**CELEBAL TECH**

**TRAINING**

**SQL DIM DATE**

**PROJECT REPORT**

**SUBMITTED BY -**

SAKSHAM AGARWAL(UE183090)

VIRAT AERON(UE188114)

**Task 1**

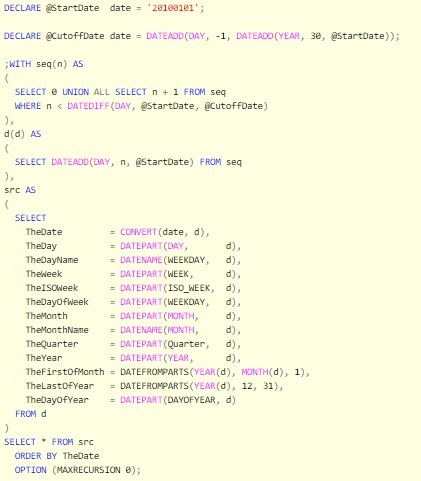
Understand the concept of dimension tables in data modelling. Learn the importance and schema structure of DimDate table (date dimension table) in modelling and implement a stored procedure code to load 25 years(from today’s date) date data and its computed date fields in date dimension table.

A calendar table can be immensely useful, particularly for reporting purposes, and for determining things like business days between two dates. I often see people struggling with manually populating a calendar or date dimension table; usually there are lots of loops and iterative code constructs being used. In this tip I will show you how to build and use a calendar table using a set-based solution that is powerful and easily customizable.

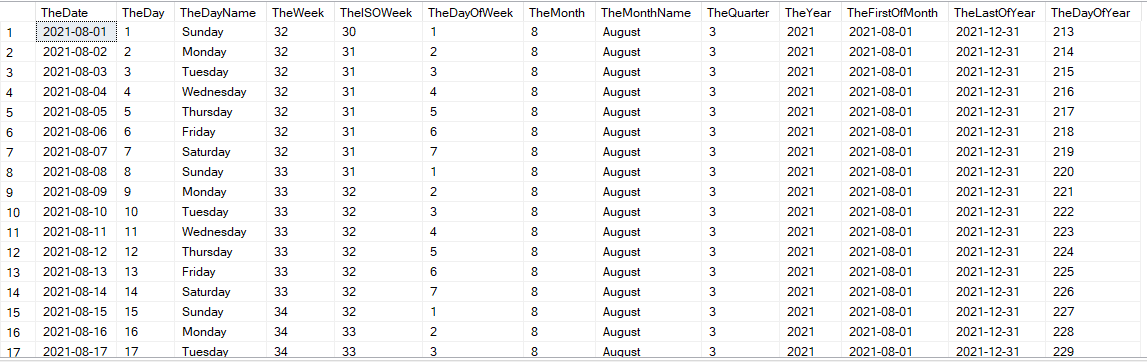
The given time dimension script will create table of time dimension and populate it with appropriate values. It also creates time buckets in table and fills it with group values, so that the user can perform aggregation of data using various combinations of hourly time buckets or day time buckets and they can do analysis of data using these time buckets and can do study of trend over the entire day.

First, we have a recursive CTE that returns a sequence representing the number of days between our start date (2010-01-01) and 30 years later less a day (2039-12-31):

we can start extending those dates with information commonly vital to calendar tables / date dimensions. Many are bits of information you can extract from the date, but it's more convenient to have them readily available in a view or table than it is to have every query calculate them inline. I'm working a little backward here, but I'm going to create an intermediate CTE to extract exactly once some computations I'll later have to make multiple times. This query:



**OUTPUT-**

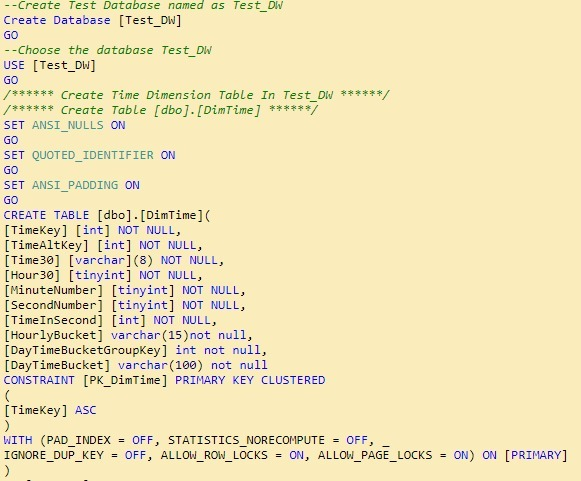


**Task 2**

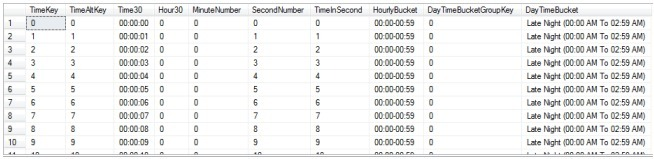
Learn the importance and schema structure of DimTime table (time dimension table) in modelling and implement a stored procedure code to load data and its computed time fields in time dimension table.

Follow the given steps to create time dimension:

* Open SQL Server Management Studio.
* Connect database engine.
* Open new query editor.
* Copy paste the script given below in new query editor window.
* Press F5 to run the given SQL script.



**OUTPUT -**



**Task 3**

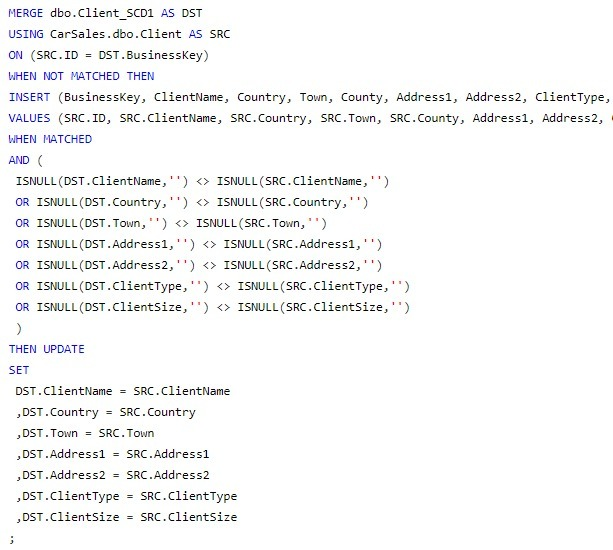
Understand the concept of Slowing Changing Dimension and its types in Data Modelling. Create a stored procedure to implement SCD-type1 logic for a sample dimension table(take it any table).

Type 1 SCD - Update changed attributes and disregard older attributes

A type 1 slowly changing dimension is by far the easiest to handle, as it consists of a simple in-place update of existing data, with no attempt to track the evolution of the changes. There are many voices which affirm that this is not really a "changing dimension" at all, but whatever the precise definitions, I will go with the flow and use it as a starting point for basic inserts and updates (or "Upserts" to use the conventional term).First, we need a "destination" table, and here it is:



Inserting and updating data is as simple as the following piece of T-SQL:



First, you have a source table (SRC), and a destination table (DST). The MERGE statement MERGEs into the destination, USING the source to get its data. As the two tables need a mapping join, there is a field - or fields - using which they can link. This is provided by the ON clause. All the other columns are the attributes of the dimension which we are monitoring for eventual changes. The MERGE statement than carries out a series of operations. Here only two are used:

WHEN NOT MATCHED

WHEN MATCHED

The former simply says "if there is no corresponding record in the destination table, then" - and it INSERTs the new data.

**Task 4**

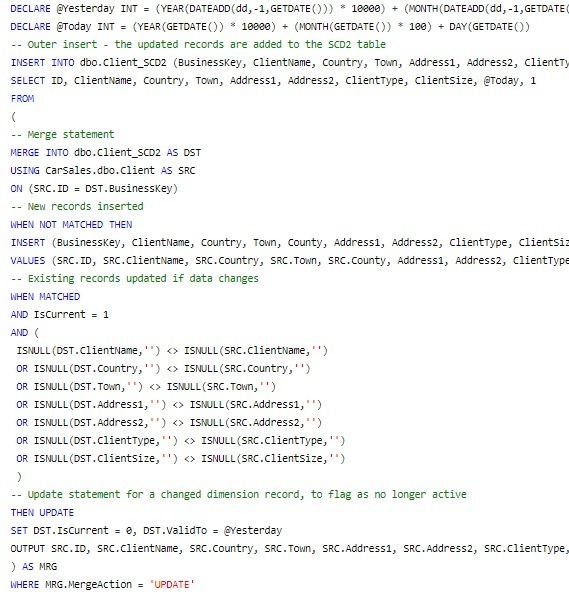
Create a stored procedure to implement SCD-type2 logic for a sample dimension table(take it any table).

Type 2 SCD - Keep a history of previous attributes as separate records, indicate the dates they were valid, and flag the currently valid record

The next - and to many, the "classic" - slowly changing dimension is the type 2. This approach will add a new record to the dimension table every time that the source data changes. To ensure that the multiple records can be used appropriately, the surrogate key will change every time new data is added, while the business key will remain the same. As is usual for SCD Type 2, three metadata (or "tracking") fields will be added:ValidFrom - to indicate the date that the record can be used,ValidTo - the date that it was ceased being valid.IsCurrent - to flag which of the dimension records is the current one.An SCD type 2 table, based on the same source data as that used in the previous example, is:



With the type 2 dimension table created, the SQL snippet given below will map the two tables on the business key and carry out the three following operations:Insert a new record into the destination table if the record referenced by the business key if the key is not already present (WHEN NOT MATCHED), as well as adding an auto-incremented surrogate key and setting today's date as the ValidFrom date.Insert a new record into the destination table if any of the attribute fields are not identical between the source and destination tables (WHEN MATCHED AND ...), as well as adding an auto-incremented surrogate key and setting today's date as the ValidFrom date and flagging this record as the current record.Updating the previous valid record for this business key. This involves setting the current record flag to False and setting yesterday's date as the ValidTo date.



As we will need to add - or update - validity dates, these are set as variables at the start of the process.New records are INSERTed, just like before.Any changed records (providing that they are the current valid record for a client) are UPDATEd, so that their IsCurrent and ValidTo fields are set to indicate that the record is now no longer current, and is historical. Where data was updated, it is now INSERTed into the dimension table based on the data in the OUTPUT table. This is detected by using the MERGE statement as an inline query which returns the data defined as the OUTPUT - and filtered on the data which was UPDATEd - which is isolated using the $Action field which is part of the OUTPUT clause.