Curry-encies never tasted so good

I was talking with my partner the other night over a nice curry about a niggly issue applying currency conversion.

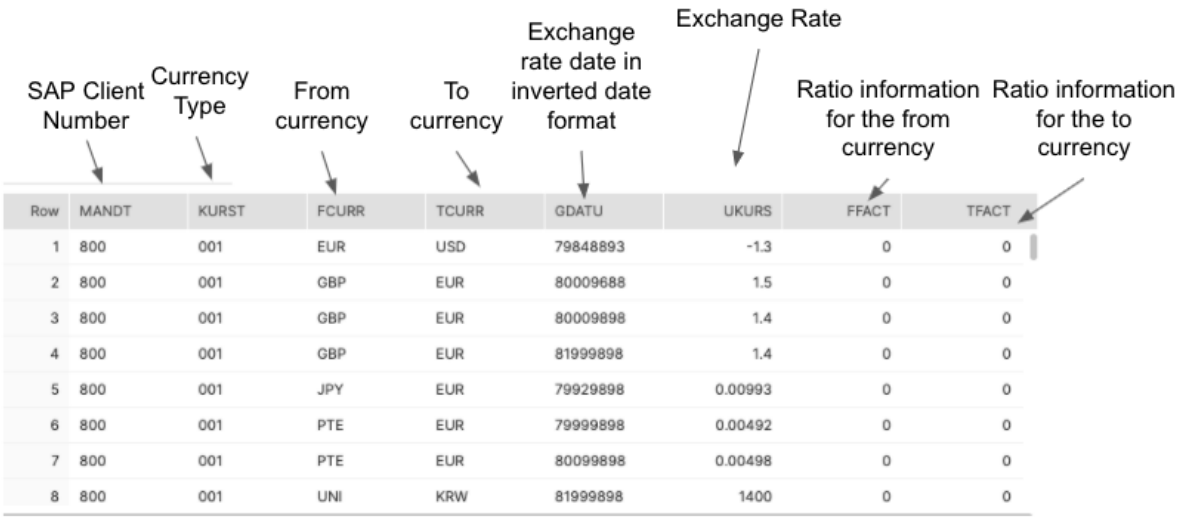
She looked at me like I had just slipped her some extra chili-pepper. "There are a ton of web pages out there that will calculate the currency for you," she pointed out. She is, of course, correct. But some operational systems that Snowflake customers want to get data out of originated in the 1970s, way before the Internet. Currency tables were updated monthly. These days, we need to apply logic within the system to compensate for missing dates. And you have to get it right. If you are off by a small fraction, business will not trust the data, and your 80% savings of replacing systems like SAP BW with Snowflake burn like an over-cooked nan. Luckily, with a couple lines of SQL in Snowflake you can get the same results!

Consider this scenario: You are the head of analytics and have an SAP ERP system as a source. To get accounts receivable information, you use SAP extractor 0FI_AR_4 or replicate tables BSID (open items) and BSAD (cleared items) from your SAP R/3 or ECC system. The postings are in one of 49 local currencies, but your corporate dunning report needs to have the global currency in euros. You have three options:

- Option 1: Get the SAP admin, Agent Smith[1], to exchange the currencies at the ERP level. You are Mr Anderson in this scenario, so feel your inner Neo, change the Matrix construct, time warp and get your project in before the due date. Yeah, right.
- Option 2: Source the currency table from SAP to perform the conversion, according to SAP rules of working with it. This article explains the flow, and gives an example of how to perform the same logic in SQL.
- Option 3: Use the Snowflake Data Marketplace to locate freely available and current currency exchange data. You can promptly join your fact table on date and currency, and within a couple of seconds, you have your currency rates. I don't talk about this at all, as it is way too easy, but check out Dmitry's Snowflake Quickstart guide[2] on how to do this.

TCURR

SAP performs currency conversions in their ERP (Enterprise Resource Planning) application, but also in SAP BusinessWarehouse (BW) using ABAP. SAP uses the TCURR table, which gets updated on a scheduled basis, and some business logic to do this.



Row	MANDT	KURST	FCURR	TCURR	GDATU	UKURS	FFACT	TFACT
1	800	001	EUR	USD	79848893	-1.3	0	0
2	800	001	GBP	EUR	80009688	1.5	0	0
3	800	001	GBP	EUR	80009898	1.4	0	0
4	800	001	GBP	EUR	81999898	1.4	0	0
5	800	001	JPY	EUR	79929898	0.00993	0	0
6	800	001	PTE	EUR	79999898	0.00492	0	0
7	800	001	PTE	EUR	80099898	0.00498	0	0
8	800	001	UNI	KRW	81999898	1400	0	0

Figure 1: SAP TCURR currency exchange table

[1] Youtube, https://bit.ly/3y0GEPv
[2] Snowflake Quickstart Guides, https://bit.ly/3CU92WQ

SAP TCURR Currency Exchange

The flow chart below shows a logic commonly found in SAP systems, where the currency exchange rate is not mapped on a daily basis, but rather takes the month median from the first day of the fiscal period. If that currency is missing, it takes the last day of the previous period, and if that is missing, it looks forward and takes the first day of the subsequent fiscal period (and so on) until it finds a currency conversion record it can use.[3] Here is a flow diagram:

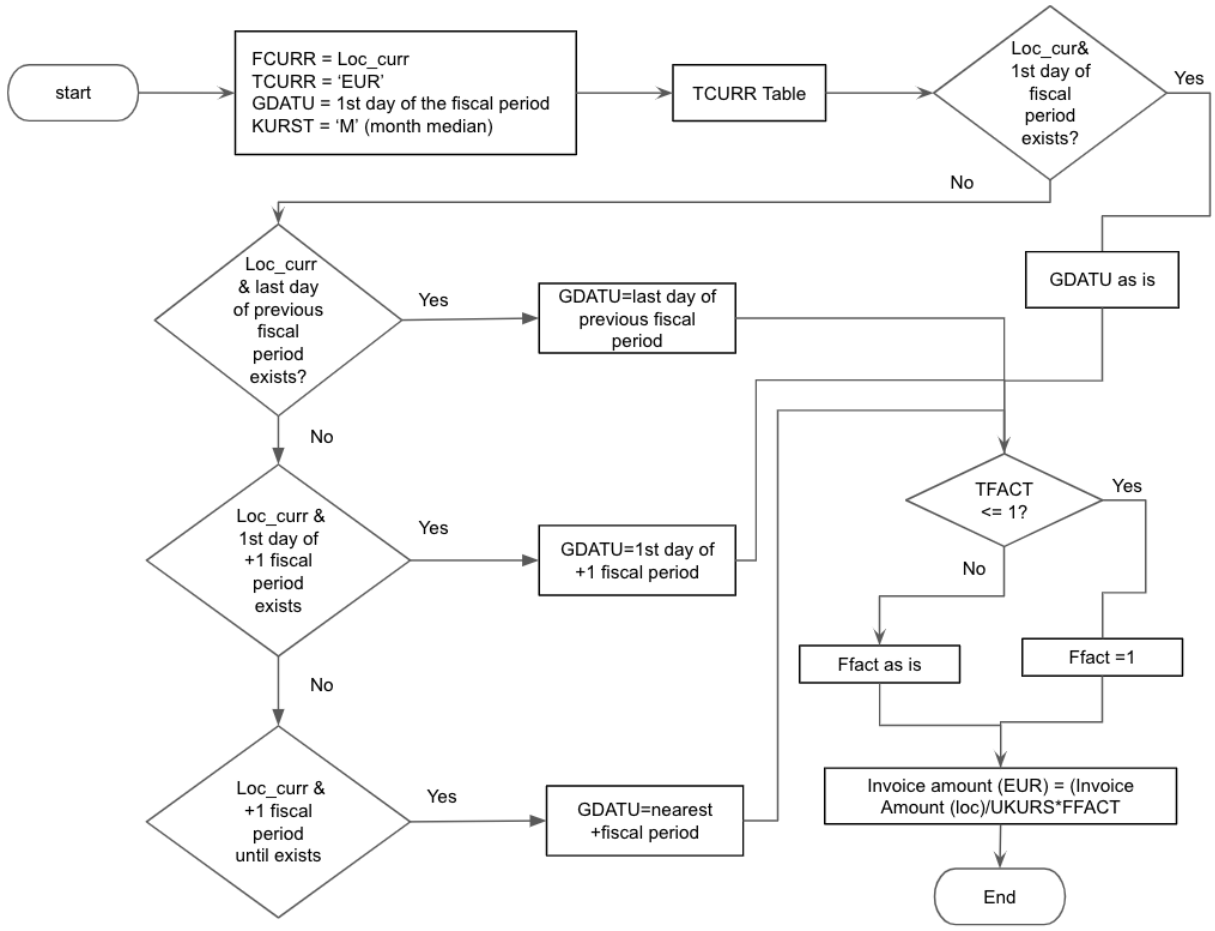


Figure 2: Flow chart in choosing substitution dates

But why?

Or as my colleague put it, "I have not seen this logic elsewhere." Well, because pre-Internet, companies did not necessarily upload the currency rates on a daily basis (and, still today, you can't get an exchange rate on a weekend). Taking the median is less work, and takes less computational power. Rounding accounting had its purposes. See how Richard Pryor gets rich (for about five minutes) on exploiting a rounding accounting practice in Superman III.[4] Fast

[3] To simplify this article I use month and fiscal period interchangeably.
[4] Youtube, https://bit.ly/3yC9dDM

forwarding to the present, also check out how you can fight cyber-fraud using Snowflake's super-powered security lake.[5]

Even if you are in doubt about using median, you want to make sure the numbers out of the SAP system match the conversion results out of Snowflake.

Back to the flow chart...

The flow, at the end, looks at the TFACT field, because not all currencies are necessarily stored in the same unit. For example, the Japanese yen could be stored in units of 100, for example 100 JPY = .94 USD, as opposed to 1 JPY = .0094. It's worthwhile to check your data, though, because the TFACT and FFACT fields may not be used anymore.

This is a good time to explain a couple other idiosyncrasies in the TCURR table:

- The GDATU date field is stored as an inverted date format. This means that to get the date, you have to subtract the stored number from 99,999,999. The logic behind this groovy 1970s move was to aid in sorting—to get the latest date first. Now-a-days we have *user interfaces*. Man, do I miss playing with those punch cards.

- The UKURS numeric field *KURS*, from German, means *course* or *exchange rate*. Negative values in currency rate should be understood as inverse : e.g. -45.77 for 1 / 45.77, so SAP can store the value without any rounding. Anything to save space and be accurate, right? That's positively negative.

OK, how to write that in SQL?

As always, there are many different ways to solve a problem. Here I propose one possible SQL solution to implement the logic in the figure above.
Let's draw this out first. I know I have a pretty sparse currency exchange table, and I would like to fill out the missing dates and exchange rates upfront so that when I join it to my fact table, the calculation will be super fast. And I want to have all the dates that exist (in my range), so it works for no matter what fact table I join it to. A client asked me to write a function call, which you can do, too. A bit slower, but nicer interface. That's for another article...

[5] Snowflake Security Lake, https://bit.ly/2VZaN41

Visual high level

My SQL solution consists of three blocks. The results of one block feed into the next.

1. First, because I am lazy (and don't like duplicates), I want to find out what's missing and then just apply my data substitution logic to that. To do that, I subtract the dates and currencies in the TCURR table from a date currency dimension table that holds *all* combinations of dates (well, the first day of the month date that is) and currencies.

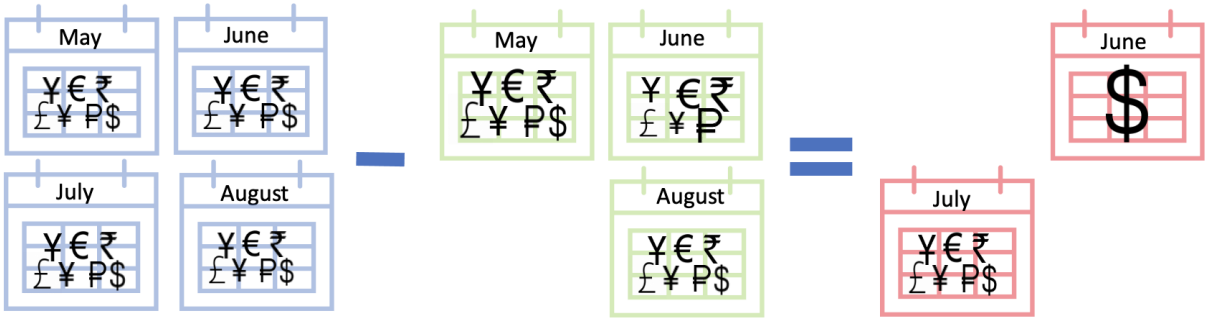
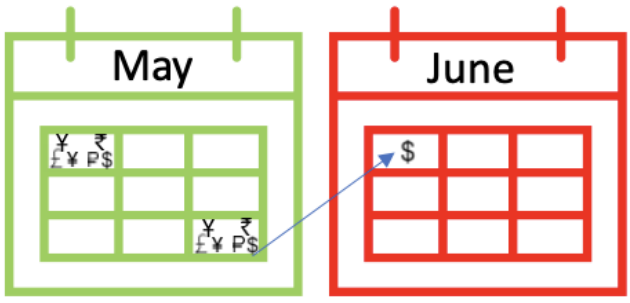


Figure 3: Finding the values that need substitutions

The blue represents all date and currency type combinations, green the values in TCURR, and red the values for which we need to find substitutions. Note that June is missing a USD value, and July is missing all currencies.

2. For each of these missing date/currency pairs I check whether I have a currency value in previous and subsequent dates to the missing date, and then choose the smallest (or oldest) of these dates.

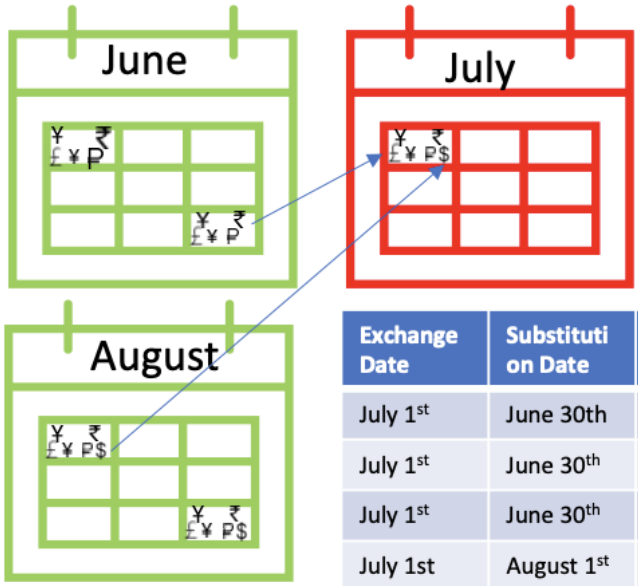
   Example 1: USD is missing from June 1st, so I look back one day to the last day in May and I find a currency value, so I perform the substitution, and get the currency exchange.



Exchange Date	Substitution Date	From Currency	To Currency	Currency Exchange
June 1st	May 31st	USD	EUR	.87

Figure 4: Last day, previous month substitution

For the next month, all currency types are missing from July, so I again look back one day, to the last day in June and I find values for everything except USD. So I take those June values for those currencies. I then look forward one month and I find a value for USD, and use that substitution value.



Exchange Date	Substitution Date	From Currency	To Currency	Currency Exchange
July 1st	June 30th	Yen	EUR	.0077
July 1st	June 30th	Ruble	EUR	.0012
July 1st	June 30th	GBP	EUR	1.17
July 1st	August 1st	USD	EUR	.85

Figure 5: Last day, previous month substitution and next month substitution

3. Once I have all my substitution values I pull all the values back together into a new table that I can join to my fact table.

SQL snippets

Precursors

Before running the currency exchange SQL I set up a couple of tables :

- a special date dimension table that only returns the first day of each month
- A transformed TCURR table that has a date field, a positive exchange number, unit consistency, and filters for target currency and currency type.

Please check the section at the end if you would like to have the SQL for this.

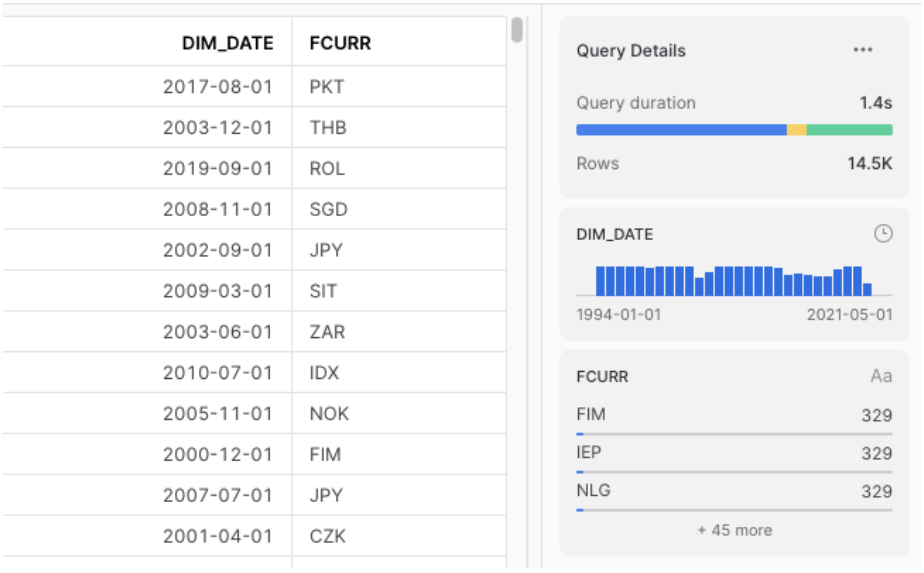
To build my TCURR table with substitution logic, I want to walk you through the three steps. The results from the previous step feed into the subsequent step. The full CTE is supplied in the appendix.

Step 1: NOT IN

As a first step I can use a `NOT IN` clause to get the missing dates and currencies, for example:

```
select distinct dim_date, from_currency
 from date_dimension
  where (dim_date, from_currency) not in (select gdatu, from_currency from
tcurr) ;
```

Running this query in Snowsight, I get the following results and understand I have about 14.5 thousand rows I will need to substitute, that the substitution dates range from 1994 to 2021 and there are 48 distinct currencies.



DIM_DATE	FCURR
2017-08-01	PKT
2003-12-01	THB
2019-09-01	ROL
2008-11-01	SGD
2002-09-01	JPY
2009-03-01	SIT
2003-06-01	ZAR
2010-07-01	IDX
2005-11-01	NOK
2000-12-01	FIM
2007-07-01	JPY
2001-04-01	CZK

Figure 6: Not In Results

Step 2: MIN

As a second step, I take the results of the first step and run them through a date comparison logic by simply taking the minimum date I find:

```
select dim_date, c.from_currency, min(gdatu) as substitution_date
from tcurr l join [results from step 1]
     on (dim_date - 1 = fxdate and l.fcurr=c.fcurr)
         or
        (dim_date < fxdate
       and DATE_TRUNC('MONTH',dim_date) < DATE_TRUNC('MONTH',fxdate)
       and dayofmonth(gdatu) = 1
       and l.fcurr=c.fcurr)
group by dim_date, c.fcurr
```

The first part of the join `dim_date - 1 = fxdate and l.fcurr=c.fcurr` checks to see if there is a value in the last day of the previous month. The second part of the join clause checks to see if subsequent values exist. Using `date_trunc` in this part allows me to compare results at the month level, which is what the logic asks for and speeds up my calculation. The predicate `dayofmonth(gdatu) = 1` keeps the min(gdatu) from looking at other dates than the first day of the month.

These intermediate results show the dimension and substitution date along with the from currency. Substitution dates must be either the end or the previous month or the first of the next or subsequent months, and this is what we see here.

	DIM_DATE	FCURR	SUBSTITUTION_DATE
1	2012-05-01	MXN	2012-11-01
2	2012-05-01	CNY	2012-11-01
3	2013-02-01	AED	2013-03-01
4	2013-06-01	USD	2013-05-31
5	2012-05-01	SAR	2012-11-01
6	2013-06-01	SGD	2013-05-31
7	2012-10-01	NOK	2012-11-01
8	2013-09-01	JPY	2013-10-01
9	2013-12-01	CNY	2014-04-01
10	2012-05-01	PLN	2012-11-01
11	2014-06-01	RUB	2014-07-01
12	2014-02-01	AED	2014-01-31
13	2012-08-01	MXN	2012-11-01
14	2012-05-01	RUB	2012-11-01
15	2012-06-01	PLN	2012-11-01
16	2014-03-01	SAR	2014-02-28

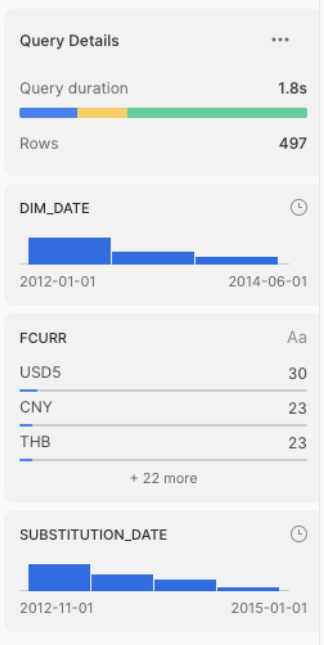


Figure 7: Substitution dates

Step 3: Union ALL

As a third step, I take the above results with substitution dates and values, and combine it with the dates and values that existed in the original TCURR table.

```
select dim_date, substitution_date, fcurr, tcurr, ukurs
from [results from step 2]
union all
select gdatu, gdatu, fcurr, tcurr, ukurs
from tcurr
where  dayofmonth(gdatu) = 1
union -- treat cases when the posting local currency is in EUR and no
conversion is necessary
select dim_date
, dim_date
, 'EUR'
,'EUR'
, 1
from date_dimension
) ;
```

The results look like this:

DIM_DATE	SUBSTITUTION_DATE	FCURR	TCURR	UKURS
2014-01-01	2014-01-01	SAR	EUR	0.193958202
2014-01-01	2015-01-01	USD5	EUR	0.9009009009
2014-01-01	2013-12-31	THB	EUR	0.02213466732
2014-01-01	2014-01-01	RUB	EUR	0.02214067075
2014-01-01	2014-01-01	PLN	EUR	0.2407608041
2014-01-01	2014-01-01	NZD	EUR	0.598820324
2014-01-01	2014-01-01	NOK	EUR	0.1206025302
2014-01-01	2014-01-01	EUR	EUR	1
2014-01-01	2013-12-31	AUD	EUR	0.6488450558
2014-01-01	2014-01-01	HKD	EUR	1.204891861
2014-01-01	2013-12-31	BRL	EUR	0.3069744597
2014-01-01	2013-12-31	MXN	EUR	0.05580357143
2014-01-01	2014-01-01	GBP	EUR	1.204891861
2014-01-01	2014-01-01	DKK	EUR	0.1340536483
2014-01-01	2014-01-01	CZK	EUR	0.03657510698
2014-01-01	2014-01-01	CHF	EUR	0.8148299043
2014-01-01	2014-01-01	CAD	EUR	0.6830834386
2014-01-01	2014-01-01	AED	EUR	0.1980492152
2014-01-01	2014-01-01	USD	EUR	0.7274314396
2014-01-01	2013-12-31	CNY	EUR	0.1197733887
2014-01-01	2014-01-01	ZAR	EUR	0.06929550723
2013-12-01	2014-01-01	DKK	EUR	0.1340536483

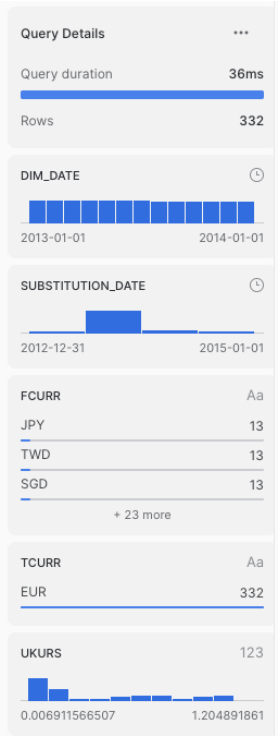


Figure 7: Currency Substitution Table

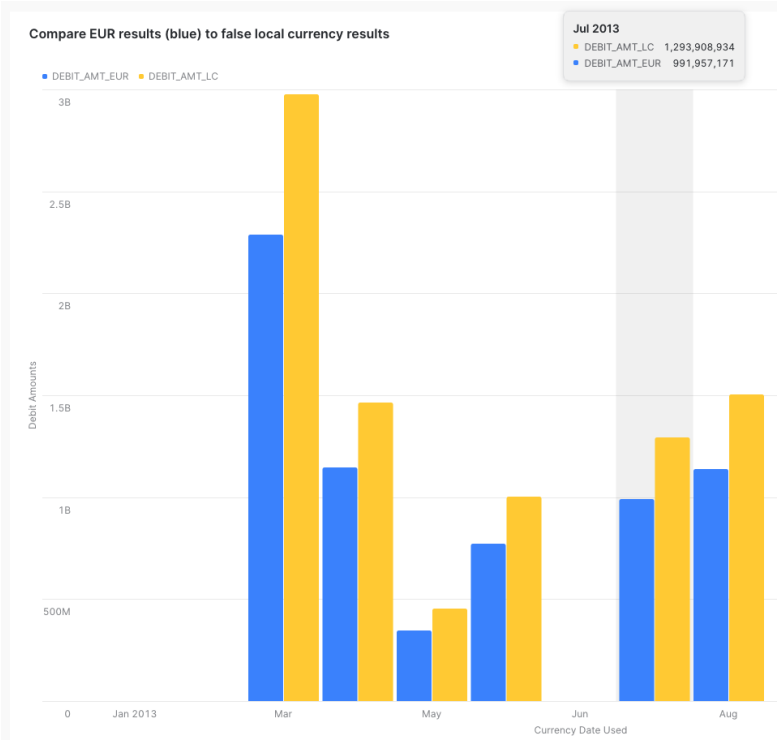
More cosmetic, but notice that I also union a EUR to EUR conversion that equals 1, so that when I join to my fact table, all the postings that were originally in EUR are filled in my substitution TCURR table.

In sum, facts, units and Mr. Faddis

I now join my new currency table to my extracted accounts receivable table:

```
Select
budat as posting_date
, belnr as document_number
, substitution_date as curr_date_used
, lcurr as local_currency
, dmsol as debit_amt_lc
, ukurs as fx_eur
, dmsol*ukurs as debit_amt_eur
from SAP.SAP."0fi_ar_4" left join tcurr_with_substitutions
on date_trunc('month', budat)= dim_date
and lcurr=fcurr
where
 dmsol <> 0 and
 posting_date between '2013-01-01' and '2013-12-31'
order by budat desc;
```

To show the numeric disaster that we avoided by correctly converting our local currencies to Euros, I built the following chart that compares the Euro results to, as my science teacher Mr Faddis would say, *absolute gibberish*, the sum of local currencies results. You can't, of course, aggregate numbers with different units.



What's the Point?

SAP BW does currency conversion with a key-click. Why go to the bother of the 40 odd lines of SQL? Well SAP BW is a great application, but it does not scale well, is not cloud native, costs a lot of money and people-power to maintain, and SAP has gutted its roadmap and removed the reporting tools. As your S/4Hana project will pull back in the operational reporting piece, SAP BW is no longer needed and it would be much better that your future analytic data foundation based on open-standard, transparent technology, such as Snowflake, where you can combine all your SAP, non-SAP, sensor, social, business partner and data exchange data in one place, breaking down silos and catapulting ahead of your competition. Use S/4Hana for your transactional and operational reporting. Use Snowflake for analytical, machine learning and analytical applications. This move to Snowflake is economical, unlocks business value, accelerates innovation. Customers using Snowflake are taking advantage of 10x-300x more data than traditional systems. Brush up on your SQL a bit, and with Snowflake you will have more time to eat that curry! Thanks!

Full SQL

Date dimension table generation

You may be wondering what my `date_dimension` table looks like and how I generated it. Here is the SQL I used:

```
create or replace table date_dimension (
   dim_date             date        not null
  ,dim_year             number    not null
  ,dim_month            number    not null
  ,day_of_mon           number    not null
)
as
  with cte_my_date as (
    select dateadd(day, seq4(), '1994-01-01') as my_date
      from table(generator(rowcount=>10000))
  )
  select my_date
        ,year(my_date)
        ,month(my_date)
        ,day(my_date)
    from cte_my_date
    where day(my_date) = 1;
```

Notice that my `where` clause is only returning the first day of each month.

TCURR table transformations

And you may be wondering what fun and exciting transformations I pre-performed on the TCURR table. And here that is:

```
/* We want to modify tcurr a little bit to get a date, a positive
exchange number, unit changes, just EUR for target currency and
currency type as EURX */
create or replace table tcurr as
(select
mandt
, kurst
,   fcurr
,   tcurr
,   to_date (substring(to_varchar(99999999 - gdatu),1,8), 'YYYYMMDD')
gdatu
, case when ukurs < 0 and tfact <= 1 then -1/ukurs
        when ukurs < 0 and tfact > 1 then -1/ukurs*ffact
        when ukurs > 0 and tfact > 1 then ukurs*ffact
 else ukurs end ukurs
, ffact
, tfact
 from
 "SAP"."SAP"."ztcurr_attr" where tcurr = 'EUR'
and kurst='EURX'
);
```

Full CTE (it was split up in the main article into 3 blocks)

This is what the CTE looks like when I bring it all together:

```
create or replace table tcurr_with_substitutions as (
with

  find_missing_dates as (
   select distinct dim_date, fcurr
 from date_curr_dimension
  where (dim_date,fcurr) not in (select gdatu, fcurr from tcurr)
--14.5k rows

    )
```

```
/*
  STEP 2: MIN. Generate substitute dates by looking for the last day
of the previous month or
  for the first available first day of a subsequent month.
*/

, substitute_dates as (
select dim_date, fmd.fcurr, min(gdatu) as substitution_date
from tcurr t join find_missing_dates fmd
     on (dim_date - 1 = gdatu and t.fcurr = fmd.fcurr)
         or
        (dim_date < gdatu
         and date_trunc('month',dim_date) < date_trunc('month',gdatu)
         and dayofmonth(gdatu) = 1
         and t.fcurr=fmd.fcurr
         )
group by dim_date, fmd.fcurr
    )

/* STEP 3: UNION ALL  */

, add_missing_columns as
(select dim_date, substitution_date, sd.fcurr, tcurr, ukurs
 from substitute_dates sd join tcurr t on substitution_date=t.gdatu
and sd.fcurr=t.fcurr
)
/*
    union to the existing dates and currencies
*/
select dim_date, substitution_date, fcurr, tcurr, ukurs
from add_missing_columns
union all
select gdatu, gdatu, fcurr, tcurr, ukurs
from tcurr
where  dayofmonth(gdatu) = 1

/*Fill currency table for when from and to date are EUR */
union
select dim_date
, dim_date
, 'EUR'
,'EUR'
, 1
from date_dimension
) ;
```