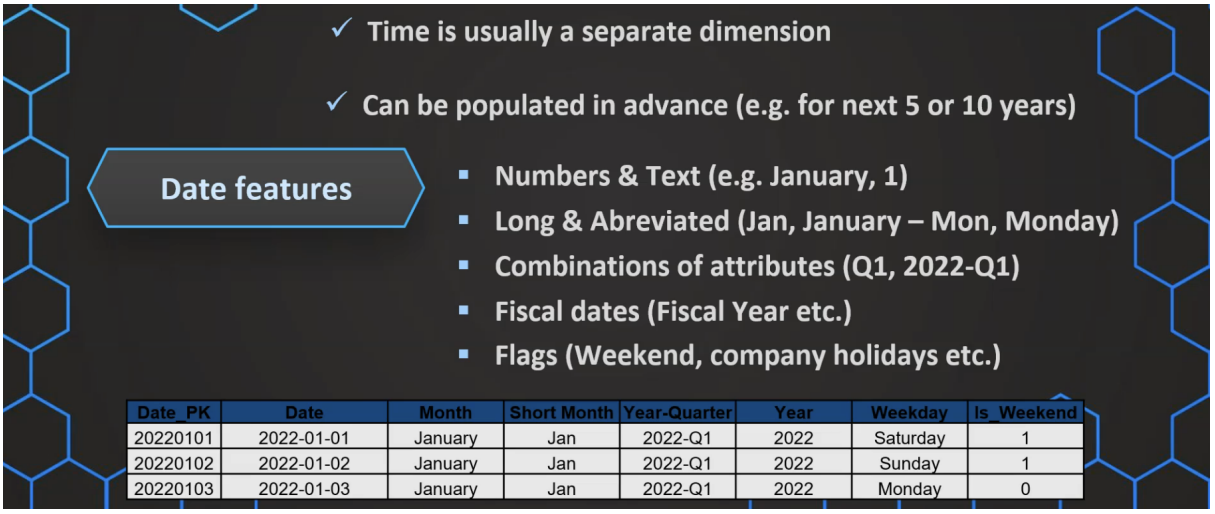
- Now we have learned that this dimension table is now used to group and filter our data. So it's also called slice and dice the data.
- And this can mean then that we can just use some of the attributes of our dimension. So for example, the product name and group the data like above.

Date dimension



- Date dimension is one of the most important aspects in our dimensional analysis because we want to measure the performance usually over time and across the different date dimensional aspects.
- It contain all of date related features that we want to analyze, it can be year, month , name of the month, number of the month, day of the month, day of the week, weekday name and it’s number
- Surrogate key in the date dimension is not just a number without any meaning that is just increasing one by one in each row but it is usually in the format YYYYMMDD, so this is just the primary key in the date dimension
- Commonly, we also have an extra row in our date dimension. So we should usually always have that. And this is just representing a dummy value if there is no date value in our fact table because in our fact table in the foreign keys we should not have null values or missing values.
- And oftentimes in the source there can be no values and therefore we need to replace them with a foreign key that is just referring or referencing a dummy date value.
- So this could be a very early days, or for example, 1900 1st of January. And like this, we have the referential integrity and all of the relationships work well. So this is what we should do in our date dimension.
- But now what we should also note is that if the time aspect is also important, so just imagine you have also a timestamp next to the date in the source system then this should be usually a separate dimension.

- So if this granularity of the time is also important which is not always the case but then we can also just separate the time from the date and also create a separate time dimension. And the date dimension is one of the few dimensions which are very calculatable.



Date_PK	Date	Month	Short Month	Year-Quarter	Year	Weekday	Is_Weekend
20220101	2022-01-01	January	Jan	2022-Q1	2022	Saturday	1
20220102	2022-01-02	January	Jan	2022-Q1	2022	Sunday	1
20220103	2022-01-03	January	Jan	2022-Q1	2022	Monday	0

- So it's very predictable and we can therefore, populate this table in advance also for days in the future that do not yet exist in our fact table.
- So of course in the fact table the future events are not recorded yet but still we can populate also for the next years this date dimension.
- We should include both numbers and text. So for example, January as month name and one as the first month in the year. Then we should also include long and abbreviated names.
- This is not a must, but is depending on the business use case, but we should consider both of these options in our data warehouse.
- So if the business users wants to have these abbreviated names in their reportings, then of course, we should also populate that in our data dimension.
- And also we can have combinations of attributes. So for example, we can not only keep the quarter itself but also the combination of 2022 Q1, and then the users can just use this feature to group the data by.
- Also something that we can populate on top of that is some fiscal dates. So for example, the fiscal year.
- And last but not least, we also oftentimes have some flags. So a flag is something that is indicating if some value is true or not. So for example, is it

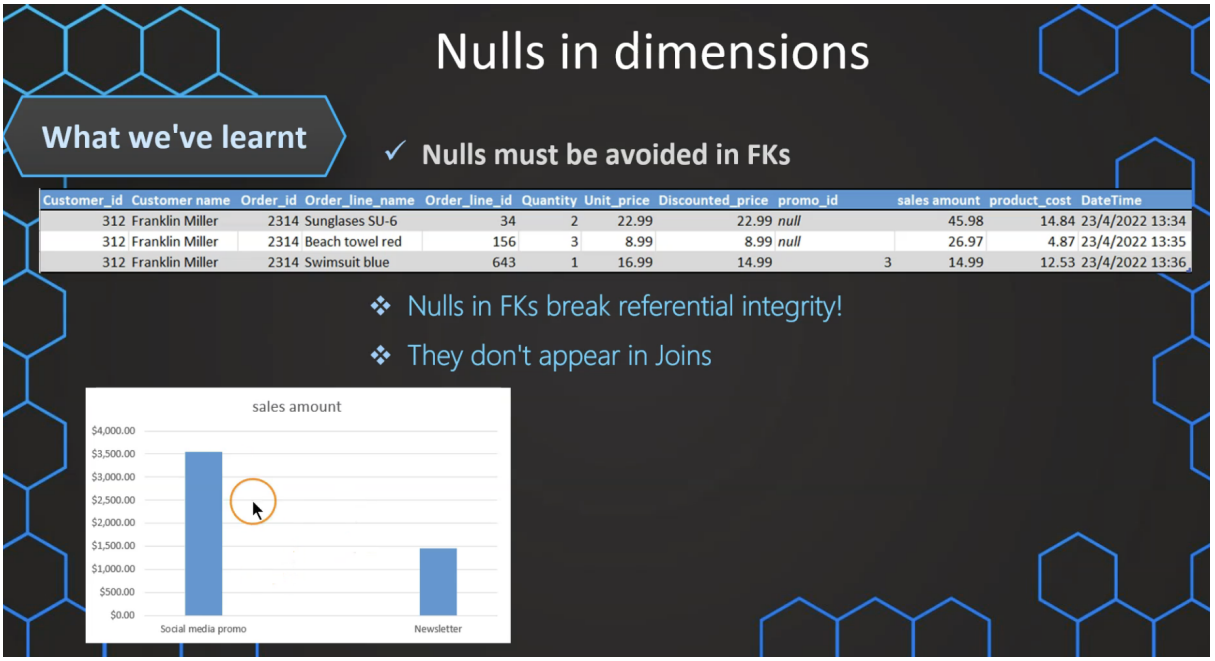
a weekend or not? Is it holidays or not? And like this, we can also analyze the data with these flags or these yes and no features.

Date_PK	Date	Month	Short Month	Year-Quarter	Year	Weekday	Is_Weekend
20220101	2022-01-01	January	Jan	2022-Q1	2022	Saturday	1
20220102	2022-01-02	January	Jan	2022-Q1	2022	Sunday	1
20220103	2022-01-03	January	Jan	2022-Q1	2022	Monday	0

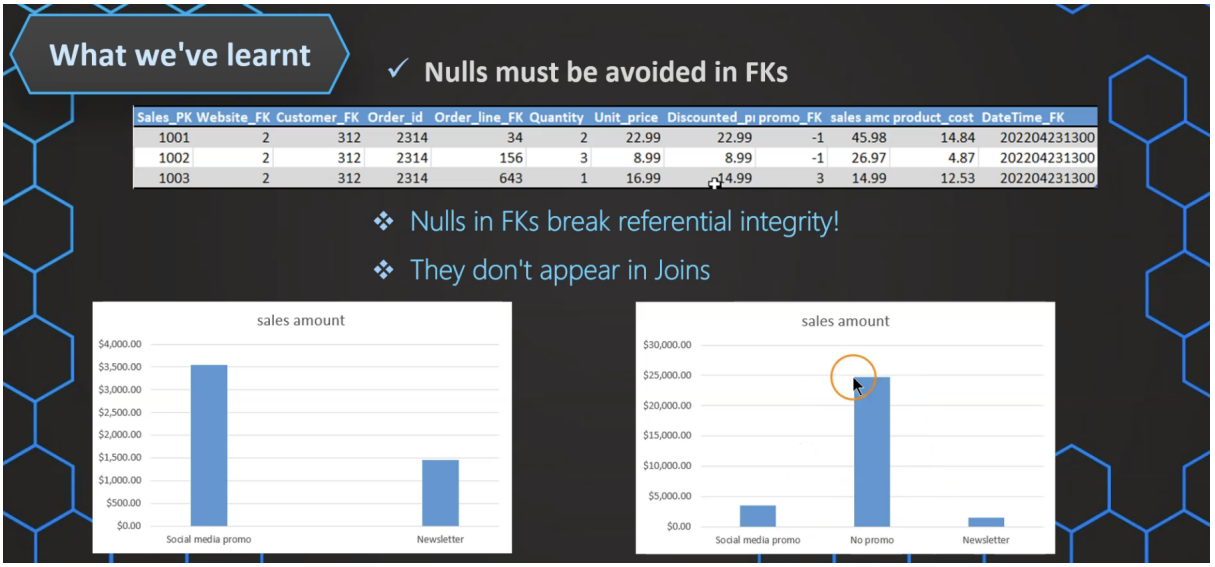
- Also note that here usage of one and zero. So one is indicating yes and zero is indicating no which has some advantages because it can be aggregated we can calculate the sum but also it is not so user friendly. And if it is not so clear to the end users it is better to use understandable words. So for example, we can say weekend or weekday so just in plain text.

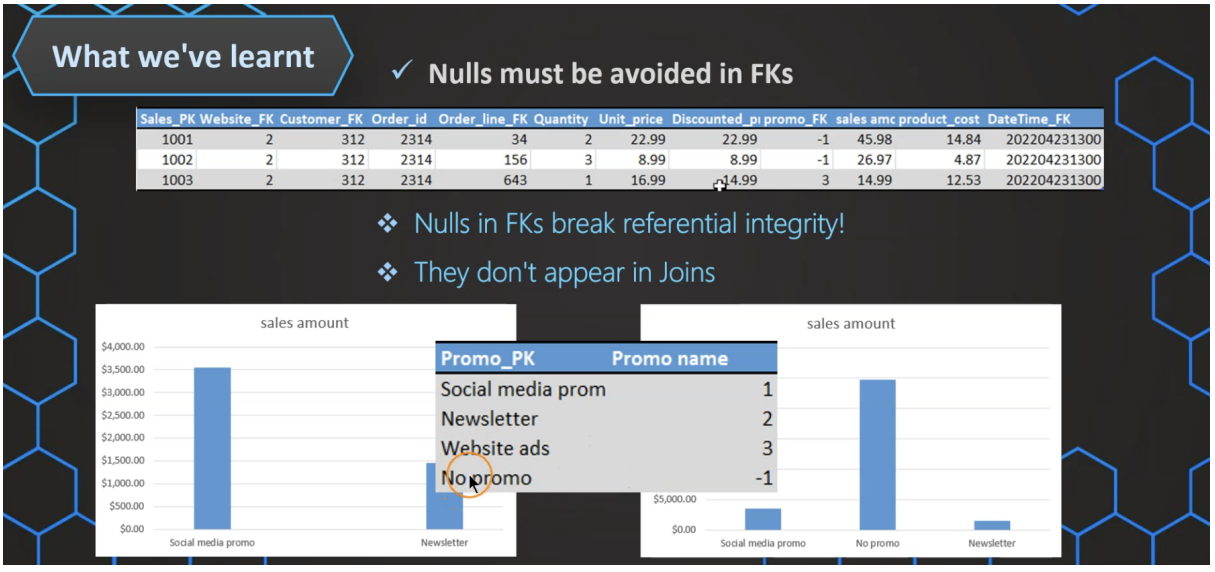
Nulls in dimensions

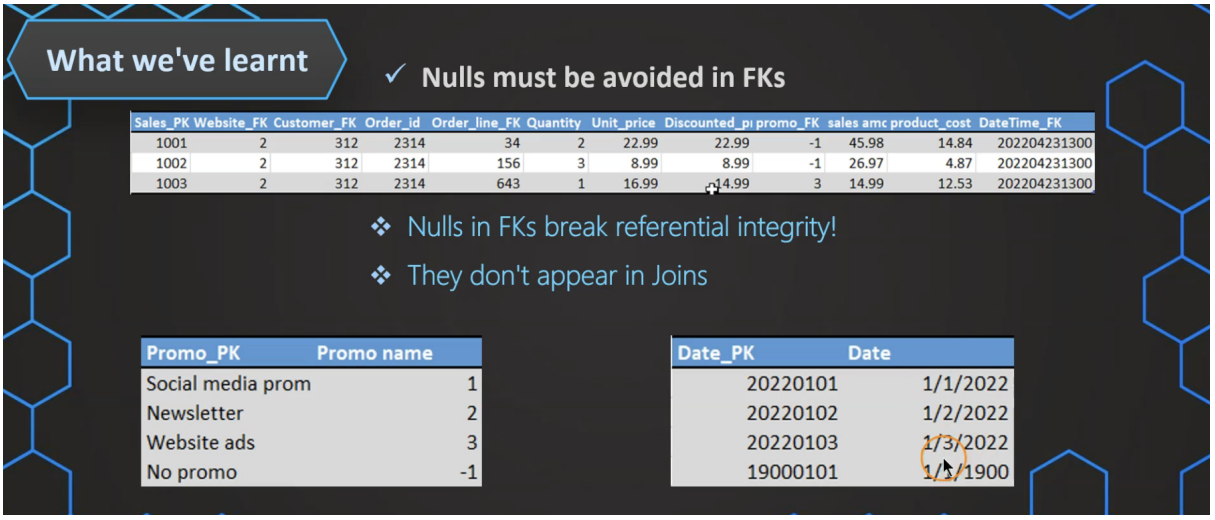
- Nulls must be avoided at all costs basically in foreign keys
- We can easily do that by just replacing those nulls with a dummy value, for example, negative one.
- If we don't do that, so if we would keep those nulls in the foreign keys that are used to create joins or relationships to our dimension tables, we would break the referential integrity and then those values would basically disappear if we want to join this to the dimension tables.



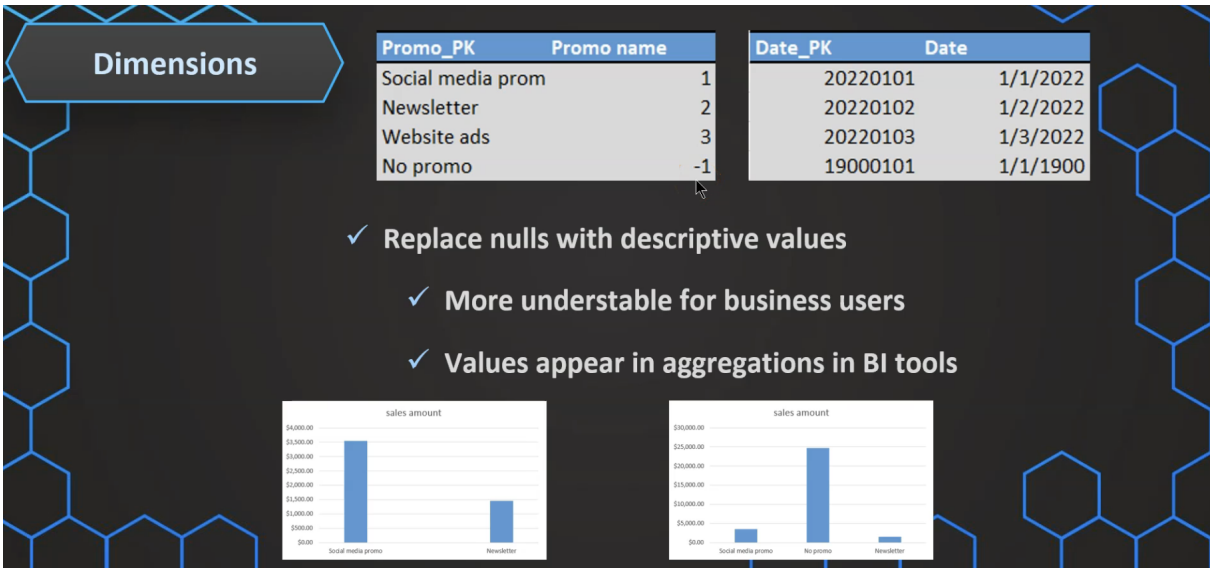
- So if we are grouping by promotion then it would look like the above image because the values with null joining will just disappear during the joins
- So use -1 instead of nulls







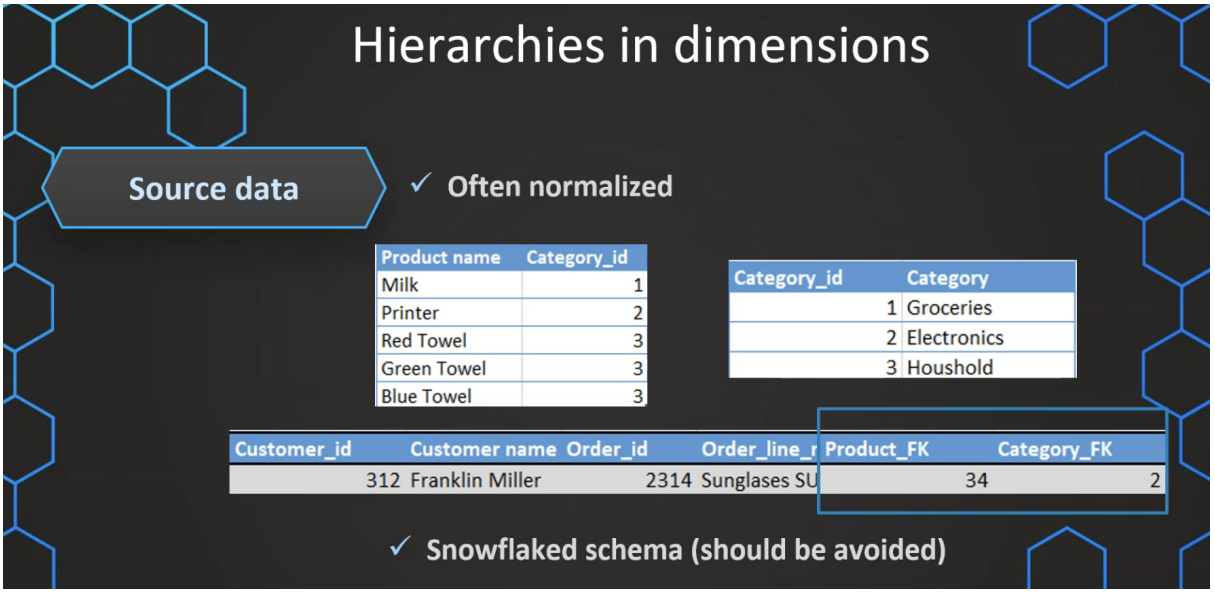
- Hence use dummy values like this wherever applicable according to the data type, in integer column use -1 and in date column have old date or large date
- Nulls can be present in the Fact tables
- So for example, if we, in general, have never sales in the weekend because our stores are closed, then it would not make sense to use zero because this would just screw the average.
- So the average would appear then lower as it is in general in the times when the store is open. And therefore, we can have potentially nulls in the fact tables. So they work pretty well with the aggregations, such as sum, such as average. And therefore, this can be present in the facts and we don't need to replace them necessarily.



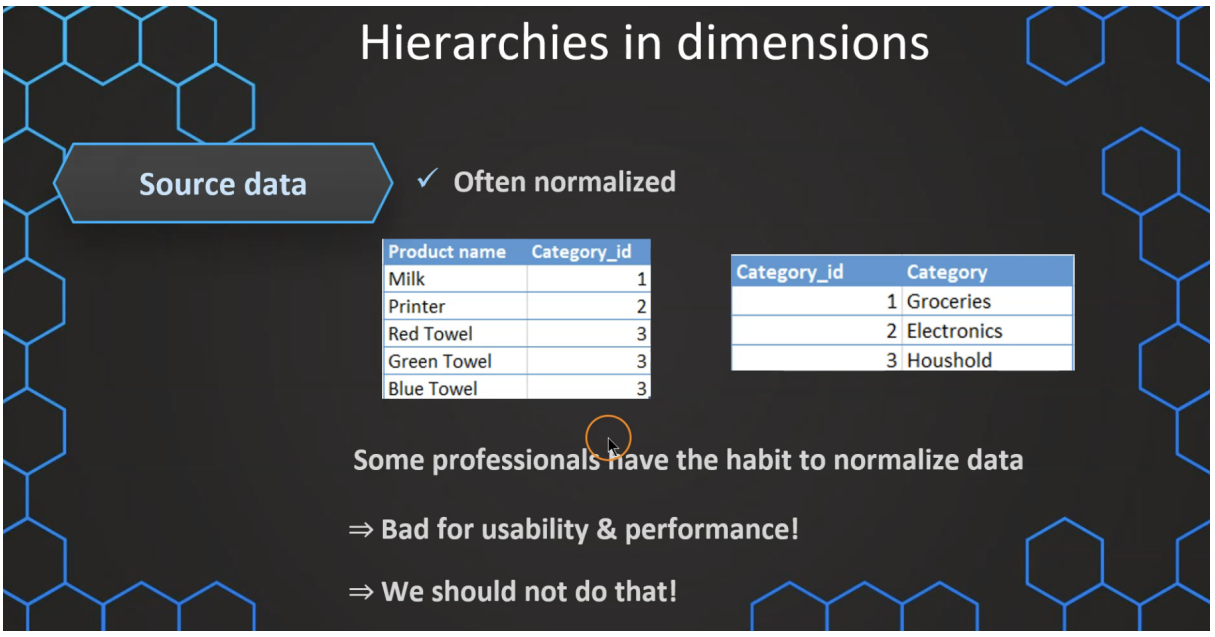
- We should, if there are any nulls in the dimensions, replace them always with descriptive values.
- For example, a descriptive value could be something like no promotion available, no category available. And also, of course, for the date, we have a dummy value, such as 1st of January 1900.
- And the reason for that is that you want to avoid that is the business users don't understand what this null means because the null is not really descriptive and can mean anything. No promotion available is much more descriptive.
- And then also the users can decide themselves if the value should appear in their aggregation, in their grouping, in their graphs or not.
- If it would be a null, it would just by default disappear and would not be shown in the graphs that are created in these BI tools.
- And therefore, we have more options with these descriptive values for the end users and it is also more understandable for the end users.
- So therefore, in the dimensions, just also replace the nulls. And if possible, use dummy values, such as the 1900 or descriptive values if there is a null initially in the fact table. So this is how we should deal with null attributes in dimensions.

Hierarchies in dimensions

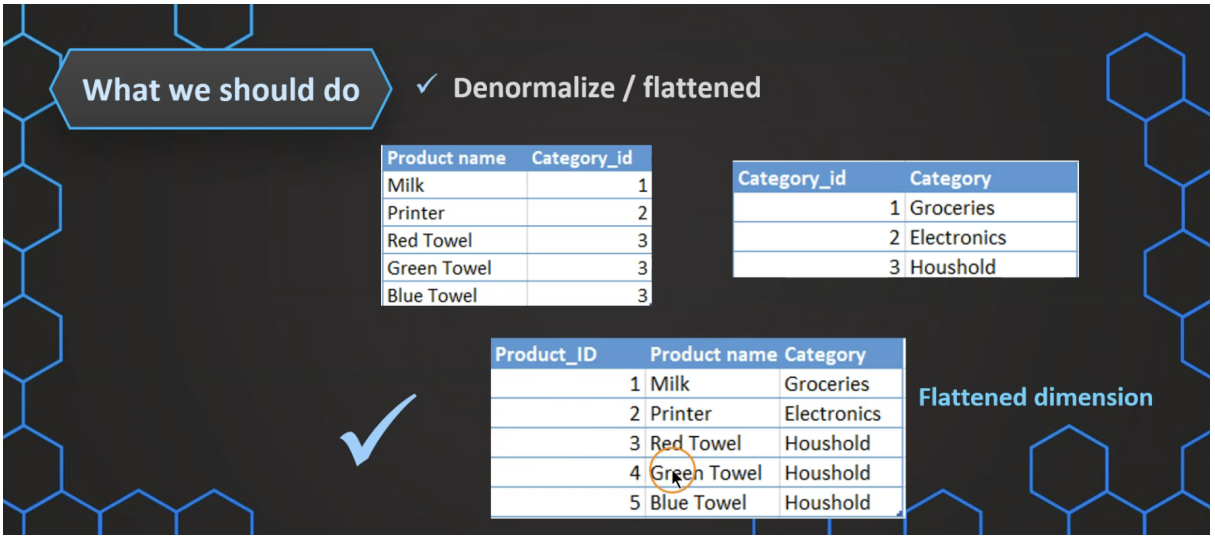
- In our dimensions, we oftentimes have hierarchies included, and therefore we want to talk about how we should deal with those hierarchies.
- So what is the best way to go about them? Because there are also some pitfalls that we can fall into. And that's why we want to talk about the best way of dealing with hierarchies in our dimensions.



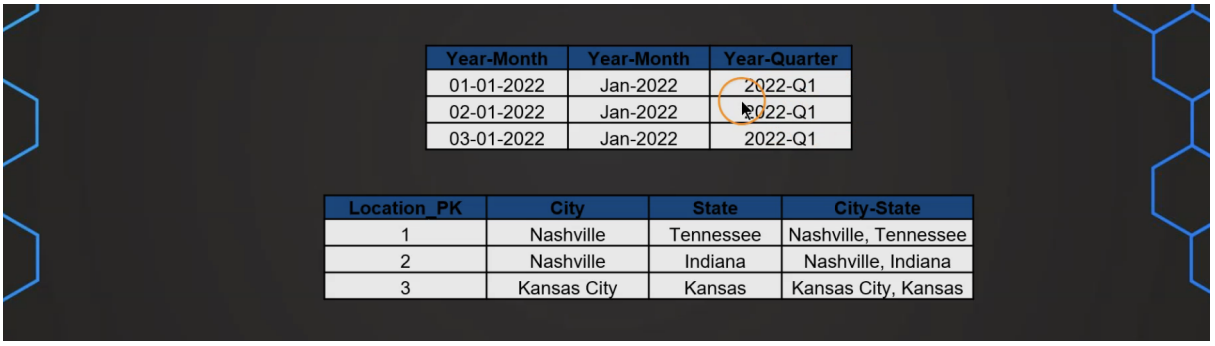
- In our source data, the data is used for transactional processing and this is why the data is oftentimes also normalized. That means that if we have, for example, this product table, we have the associated category also and then there is a separate table.
- This of course saves disk space and is also good for the right performance and therefore it is just a good fit for transactional data processing.
- But for us, since we want to use the data warehouse for analytical data processing, we are looking for high performance to read the data, so to get the data out of the warehouse, and also we want to have a high usability.
- And if we would have such a case where everything is denormalized, we have many foreign keys in our table, and this can really make the fact table very wide. And this is something that we should avoid.
- If we do that for all of the different dimensions and all of the hierarchies we get something that is called a snowflaked schema.
- So we have only this product name as a dimension and then we have another level of hierarchy. So we can connect the hierarchy from the product with the category via this column. And then this basically looks like a snowflake and therefore this is called a snowflaked schema, if all of the dimensions are denormalized.
- But this is something that should be avoided because we really don't have any benefits with that for our data warehouse because we just have a lower performance and a worse usability, and therefore we should not do that.



- And even there are some people that are, especially the ones that are a little bit more experienced in data modeling, they have the kind of habit to normalize the data.
- So even if the data is not normalized in the source data, they tend to, because it is good for certain cases, normalize the data.
- But this is not the case, as we've learned, in the data warehouse. And here we should really resist that habit to normalize the data because it's bad for usability and we should avoid it.
- Instead, we should try to flatten the data. So we should try to denormalize the data, and this is now the way to go.



- For example, we could just create a join between Category_id and then the category table. So we could write the data in our data warehouse just in one table and like this, have flattened the dimension. This is what we should do in our dimensions in dealing with these hierarchies.
- And what we can do as well is we can combine attributes of different levels of hierarchies together in one column.

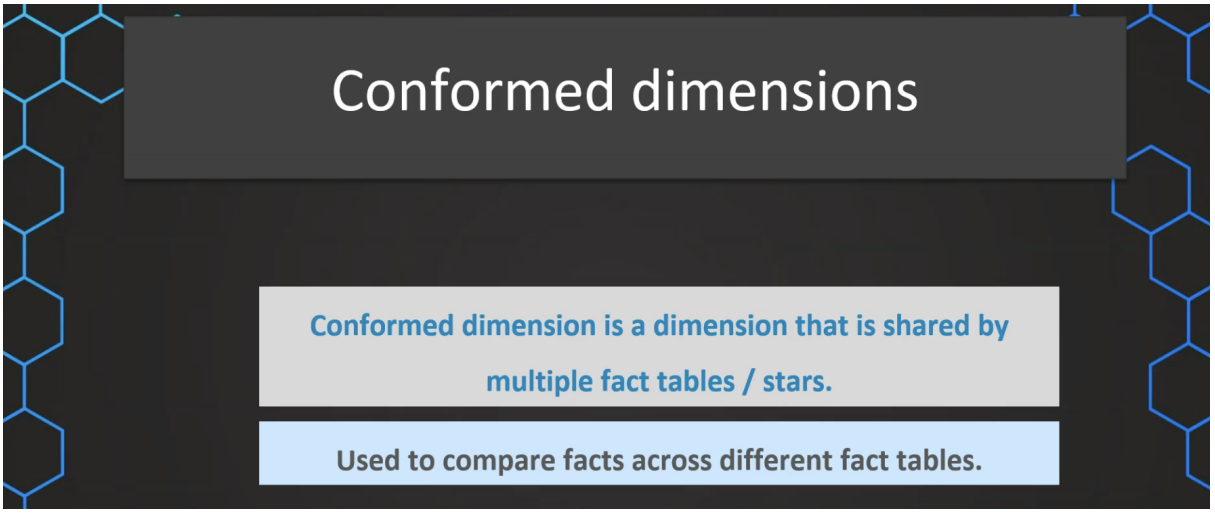


Year-Month	Year-Month	Year-Quarter
01-01-2022	Jan-2022	2022-Q1
02-01-2022	Jan-2022	2022-Q1
03-01-2022	Jan-2022	2022-Q1

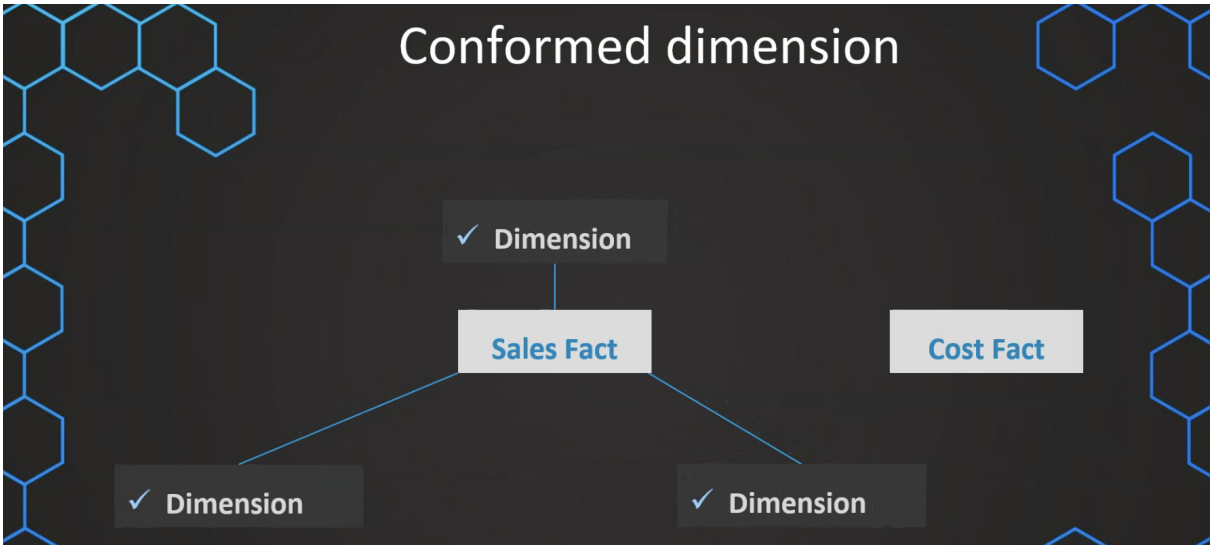
Location_PK	City	State	City-State
1	Nashville	Tennessee	Nashville, Tennessee
2	Nashville	Indiana	Nashville, Indiana
3	Kansas City	Kansas	Kansas City, Kansas

- So this combination can be then more user friendly if the user wants to also have the combination of, for example, year and quarter, and then this can be used directly out-of-the-box.
- And especially this can be helpful if we have duplicate values, or let's say Nashville, a city, that can occur in the state and not only in one state, but the same city also in another state.
- In that case, it can also make sense to have some pre-calculated column that is just combining the city and the state.
- Especially in those cases where one attribute can occur in different levels of the hierarchies multiple times.
- In that case, it can especially make sense for the user friendliness to just create an out-of-the-box combination of those two columns.
- So these are things that we should keep in mind, but the most important thing is we should try to combine those hierarchies and collapse them into one table when it makes sense, so we have less dimensions and a more flattened table. So this is how we should deal with hierarchies in the dimensions.

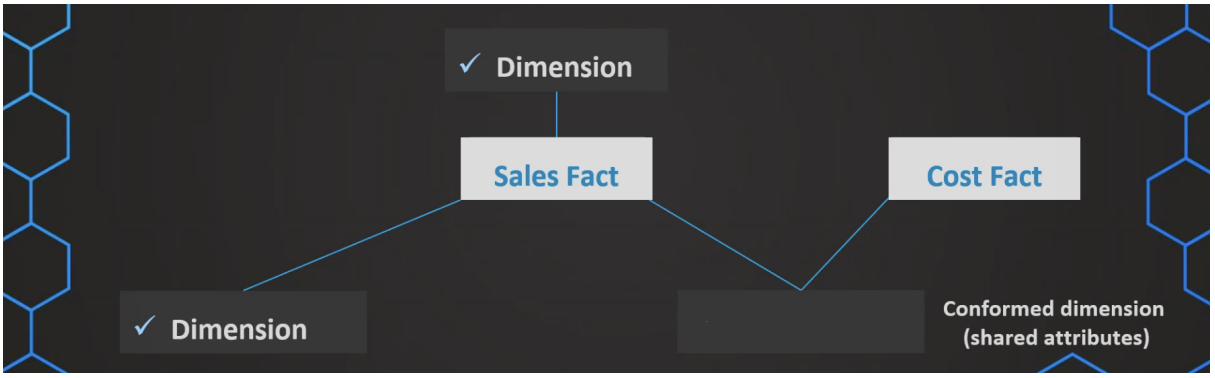
Conformed dimensions



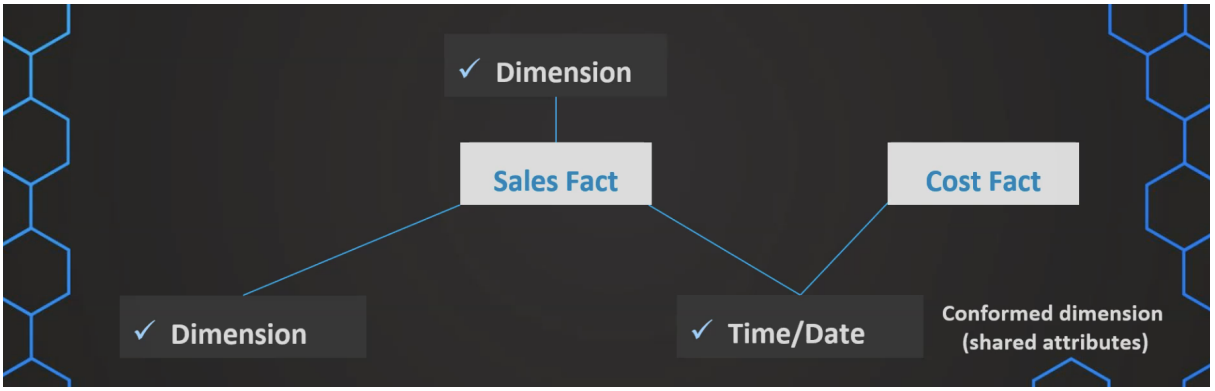
- purpose for those is to compare multiple facts, and use them together in one report or in one analysis, so we want to compare facts across multiple facts.



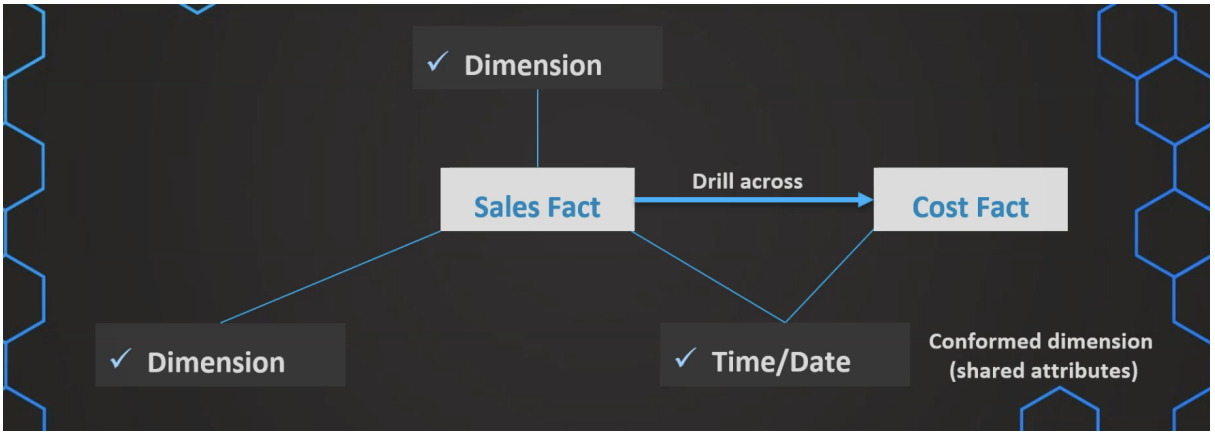
- So let's imagine we have a sales report, we have a star schema, so a data mart around a sales table, and in our case, we now want to also analyze the costs.
- So we have a second fact, and now, we want to compare, of course, the sales with the costs, and now, what we could do is use a so-called conformed dimension,



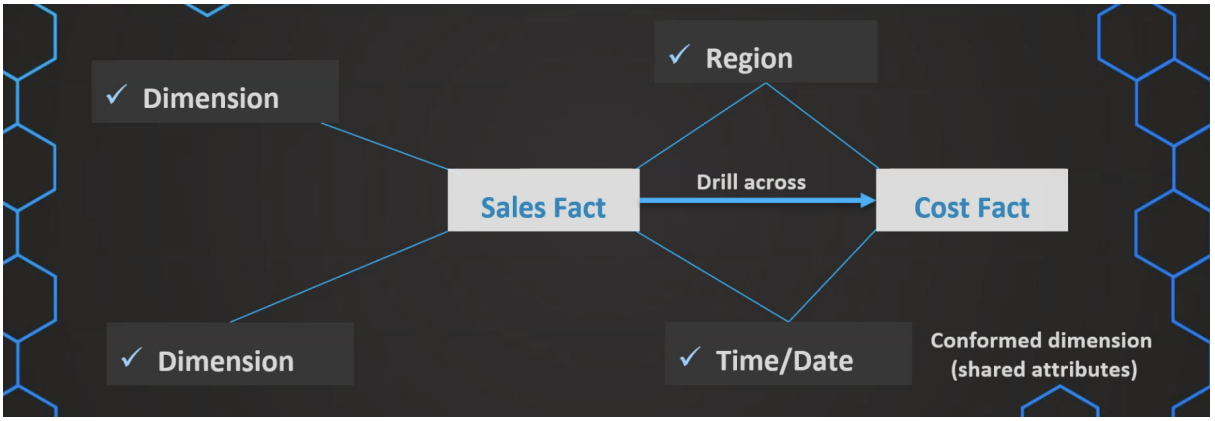
- so a dimension that shares the same attributes. So for example, this is something that can be the date or the time, which is the most typical example for a conformed dimension.



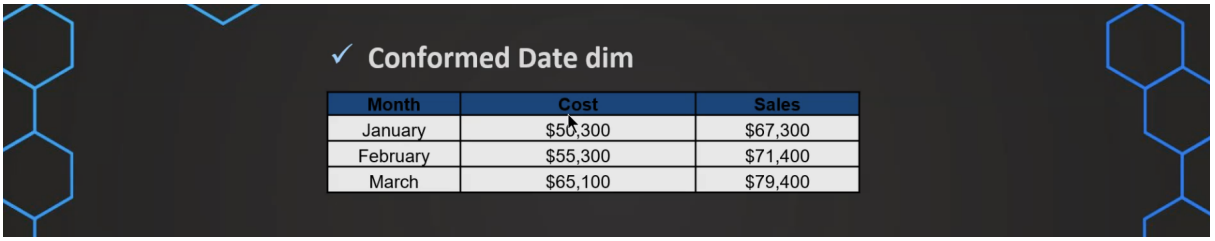
- And now, with that connection, so this conformed dimension, we can now create a report or analysis that is comparing the sales and the fact, so we call this a drill across.



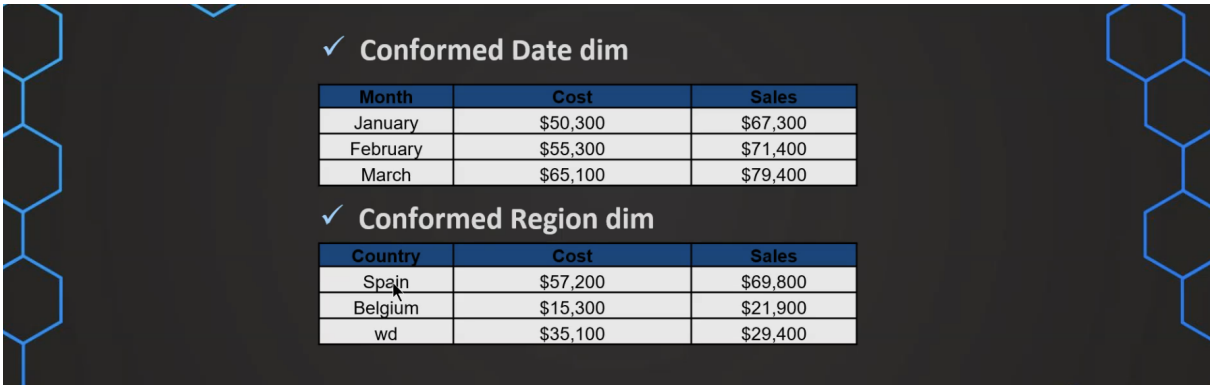
- Another example of that could be that we want to just use the region as a conformed dimension.



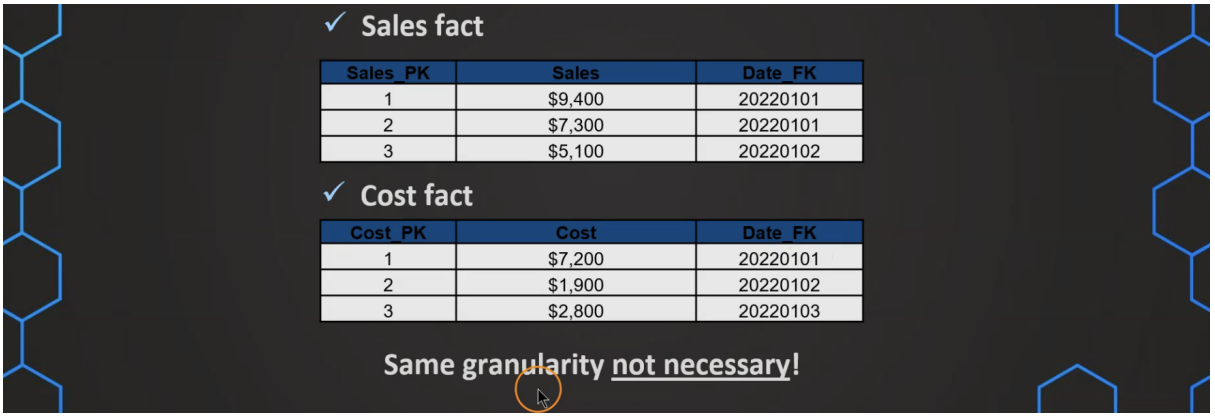
- So for example, both in the sales table, and in the cost fact, we have the region included, and these are now both attributes that are shared by both of these facts, and now, we can in one analysis, in one report, use the sales and compare it also with the cost using this shared attribute of the region.
- So let's have a look at the following example. We want to use a conformed date dimension. That means we use the date, so a shared attribute, which is available in the cost and sales to just compare the cost and the sale, and we can now do that because we have both facts connected with this date dimension.



- And with that, the result of that is that we can use this drill across, meaning that we can compare facts, so measures from those two independent fact tables.
- So even though they are separate fact tables, we can now compare those values with each other.
- Another example would be to use a conformed region. So in that case, we have the shared attribute of, for example, the country, and in that case, we can again compare the costs with the sales, in our case, so the measures of those two fact tables.

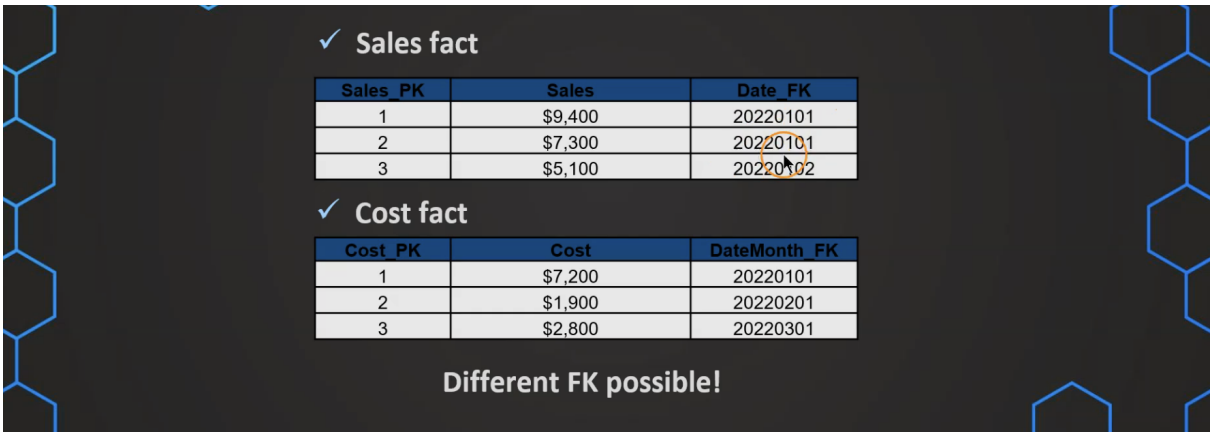


- Of course, what we need to have if we have a conformed dimension, we need to have the identical attributes, or at least a subset of attributes used for both of these facts.

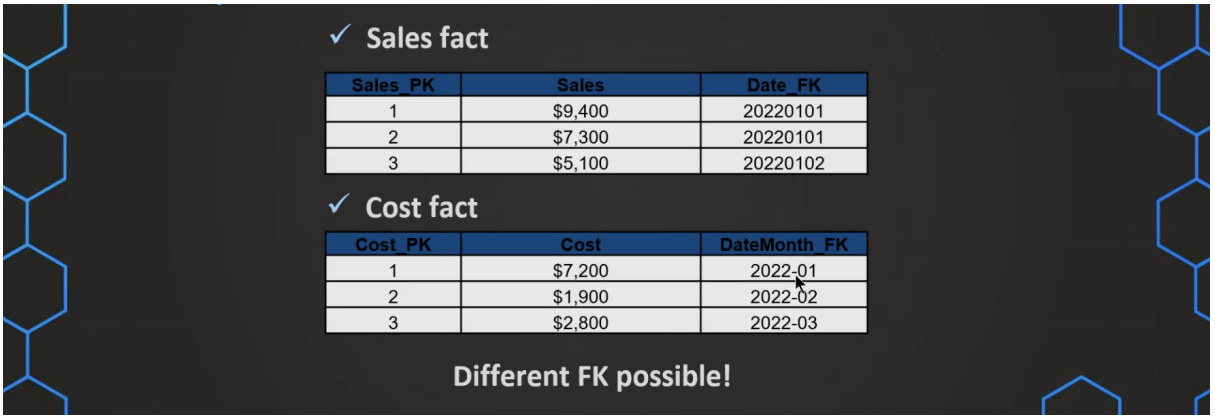


- So this could be, in our case, if we have the sales fact, that we of course have a date foreign key, and then also in the cost fact, we need to also have this date foreign key included, and note that it is not necessary to have the same granularity.
- So for example, in this case, the cost fact is one row and one date, so therefore, every foreign key, so every value here is also unique.
- So one row is equal to one day, and this is not necessarily the case in the sales fact table.
- So in that case, here, there can be duplicate values, and there can be a different granularity, and this makes now this conform dimension so powerful, because now even though we need to have separate fact tables, we can still compare those values with these conformed dimensions.
- Another option would be that we can also have, in this case, only the monthly costs, but in the sales table, we have maybe just transactional, or

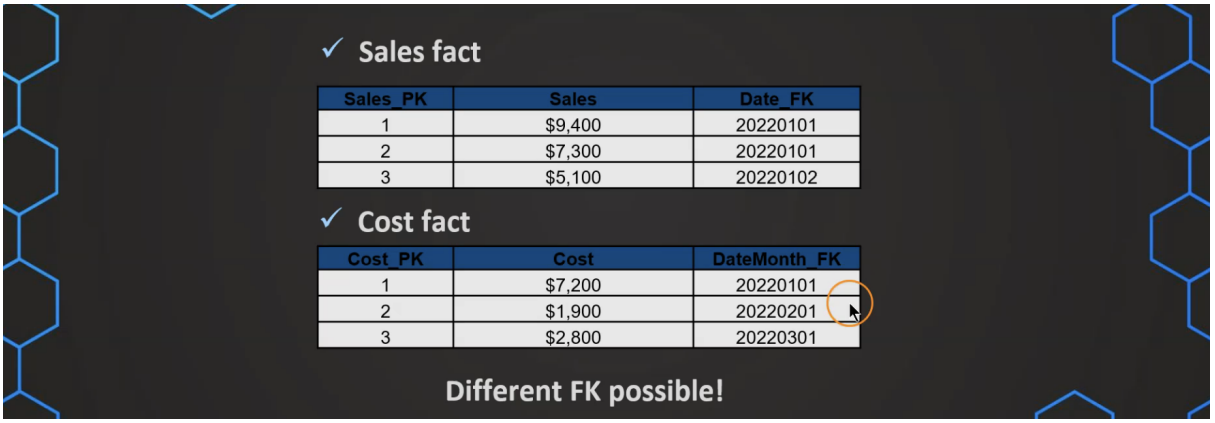
also, we have just a periodical sales fact,



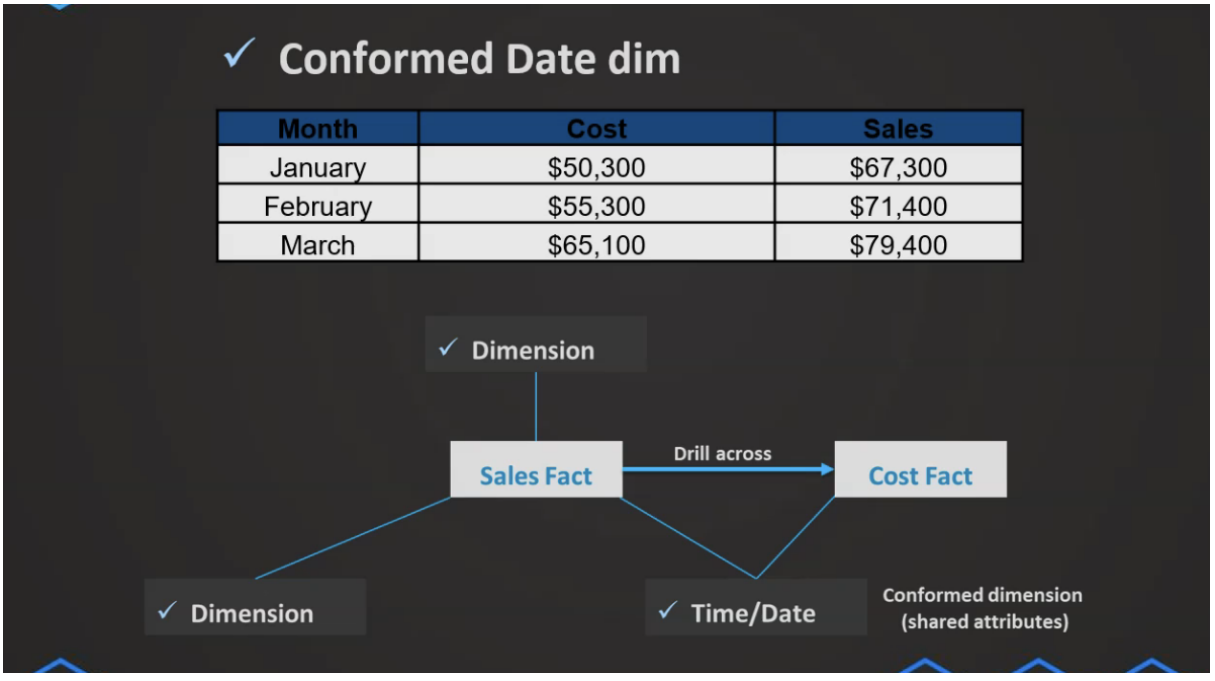
- so every day is one row, but in this case, we can still have different foreign keys.
- So for example, we can have just the foreign key, in this case, that is using the month and the year to connect to the dimension, and in the other case, we can have the daily key.



- So this is using now different foreign keys, but still, it is possible to use this same dimension, and then only, of course, in the cost fact, we need to be careful to use only the correct dimension attributes.
- So this is possible to use also different foreign keys, but also, for example, if we have a monthly value, another alternative would be to just use the first of the month and still use this foreign key.



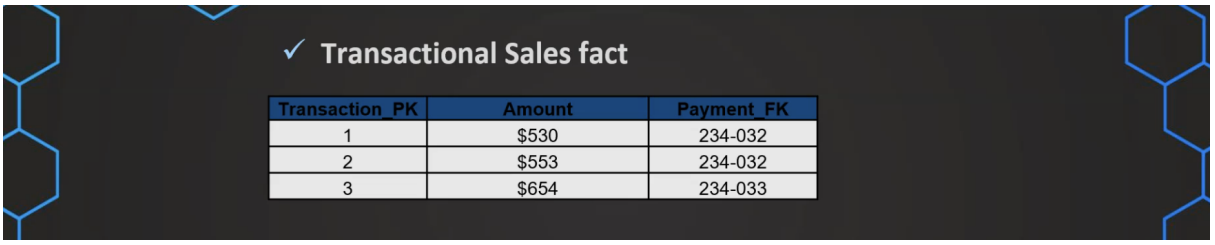
- This is also something that is possible
- So this is how we can use this conformed dimension to drill across and use multiple fact tables together in one analysis, in one table, or in one report, and therefore, if we want to have multiple facts, it is good to have those conformed dimensions in our data model.



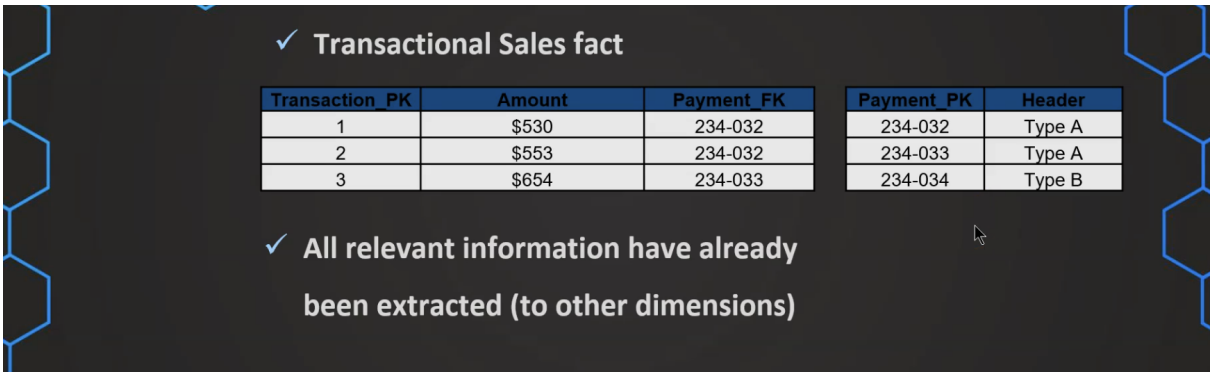
Degenerate dimensions

- Sometimes we have a dimension in our data model that is not really a dimension.
- So this dimension does not have a separate dimension table, but still it functions in a way as a dimension. And that is a so-called degenerate dimension.

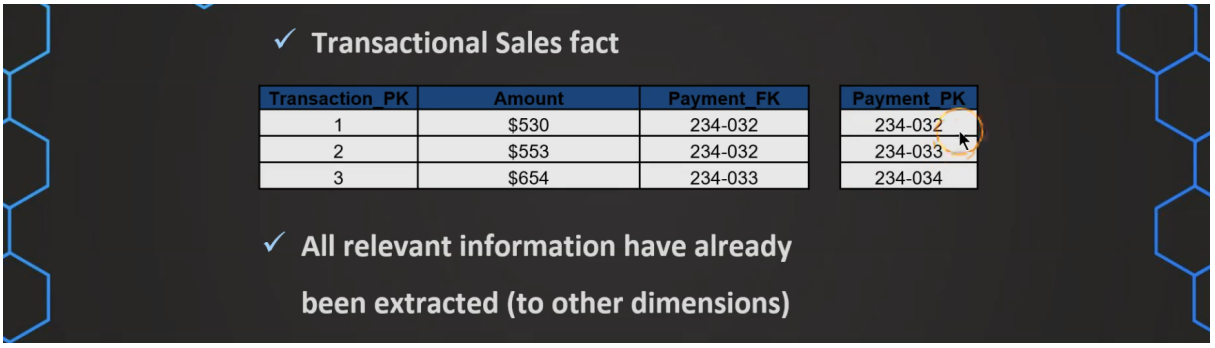
- Let's assume we have the following transactional sales fact table.



- In this case, we have different transactions and there are different amounts and they can be also summarized together in one payment.
- So we can have a payment foreign key and a payment dimension.

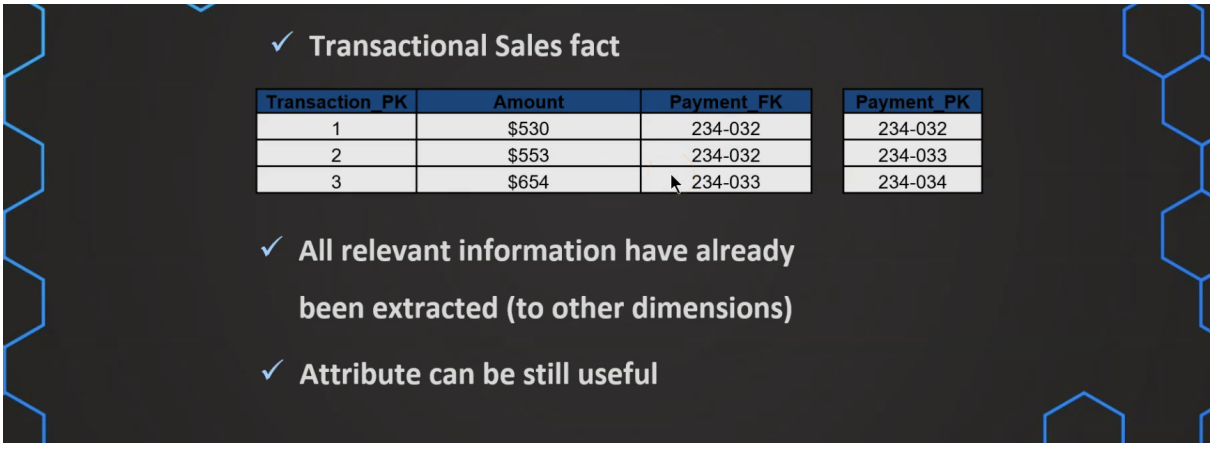


- But sometimes all of these relevant attributes have already been extracted by other dimensions, or sometimes also this value of the header is not really important, and therefore we already have all of the relevant information.
- In that case, we are left only with the primary key of this payment dimension.

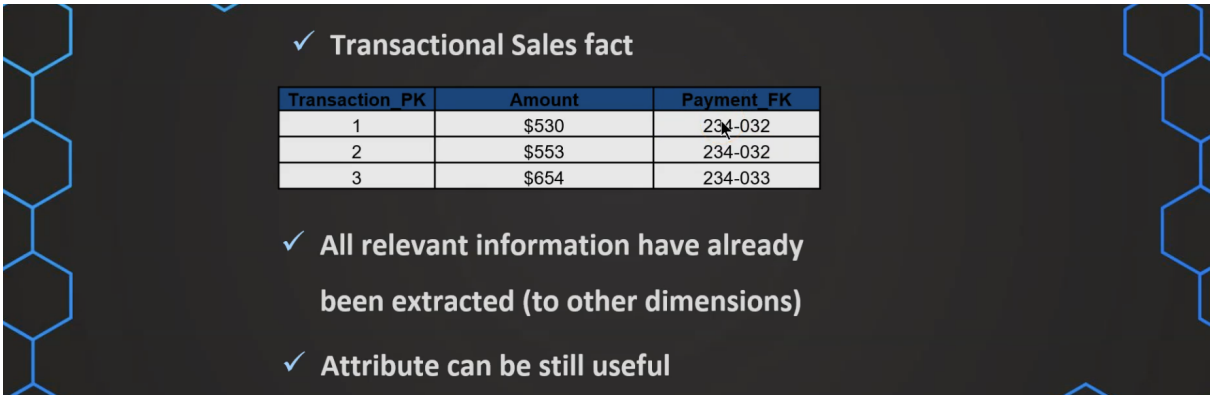


- So of course, in that case, it doesn't really make sense to have this separate dimension because there's no additional information.

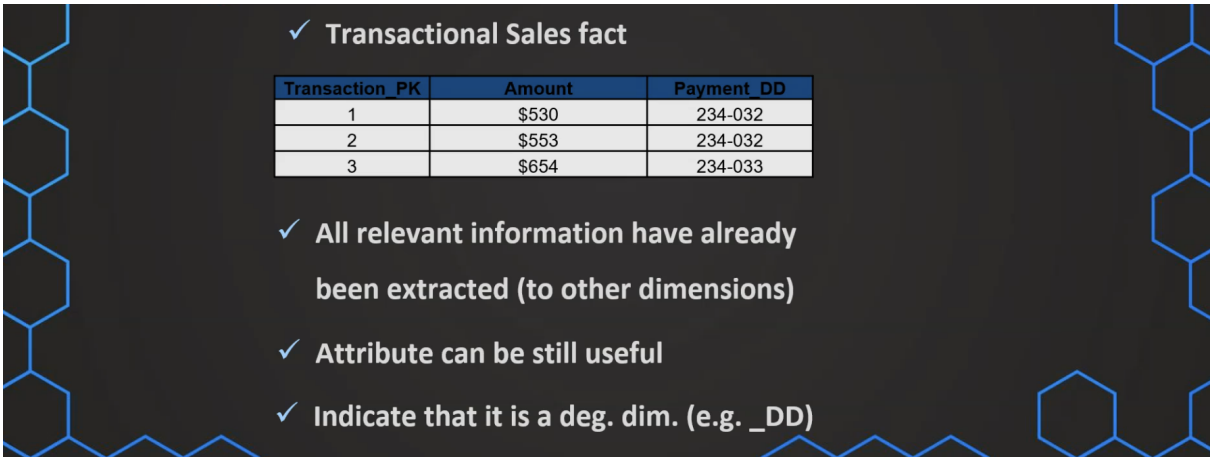
- And therefore, this can be still important that we, even though we don't have this dimension of the payment, this value of the payment ID can still be of value for our analysis.
- For example, we want to summarize the amount of each of the payments, or we also want to maybe calculate the average of the payments, so of the amount of the payments, and therefore, we need to be able to group also by this column.



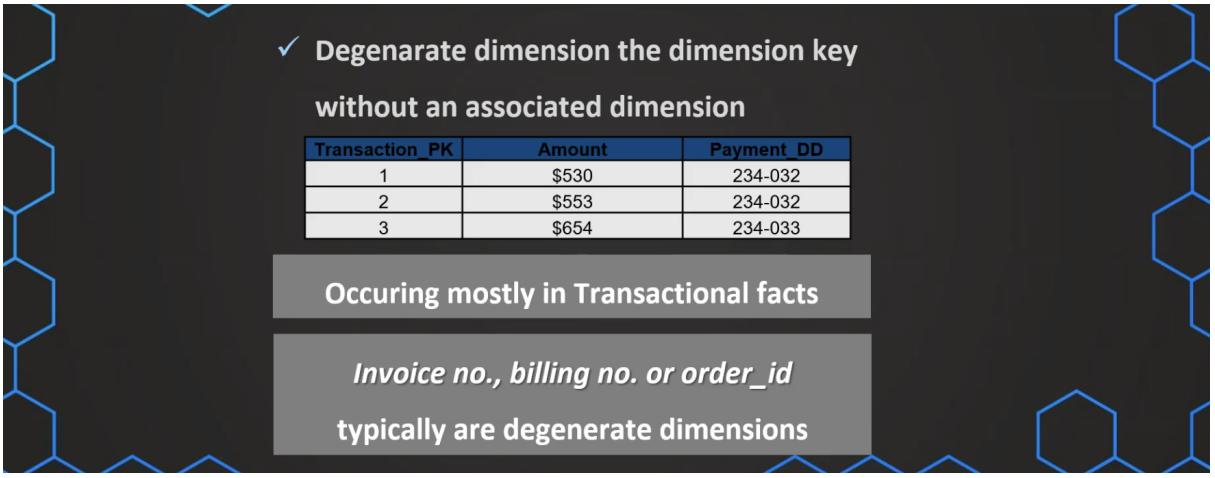
- So in that case, we would not have this separate dimension table, but we would keep the value of this fact table of, in this case, the payment foreign key.



- Yet, of course, in that case, it is not a foreign key anymore, and we should also denote that.
- So we should explicitly make it clear that this is not a foreign key and that there's no dimension table associated with that.



- And therefore we could also just add the suffix, in this case, for example, DD.
- So this is suggested by Kimball, and this is what we can do as well.
- So we just have to denote that this is a special type of dimension. So we don't have a foreign key, and this is not referencing a real other dimension table, but still it can be of value and it can be used in the analysis.



- So we've learned that a degenerate dimension is a dimension key only, so only the dimension attribute in our fact table without the associated dimension.
- So this can be, for example, a payment ID or anything else that is just something that has no associated dimension table.
- So this mostly occurs in transactional effects. And oftentimes this is something like order IDs, billing numbers, or invoice numbers that already have all of the other information extracted and we therefore are left only with, for example, the order ID.

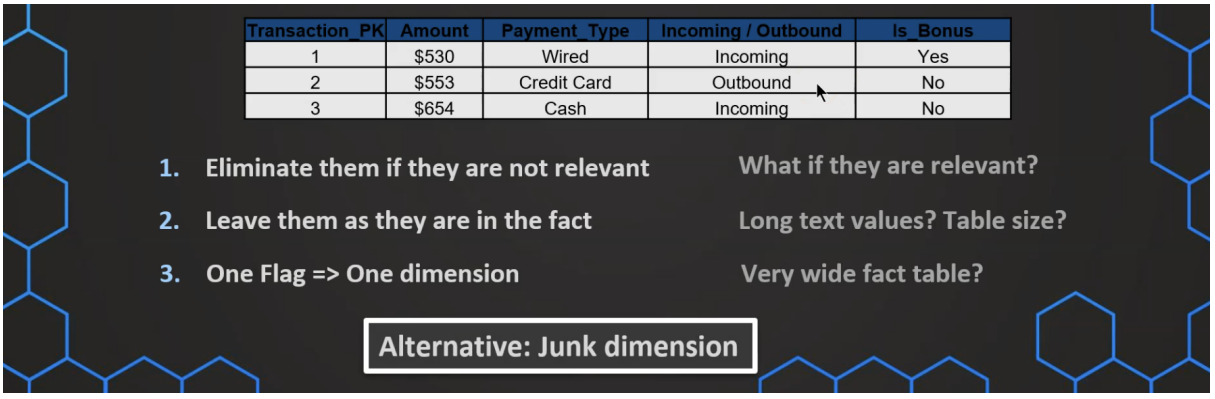
- But still it can be a relevant attribute and therefore we keep it even though there is no additional dimension attached.
- So that's why it can be important to still have those degenerate dimensions even though there is no other dimension table attached to it.

Junk Dimension

- Sometimes in our transactional fact table, we can have a lot of indicators or flags that are actually dimensions but they don't really fit into a given dimension. And therefore, we could create a so-called junk dimension.
- Let's have a look at the following example. In our case, we are looking here at a transactional table.
- So there are some sales transactions and we have a few flags or indicators. So for example, what is the payment type? Is it incoming or an outbound payment? And also is it associated with a bonus or not?

Transaction_PK	Amount	Payment_Type	Incoming / Outbound	Is_Bonus
1	$530	Wired	Incoming	Yes
2	$553	Credit Card	Outbound	No
3	$654	Cash	Incoming	No

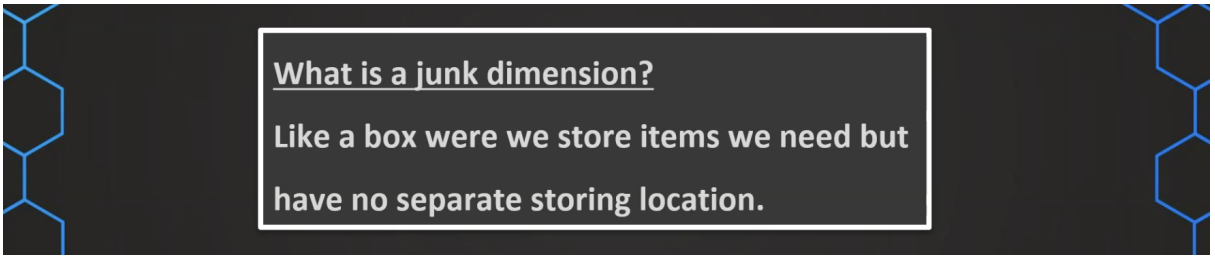
- And in this case, we have now a few options of how we could deal with these dimensional attributes in our fact table.



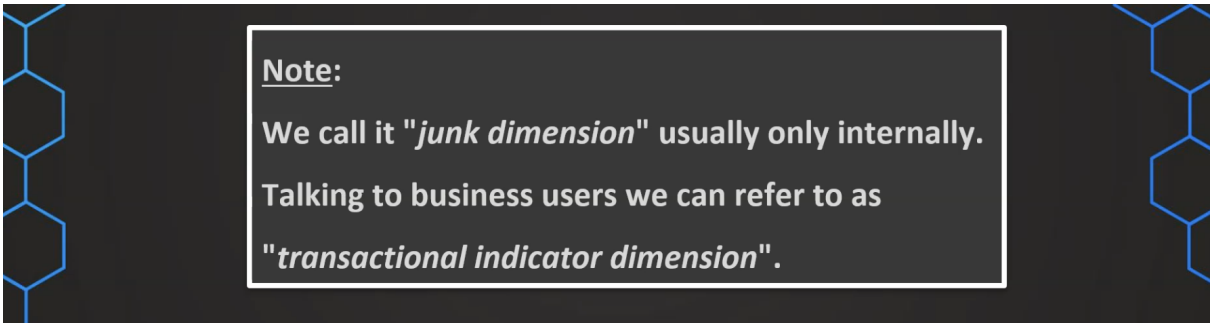
- So first of all, what we could do as data modelers, we can suggest that we can just eliminate them if they are not relevant.
- But of course, there might be some business users that say this is very relevant for me and I still want to use these indicators.

- And in that case, we can, if we want, of course, leave them in the fact table.
- This could be also an option so it's like a dimensional value but we are still just having that in the fact table.
- This is possible if it's just an indicator and we don't want to create an additional dimension for that.
- But now what happens if these are very long text values that are really bulky? And then also it could be that this table size if it's a very long fact table, is just increasing dramatically. And in that case it would also not be a really nice situation.
- What we also could do in such a situation, we can create a dimension. So for one of these indicators, we can create one separate dimension.
- Now again, the problem with that would be that if we have already a very wide fact table, this could just again increase the size and the width of this fact table, which is not so good for the performance and not so good for the user friendliness.
- And therefore, we can go also with an alternative. And that is to create ajunk dimension.
- What is a junk dimension? A junk dimension is basically a separate dimension that contains various flags or indicators.

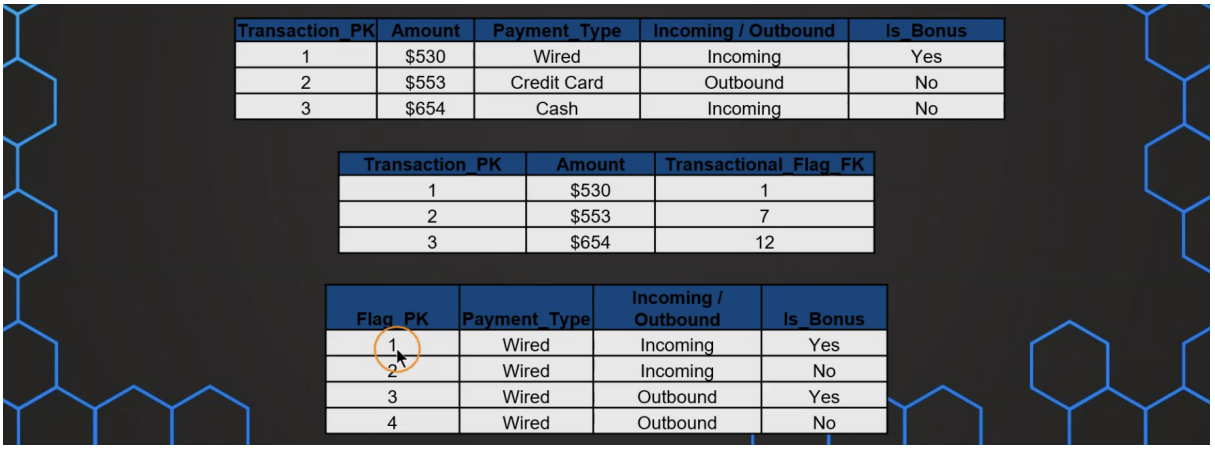
- So this could be yes, no or anything else with a very low cardinality. So the options of values are not so many.
- So for example, there are only two, three or four different values. And then we refer to this as a junk dimension.



- So we can understand this junk dimension like a box in our household where we just put in some items that don't fit anywhere else.
- So we don't want to create a separate storage location for these various items and therefore we put them all just in one box. So we have here all of the items that don't really fit somewhere else. And this is how we can also think of a junk dimension.



- use the term ***transactional indicator dimension*** or ***transactional flag dimension***.
- Let's have a look at the example of how this could be implemented. Let's assume we have the following fact table.



Transaction_PK	Amount	Payment_Type	Incoming / Outbound	Is_Bonus
1	$530	Wired	Incoming	Yes
2	$553	Credit Card	Outbound	No
3	$654	Cash	Incoming	No

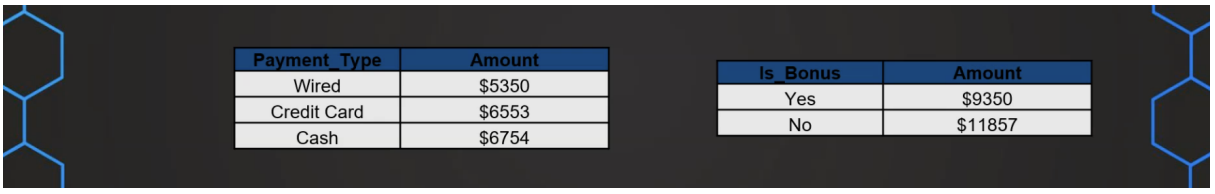
Transaction_PK	Amount	Transactional_Flag_FK
1	$530	1
2	$553	7
3	$654	12

Flag_PK	Payment_Type	Incoming / Outbound	Is_Bonus
1	Wired	Incoming	Yes
2	Wired	Incoming	No
3	Wired	Outbound	Yes
4	Wired	Outbound	No

- We see we have these three indicators and now what we could do is just replace them all with a foreign key and of course, we now need to create all

of the possible combinations and have also this flag, primary key.

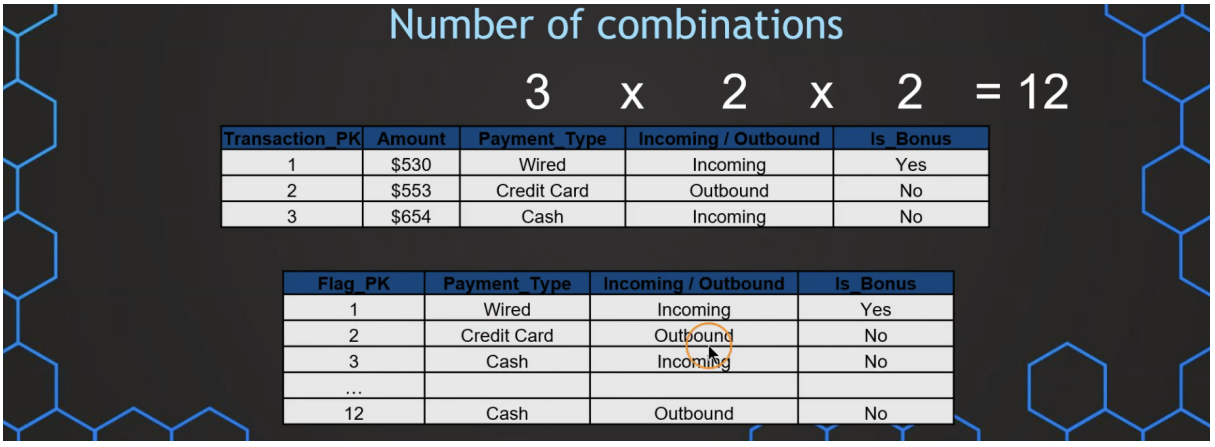
- So in that case, we can create the reference from the fact to the dimension. And then we have all of these possible combinations in this dimensional table.



Payment_Type	Amount
Wired	$5350
Credit Card	$6553
Cash	$6754

Is_Bonus	Amount
Yes	$9350
No	$11857

- This can be used very easily for the users. So we can use the payment type to group the data by the amount, we can use the other attributes such as the bonus and so on.
- And this is our goal with our dimensional modeling.
- Now we have to be a little bit careful because sometimes, there can be quite a lot of combinations.

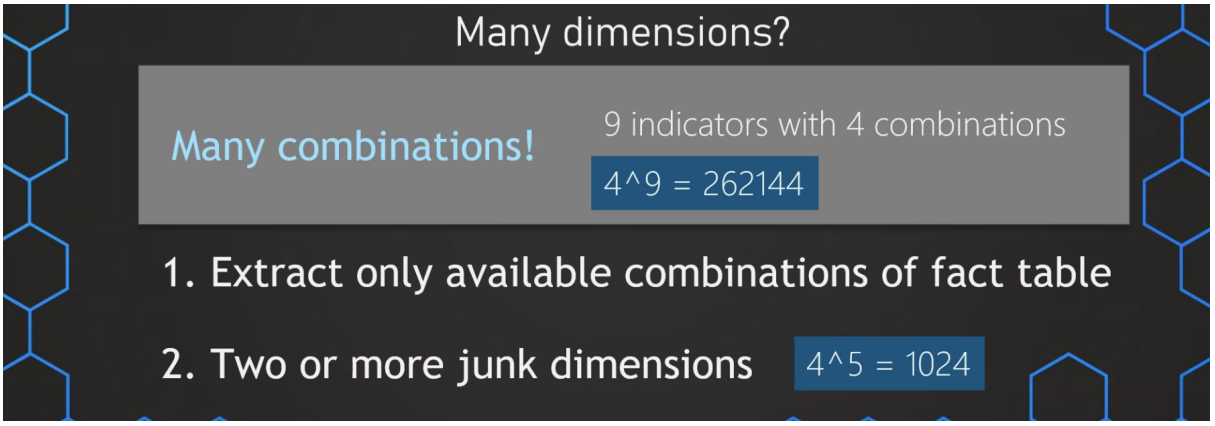


Transaction_PK	Amount	Payment_Type	Incoming / Outbound	Is_Bonus
1	$530	Wired	Incoming	Yes
2	$553	Credit Card	Outbound	No
3	$654	Cash	Incoming	No

Flag_PK	Payment_Type	Incoming / Outbound	Is_Bonus
1	Wired	Incoming	Yes
2	Credit Card	Outbound	No
3	Cash	Incoming	No
...			
12	Cash	Outbound	No

- Assume for example, that we have in this case, three different columns. And in the first one, there are three potential values. In the second one, only two, and in the last one, also only two. In that case, we have a total number of combinations which is 12.
- So of course then, our dimensional table should have now 12 rows to reference all of these different combinations.
- But sometimes if we have a lot of value, so a lot of indicators, this can increase exponentially. So we can end up with a lot of combinations.

- For example, if we have nine indicators, with four dimensions, we have over 260,000 combinations.

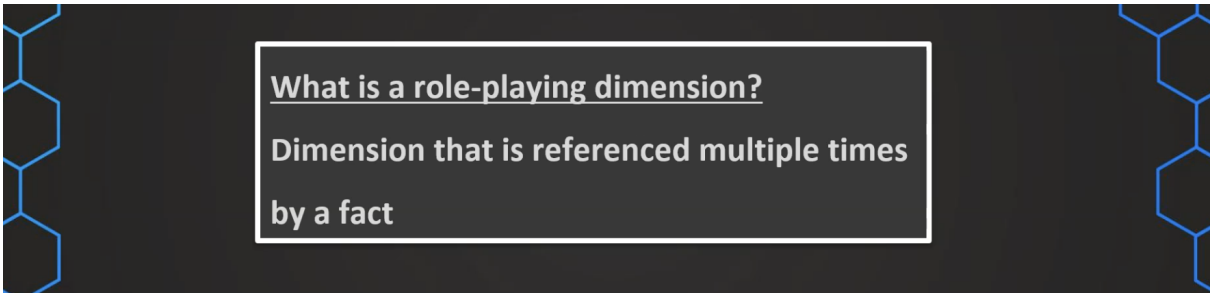


- And this would be a table that is really, really large and therefore this is not always a very good situation.
- So how could we deal with the situation of many indicators with maybe a lot of combinations in total?
- So in this case, we could only extract the combinations from the fact table that have actually occurred and disregard all of the combinations that in practice, would never occur.
- And we can see that by just extracting the available combinations from the fact table that have already occurred in the past.
- Of course, we have a little bit the risk that there might be some new combinations that are maybe super rare and that have not been created or have not been occurred in the past.
- In that case, what we could also do is to just split up these dimensions into multiple junk dimensions.
- So for example, if we, instead of putting them all all of these nine indicators in one dimension if we put them in two, we get a lot lower number of total combinations.
- For example, the one with five indicators, we have only in total a little bit more than 1000 combinations.
- So in such a situation, if we have a lot of indicators, we could also split them up into multiple dimensions.

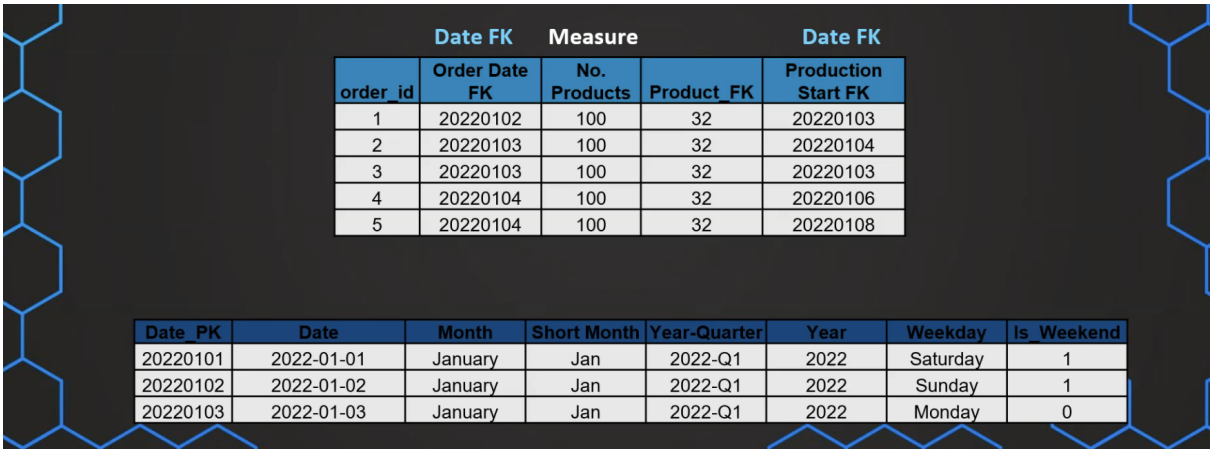
- But in general, these junk dimensions can now be used to just have all of these indicators or flags that don't suit in other dimensions. And that's why those junk dimensions can also be useful when we want to model our data.

Role playing dimension

- Applicable for date dimension. The date dimension is we know the most important dimension, and there can be in some facts, multiple date keys and then our date dimension can play multiple roles.
- So first of all, what is a role-playing dimension? So a role-playing dimension is a dimension, usually the data dimension that is referenced in a given fact table multiple times.



- So let's have a look at an example. In this case, we go back to our production table. So in this case we have one fact table that also contains order dates for specific products. And then also when we started with the production of those products.



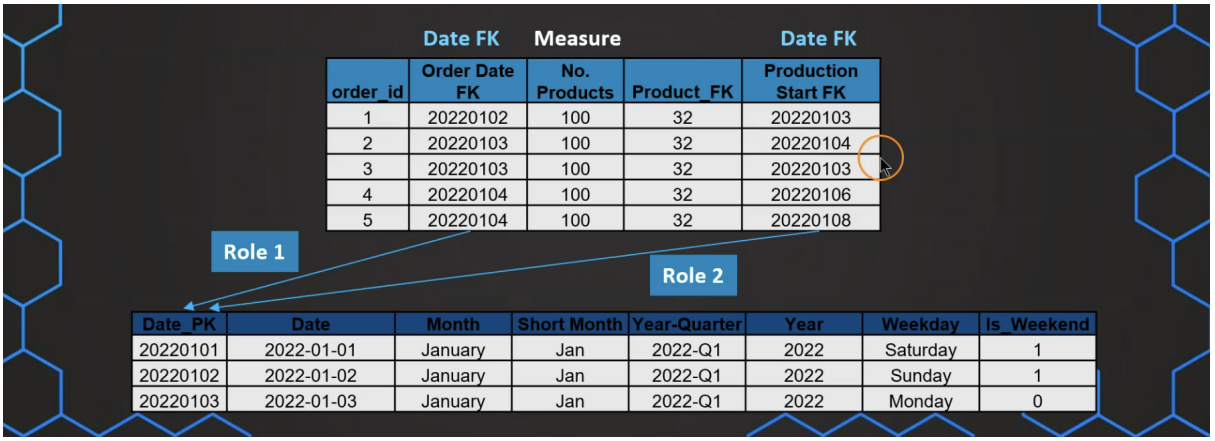
	Date FK	Measure		Date FK
order_id	Order Date FK	No. Products	Product_FK	Production Start FK
1	20220102	100	32	20220103
2	20220103	100	32	20220104
3	20220103	100	32	20220103
4	20220104	100	32	20220106
5	20220104	100	32	20220108

Date_PK	Date	Month	Short Month	Year-Quarter	Year	Weekday	Is_Weekend
20220101	2022-01-01	January	Jan	2022-Q1	2022	Saturday	1
20220102	2022-01-02	January	Jan	2022-Q1	2022	Sunday	1
20220103	2022-01-03	January	Jan	2022-Q1	2022	Monday	0

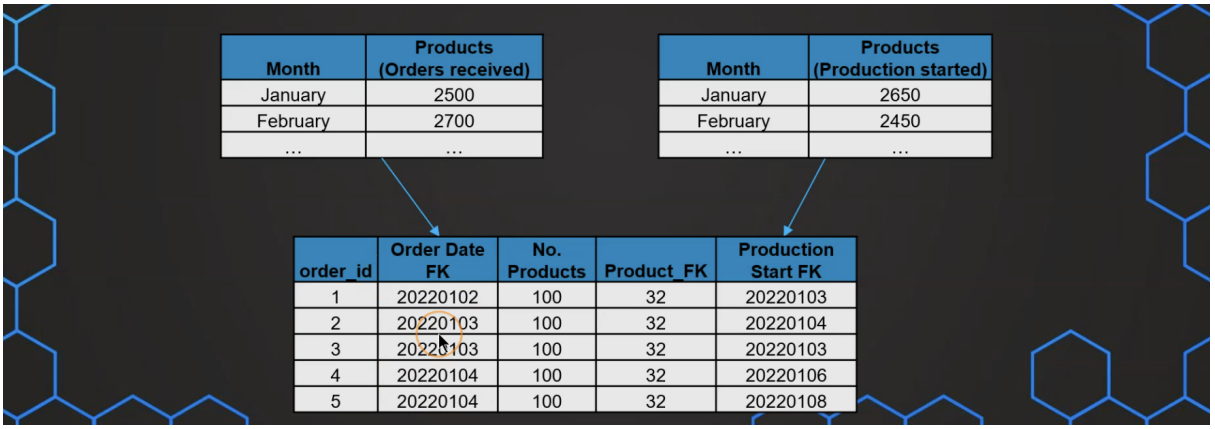
- And we can see now that we have two date foreign keys and we have one date dimension. And now it wouldn't make sense to duplicate this

dimension physically in our database and have now one dimension that is for the order date and one dimension that is for the production date.

- And therefore we can use that same table. But now with different role.
- So we can create, for example, a join or a relationship to this same date table, but in two different ways.

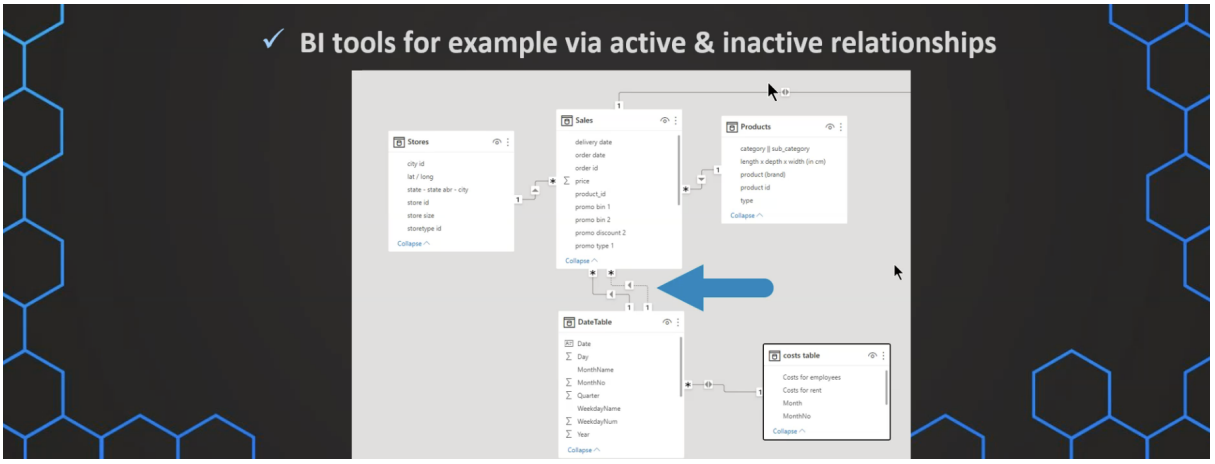


- So on the first one, we use that date foreign key, and in the second one we use that production foreign key.
- The first role is just the order date. And then the second role that this data dimension plays is the production date. So these are now the two roles that this date dimension is playing.
- And now with that, we can analyze both the orders received so based on the date dimension of the month, but now if we want to use the other role, so the other relationship or the other join with the other column, then we can also display the products based on the date of the production. So when the production has started.

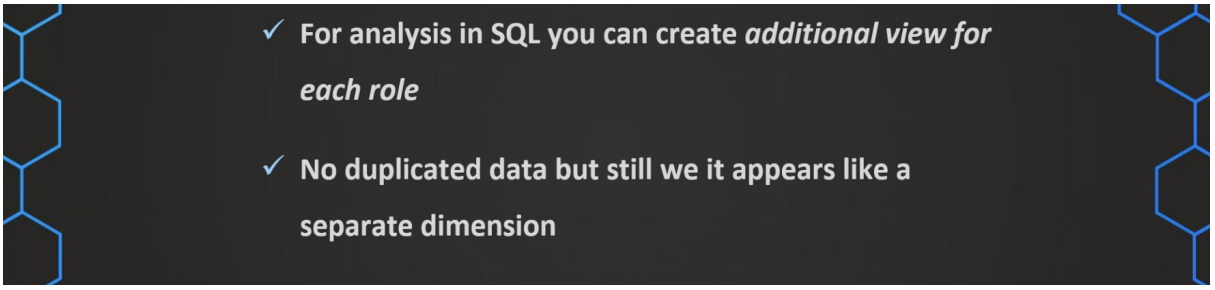


- So in this case, we just use two different relationships.
- For example, if we are in Power BI or in Tableau in any other BI tool, or we can also, if we are just in SQL analyzing the data, use different joins.
- So once we just join this date table with the production foreign key, and the second time we join the same date dimension with a different date key.
- So for example, let's see how this looks like in practice we have seen that we use different BI applications. The most common one is probably Power BI.

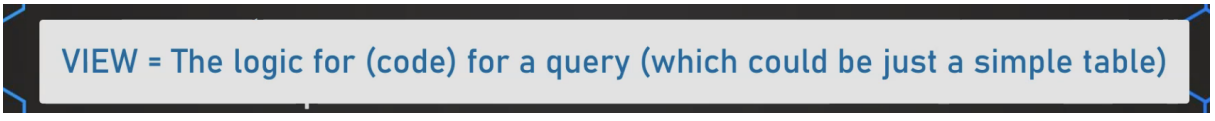
- In this case, we can set up our data model and we can set up relationships between our tables. And in this case, we can create one active relationship. So this will be the default relationship.
- We can just drag and drop, for example the month and then the orders. So we can count the orders and then the default relationship will be used.
- But then we can also change that and use the inactive relationship in some of the calculations, some of the columns.



- This role-playing dimension can be referenced multiple times. So by different foreign keys in one fact table. So this is possible in BI tools but also in SQL we can create
- If we want to analyze the data a lot in SQL, we can also create an additional view for each of the roles that this date dimension is playing.



- So for example, in our case we can just set up a second view. A view is something that is just basically referencing a physically available table but it is not physically available.

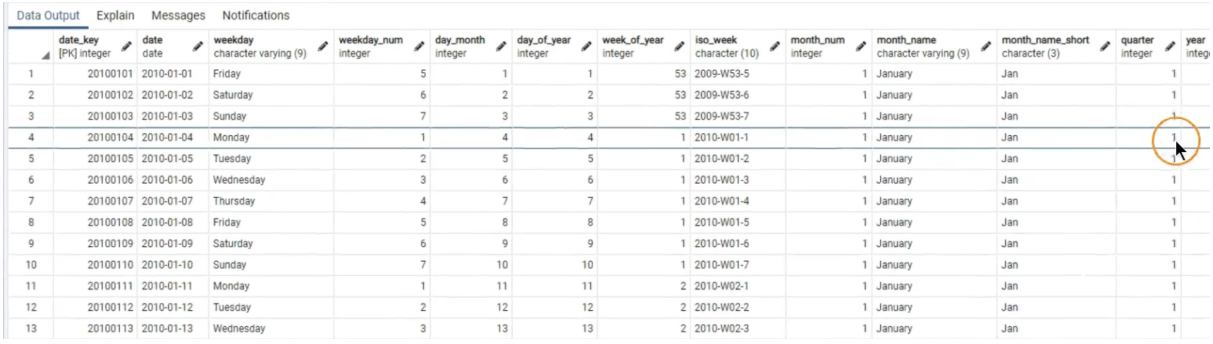


- So it has a new name and it appears like a separate table, but in fact it's just referencing the original physically available table.
- And with that, we don't create duplicate data which is of course not so nice because it's redundant data but it still appears as a second dimension that is dedicated, for example the production start date.
- And with that, we can create multiple joins and we can now use this dedicated view for the role this view is dedicated to.

- So to summarize things this role-playing dimension is commonly used if we have a date table. So this is the most typical case.
- And we have in our fact table multiple references to this same dimension. For example, the same date dimension.
- But we can still implement that by having, for example in BI tools, then active and inactive relationships.
- Or if we are analyzing the data mainly on SQL then we can also create additional views for each of the roles, which just makes it easier for the analysts to analyze the data.

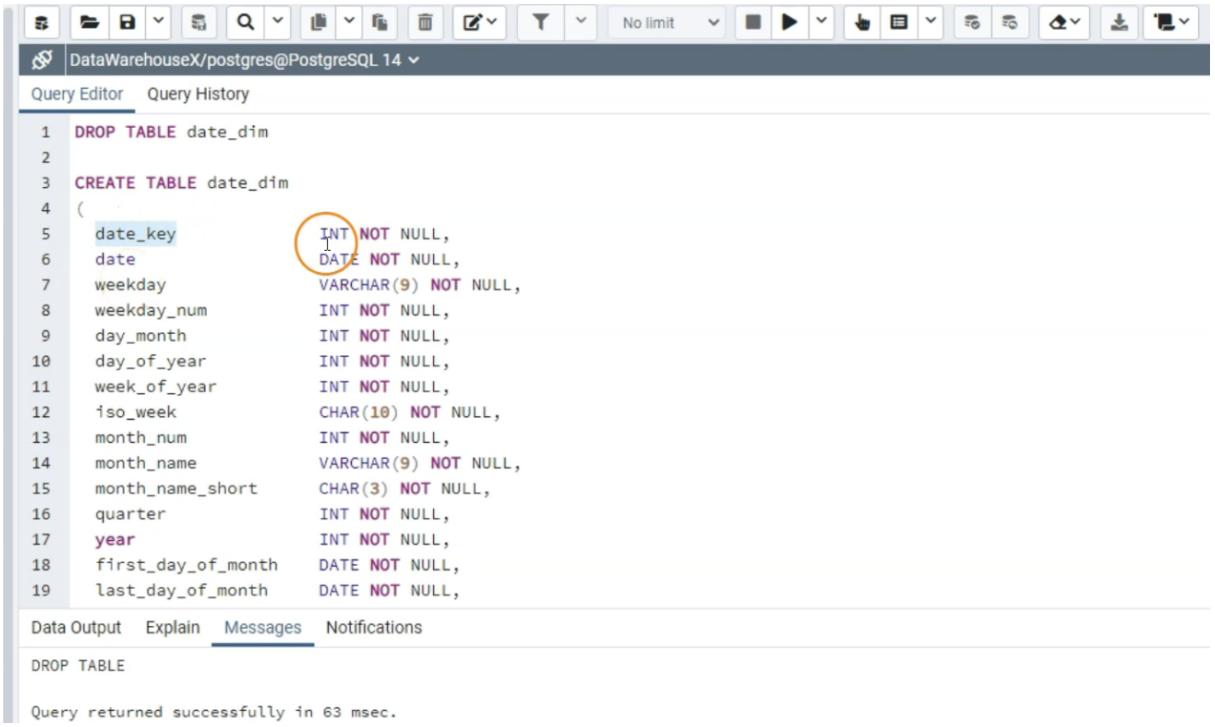
Case study: Date dimension

- Date dimension is probably the most important dimension, but luckily we can set this up very easily in our database management system using SQL code, and we can just pre-populate that without any source data. So this is done very easily using the SQL code.



Data Output | Explain | Messages | Notifications

	date_key [PK] integer	date date	weekday character varying (9)	weekday_num integer	day_month integer	day_of_year integer	week_of_year integer	iso_week character (10)	month_num integer	month_name character varying (9)	month_name_short character (3)	quarter integer	year
1	20100101	2010-01-01	Friday	5	1	1	53	2009-W53-5	1	January	Jan	1	
2	20100102	2010-01-02	Saturday	6	2	2	53	2009-W53-6	1	January	Jan	1	
3	20100103	2010-01-03	Sunday	7	3	3	53	2009-W53-7	1	January	Jan	1	
4	20100104	2010-01-04	Monday	1	4	4	1	2010-W01-1	1	January	Jan	1	
5	20100105	2010-01-05	Tuesday	2	5	5	1	2010-W01-2	1	January	Jan	1	
6	20100106	2010-01-06	Wednesday	3	6	6	1	2010-W01-3	1	January	Jan	1	
7	20100107	2010-01-07	Thursday	4	7	7	1	2010-W01-4	1	January	Jan	1	
8	20100108	2010-01-08	Friday	5	8	8	1	2010-W01-5	1	January	Jan	1	
9	20100109	2010-01-09	Saturday	6	9	9	1	2010-W01-6	1	January	Jan	1	
10	20100110	2010-01-10	Sunday	7	10	10	1	2010-W01-7	1	January	Jan	1	
11	20100111	2010-01-11	Monday	1	11	11	2	2010-W02-1	1	January	Jan	1	
12	20100112	2010-01-12	Tuesday	2	12	12	2	2010-W02-2	1	January	Jan	1	
13	20100113	2010-01-13	Wednesday	3	13	13	2	2010-W02-3	1	January	Jan	1	




```
ALTER TABLE public.date_dim ADD CONSTRAINT date_dim_pk PRIMARY KEY (date_key);

CREATE INDEX d_date_date_actual_idx
  ON date_dim(date);

COMMIT;

INSERT INTO date_dim
SELECT TO_CHAR(datum, 'yyyymmdd')::INT AS date_key,
       datum AS date,
       TO_CHAR(datum, 'TMDay') AS weekday,
       EXTRACT(ISODOW FROM datum) AS weekday_num,
       EXTRACT(DAY FROM datum) AS day_month,
       EXTRACT(DOY FROM datum) AS day_of_year,
       EXTRACT(WEEK FROM datum) AS week_of_year,
       TO_CHAR(datum, 'TMMonth') AS month_name,
       TO_CHAR(datum, 'Mon') AS month_name_short,
       EXTRACT(QUARTER FROM datum) AS quarter,
       EXTRACT(YEAR FROM datum) AS year,
       datum + (1 - EXTRACT(DAY FROM datum))::INT AS first_day_of_month,
       (DATE_TRUNC('MONTH', datum) + INTERVAL '1 MONTH - 1 day')::DATE AS last_day_of_month,
       CONCAT(TO_CHAR(datum, 'yyyy'),'-',TO_CHAR(datum, 'mm')) AS mmyyyy,
       CASE
           WHEN EXTRACT(ISODOW FROM datum) IN (6, 7) THEN 'weekend'
           ELSE 'weekday'
           END AS weekend_indr
FROM (SELECT '2010-01-01'::DATE + SEQUENCE.DAY AS datum
      FROM GENERATE_SERIES(0, 7300) AS SEQUENCE (DAY)
      GROUP BY SEQUENCE.DAY) DQ
ORDER BY 1;

COMMIT;
```

Data Output | Explain | Messages | Notifications

	date_key [PK] integer	date date	weekday character varying (9)	weekday_num integer	day_month integer	day_of_year integer	week_of_year integer	iso_week character (10)	month_num integer	month_name character varying (9
1	20100101	2010-01-01	Friday	5	1	1	53	2009-W53-5	1	January
2	20100102	2010-01-02	Saturday	6	2	2	53	2009-W53-6	1	January
3	20100103	2010-01-03	Sunday	7	3	3	53	2009-W53-7	1	January
4	20100104	2010-01-04	Monday	1	4	4	1	2010-W01-1	1	January
5	20100105	2010-01-05	Tuesday	2	5	5	1	2010-W01-2	1	January
6	20100106	2010-01-06	Wednesday	3	6	6	1	2010-W01-3	1	January
7	20100107	2010-01-07	Thursday	4	7	7	1	2010-W01-4	1	January
8	20100108	2010-01-08	Friday	5	8	8	1	2010-W01-5	1	January
9	20100109	2010-01-09	Saturday	6	9	9	1	2010-W01-6	1	January
10	20100110	2010-01-10	Sunday	7	10	10	1	2010-W01-7	1	January
11	20100111	2010-01-11	Monday	1	11	11	2	2010-W02-1	1	January

- We can also have future dates populated

Data Output | Explain | Messages | Notifications

	date_key [PK] integer	date date	weekday character varying (9)	weekday_num integer	day_month integer	day_of_year integer	week_of_year integer	iso_week character (10)	month_num integer	month_name character varying (9)	month_name_short character (3)	quarter integer
7289	20291215	2029-12-15	Saturday	6	15	349	50	2029-W50-6	12	December	Dec	4
7290	20291216	2029-12-16	Sunday	7	16	350	50	2029-W50-7	12	December	Dec	4
7291	20291217	2029-12-17	Monday	1	17	351	51	2029-W51-1	12	December	Dec	4
7292	20291218	2029-12-18	Tuesday	2	18	352	51	2029-W51-2	12	December	Dec	4
7293	20291219	2029-12-19	Wednesday	3	19	353	51	2029-W51-3	12	December	Dec	4
7294	20291220	2029-12-20	Thursday	4	20	354	51	2029-W51-4	12	December	Dec	4
7295	20291221	2029-12-21	Friday	5	21	355	51	2029-W51-5	12	December	Dec	4
7296	20291222	2029-12-22	Saturday	6	22	356	51	2029-W51-6	12	December	Dec	4
7297	20291223	2029-12-23	Sunday	7	23	357	51	2029-W51-7	12	December	Dec	4
7298	20291224	2029-12-24	Monday	1	24	358	52	2029-W52-1	12	December	Dec	4
7299	20291225	2029-12-25	Tuesday	2	25	359	52	2029-W52-2	12	December	Dec	4
7300	20291226	2029-12-26	Wednesday	3	26	360	52	2029-W52-3	12	December	Dec	4
7301	20291227	2029-12-27	Thursday	4	27	361	52	2029-W52-4	12	December	Dec	4

Quiz

Question 1:

Why do we even use dimension in our data model?

- ○ It helps with update and insert operations
- ◉ **It helps with performance and usability**
- ○ It helps to deal with heterogeneous data

Question 2:

What is a common key format for date dimensions?

- ○ We commonly use a standard (and meaningless) surrogate key like 1, 2, 3, 4,...
- ◉ **We commonly use a meaningful surrogate key in the format YYYYMMDD.**
- ○ We commonly use no surrogate key but just the date column in the format "YYYY-MM-DD".

✓ **Good job!**
Yes. We create multiple timestamp keys that can be all used together with our date dimension. That means our date dimension can play different roles - depeinding on which key is used to create the relationshipt to the date table.

Question 3:

In your order processing fact table you have multiple timestamps for different steps of the order processing. The analysts/business users want to analyze the data based on all these different timestamps. How can you fulfill their requirement?

- ◉ **Using a role-playing dimension**
- ○ Using a degenerate dimension
- ○ Using a junk dimension
- ○ Using a conformed dimension