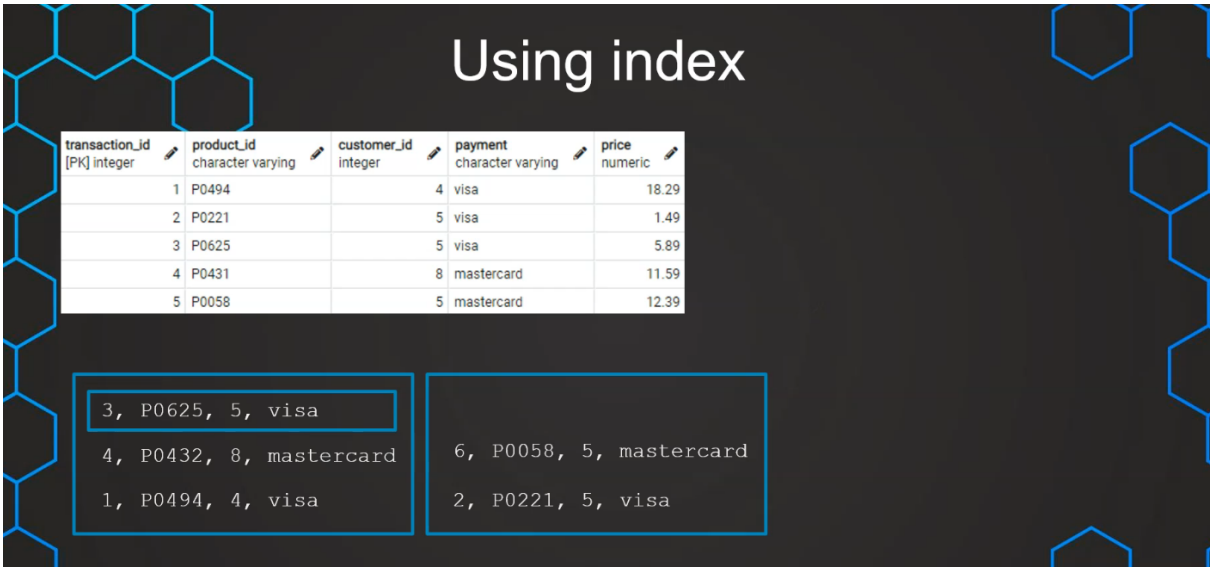
13. Optimizng a Data Warehouse

Using Indexes

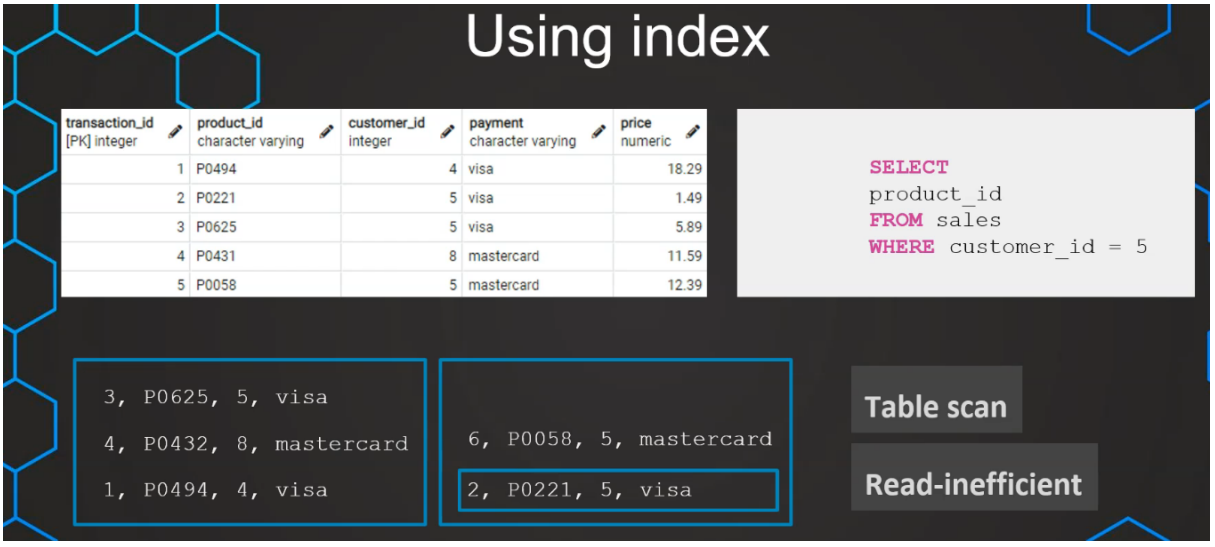
- After we have set up our data warehouse, it may happen that the business users in some situations may experience a bad query performance.
- This means that they would have to wait very long for their results and their reports. And also if they want to work on the database they have to just wait very long until the data is retrieved.
- And in such cases, the first thing that we want to think about when we want to improve the performance of our queries is to use indexes.
- We first want to start with what are actually indexes and why do they help us? So to understand that we first need to understand that in a database, in a given table the data is not stored in a systematic order but just wherever there's some free space available.
- So there's no particular order just how it can be arranged on the disc space. And this is then a little bit problematic if we are trying to retrieve the data.



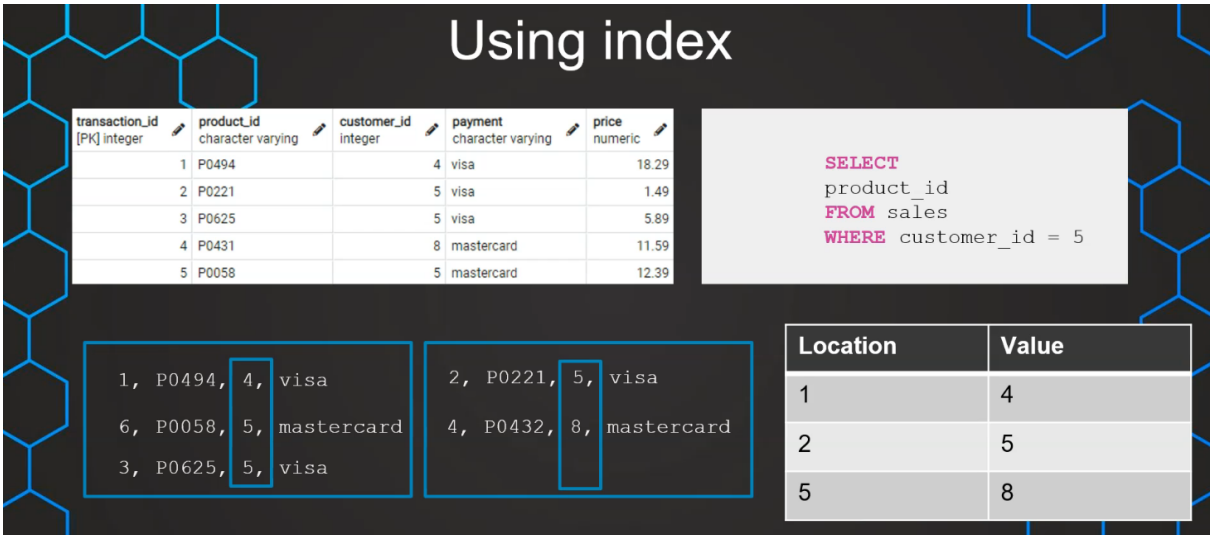
- So of course it's very efficient if the data is written but if we now want to retrieve that data, for example we want to filter just based on one specific

customer then the query has to scan the entire table.

- So it has to go through all of the different rows and search for query number five. And also if it finds one how do we know that there's no more search required? Maybe we have already found all of them, but still the query needs to scan the entire table.

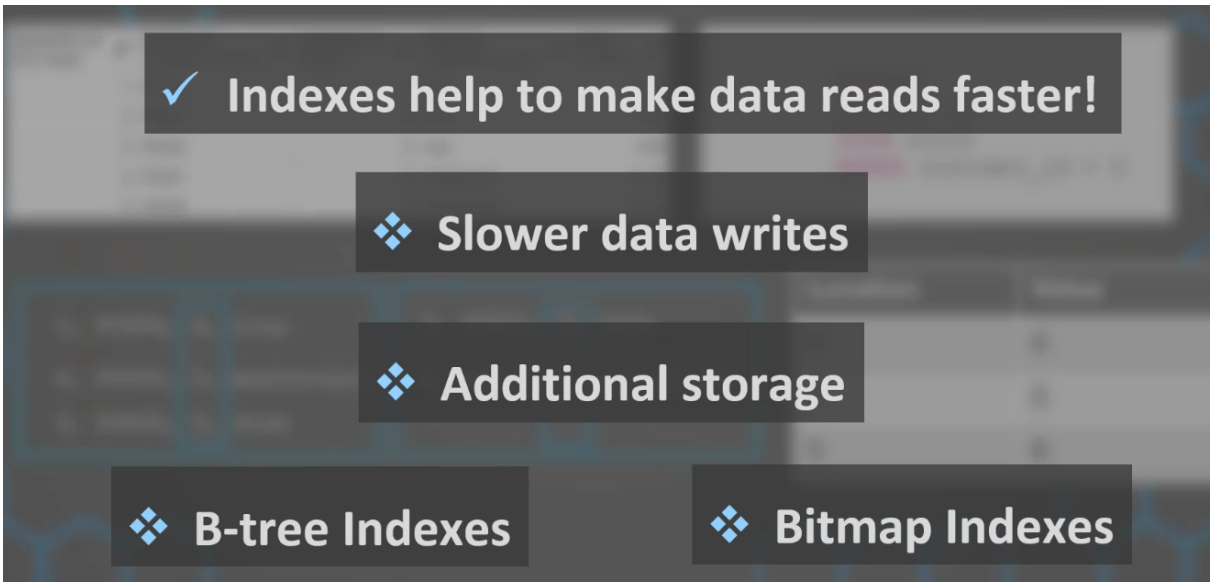


- So this is a full table scan, and it's very inefficient in terms of getting the data out.
- And that is why we want to use indexes. So those indexes are basically storing the data in a specific order.



- So for example, if we place an index on the customer ID this will be in a particular order and we can then just find a pointer basically.

- So for example, the value four for the customer ID is in location. So row ID number one. Then we have value number five. This starts at row number two, and then the next value for the customer ID is eight. And this starts then at the location number five.
- So this is how we have then pointers that just help to find where are the specific rows stored so that we don't need a full table scan but we can just directly go to the location.
- And with that, we can read the data much quicker. So this is the use of indexes.

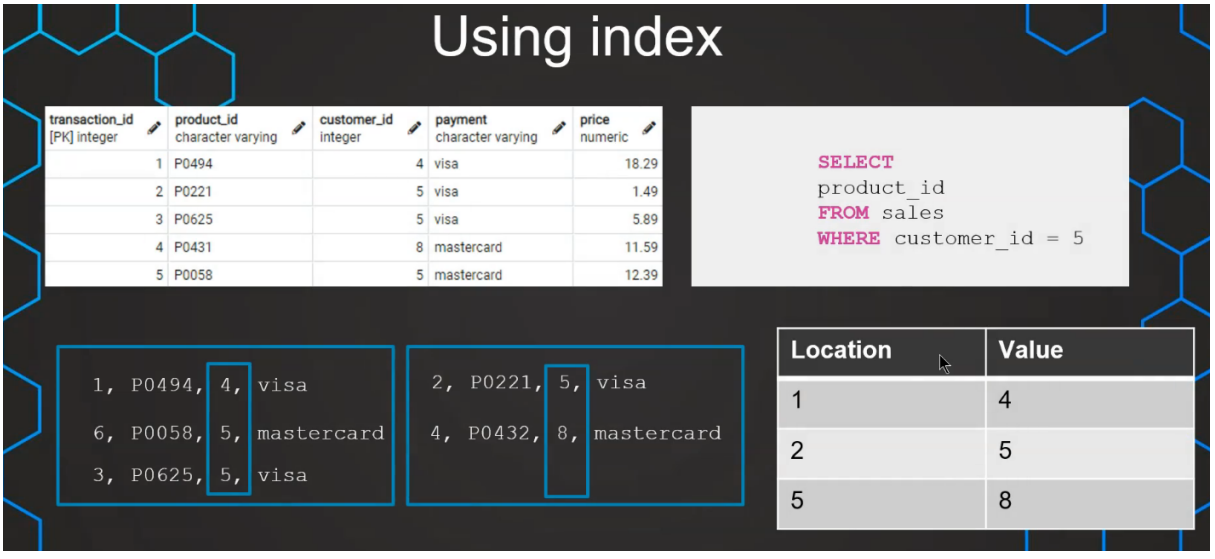


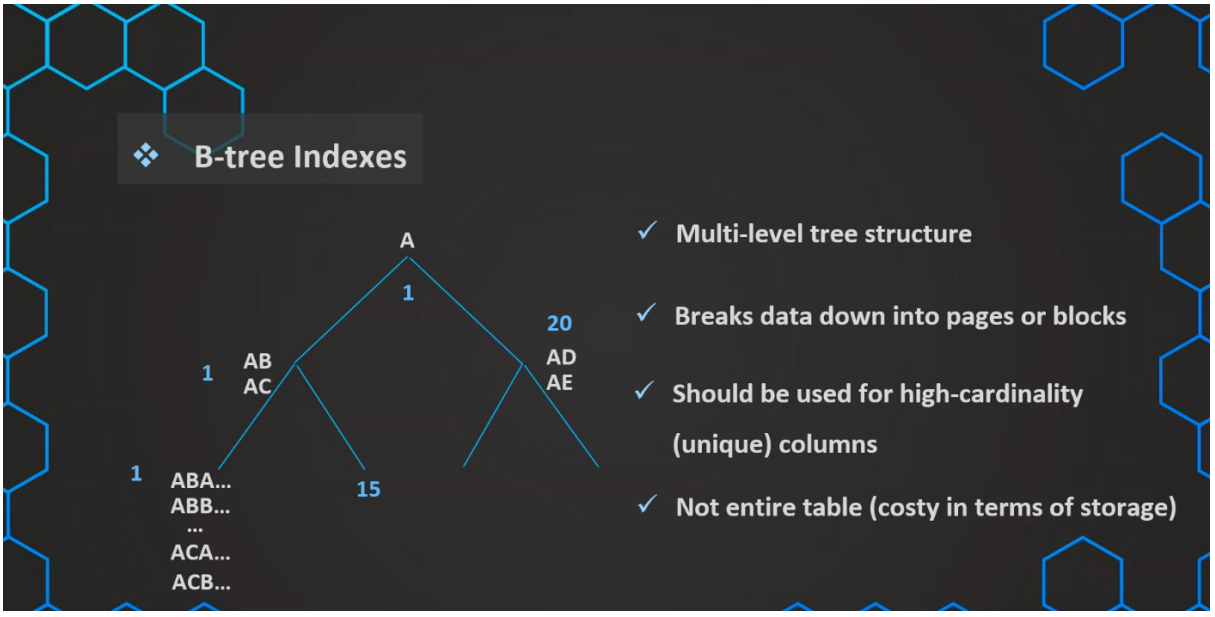
- They help us to make the data retrievals faster because we don't need to do a full table scan anymore.
- And oftentimes, in particular, in large tables the full table scan is what takes up most of the query time till the data is retrieved. And that's why optimizing that can be very helpful.
- On the other side, there is also downsides to these indexes because they, for example make writing the data and updating the data much slower because now also the key needs to be maintained and we can now not write the data just wherever we want to.
- And on top of that there's also some additional storage needed.
- So if we put many different indexes this can be sometimes quite a lot of storage actually that is needed on top of that.

- So therefore, we now want to talk about different keys that we can use and how we should use them.
- And in particular, in our case we talk about so-called B-tree indexes. Those are basically the standard indexes.
- And then there are also so-called bitmap indexes and they are particularly helpful for data warehouses.
- And we now want to understand what are these different indexes, how do they work, how do they differ? And that's why we want to, in the next topic we can have a closer look at how those two different types of indexes work and how we should use them.

B-tree index

- we've seen that indexes can help us to make data reading faster, especially if the data needs to be filtered so that we can avoid full table scans. So that's why we first want to now start with the first type of index, which is the B-tree index.

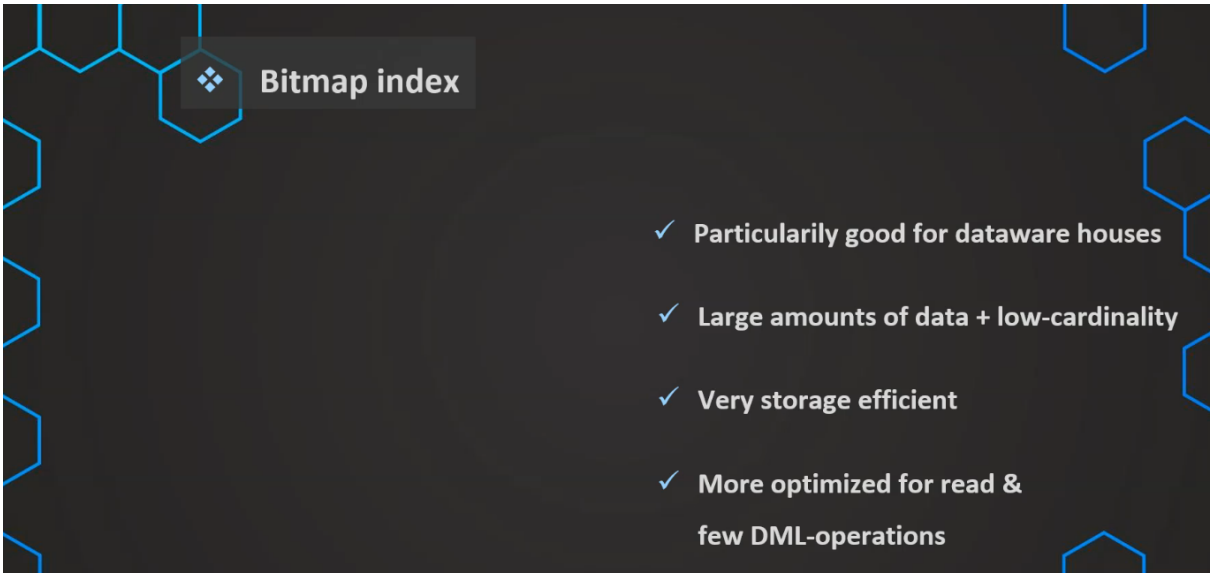




- It's mentioned, this is the standard index. So if there's no particular type of index mentioned, we usually talk about this B-tree index.
- And this is just a multi-level tree structure that is just using a tree logic to break the data down into different pages or blocks.
- So, for example, if we put an index on a name of a person, this can be broken down by the first character. And then we have, for example, the first character is an A, then we look at the second character. If the second is then a B, then this can be broken down even more. And then on top of those values, we also of course have the information, where is this data stored? So the row number is also then included.
- So for example, in here, if we want to look at Adam, we have AD, we should start looking at row ID number 20. And like this, we can just break down our data in pages and blocks with this tree structure. And like this, avoid our full table scans.
- So this type of tree is used usually if we have, usually, unique values in our column. So, oftentimes, it is a surrogate key, our primary key, that is just containing unique values. That means every single value occurs only once. So we have a very high range of different values. This means we have a high-cardinality. Just a high range of values.
- And we should note that we should never put an index just blindly on all columns in our tables. Because we have seen that there are some downsides like cost, in terms of storage, in terms of update and insert operations.

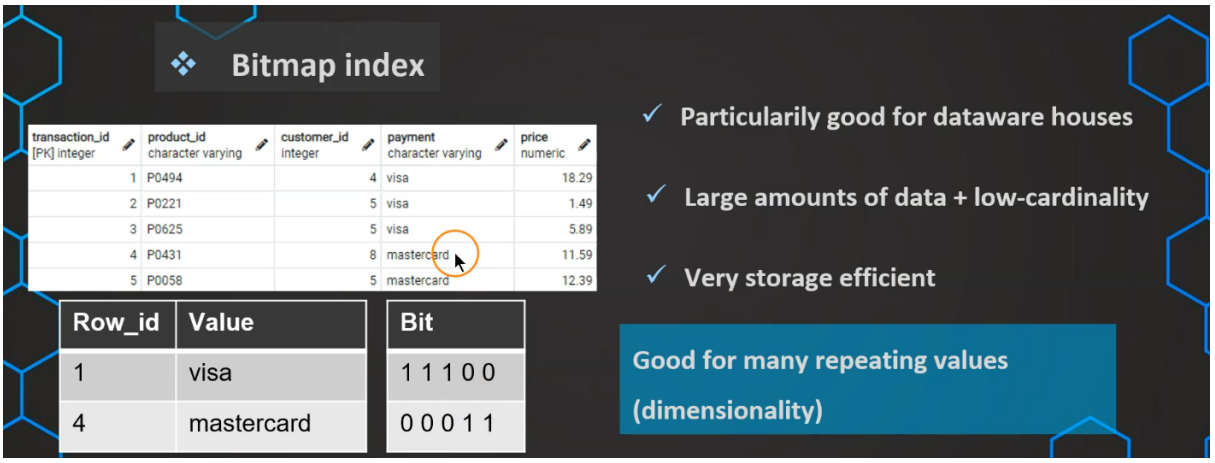
Bit map index

- But now there's also the so-called bitmap index, which is a special type of index that can be especially good if we are working on a data warehouse.
- So this is a type of index that is beneficial, especially for large amounts of data in combination with a low cardinality.
- So for example, if we have only a few different values in our table, but the table itself is huge, so for example, one column can only take three different values or maybe only two different values, then this is a low cardinality and then a bitmap index is a good choice in our data warehouse.
- Because the benefit of that is since the data is stored in so-called bits, we will see later on how this works in a little bit more detail. It is very storage efficient, and on top of that, it is also very good for the read performance with the downside that it is not so efficient for our data manipulation.
- So if the data needs to be updated or inserted, in this case, it is not so efficient, but we can basically live with that a lot better because we are in a data warehouse, and the most part is getting the data out.
- So reading the data and that's why a bitmap index is usually a very good choice if we have a low cardinality in our data warehouse.

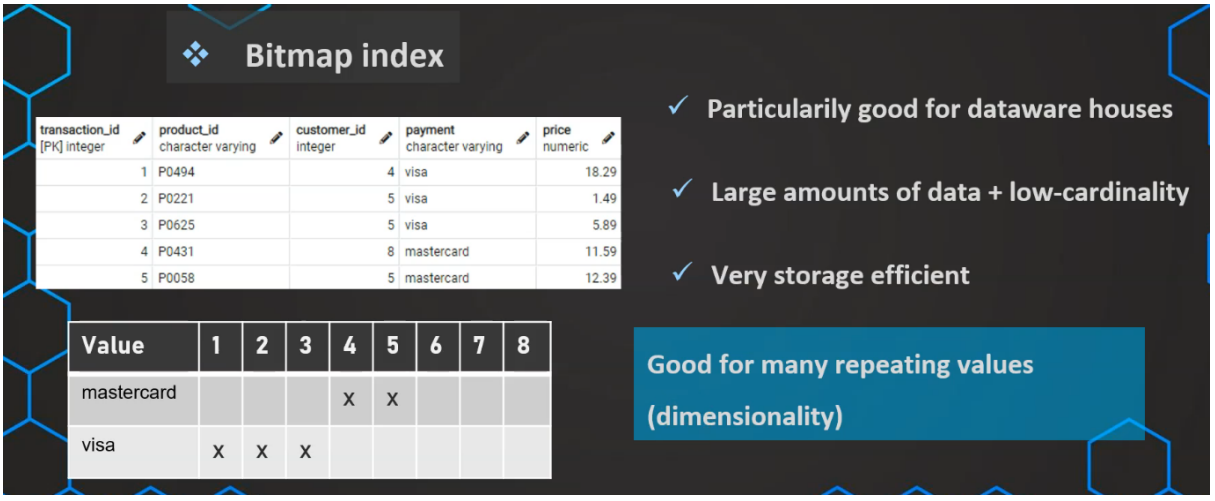


- But now, how does this bitmap index work? So we have this table, and of course we now also want to create a structure that helps us to avoid full table scans. And in this case we don't use a list, but we use bits.

- So this can look something like this.



- We have the different payment types and we see that there are only two different types in our case. And this is why we can now create different maps basically that are telling us in which rows we find which values.

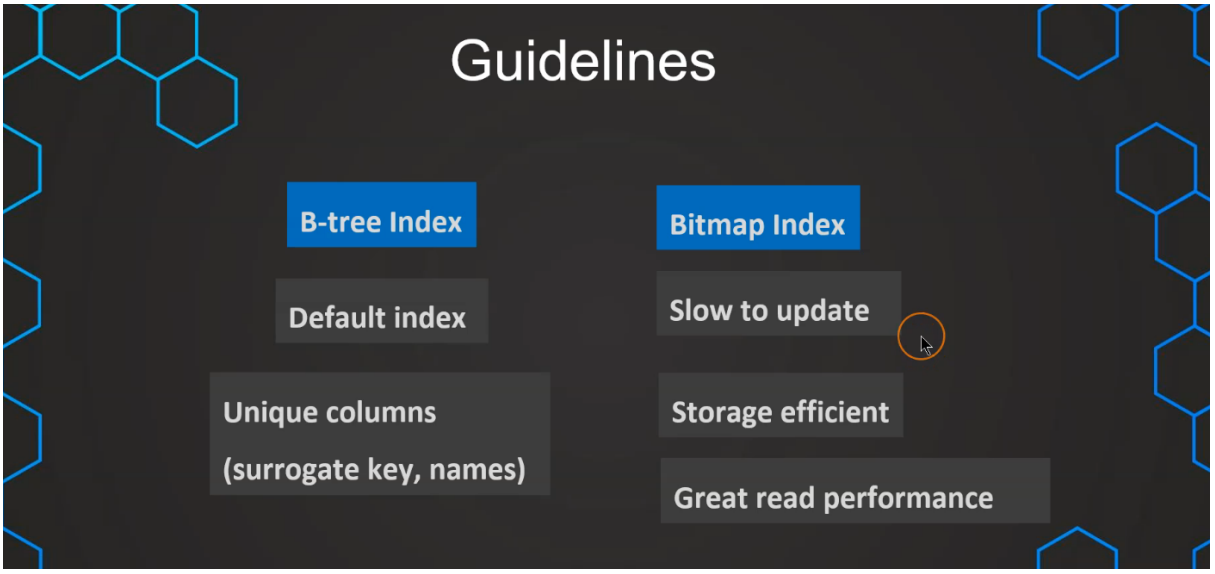


- So for example, for the value visa, we can find this value in row number one because there's a 1. And in row number two, also because there's also a 1, in row number 3, there's also a one. And then in the next row we see that in row number 4 there is a zero. And that's why we cannot find the value visa. But if we now go to the Value MasterCard, we find in row number four, yes, it is turned to one. That means yes, there is a value MasterCard, and this is then how basically there can be a map created that helps to find in which rows we find which value.
- And as mentioned, this is a very good strategy if we have not so many repeating values, because then this is a very efficient method.

- So now that we have had a quick introduction into those two different types of indexes, we now also quickly want to get some guidelines on how we should go about using those indexes for which columns, in which tables, in what situations do we want to use, which of those indexes,

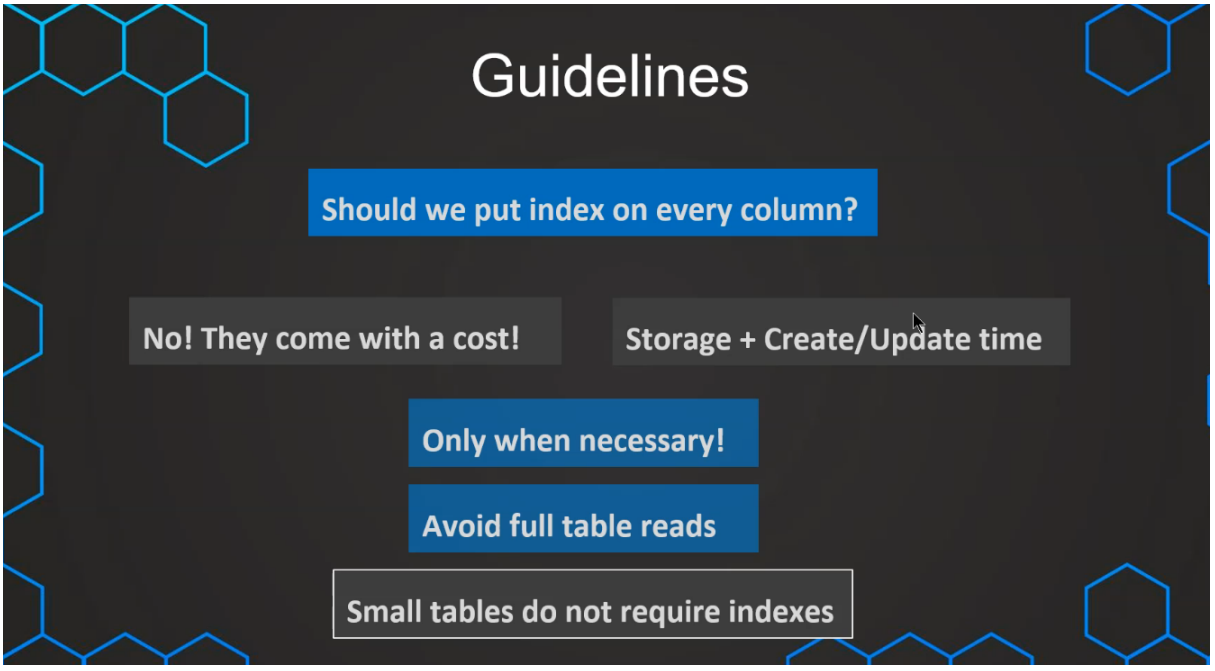
Guidelines for Indexes

- we've already learned that indexes can help us to get the data faster out of the database. And with that, increase the performance of our database.



- We've heard first about the B-tree index which is the default index.
- So if there's nothing else specified and we just talk about an index, usually we refer to the standard B-tree index.
- And this is good choice if we have unique columns or at least columns with a very high range of different values, that means a high cardinality in this column.
- And this is typically something on the surrogate keys and for example, also on phone numbers, on maybe addresses, on names if it's a full name, because those are usually very rare in having duplicates. So this is something that can be helpful there.
- And note also that this B-tree index is usually also, per default, set on our primary key. So we don't always need to explicitly denote this index, but it can be already set up automatically on our primary key.

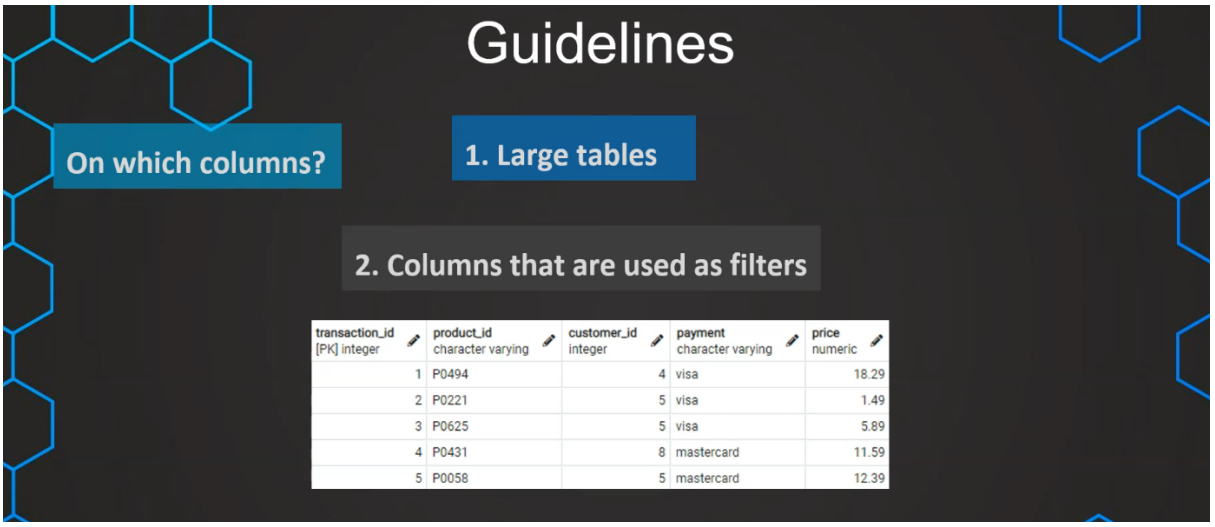
- If, for example, we have denoted our surrogate key as our primary key in our database then there will be also usually automatically an index created on this column.
- Then on the other side we have the bitmap index which we've learned has the downside that it is slow in the updating and inserting, but as we are working in a data warehouse we can very easily live with this downside and we have the benefit that it's super storage efficient due to saving the row locations in bits and it is also extremely good in the read performance.



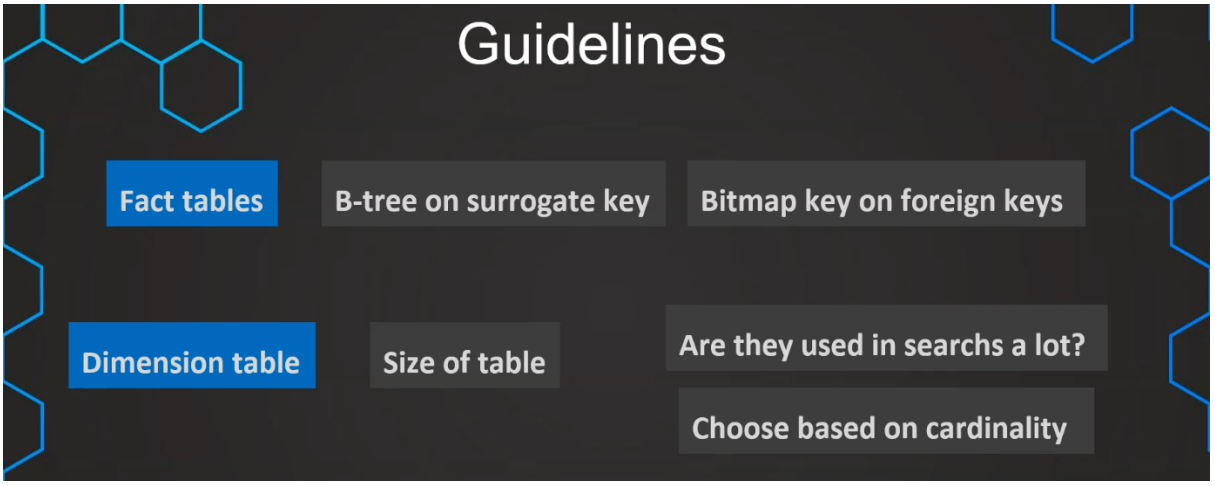
- The first question is, should we put now, since we have these benefits, an index on every single column in our entire table? So of course the answer to that is no, because we've already seen that those indexes come with a cost.
- First, because they can really require significant amount of additional storage. And also in the creation and the updating, it is really a bit slower. And that's why also we don't want to go too far into just creating an index for every single column because this will then also slow our update and our inserts down. And all of that needs to be maintained.
- So that's why we should really think about when we want to use those indexes and we should only use them if they are necessary. That means if we experience slow query performance due to the full table scans because with these indexes we have learned that we want to avoid those full table

reads. And this brings us to our first guideline, and that is, small tables don't require any indexes.

- Because even if we have an index on them, usually the query optimizer is not even using that index or it is not giving any additional benefit in comparison to a full table read. So this is our first guideline to use only these indexes on large tables, especially if we experience a low query performance and we need to filter the data a lot.
- So with that first rule of thumb, that we should only use those indexes for large tables with a low query performance



- we then want to ask the question, now on which columns should we place an index? And the answer to that is to those columns and also to those tables that also often need to have the data filtered.
- And then if we, for example, use a filter on, let's say, the customer_id very frequently, and this is a column that is used to just get maybe 10% of the table, then we can place an index on that exact column because this will then help to avoid this full table scan, if we only want to retrieve, let's say, 10% of the data.
- So this is the column that are then the good columns for using an index.



- And if we now look on our data warehouse specifically, we want to now see on which tables, fact tables, dimension tables, we should place our indexes and also what type of indexes should we use.
- We've already learned that the B-tree index is good for our primary keys, and, in fact, if we have just set this as a primary key in our database, there will be also automatically usually already set up this index and it'll be a B-tree index.
- On the other side though, we also want to consider an index on the foreign key because the foreign key is oftentimes used to filter the data, and it's also used to join to different tables. And putting a key on this join column will also increase the performance.
- That's why we can get additional benefits by placing a bitmap key on our foreign key. Of course, again, it's depending on the cardinality.
- So if we have a huge range of different values in a huge dimension table then it could be also more beneficial to use a B-tree index.
- So this is depending on the amount of values. So five different values of course, or 10 different, or 20, or 100, a bitmap key is usually then the better choice.
- And now on our dimension table, we first have to ask the question do we really need an index? And the answer to that is it really depends on two things.
- First of all, on the size of the table. If it's a very small table with, let's say, a few hundred rows then we probably don't even need an index. We wont benefit from it.

- But if we have millions of rows, then we can ask the next question, are there columns that are used a lot in searches to filter the data? And if the answer is yes, we can use either a bitmap key or a B-tree index, and which one should we use? This is, again, depending on the cardinality.
- If we have only two different values, for example, we have only 10 different categories in our huge dimension table, because we have maybe a lot of subcategories, then in this low cardinality, again, we use bitmap key, but if we have a very high cardinality, we typically benefit more from the B-tree index.
- So these are the very general guidelines that we can use to increase the performance of our data warehouse by using indexes.

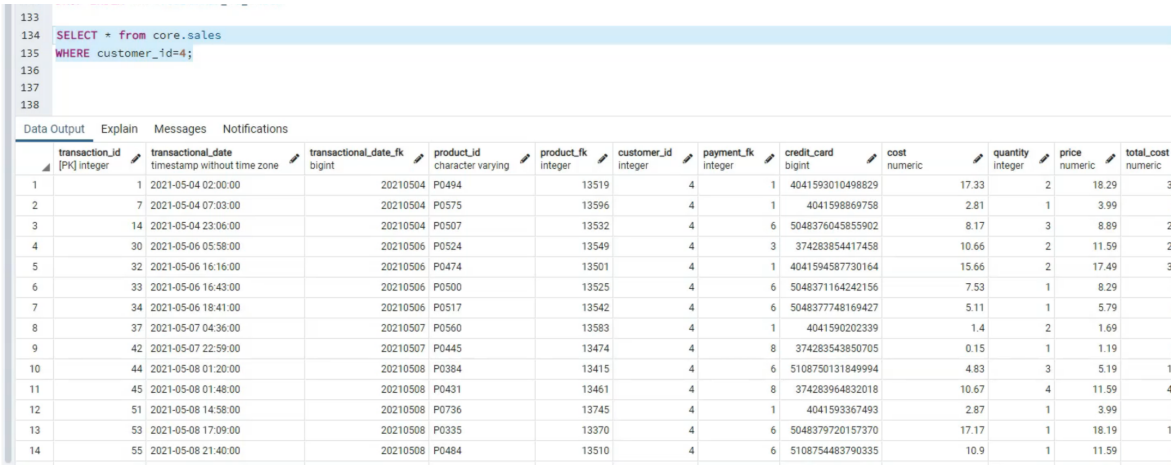
Demo: Setting indexes

- If we want to place an index on a column in our database we just have to go to our database management system and set up an index there.
- So this can be done via the following syntax, of course. Again, this can depend on the database system. Maybe on Microsoft it's a little bit different than on Oracle or on Postgres, but it is almost the same in all of those types.
- Again, they also can have different methods for indexes but these are usually rather small differences and in the end, they work pretty much in the same way.

```
CREATE INDEX index_name ON table_name [USING method]
    (
     column_name [ASC | DESC],
     ...
    );
```

- So for Postgres, we can use this index so we can use the command create index the name of the index, and then on which table we want to use that index.

- So this can be then also optionally specified which method we want to use. And if we don't specify anything, again the default is the B three index And then on which columns do we want to place this index.

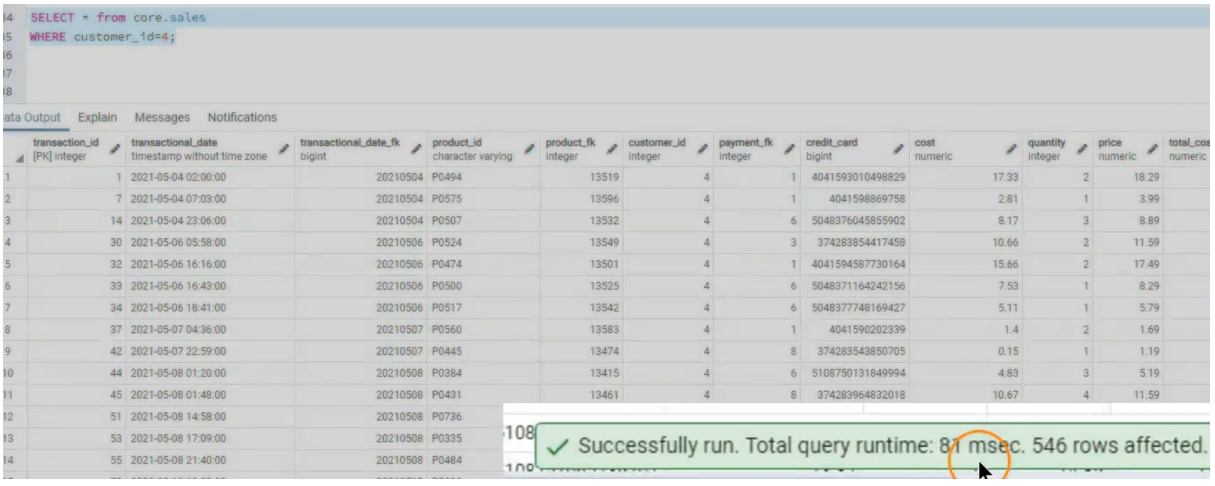


```
133
134 SELECT * from core.sales
135 WHERE customer_id=4;
136
137
138
```

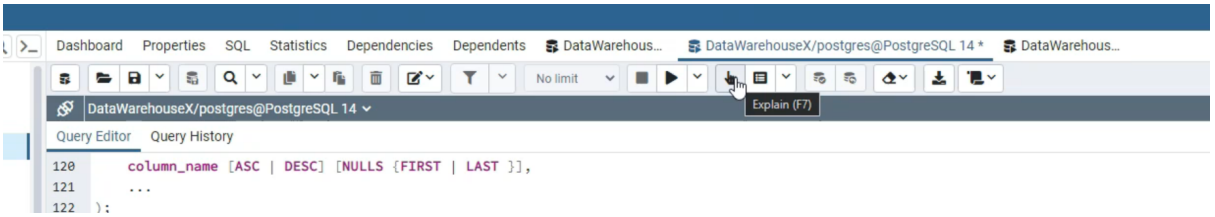
Data Output | Explain | Messages | Notifications

	transaction_id [PK] integer	transactional_date timestamp without time zone	transactional_date_fk bigint	product_id character varying	product_fk integer	customer_id integer	payment_fk integer	credit_card bigint	cost numeric	quantity integer	price numeric	total_cost numeric
1	1	2021-05-04 02:00:00	20210504	P0494	13519	4	1	4041593010498829	17.33	2	18.29	
2	7	2021-05-04 07:03:00	20210504	P0575	13596	4	1	4041598869758	2.81	1	3.99	
3	14	2021-05-04 23:06:00	20210504	P0507	13532	4	6	5048376045855902	8.17	3	8.89	
4	30	2021-05-06 05:58:00	20210506	P0524	13549	4	3	374283854417458	10.66	2	11.59	
5	32	2021-05-06 16:16:00	20210506	P0474	13501	4	1	4041594587730164	15.66	2	17.49	
6	33	2021-05-06 16:43:00	20210506	P0500	13525	4	6	5048371164242156	7.53	1	8.29	
7	34	2021-05-06 18:41:00	20210506	P0517	13542	4	6	5048377748169427	5.11	1	5.79	
8	37	2021-05-07 04:36:00	20210507	P0560	13583	4	1	4041590202339	1.4	2	1.69	
9	42	2021-05-07 22:59:00	20210507	P0445	13474	4	8	374283543850705	0.15	1	1.19	
10	44	2021-05-08 01:20:00	20210508	P0384	13415	4	6	5108750131849994	4.83	3	5.19	
11	45	2021-05-08 01:48:00	20210508	P0431	13461	4	8	374283964832018	10.67	4	11.59	
12	51	2021-05-08 14:58:00	20210508	P0736	13745	4	1	4041593367493	2.87	1	3.99	
13	53	2021-05-08 17:09:00	20210508	P0335	13370	4	6	504837972015737	17.17	1	18.19	
14	55	2021-05-08 21:40:00	20210508	P0484	13510	4	6	5108754483790335	10.9	1	11.59	

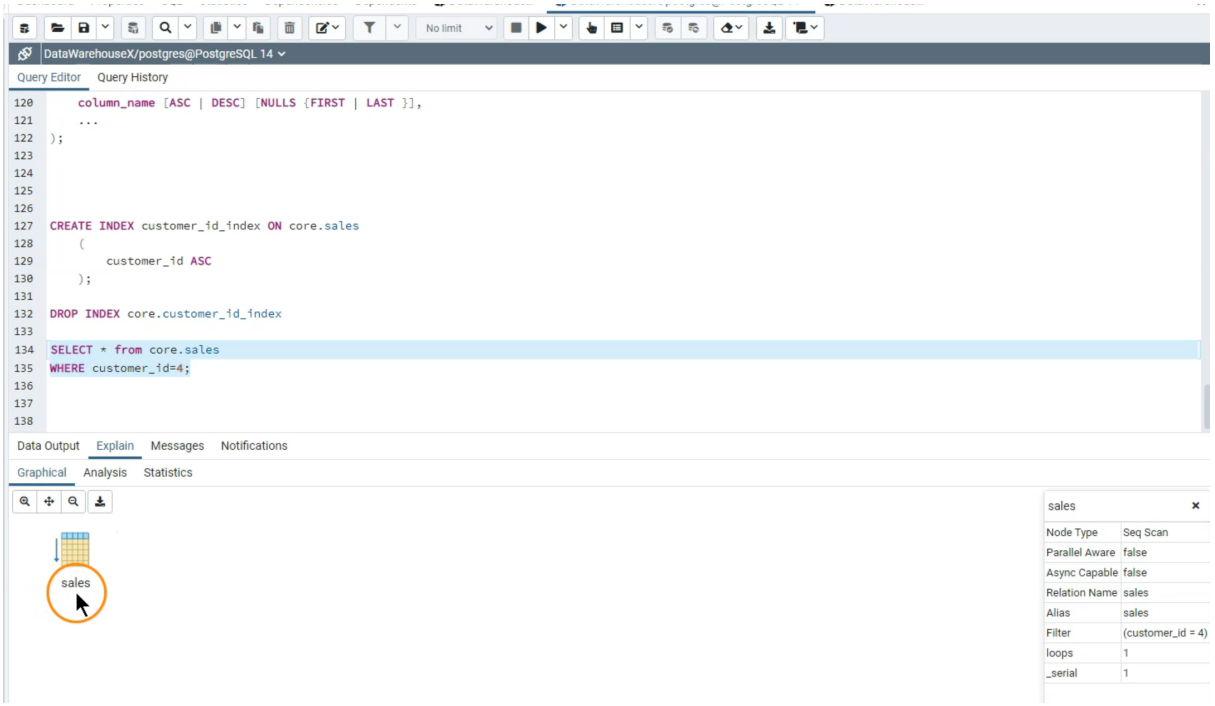
- So first of all, I want to use this query. I want to filter the data based on the customer ID.
- So we see now we get all of these values and we've seen this took about only 81 milliseconds and we can now also go and see how the query is retrieved.



- So we've talked about that there's a query optimizer and we can see how exactly the data is retrieved.

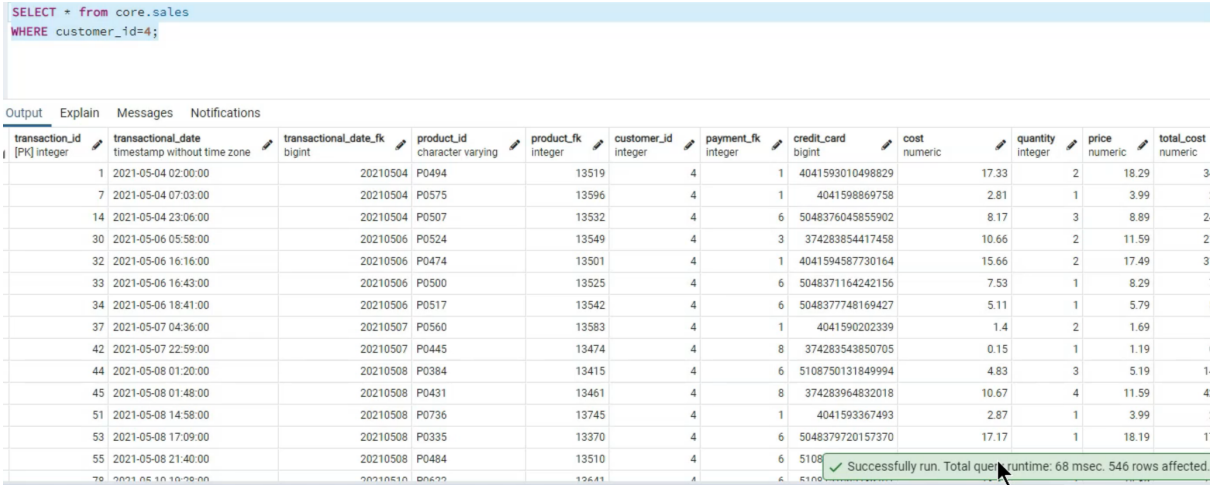


- So if we click here on explain we see that there is just a full table scan basically and we can see that in here that we have a full table scan and we can now set up an index on this column.

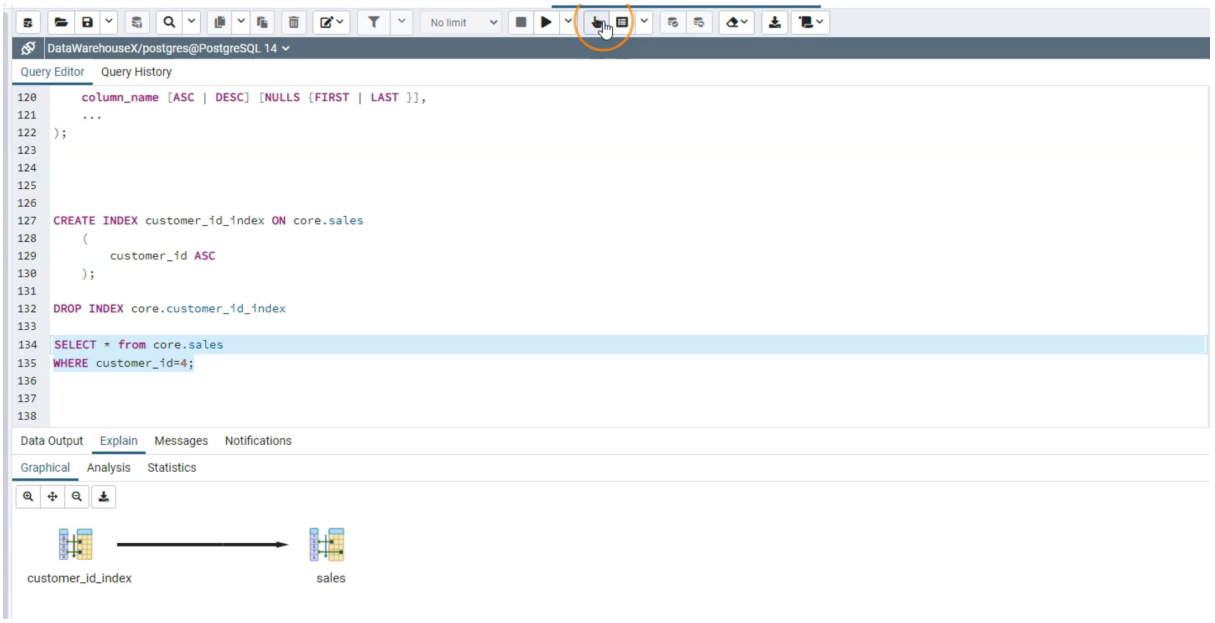


- So we say create index. This is the index name on that table, and this is on the column that should have this index.

- So now if we execute that, we can again get this same query and we see it is already at least a little bit better. It's not necessarily getting better as we've mentioned. So don't worry if in your case it's not even getting better. This is because our table is pretty small and that's why it's not benefiting so much from this index.



- So that's why don't worry if your time is here even a little bit higher. This is because our table is not so huge, but we've seen in our case, we have a very small increase in performance.



- And now we also want to see how the data is now retrieved. And now we can see that in here, we have now used this index, and then afterwards we have retrieved the specific rows that were needed.

- So this is something that we can usually also see in the different database management systems how the query is retrieved.
- And oftentimes there's also an internal tool that can be used to help us to tell how we can increase the performance of a specific query, and they can then make suggestions.
- For example, they can say if you want to use this query very frequently you can add an index on that column, for example. So we can also get suggestions with some additional query optimization tools.
- But this was just supposed to be a very quick demonstration on how an index can be set on a database.