Prepared by Team2:

* + Sara Ashraf
  + Mai Atef
  + Mahmoud Mohamed Sadek
  + Nourhan Rezk
  + Ahmed Shaaban

Types of indexes used in Data warehousing

TABLE OF CONTENTS

1. **Introduction …………………………………………………………………………………………………….. 2**
2. **Types of indexes used in Data warehousing ……………………………………………………… 2**
   1. **Bitmap Indexes ………………………………………………………………………………….2**
   2. **B-Tree Indexes …………………………………………………………………………………..3**
   3. **Bit map join Indexed ………………………………………………………………………….3**
3. **On which tables/fields shall we use which indexes? ……………………………….………….4**

**Introduction**

In a query-centric system like the data warehouse environment, the need to process queries faster dominates. There is no surer way of turning your users away from the data warehouse than by unreasonably slow queries. For the user in an analysis session going through a rapid succession of complex queries, you have to match the pace of the query results with the speed of thought. Among the various methods to improve performance, indexing ranks very high.

Let us consider the technique of indexing from the perspective of the data warehouse. The data tables are "read-only." This feature implies that you almost never update the records or delete records. And records are not inserted into the tables after the loads. When you do add, updates, or deletes, you incur additional overhead for manipulating the index files. But in a data warehouse this is not the case.

**Types of indexes used in Data warehousing**

1. **Bitmap Indexes in Data Warehouses**

A bitmap index is a special kind of [database index](https://en.wikipedia.org/wiki/Index_(database)) that uses [bitmaps](https://en.wikipedia.org/wiki/Bit_array). Bitmap indexes have traditionally been considered to work well for low-[cardinality](https://en.wikipedia.org/wiki/Cardinality) [columns](https://en.wikipedia.org/wiki/Column_(database)), which have a modest number of distinct values, either absolutely, or relative to the number of records that contain the data. The extreme case of low cardinality is [Boolean data](https://en.wikipedia.org/wiki/Boolean_data) which has two values, True and False. Bitmap indexes use [bit arrays](https://en.wikipedia.org/wiki/Bit_array) called bitmaps and answer queries by performing [bitwise logical operations](https://en.wikipedia.org/wiki/Bitwise_operation) on these bitmaps. Bitmap indexes have a significant space and performance advantage over other structures for query of such data.

The environment of data warehousing has large amounts of data and ad hoc queries, but a low level of concurrent DML transactions. For such applications, bitmap indexing provides:

* Reduced response time for large classes of ad hoc queries.
* Reduced storage requirements compared to other indexing techniques.
* Dramatic performance gains even on hardware with a relatively small number of CPUs or a small amount of memory.
* Efficient maintenance during parallel DML and loads.

**It’s usage:**

Parallel query and parallel DML work with bitmap indexes. Bitmap indexing also supports parallel create indexes and concatenated indexes.

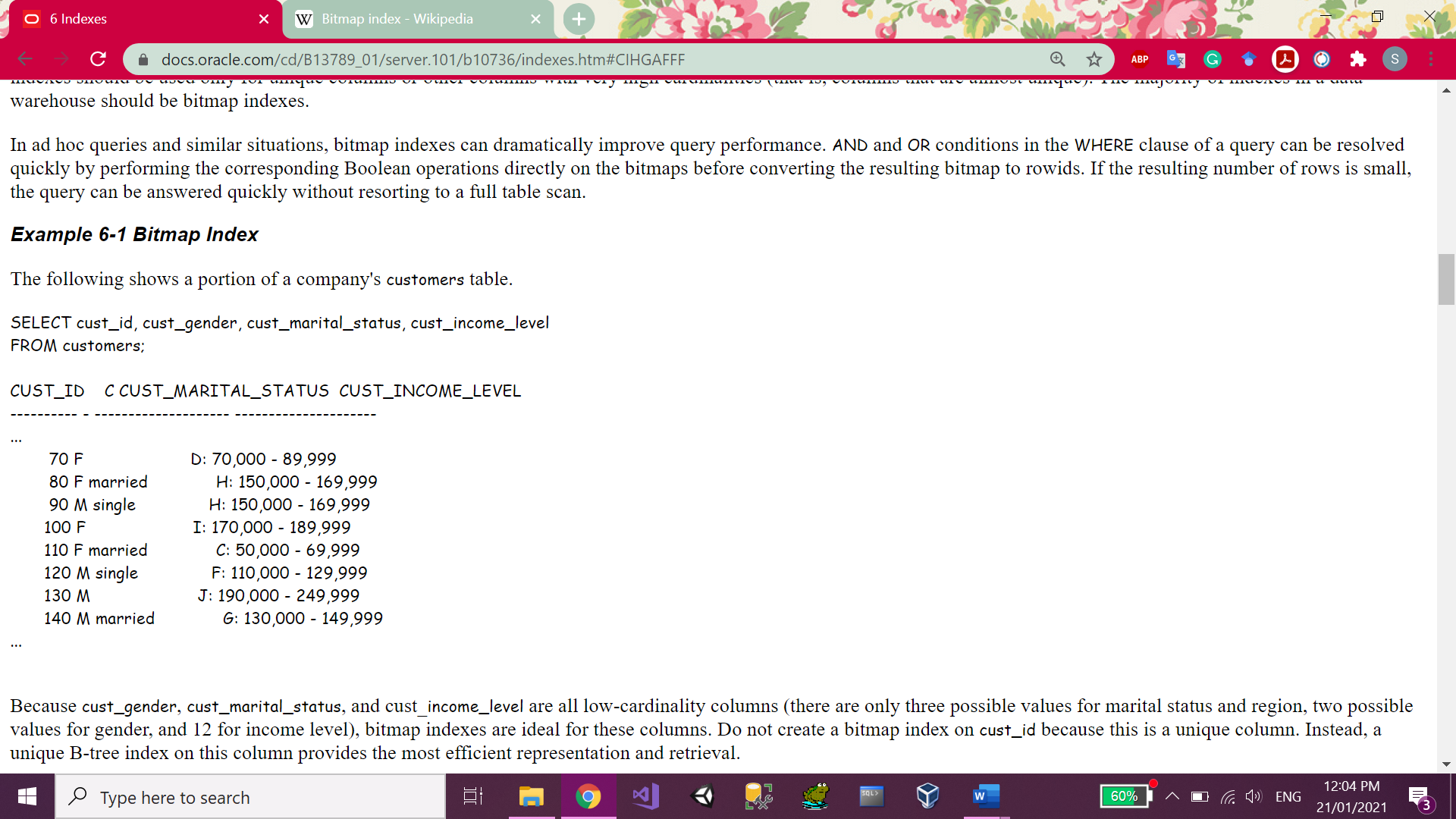
The advantages of using bitmap indexes are greatest for columns in which the ratio of the number of distinct values to the number of rows in the table is small ([**degree of cardinality**](https://docs.oracle.com/cd/B13789_01/server.101/b10736/glossary.htm#i996860))

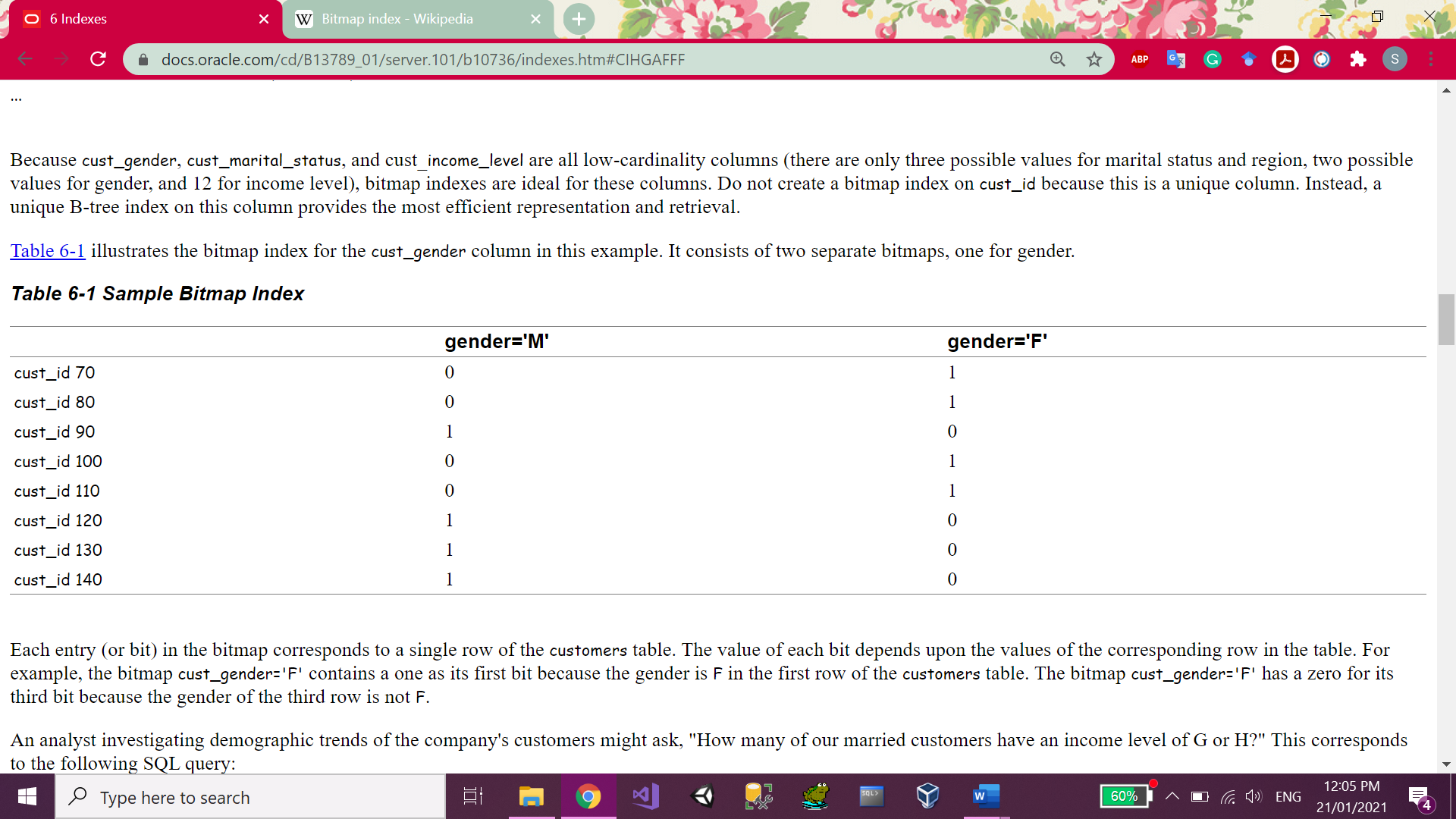
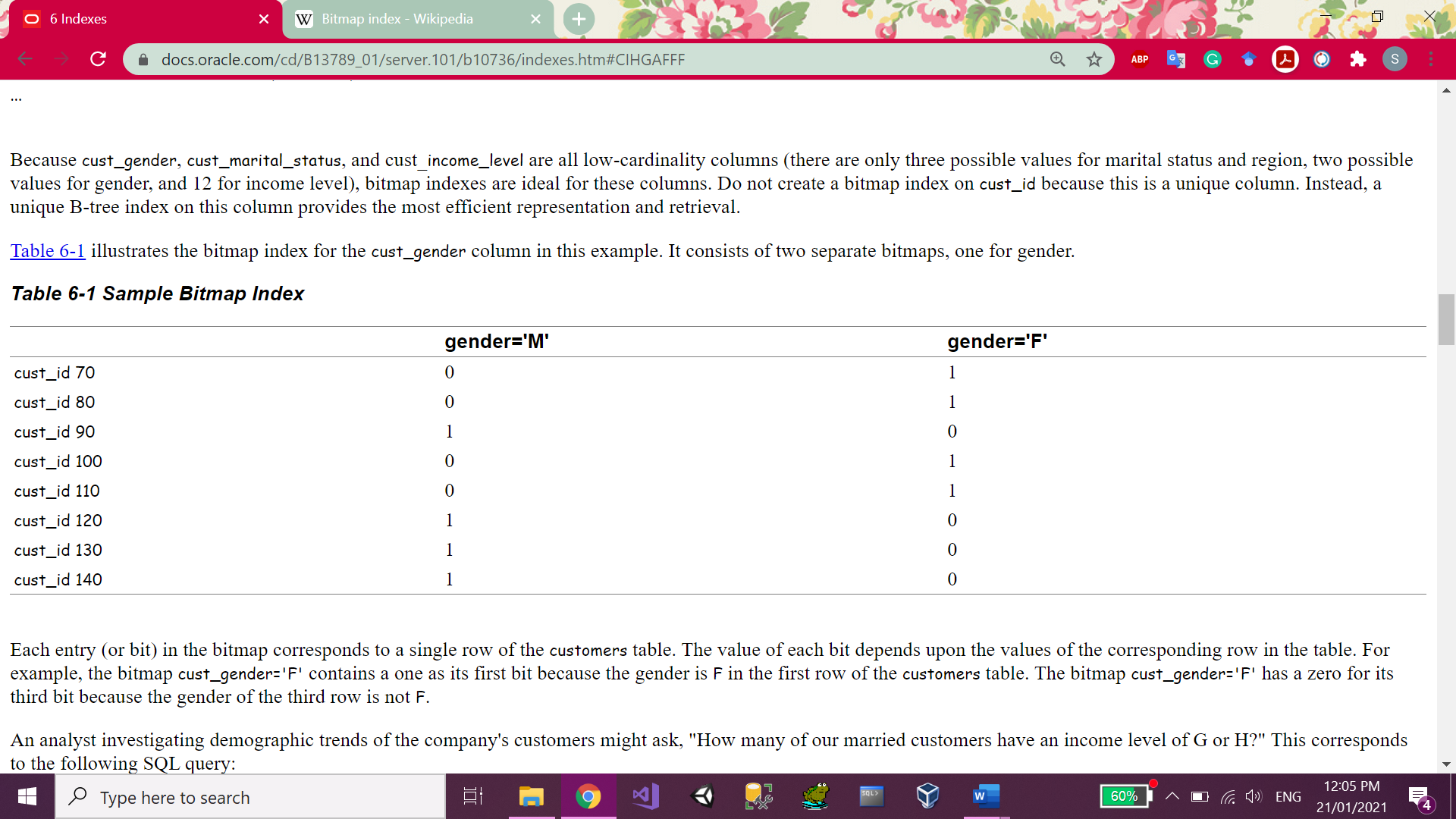
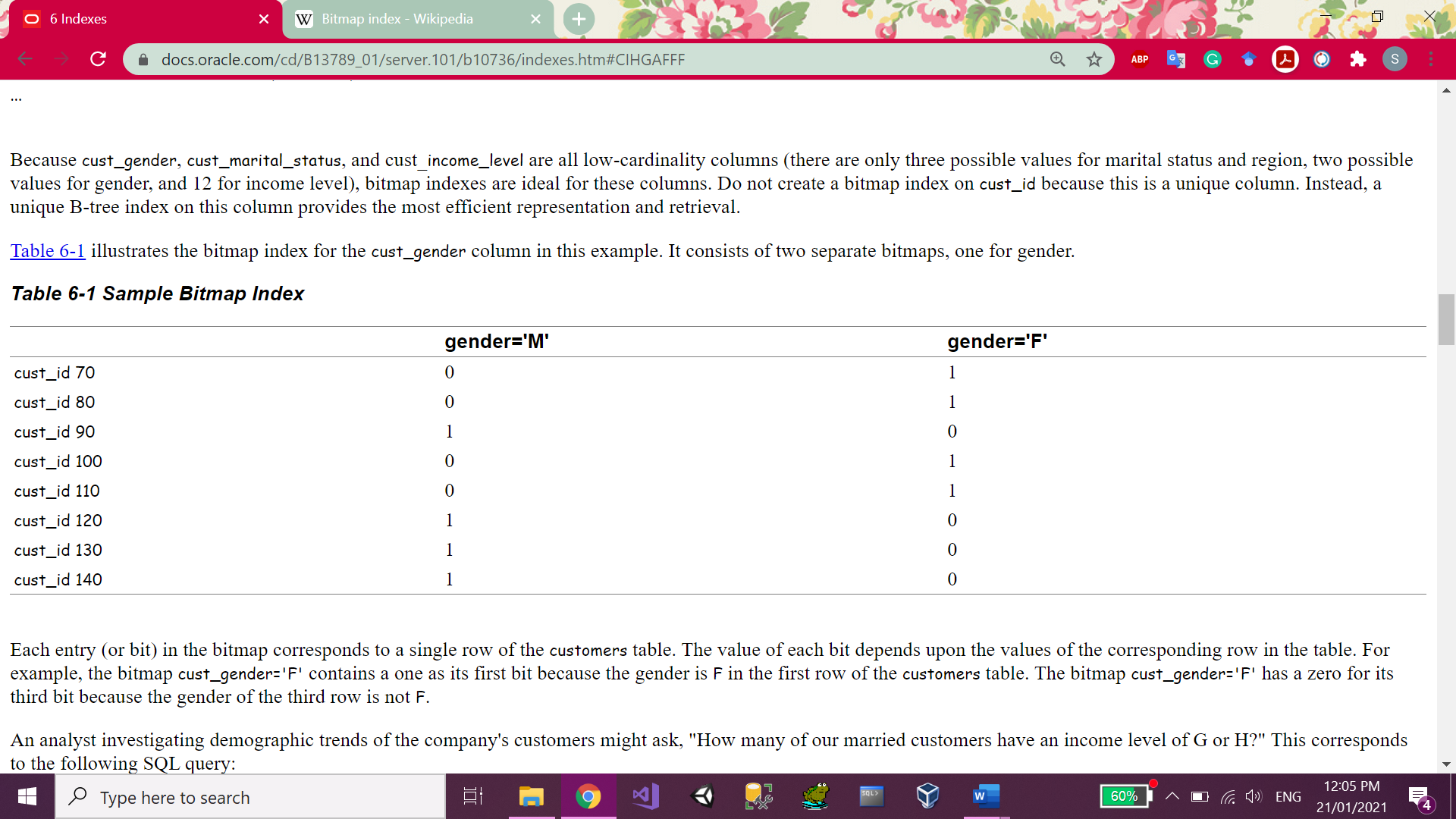
In ad hoc queries, bitmap indexes can dramatically improve query performance. AND and OR conditions in the WHERE clause of a query can be resolved quickly by performing the corresponding Boolean operations directly on the bitmaps before converting the resulting bitmap to row ids. If the resulting number of rows is small, the query can be answered quickly without resorting to a full table scan.

**Example:**

The following shows a portion of a company's customers table.

bitmap index for the cust\_gender column in this example. It consists of two separate bitmaps, one for gender.





**Mapping it to our case study we can add this index on the Reservation Channel FK in the fact tables as this column has only 3 categories whether Online, Flight reservations offices or Airlines companies.**

## B-Tree Indexes in Data Warehouses

## A B-tree index is organized like an upside-down tree. The bottom level of the index holds the actual data values and pointers to the corresponding rows, much as the index in a book has a page number associated with each index entry. Use B-tree indexes when you know that your typical query refers to the indexed column and retrieves a few rows. In these queries, it is faster to find the rows by looking at the index. So, if you are retrieving most of the rows in a table, it might not make sense to look up the index to find the table rows. Instead, you might want to read or scan the table.

## It’s usage:

## B-tree indexes are most commonly used in a data warehouse to index unique or near-unique keys. In many cases, it may not be necessary to index these columns in a data warehouse, because unique constraints can be maintained without an index, and because typical data warehouse queries may not work better with such indexes. B-tree indexes are more common in environments using third normal form schemas. In general, bitmap indexes should be more common than B-tree indexes in most data warehouse environments.

## Bitmap Join Index

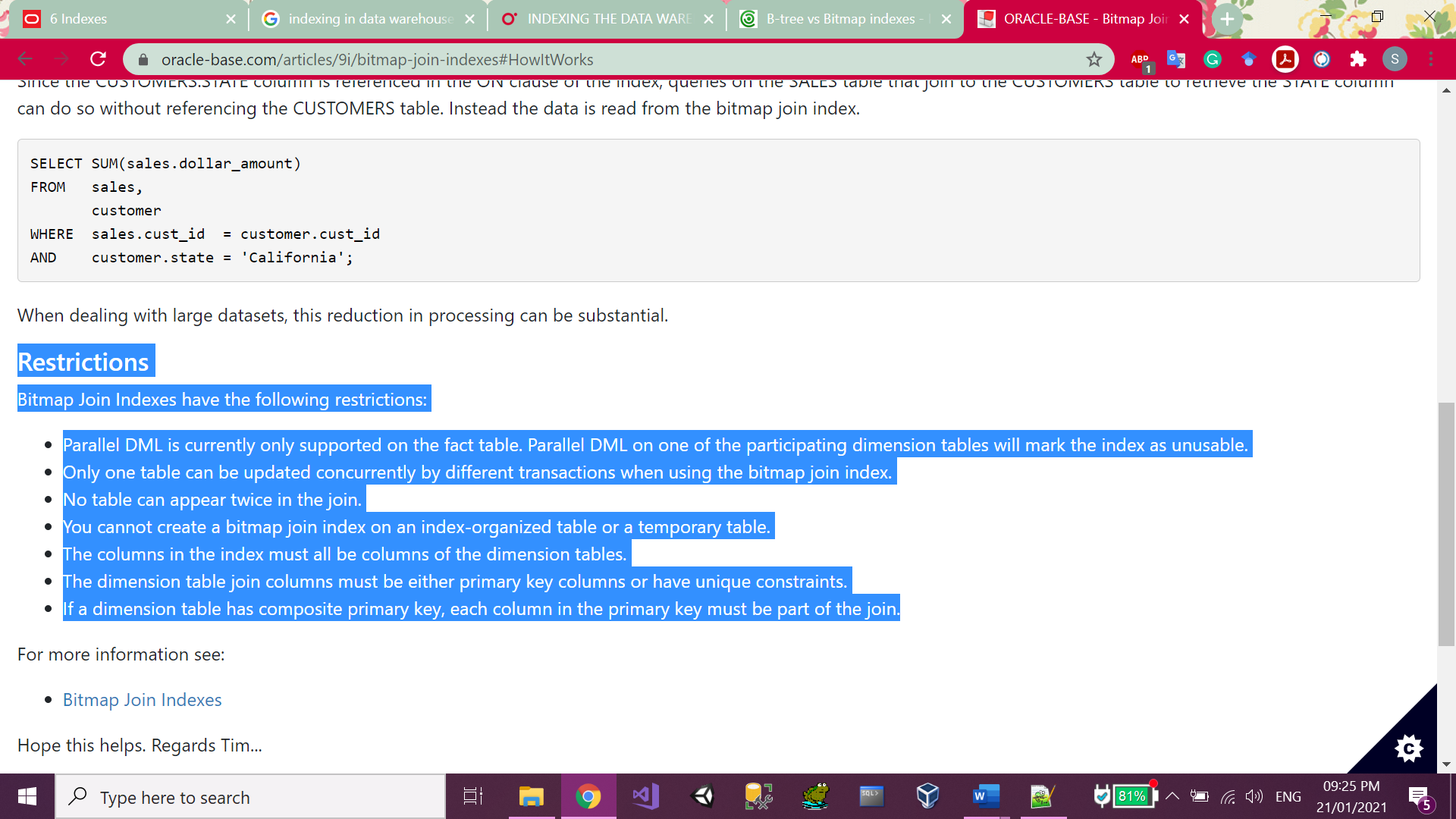
## Bitmap Join Indexes extend the concept of bitmap index such that the index contains the data to support the join query, allowing the query to retrieve the data from the index rather than referencing the join tables. Since the information is compressed into a bitmap, the size of the resulting structure is significantly smaller than the corresponding materialized view.

## Example:

## The index is created with reference to the columns in the joined tables that will be used to support the query. In the following example an index is created where the SALES table is joined to the CUSTOMERS table.

## 

Since the CUSTOMERS.STATE column is referenced in the ON clause of the index, queries on the SALES table that join to the CUSTOMERS table to retrieve the STATE column can do so without referencing the CUSTOMERS table. Instead the data is read from the bitmap join index.



## Restrictions:

Bitmap Join Indexes have the following restrictions:

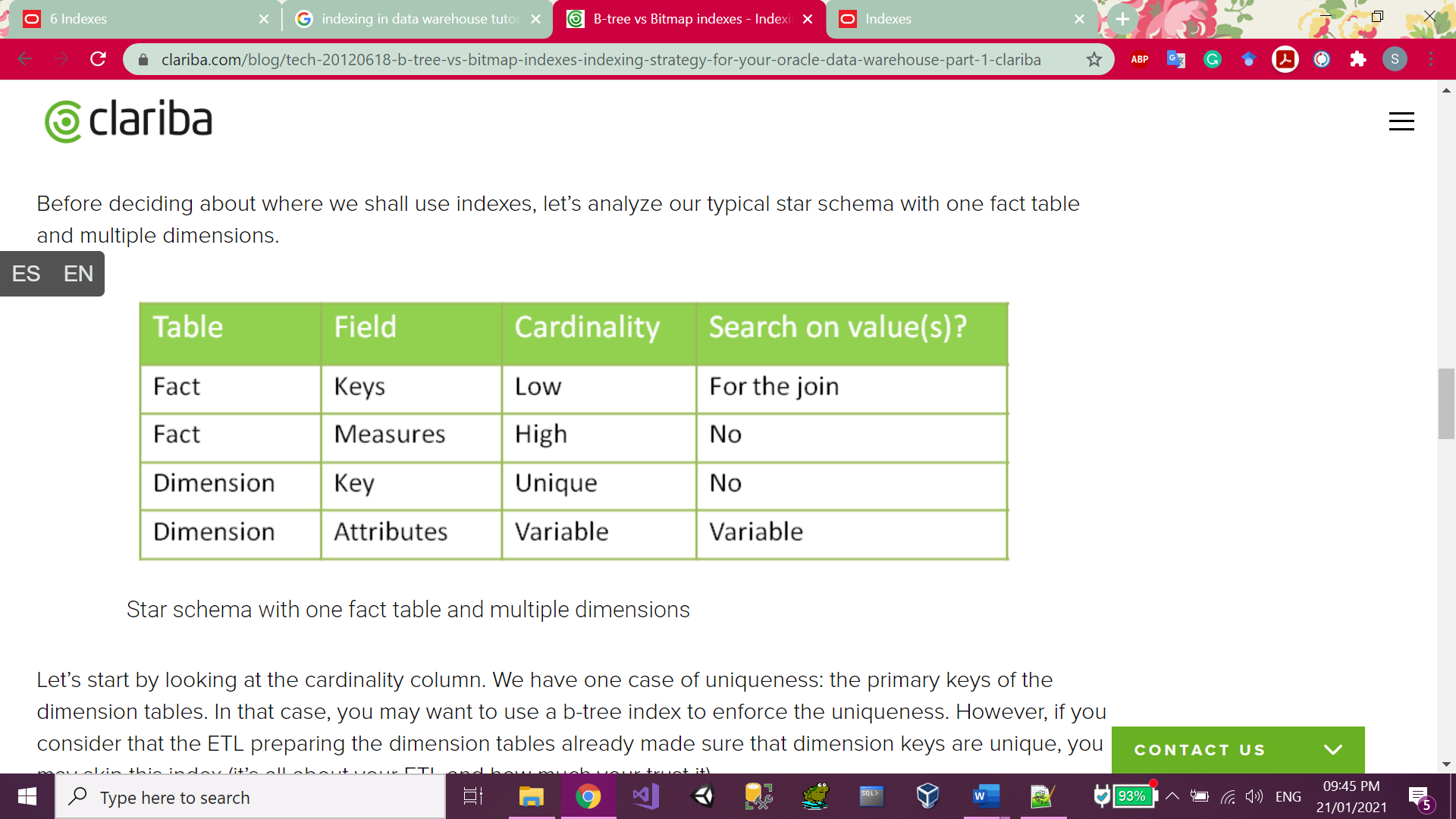
* Parallel DML is currently only supported on the fact table. Parallel DML on one of the participating dimension tables will mark the index as unusable.
* Only one table can be updated concurrently by different transactions when using the bitmap join index.
* No table can appear twice in the join.
* You cannot create a bitmap join index on an index-organized table or a temporary table.
* The columns in the index must all be columns of the dimension tables.
* The dimension table join columns must be either primary key columns or have unique constraints.
* If a dimension table has composite primary key, each column in the primary key must be part of the join.

**On which tables/fields shall we use which indexes?**

Indexes come with costs (creation time, update time, storage) and should be created only when necessary.

Remember also that the goal is to avoid full table reads – if the table is small, then the Oracle optimizer will decide to read the whole table anyway. so, we don’t need to create indexes on small tables

Before deciding about where we shall use indexes, let’s analyse our typical star schema with one fact table and multiple dimensions.



Let’s start by looking at the cardinality column. We have one case of uniqueness: the primary keys of the dimension tables. In that case, you may want to use a b-tree index to enforce the uniqueness. However, if you consider that the ETL preparing the dimension tables already made sure that dimension keys are unique, you may skip this index (it’s all about your ETL and how much your trust it).

We then have a case of high cardinality: the measures in the fact table. One of the main questions to ask when deciding whether to apply an index is: “Is anyone going to search a specific value in this column?” In this example I´ve developed I assume that no one is interested in knowing which account has a value of  43453.12. so, no need for an index here.

What about the attributes in the dimension? The answer is “it depends”. Are the users going to do searches on column X? Then you want an index. You’ll choose the type based on the cardinality: bitmap index for low cardinality, b-tree for high cardinality.

Concerning the dimension keys in the fact table, is anyone going to perform a search on them? Not directly (no filters by dimension keys!) but indirectly, yes. Every query which joins a fact table with one or more-dimension tables looks for specific dimension keys in the fact table. We have got two options to handle that: putting a bitmap key on every column, or using bitmap join keys.