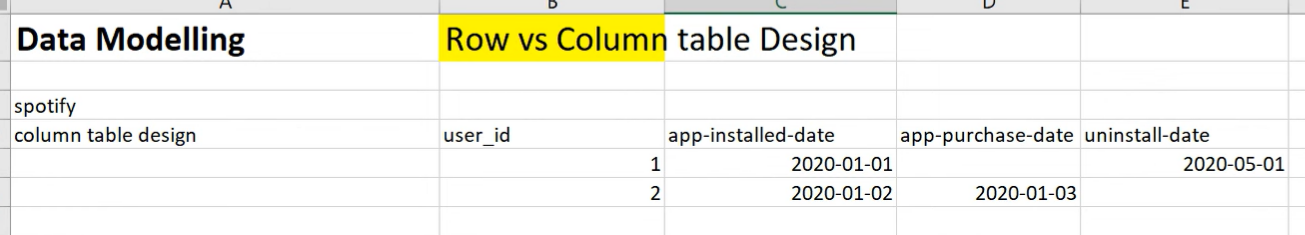
**Row Vs Column Table Design**

The biggest disadvantage of column-based design is that there are multiple rows in the database and if you want to update any row you need to scan the whole database and then make update.

Update is also a costly operation.

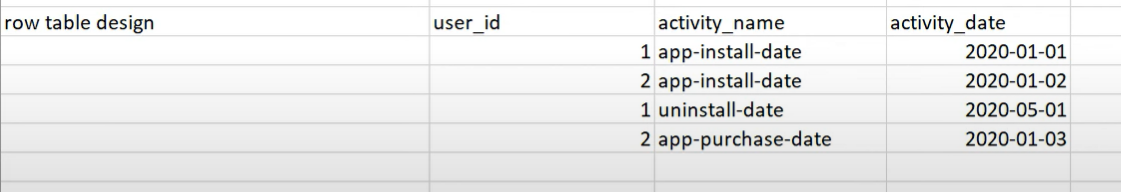
Null values would be present in the database where there in no update made, ex – app-purchase date would be null for those who haven’t made and purchase.

****

**## Row-table design**

When the data columns are dynamic, we generally prefer row table design. Adding a new column in production table is tedious task and adding a row in table is very easy.

Ex- if we need a new column as ‘premium-app-subscription-date’, adding the column in columnar table design would be very hard, but in row table design we can add that as new row.

****

**## SCD Type-1**

There are different ways to store data within a table. In SCD type 1 –

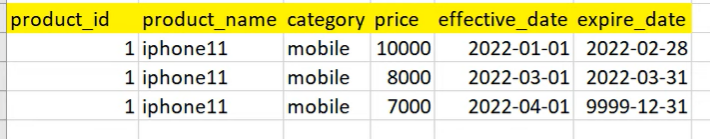
1. The table contain the current data. Old data will be replaced with new values.
2. There is no history maintained.
3. Not recommended for analytical use cases

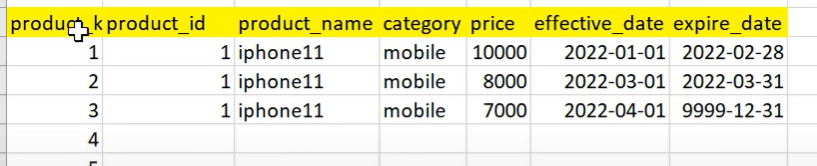
|  |  |
| --- | --- |
| Old Table | New Table |

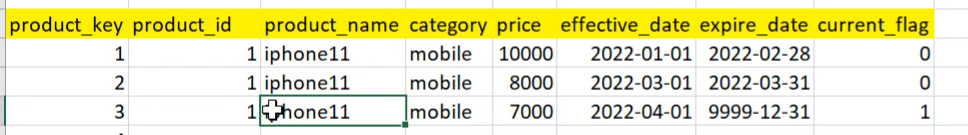
**## SCD Type-2**

In SCD Type 2 Table:

1. History would be maintained
2. Old records would be terminated with expiry date and new active record will be updated.
3. The table would contain multiple record with same key(Product\_id),there is no unique id for the records so, to overcome this and update each record with unique key a **surrogate key** (product\_key)is used. It is updated in incremental order.
4. Sometime current\_flag or active\_flag is also present as a column, to know which is the latest record.

****

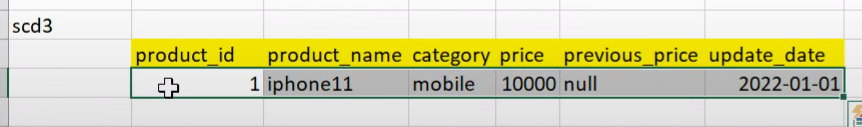
****

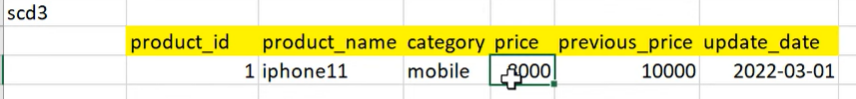
****

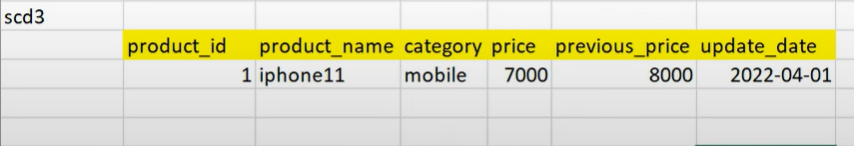
**## SCD Type-3**

In SCD type3 –

1. We will keep one previous information.(previous\_price)
2. Initially previous information would be null, with each and every update the information will be updated.



****

****

**## Difference between Where and Having Clause in SQL**

>> When we use **where clause** the pointer will go to each rows and check if the condition is qualified. It will go to each row and apply the filter.

Ex – when you want to know the emp details whose salary is > 10000. This would be a row wise operation and each row within the table would be matched with condition and required rows will be filtered out.

>> when we have to apply filter on aggregated value we need to use **having clause**

When some ask to get the data for department id whose avg salary is > 10000. In this case we cannot go row by row to check the individual salary.

|  |  |
| --- | --- |
|  |  |

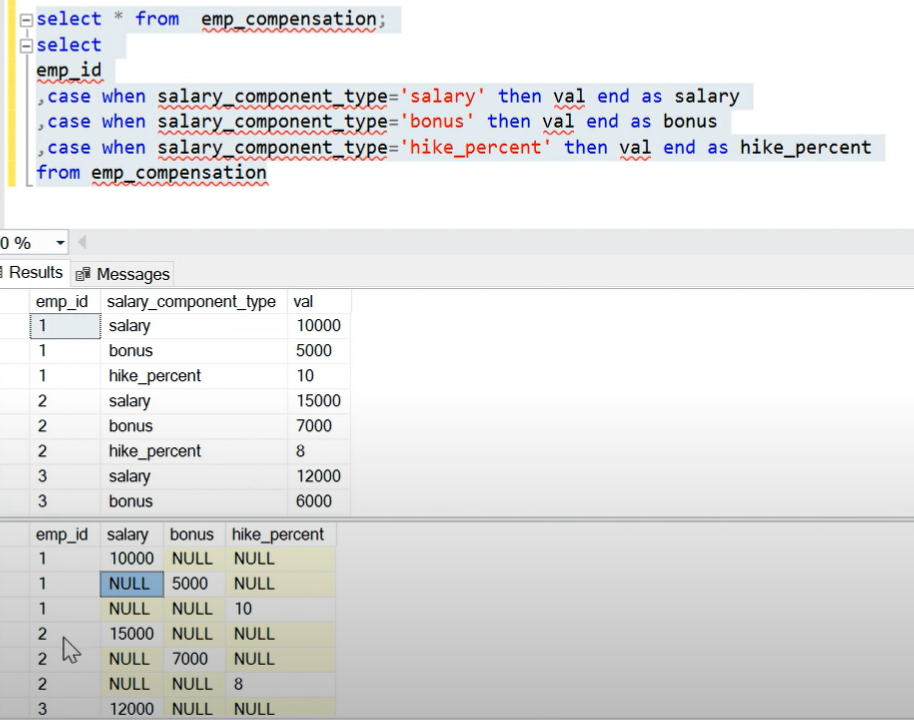
When we have to use both where and having together, then first use Where filter and then having filter.

Example – Find the department Id of employee having salary > 10000 and department avg is > 12000.

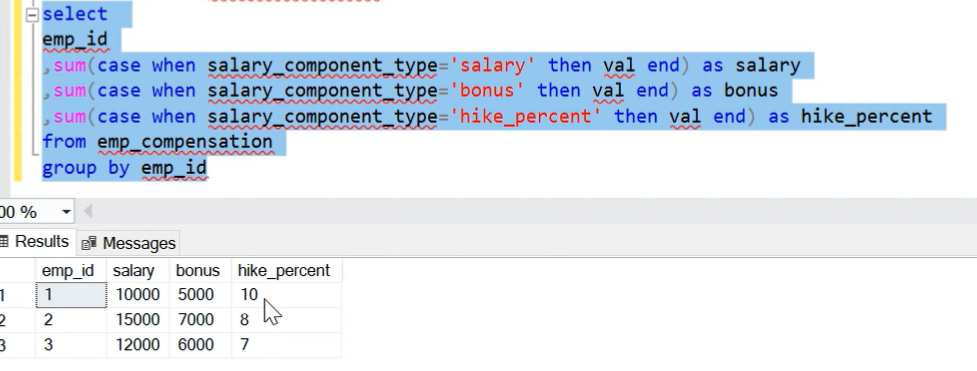
**## Pivoting and Unpivoting (Case when with Sum)**

|  |  |
| --- | --- |
| **Input:** | **Output:** |

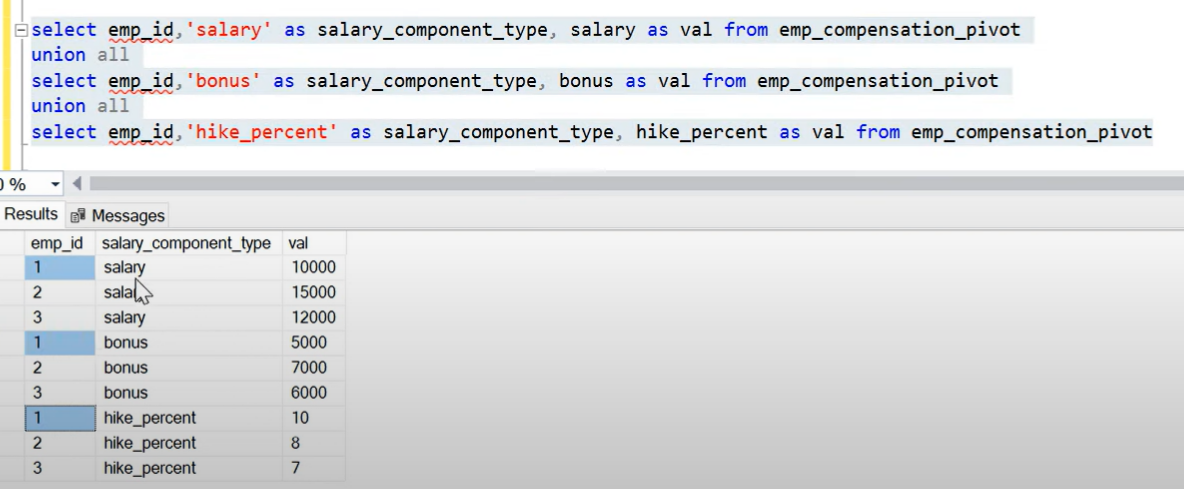
Solution:



When using “Case When” statement it will create separate column with values and NULL. Above we got Salary, Bonus and hike\_percent as separate column with NULL values. Also we have three rows for emp\_id 1 & 2, So if we need to combine it into 1 row we need the summation of each row value in three columns grouping by emp\_id.



**## Going back to pivoted data:**



**## SELF JOIN**

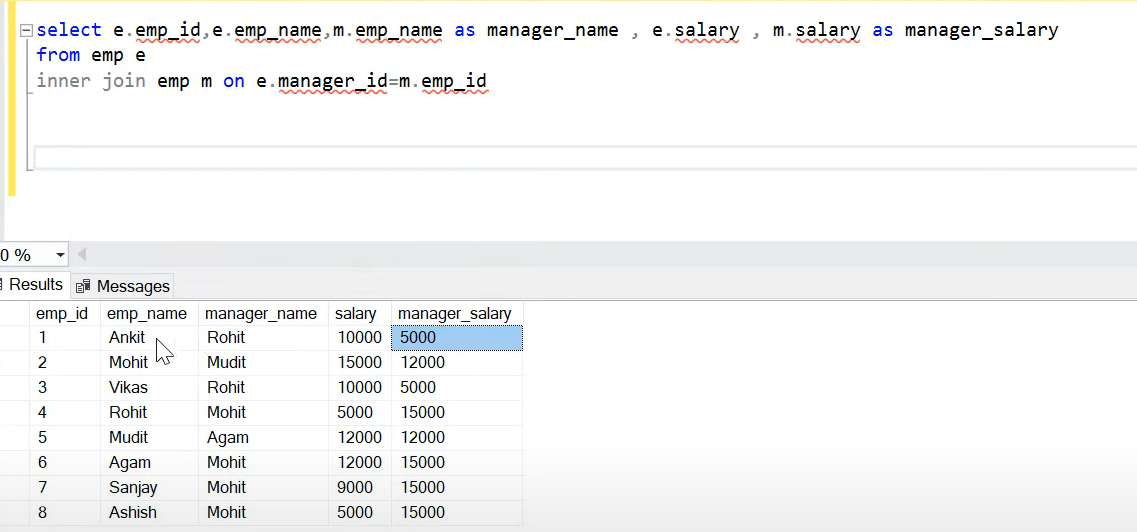
We use self-join where one column of the table is referring to other column of same table. In below table Manager\_id is referring to emp\_id of the same table.



When doing self join we create self instance of same table. Let suppose we make two instance of the table and one table is EMP\_tbl and 2nd table is MGR\_tbl.

To know the manager details of the employee we will join both table on

EMP\_tbl.manager\_id = MGR\_tbl.emp\_id



**## CROSS JOIN**

When joining two table we join on a particular key which is common, but in cross join we don’t give any condition. In join cross there will be Cartesian product of the records. 1st row joined to all rows of 2nd table, 2nd row to all the rows, 3rd rows to all and so on…

Use Case:-

1. Prepare Master Table

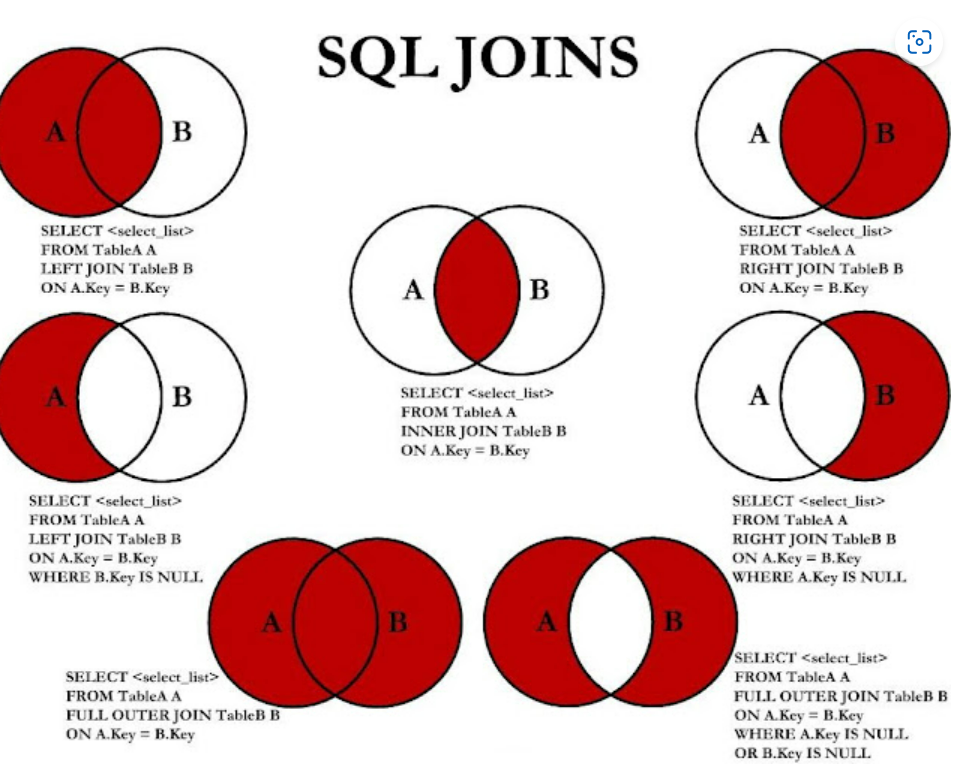
**## Most Asked SQL JOIN based Interview Question**

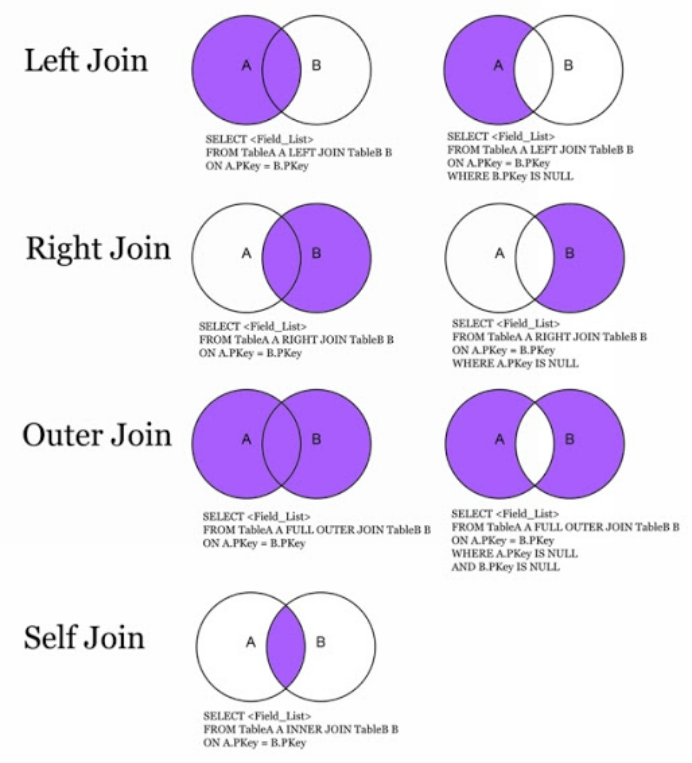
**Inner Join –** We get matching records from both the tables. If we have duplicate record in the tbl1 it will also get matched with table2. I table has NULL value it will not get joined since NULLS are not equal to any values.

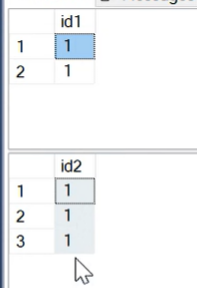
LEFT Join – All the records from left table will come irrespective of match found in right table. NULL will be returned for those records where there is no match found in right table

RIGHT Join – All the records from right table will come irrespective of match found in left table. NULL will be returned for those records where there is no match found in left table.

FULL JOIN – ALL the matching and non-matching records from both the tables







Rule – If there is matching records, all the joins will return the same number of records.

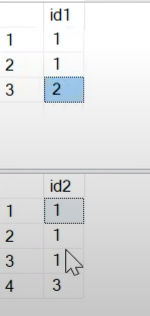
INNER JOIN – (1\*3) +(1\*3) = 6 records

LEFT JOIN – 6 records (1 joined with 3 records, 2nd 1 joined with 3 records)

RIGHT JOIN – 6records

Outer Join – 6 records

Outer join – 6 records



INNER JOIN – 6 Records (11,11,11,11,11,11)

LEFT JOIN – 7 records (11,11,11,11,11,11,2null)

RIGHT JOIN – 7 records (11,11,11,11,11,11,null3)

OUTER JOIN – 8 records (11,11,11,11,11,11,2NULL,NULL3)



INNER JOIN – 8 records (11,11,11,11,11,11,22,22)

LEFT JOIN – 8 records (11,11,11,11,11,11,22,22)

RIGHT JOIN – 9 records (11,11,11,11,11,11,3NULL,22,22)

OUTER JOIN – 9 records (11,11,11,11,11,11,3NULL,22,22)



NULL cannot be Joined with another NULL.

INNER JOIN – 10 records (11,11,11,11,11,11,22,22,22,22)

LEFT JOIN – 12 records (11,11,11,11,11,11,22,22,22,22,4NULL, NULL-NULL)

Here NULL is not matching , it is NULL from LEFT table

RIGHT JOIN – 12 (11,11,11,11,11,11,3NULL,22,22,22,22,NULL-NULL)

OUTER JOIN – 14(11,11,11,11,11,11,22,22,22,22, 4NULL, 3NULL,NULL-NULL,NULL-NULL)

1 NULL-NULL from RIGHT join and 1 NULL-NULL from LEFT join