Part 1: The Star Schema Data Model

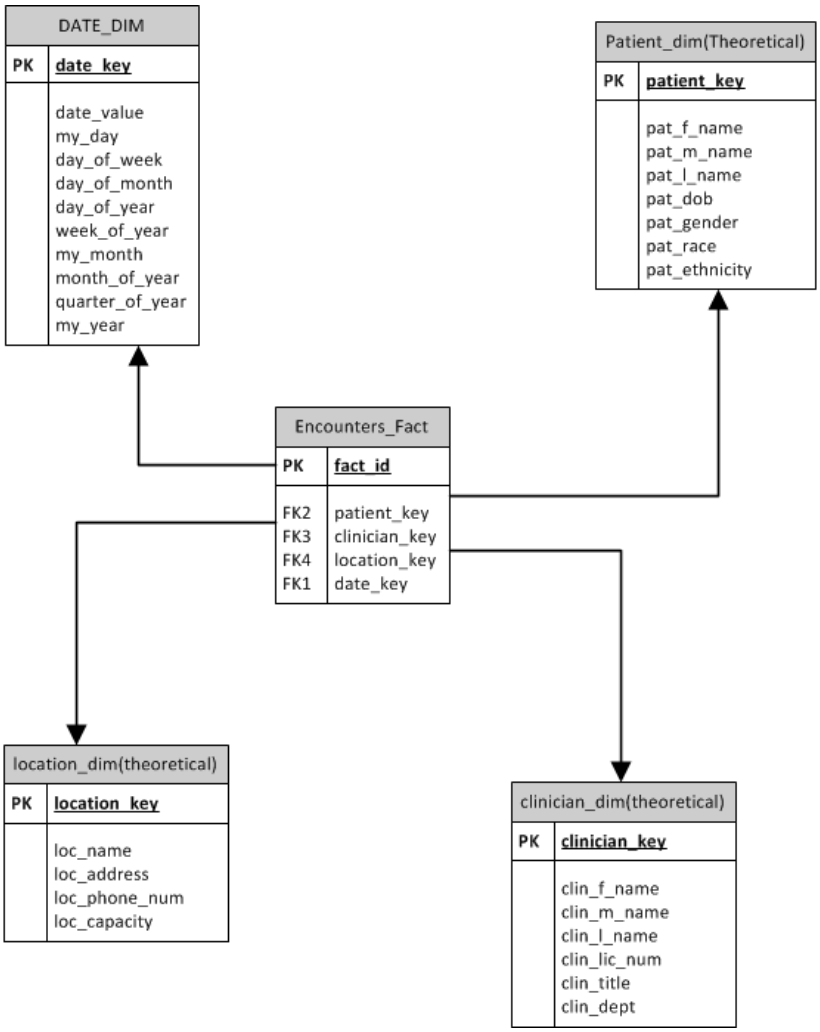
A star schema model is comprised of Dimension Tables and Fact Tables.

Dimension tables. Store your 'Nouns', People, Places, Things, and Time. For the purposes of this exercise assume that your dimension table will have exactly one row, aka one entry, for each noun. For example, a patient dimension will have exactly one row/entry for each patient at your institution. Each 'Noun' is uniquely identified by a Primary Key.

Fact tables. Are a bit different. Consider encounters as an example. Each encounter can be viewed as an interaction between many 'Nouns'. There will be a patient, a clinician, a place (e.g., office/hospital/lab), and a time. When all of these 'Nouns' intersect, we have an encounter. 'Nouns' exist in the fact table only as a reference back to their home in their respective dimension table. There is no point describing the same noun, in the same way, in a large number of fact tables.

In this case, the key is called a foreign key because it uniquely identifies a row in a foreign table, specifically the dimension table of interest.

Here is an example of a star schema representing this scenario:



Step 1: Review the SQL Used to Create the Date Dimension Table: DATE_DIM

This SQL creates a table named DATE_DIM which contains one row for every day from January 3rd 2000 – December 31st 2015. (January 3rd was selected because it was the first Monday of the year 2000). This table should have an entry for every day that you wish to report on. Notice how each row describes the same day in a number of ways (i.e., Day of week, day of month, day of year). This is part of the power of describing all dates in single table; all of this data is created once and can be re-used in every query. The downside is that it takes time to create and maintain these tables.

```
CREATE TABLE Date_Dim(
Date_Key number(18,0) NOT NULL,
Date_Value Date NOT NULL,
my_Day Char(10 ),
Day_Of_Week number(18,0),
Day_Of_Month number(18,0),
Day_Of_Year number(18,0),
Week_Of_Year number(18,0),
my_Month Char(10),
Month_Of_Year number(18,0),
Quarter_Of_Year number(18,0),
my_Year number(18,0)
```

```
);

--SQL To populate the table:

INSERT INTO
        Date_Dim
SELECT
        to_number(to_char(CurrDate, 'YYYYMMDD')) as Date_Key,
        CurrDate AS Date_Value,
        TO_CHAR(CurrDate,'Day') as Day,
        to_number(TO_CHAR(CurrDate,'D')) AS Day_Of_Week,
        to_number(TO_CHAR(CurrDate,'DD')) AS Day_Of_Month,
        to_number(TO_CHAR(CurrDate,'DDD')) AS Day_Of_Year,
        to_number(TO_CHAR(CurrDate+1,'IW')) AS Week_Of_Year,
        TO_CHAR(CurrDate,'Month') AS my_Month,
        to_number(TO_CHAR(CurrDate,'MM')) AS Month_of_Year,
        to_number((TO_CHAR(CurrDate,'Q'))) AS Quarter_Of_Year,
        to_number(TO_CHAR(CurrDate,'YYYY')) AS my_Year
FROM
(
select
        level n,
        --The date which starts the reporting period
        TO_DATE('01-02-2000','MM-DD-YYYY') + NUMTODSINTERVAL(level,'DAY')
CurrDate
from
        dual
connect by
        --number of days to get to the end of the reporting period
        level <= 5842
);
commit;
```

You should review the DATE_DIM that was provided to you.

Now that we have seen the creation of the DATE_DIM table we can discuss its use. The purpose of this table is to have a single interface through which to query for date. For example, think about a table with 30,000 rows that spans data for 15 years. Over the course of 15 years there are more than 6,000 unique days (rough estimate, leap years and other calendar anomalies can impact) yet a date will be stored in each of the 30,000 rows of this table. This means there will be a lot of duplicate dates. Imagine now that you are asked to find all rows that have a date from the first year of the 15 year span. If you were to query the table directly, your database would have to iterate through all 30,000 rows and test whether or not the date on that row is in your year of interest*.

By using the DATE_DIM interface, we can significantly improve our database performance. Refer back to the SQL that created the DATE_DIM as well as your review of the table. Now consider querying this table to find data from your specific year. In this case, your database will have to scan less than 6,000 rows to find the applicable dates (Indexes change the truth of this statement quite a bit. I am excluding them for simplicity.).

Now the question becomes how we connect DATE_DIM to our data to achieve this performance benefit. This is where we exercise the FOREIGN KEY-PRIMARY KEY relationship between the fact table and the dimension table.

Since each fact carries a copy of the dimensions primary key as a foreign key we can use these data to JOIN the two tables (reference: http://www.w3schools.com/sql/sql_join.asp). We will walk through an example below to illustrate this. But first, let us provide an introduction to our fact table.

ENCOUNTERS_FACT

Since this is our fact table the create script is not present. Rather, this table would be loaded with data from your institutions EMR. Notice how this table contains foreign keys which act as the connection back to the dimension tables. It is unlikely your institution captures the data in this format; rather this table will be created by a database expert who will make it available for reporting. There can be a significant amount of processing required to put data in this format. In this example we have done it for you.

In a SQL environment you could execute this query:

```
Select *
  from encounters_fact;
```

In our case, please study the **ENCOUNTER_FACT** table that was provided to you.

Take special notice of these fields:
1. Enc_start_datetime – this is a standard date field
2. Date_key – this is a foreign key which uniquely identifies one row in the DATE_DIM table
3. Time_key – this is a foreign key which uniquely identifies one row in the TIME_DIM table

Step 2 – Compare Standard Queries vs Those Written for a Star Schema

Here we illustrate the example proposed above, selecting one year from the 15 year span. We choose calendar year 2008.

Standard Query for all encounters that take place during calendar year 2008

```
select
        *
```

```
from
        encounters_fact
where
        trunc(enc_start_datetime) >= to_date('01-01-2008','MM-DD-YYYY')
        and trunc(enc_start_datetime) < to_date('01-01-2009','MM-DD-YYYY');
```

Notice here how the date field in the encounters_fact table is referenced directly. This query took 0.536 seconds on my machine.

Query using star_schema and DATE_DIM

```
select
        *
from
        encounters_fact enc
        join date_dim ddim on enc.date_key = ddim.date_key
where
        ddim.my_year = 2008;
```

Notice here how the ENCOUNTERS_FACT.DATE_KEY foreign key is joined to the DATE_DIM.DATE_KEY primary key. This is the join which allows us to exercise the efficiency of the star_schema. This query took 0.398 seconds on my machine which is about 25% faster. Now imagine how much of an improvement there would be if the encounters table had 300,000 or 300,000,000 records. The star schema query would still only have to search the less than 6000 date_dim records even as the ENCOUNTERS_FACT grows to a large scale.

Disclaimer:
The performance will still degrade as ENCOUNTERS_FACT grows large but, this is due to the fact that your database has to connect DATE_DIM with ENCOUNTERS_FACT through a join. This join will consume more resources as the table size increases, but most databases are highly optimized for joins and the expense can be mitigated through indexing.