Slowly Changing Dimensions Type 2

What is a Slowly Changing Dimension?

A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. It is considered and implemented as one of the most critical ETL tasks in tracking the history of dimension records.

Type 2 SCDs - Creating another dimension record

A Type 2 SCD retains the full history of values. When the value of a chosen attribute changes, the current record is closed. A new record is created with the changed data values and this new record becomes the current record. Each record contains the effective time and expiration time to identify the time period between which the record was active.

Objective:

The Target of the project is to handle changes that happens in the Employee address.

The project lifecycle:

1- Create Source Table

2- Create Target Table

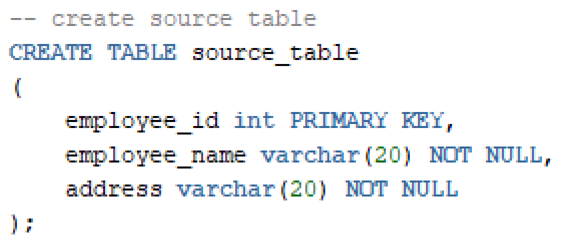
3- Create Sequence for the Surrogate Key

4- Create Stored Procedure

5- Testing the Results

The First Step:

The first step is to create the source table for the employees with columns employee id that is the primary key for this table that can't be redundant, it's a unique for the employee, employee name and address for that employee these columns can't have a null value at all.



```sql
-- create source table
CREATE TABLE source_table
(
    employee_id int PRIMARY KEY,
    employee_name varchar(20) NOT NULL,
    address varchar(20) NOT NULL
);
```

The Second Step:

The second step is to create the target table for the employees with columns employee id , employee name, address, last modified date that store the employee's address changes over time, status that is by default 1 as at first there is no change for that employee address, and surrogate key that is generated every time when the address change, these columns can't have a null value at all.

```
-- create table target table
CREATE TABLE target_table
(
    employee_id NUMBER NOT NULL,
    employee_name VARCHAR2(20) NOT NULL,
    address VARCHAR2(20) NOT NULL,
    last_modified_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    status NUMBER DEFAULT 1 NOT NULL,
    surrogate_key NUMBER NOT NULL
);
```

The Third Step:

The third step is to create a sequence for the Surrogate key that start with 1 and increment by 1 every time.

```
-- create sequence for the surrogate key to generate every time
CREATE SEQUENCE surrogate_key
  START WITH 1
  INCREMENT BY 1
  MINVALUE 1;
```

The Fourth Step:

The fourth step is to create the stored procedure It implements a Type 2 slowly changing dimension logic, a common technique used in data warehousing to handle historical changes to data over time.

Procedure Name: The procedure is named slowly_changing_dimension_type2.

MERGE Statements: The procedure contains two MERGE statements, which are used for performing INSERT, and UPDATE operations based on conditions.

```
-- this step is for stored procedures with slowly changing dimension type 2
CREATE OR REPLACE PROCEDURE slowly_changing_dimension_type2
```

First MERGE Statement:

```
MERGE INTO target_table t1
USING source_table s1
ON (t1.employee_id=s1.employee_id)
WHEN MATCHED THEN
UPDATE SET t1.status = 0 WHERE s1.address <> t1.address ;
```

It merges data from the source table (s1) into the target table (t1) and then It matches records in both tables based on the employee id. When a match is found, it updates the status column in t1 to 0 if the address in source table is different from the address in target table.

Second MERGE Statement:

```
MERGE INTO target_table t2
USING source_table s2
ON (t2.employee_id=s2.employee_id AND t2.address = s2.address )
WHEN MATCHED THEN
    UPDATE SET t2.status = CASE WHEN t2.address <> s2.address THEN 0 ELSE t2.status END
WHEN NOT MATCHED THEN
INSERT VALUES(s2.employee_id,s2.employee_name,s2.address,CURRENT_TIMESTAMP,1,surrogate_key.nextval);
```

merges data from source table (s2) into target table (t2) and then It matches records based on both employee id and address. When a match is found, It updates the status column in t2.

If the address in source table differs from the address in target table, it sets status to 0; otherwise, it keeps the existing status.

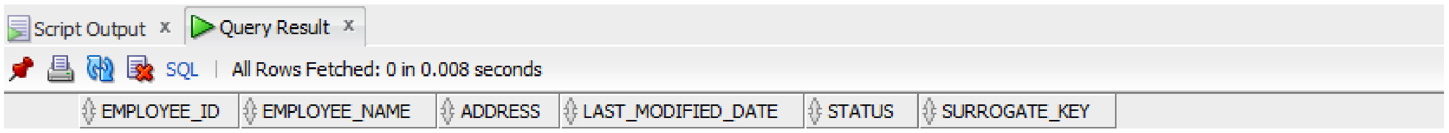
If no match is found based on employee id and address, it inserts a new record into target table with values from source table, along with a timestamp, a status of 1, and a surrogate key generated.

The Fifth Step:

The fifth step is to test the results of the stored procedure

First truncate table target table and select all columns in it to see the structure of the table

```
TRUNCATE TABLE target_table ;
SELECT * FROM target_table ;
```



All Rows Fetched: 0 in 0.008 seconds

EMPLOYEE_ID	EMPLOYEE_NAME	ADDRESS	LAST_MODIFIED_DATE	STATUS	SURROGATE_KEY

Then Runs the stored procedure and execute it

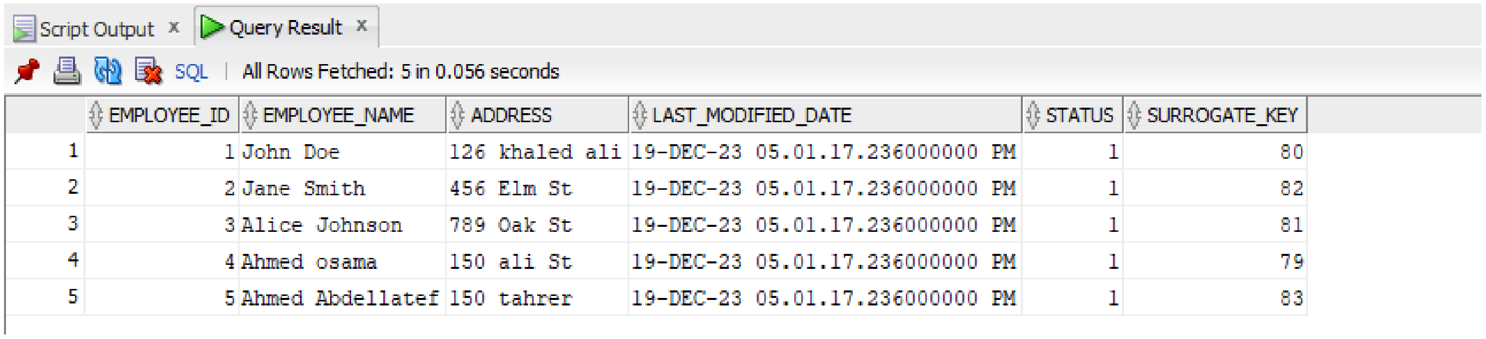
```
-- Execute the stored procedure
exec slowly_changing_dimension_type2 ;
```

```
Procedure SLOWLY_CHANGING_DIMENSION_TYPE2 compiled


PL/SQL procedure successfully completed.
```

And then run the code of the select statement in the target table to see the result

```sql
-- see the changes in the target table
SELECT * FROM target_table
ORDER BY employee_id,last_modified_date ;
```


Script Output × Query Result ×

SQL | All Rows Fetched: 5 in 0.056 seconds

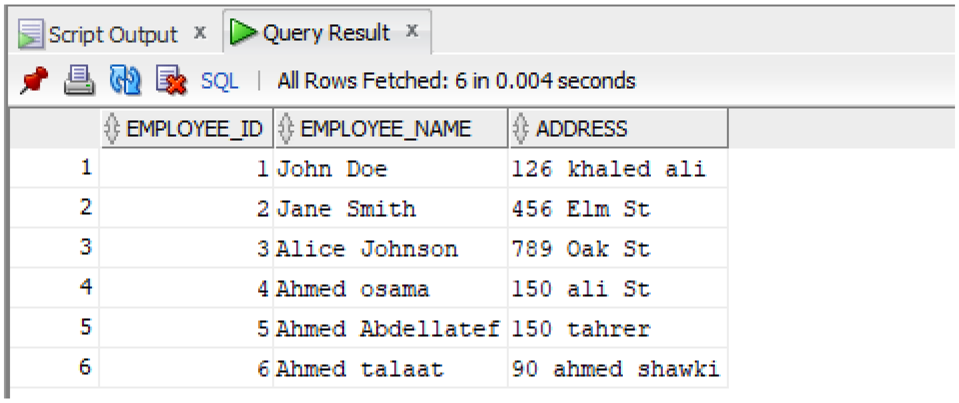
	EMPLOYEE_ID	EMPLOYEE_NAME	ADDRESS	LAST_MODIFIED_DATE	STATUS	SURROGATE_KEY
1	1	John Doe	126 khaled ali	19-DEC-23 05.01.17.236000000 PM	1	80
2	2	Jane Smith	456 Elm St	19-DEC-23 05.01.17.236000000 PM	1	82
3	3	Alice Johnson	789 Oak St	19-DEC-23 05.01.17.236000000 PM	1	81
4	4	Ahmed osama	150 ali St	19-DEC-23 05.01.17.236000000 PM	1	79
5	5	Ahmed Abdellatef	150 tahrer	19-DEC-23 05.01.17.236000000 PM	1	83

Then I will see the output of the target table if we insert a new row in the source table

```sql
-- insert in the source table
INSERT INTO source_table (employee_id, employee_name, address)
VALUES (6, 'Ahmed talaat', '90 ahmed shawki');
```

```
1 row inserted.
```

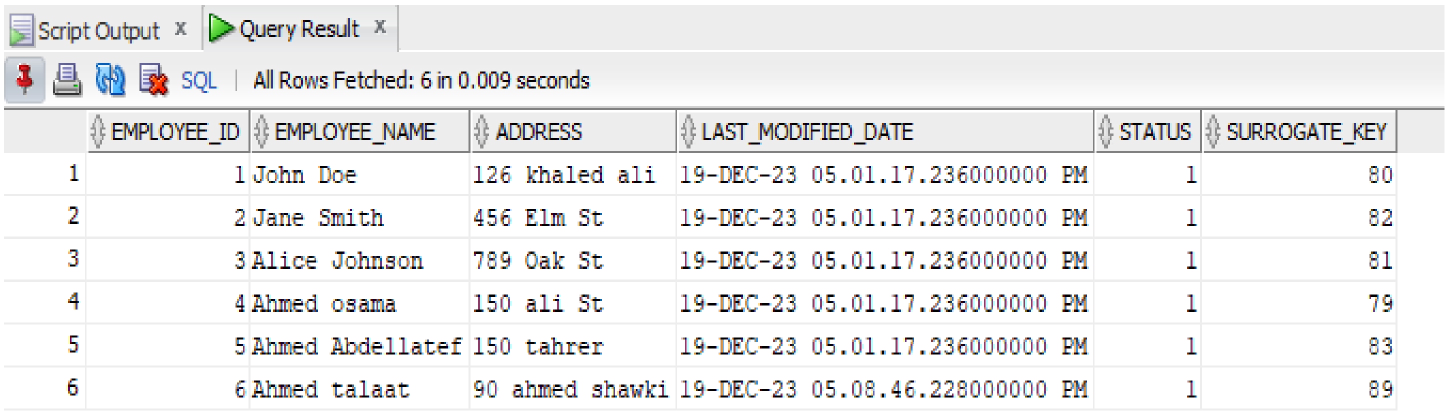
This the source table data after insert a new row in it


Script Output × Query Result ×

SQL | All Rows Fetched: 6 in 0.004 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	ADDRESS
1	1	John Doe	126 khaled ali
2	2	Jane Smith	456 Elm St
3	3	Alice Johnson	789 Oak St
4	4	Ahmed osama	150 ali St
5	5	Ahmed Abdellatef	150 tahrer
6	6	Ahmed talaat	90 ahmed shawki

And the Execute the procedure again and see the result of the target table again after insert in the source table


Script Output × Query Result ×

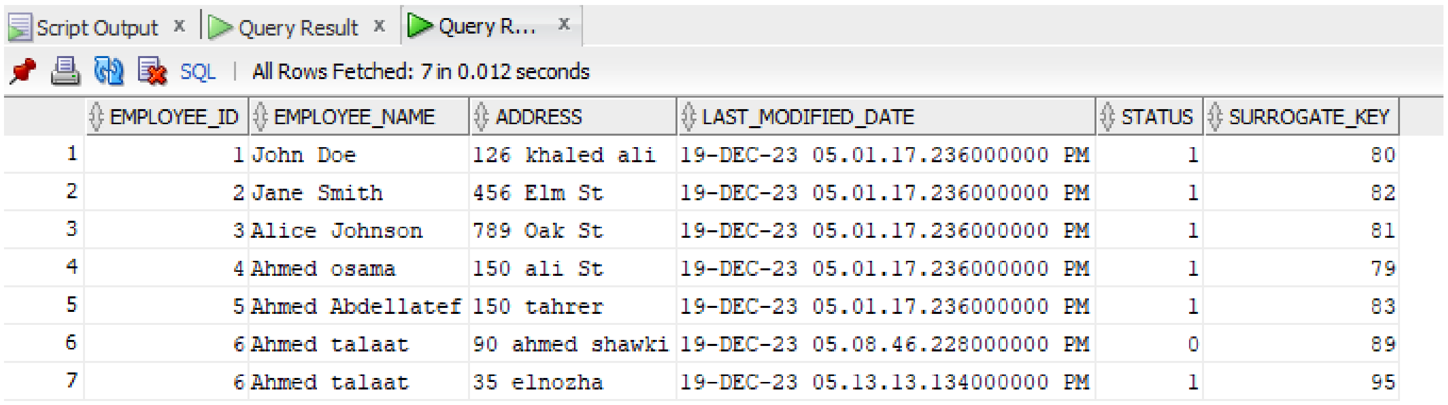
SQL | All Rows Fetched: 6 in 0.009 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	ADDRESS	LAST_MODIFIED_DATE	STATUS	SURROGATE_KEY
1	1	John Doe	126 khaled ali	19-DEC-23 05.01.17.236000000 PM	1	80
2	2	Jane Smith	456 Elm St	19-DEC-23 05.01.17.236000000 PM	1	82
3	3	Alice Johnson	789 Oak St	19-DEC-23 05.01.17.236000000 PM	1	81
4	4	Ahmed osama	150 ali St	19-DEC-23 05.01.17.236000000 PM	1	79
5	5	Ahmed Abdellatef	150 tahrer	19-DEC-23 05.01.17.236000000 PM	1	83
6	6	Ahmed talaat	90 ahmed shawki	19-DEC-23 05.08.46.228000000 PM	1	89

Then update the id = 6 for the employee Ahmed Talaat to ‘ 35 elnozha ‘ address and execute the procedure again so we can see the changes in the target table

```
-- Update in the source table
UPDATE source_table SET address = '35 elnozha ' WHERE employee_id = 6;
```

```
1 row updated.
```



Script Output × | Query Result × | Query R... ×

SQL | All Rows Fetched: 7 in 0.012 seconds

	EMPLOYEE_ID	EMPLOYEE_NAME	ADDRESS	LAST_MODIFIED_DATE	STATUS	SURROGATE_KEY
1	1	John Doe	126 khaled ali	19-DEC-23 05.01.17.236000000 PM	1	80
2	2	Jane Smith	456 Elm St	19-DEC-23 05.01.17.236000000 PM	1	82
3	3	Alice Johnson	789 Oak St	19-DEC-23 05.01.17.236000000 PM	1	81
4	4	Ahmed osama	150 ali St	19-DEC-23 05.01.17.236000000 PM	1	79
5	5	Ahmed Abdellatef	150 tahrer	19-DEC-23 05.01.17.236000000 PM	1	83
6	6	Ahmed talaat	90 ahmed shawki	19-DEC-23 05.08.46.228000000 PM	0	89
7	6	Ahmed talaat	35 elnozha	19-DEC-23 05.13.13.134000000 PM	1	95