BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DATA WAREHOUSING SS-G515
LAB-3 [Lookup Tables, Handling Multiple time zones & Currencies, Introduction to Slowly Changing Dimensions]

DATE: 13/04/2021 **TIME: 02 Hours**

Lookup Tables

A lookUp table is the one which is used when updating a warehouse. When the lookup is placed on the target table (fact table/warehouse) based upon the primary key of the target, it just updates the table by allowing only new records or updated records on the basis of lookup condition.

Look up tables hold data in the form of key-value(s). The keys are usually integers or short string codes. The keys are being used by other tables. There are multiple motivations to use lookup tables:

1)Storage saving: It would take less storage to save the information about the customers' countries as 1,2,3 or 'CN','IN','US' other than 'China','India' and 'United States'

2)Performance enhancement (for some types of queries): Since the data takes less storage, the volume of the data read and processed is smaller which leads to reduced IO, CPU and memory consumption.

3)Data centralization: All tables and all columns that need to refer to a specific value, e.g. a name of a country, will use the lookup key (e.g. 'US') therefore preventing a case where some tables/columns use the name 'United States', some use ' United States of America' and others use 'U.S.A'.

4)Data modification flexibility: If a value of a key-value pair is need to be modified, e.g. change 'United States' to 'United States of America', there is only a single row in the whole database, the row in the lookup table, that needs to be modified. Since all other tables are using the key ('US') and not the value there is no need to modify them.

5)Additional information: With lookup tables we can implement the concept of key-values, holding other essential information other than the the description of the key, e.g. - for the key 'US' we can hold not only the country name 'United State' but also additional information like 3 characters abbreviation ('USA') and international dialing code (1).

6)Fast answer to some questions: Our lookup tables can give us fast answers to questions like "which countries are potentially being used in our tables?" only by checking the lookup table and without the need to scan huge data tables.

Example 1: Create an order table as:

OrderID
OrderDate
OrderCountry
OrderAmount

For this OrderCountry column create a lookup table as CountryTable, which contains country ID and Country Name. In Order table store CountryID only and while retrieving the records retrieve CountryName from CountryTable.

Solution: For this OrderCountry column create a lookup table as CountryTable, which contains country ID and Country Name. In Order table store CountryID only and while retrieving the records retrieve CountryName from CountryTable.

```
CREATE TABLE dbo.Orders (
        OrderID int not null primary key,
        OrderDate datetime,
        OrderCountry varchar(3),
        OrderAmount float
);

CREATE TABLE dbo.CountryTable (
        CountryID varchar(3) not null primary key,
        CountryName varchar(255)
);


-----------------------------------------------------------------------------------------------
INSERT INTO Orders VALUES
(1, CAST('12/17/2015' as datetime), 'USA', 123),
(2, CAST('4/3/2010' as date), 'IND', 1224),
(3, CAST('7/4/2011' as date), 'ENG', 1747),
(4, CAST('3/27/2030' as date), 'NET', 2846),
(5, CAST('9/15/2024' as date), 'AUS', 6913),
(6, CAST('1/13/2014' as date), 'SRI', 6544),
(7, CAST('1/19/2021' as date), 'BAN', 2473),
(8, CAST('2/21/2022' as date), 'PAK', 4512),
(9, CAST('1/11/2022' as date), 'CHI', 7412),
(10, CAST('9/12/2041' as date), 'KEN', 4741)

-----------------------------------------------------------------------------------------------
INSERT INTO CountryTable VALUES
('USA', 'United States of America'),
('IND', 'India'),
('ENG', 'England'),
('NET', 'Netherlands'),
```

```
('AUS', 'Australia'),
('SRI', 'Sri Lanka'),
('BAN', 'Bangladesh'),
('PAK', 'Pakistan'),
('CHI', 'China'),
('KEN', 'Kenya')


-----------------------------------------------------------------------------------------------
select Orders.OrderId, Orders.OrderDate, CountryTable.CountryName, Orders.OrderAmount
from Orders
inner join CountryTable
on Orders.OrderCountry = CountryTable.CountryID
```

Example 2. Create two tables namely, TableColor and TableClothes.

TableColor:

- ColourID
- Colour

TableClothes:

- Day(Integer)
- PantsColourID
- ShirtColourID

Write a query to return Day,PantColourID, Colour(Pant), ShirtColourID, Colour(Shirt).

```
create table TableColour (
        ColourID varchar(7),
        Colour varchar(255)
);

create table TableClothes (
        WeekDay int,
        PantsColourID varchar(7),
        ShirtColourID varchar(7),
);

-----------------------------------------------------------------------------------------------
insert into TableColour values
('#FFA07A', 'light salmon'),
('#B0E0E6', 'powder blue'),
('#4169E1', 'royal blue'),
('#0000FF', 'blue'),
('#0000CD', 'mediumblue'),
```

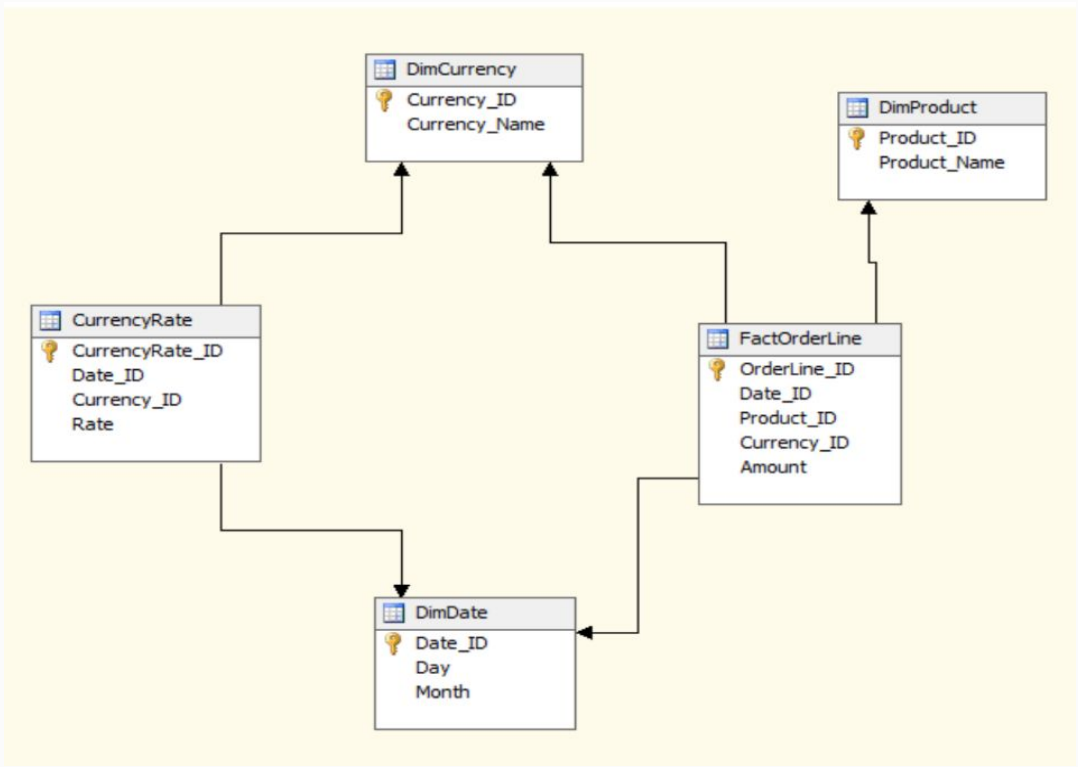
```
('#00008B','darkblue'),
('#000080','navy'),
('#191970','midnightblue'),
('#6A5ACD','slateblue'),
('#483D8B','darkslateblue')

-----------------------------------------------------------------------------------------------
insert into TableClothes values
(1, '#FFA07A', '#000080'),
(2, '#483D8B', '#6A5ACD'),
(3, '#00008B', '#B0E0E6'),
(4, '#FFA07A', '#000080'),
(5, '#483D8B', '#6A5ACD'),
(6, '#00008B', '#B0E0E6'),
(7, '#483D8B', '#6A5ACD')


-----------------------------------------------------------------------------------------------
select TableClothes.WeekDay, TableClothes.PantsColourID, c1.Colour,
TableClothes.ShirtColourID, c2.Colour
from TableClothes
inner join TableColour c1
on TableClothes.PantsColourID = c1.ColourID
inner join TableColour c2
on TableClothes.ShirtColourID = c2.ColourID
```

Handling Multiple Currencies

Fact tables that record financial transactions in multiple currencies should contain a pair of columns for every financial fact in the row. One column contains the fact expressed in the true currency of the transaction, and the other contains the same fact expressed in a single standard currency that is used throughout the fact table. The standard currency value is created in an ETL process according to an approved business rule for currency conversion. This fact table also must have a currency dimension to identify the transaction's true currency.



Insert some products in a fact table with different currencies (EUR,GBP,MEX,USD). In the currency table, maintain exchange rate with pivot currency as EURO. A pivot currency is the central currency that interacts with contra currencies.
Display amount in all the currencies for all the products in fact table.

```
create table DimDate(
        Date_ID int PRIMARY KEY,
        Day date,
        Month varchar(10));

create table DimProduct(
        Product_ID int PRIMARY KEY,
        Product_Name varchar(20));

create table DimCurrency(
        Currency_ID int PRIMARY KEY,
        Currency_Name varchar(10));

create table CurrencyRate(
        CurrencyRate_ID int PRIMARY KEY,
        Date_ID int,
        Currency_ID int,
        Rate float,
        FOREIGN KEY(Date_ID) references DimDate(Date_ID),
```

```
        FOREIGN KEY(Currency_ID) references DimCurrency(Currency_ID)
        );

-----------------------------------------------------------------------------------------------------------------------
create table FactOrderLine(
        OrderLine_ID int,
        Date_ID int,
        Product_ID int,
        Currency_ID int,
        Amount Float,
        FOREIGN KEY(Date_ID) references DimDate(Date_ID),
        FOREIGN KEY(Product_ID) references DimProduct(Product_ID),
        FOREIGN KEY(Currency_ID) references DimCurrency(Currency_ID));

-----------------------------------------------------------------------------------------------------------------------
insert into DimProduct values
        (1,'Nike Shoes'),
        (2,'Puma Shoes'),
        (3,'Adidas Shoes')
-----------------------------------------------------------------------------------------------------------------------
insert into DimCurrency values
        (1,'EUR'),
        (2,'USD'),
        (3,'INR'),
        (4,'MEX')
-----------------------------------------------------------------------------------------------------------------------
insert into DimDate values
        (1,'2019-03-01','March'),
        (2,'2019-03-02','March'),
        (3,'2019-04-04','April')
-----------------------------------------------------------------------------------------------------------------------
insert into CurrencyRate values
        (1,1,1,1),
        (2,1,2,0.8),
        (3,1,3,0.2),
        (4,1,4,0.1),
        (5,2,1,1),
        (6,2,2,0.7),
        (7,2,3,0.3),
        (8,2,4,0.2),
        (9,3,1,1),
        (10,3,2,0.9),
        (11,3,3,0.4),
        (12,3,4,0.2)
-----------------------------------------------------------------------------------------------------------------------
insert into FactOrderLine values
        (1,1,1,1,20),
```

```
        (2,2,2,1,25),
        (2,3,3,1,30)
-------------------------------------------------------------------------------------------------------------------
select
        DP.Product_Name, D.Day, FOL.Currency_ID,FOL.Amount,
        FOL.Amount*euro.Rate as Amount_in_Euro,
        FOL.Amount*usd.Rate as Amount_in_USD,
        FOL.Amount*inr.Rate as Amount_in_INR,
        FOL.Amount*mex.Rate as Amount_in_MEX
from FactOrderLine FOL
inner join DimProduct DP
        on FOL.Product_ID = DP.Product_ID
inner join DimDate D
        on FOL.Date_ID = D.Date_ID
inner join
        (select cr.Rate, cr.Date_ID from DimCurrency c,CurrencyRate cr
where c.Currency_Name='EUR' and c.Currency_ID=cr.Currency_ID)  as euro on euro.Date_ID =
D.Date_ID
inner join
        (select cr.Rate, cr.Date_ID from DimCurrency c,CurrencyRate cr
where c.Currency_Name='USD'and c.Currency_ID=cr.Currency_ID) as usd on usd.Date_ID =
D.Date_ID
inner join
        (select cr.Rate, cr.Date_ID from DimCurrency c,CurrencyRate cr
where c.Currency_Name='INR'and c.Currency_ID=cr.Currency_ID) as inr on inr.Date_ID =
D.Date_ID
inner join
        (select cr.Rate, cr.Date_ID from DimCurrency c,CurrencyRate cr
where c.Currency_Name='MEX'and c.Currency_ID=cr.Currency_ID) as mex on mex.Date_ID =
D.Date_ID
```

Multiple Time Zones

To capture both universal standard time, as well as local times in multi-time zone applications, dual foreign keys should be placed in the affected fact tables that join two rdate (and potentially time-of-day) dimension tables.

Just like the previous multiple currency example, create a fact table which has order timestamp in local time zone and another time zone table which has time zone conversions information from different time zones to UTC. Display results in different time zones.

```
create database lab3_4

use lab3_4

create table product_dim(
	product_key int primary key,
	product_name varchar(20)
	)
create table timezone_dim(
	timezone_key int primary key,
	timezone_name varchar(20),
	timezone_diff_utc int
	)
create table order_fact(
	order_id int primary key,
	product_key int references product_dim(product_key),
	order_timestamp datetime,
	timezone_key int references timezone_dim(timezone_key)
	)
-------------------------------------------------------------------------------------------------------------

GO
create function dbo.find_diff(@time_key int)
returns int
as
	begin
		declare @time_diff int
		select @time_diff=timezone_diff_utc from timezone_dim
			where timezone_key like @time_key
		return @time_diff
	end

GO
-------------------------------------------------------------------------------------------------------------
insert into product_dim values
	(10,'Nike Shoes'),
	(20,'Reebok Shoes'),
	(30,'Puma Shoes')
insert into timezone_dim values
	(1,'IST',-330),
	(2,'UST',240),
	(3,'AST',300)
insert into order_fact values
	(1,20,'2018-04-21 23:00:00',1),
	(2,10,'2016-08-11 10:00:00',2),
	(3,30,'2019-09-15 12:00:00',3)
```

```
-----------------------------------------------------------------------------------------------
select p.product_name,o.order_timestamp as Order_Timestamp_Local,
        DATEADD(minute,dbo.find_diff(o.timezone_key),o.order_timestamp) as
Order_Timestamp_UTC,
        DATEADD(minute,(dbo.find_diff(o.timezone_key)-dbo.find_diff(1)),o.order_timestamp) as
Order_Timestamp_IST,
        DATEADD(minute,(dbo.find_diff(o.timezone_key)-dbo.find_diff(2)),o.order_timestamp) as
Order_Timestamp_UST,
        DATEADD(minute,(dbo.find_diff(o.timezone_key)-dbo.find_diff(3)),o.order_timestamp) as
Order_Timestamp_AST
        from order_fact o
        join product_dim p
        on o.product_key = p.product_key
```

Slowly Changing Dimensions

Slowly Changing Dimensions (SCD) are the most commonly used advanced dimensional technique used in dimensional data warehouses. Slowly changing dimensions are used when you wish to capture the changing data within the dimension over time. There are three methodologies for slowly changing dimensions.

Type 1 (***Overwriting history)***– For this type of slowly changing dimension you simply overwrite the existing data values with new data values. This makes the updating of the dimension easy and limits the growth of the dimension table to only new records. The drawback of this is you lose the historical value of the data because the dimension will always contain the current values for each attribute. For instance, you have a store dimension which has an attribute for a geographic region. If there is a redesign in the regional boundaries some stores may move from one region to another. Because the record is simply updated a store which may have been reporting results in the Northeast district will now report their results to the Mid-Atlantic region. But, as a result of the update, now all of the history for that store before the move is essentially removed from the Northeast and moved to the Mid-Atlantic district. This change will skew the historical reports and the reports run before the update will no longer match the reports run after the update for the same timeframe.

Original Record

Key	ID	Name	Region
123	VA-13	ACME Products	Northeast
234	PA-07	Ace Products & Services	Northeast

Updated Record

Key	ID	Name	Region
123	VA-13	ACME Products	Mid-Atlantic
234	PA-07	Ace Products & Services	Northeast

Type 2 (*Preserving history)* – This is the most commonly used type of slowly changing dimension. For this type of slowly changing dimension, add a new record encompassing the change and mark the old record as inactive. This allows the fact table to continue to use the old version of the data for historical reporting purposes leaving the changed data in the new record to only impact the fact data from that point forward. Several columns should be added to the dimension table (active record start/end dates and a current active record flag) to provide historical change management and ensure optimal use of the active record. Using the same example from the Type 1 dimension above, the change in the district will cause the updating of the current active dimension record's active record end data and active record flag denoting this record is no longer actively in use. This will also spawn the creation of a new active record with a new dimension key. This new dimension key will be used in the generation of the fact table moving forward. This allows the fact table to still use the data stored under the old dimension key for historical reporting. This will ensure that the data remains the same and a historical report for the same timeframe run before the update was made will continue to display the exact same data as before the change was made.

Original Record

Key	ID	Name	Region	ACTV RCRD	ACTV START	ACTV END
123	VA-13	ACME Products	Northeast	1	20140328	99999999
234	PA-07	Ace Products	Northeast	1	20140508	99999999

Inserted / Updated Records

Key	ID	Name	Region	ACTV RCRD	ACTV START	ACTV END
123	VA-13	ACME Products	Northeast	0	20140328	20160728
234	PA-07	Ace Products	Northeast	1	20140508	99999999
784	VA-13	ACME Products	Mid-Atlantic	1	20160729	99999999

Type 3 (*Preserving a version of history)*– This is a seldom used type of slowly changing dimension. In this type of slowly changing dimension you add a second column to store the most recent past value of the column(s) you wish to be able to report on. When the data is updated the existing value is "moved" to the column defined to store the previous past value and the new value is placed into the reportable column. This allows you the ability to look back at what the value of the data was previously. This can be a challenge when loading/updating the data. The

amount of work to design and maintain this solution far exceeds the benefit the “fallback snapshot” provides.

Original Record

Key	ID	Name	Region	Previous Region
123	VA-13	Ace Hardware	Northeast	
234	PA-07	Ace Products	Northeast	

Updated Record

Key	ID	Name	Region	Previous Region
123	VA-13	Ace Hardware	Mid-Atlantic	Northeast
234	PA-07	Ace Products	Northeast	

Why is it named ‘Slowly’ Changing Dimension?
It is so because ideally, this component should be used on those tables which are not frequently updated.

For understanding, let us take an example of currency i.e. we will try to update and load currency data as we know that once in a week or month this data changes. Since its data changes slowly we can apply a slowly changing component to it.