**Storing a date range:**

Rows report on factual count which don not change substantially overtime.

In date range: insert new record only when factual count has changed

-not every product sell every day.

If stock position is same as yesterday, update existing record by incrementing date range.

-Structure of query needs to change to account for date range.

-to query against date range use between statement

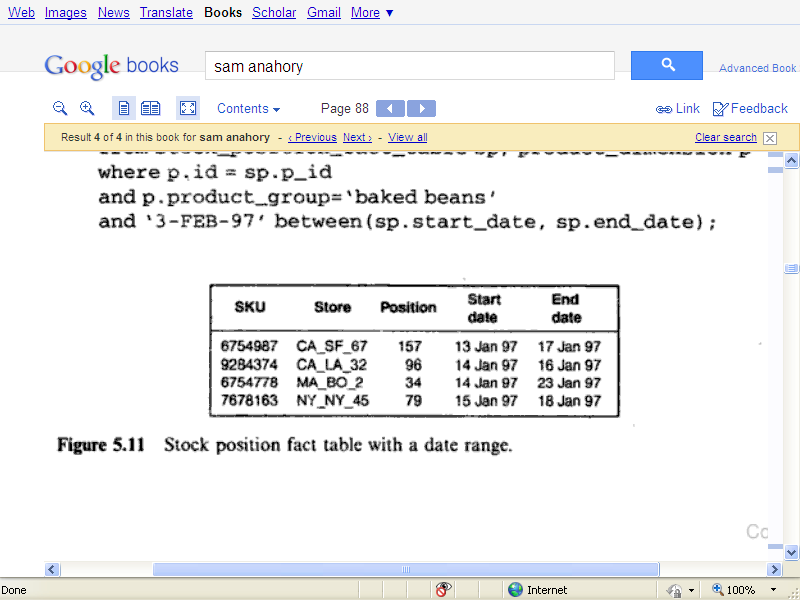
*select sum(sp.Position)*

*from stock\_position\_fact\_table sp, product\_dimension p*

*where p.id=sp.p\_id*

*and p.product\_group=‘baked beans’*

*and ‘3-Feb-11’ between (sp.start\_date,sp.end\_date);*



This technique can produce significant saving in disk capacity and query performance

A number of access tools may not be able to process fact tables in this format..

Solution: Create a database view, to make fact table appear to have a row for every day within date range. This view will utilize time dimension which has row for every day of year for every year in ware house. This table is joined against date range in order to produce a Cartesian product.

Unfortunately, cost of creating Cartesian product is very high, require significant processing power and space. Practically unacceptable, so only use date ranges when access tools can cope directly with structure

**Partition the fact table:**

Very detailed and will be explained in chapter 6.

**DESIGNING DIMENSION TABLE**

**Creating a star dimension:**

Speed up query performance by denormalizing reference info into a single table.

-Rely on perceived use of information by typical queries , where bulk of queries are likely to be analyzing facts by applying number of constraints against a single dimension

E.g. in retail sales ware house, typical queries will analyze sales info by product dimension.

As in most cases, query analyze facts by constraining products dimension in variety of ways so query can be speed up if constraining info will also be in same table.

Take all product hierarchy info and place it to same table.

Performance saving of not having join.

Inappropriate method if bulk of queries not often access those columns, this technique will speed up minorities of queries and slow down majority of them.

Star dimension is generated from snowflake data, so if you need to change the star dimension in future to add extra columns you can change it easily.

**HIERARCHIES AND NETWORKS:**

It’s not possible to denormalize all entities into star dimension.

Specifically the ones with many to many relationship should not be denormalize into a star dimension.

Determine the hierarchy that is most likely to be used by largest number of queries .they denormalize it into star dimension table.

The technique is effective when route is unlikely to change in future, query profile should not change or hierarchy itself is not stable.

Good compromise , if query profile changes: add new columns to star dimension, as long as existing columns are not replaced there should not be impact on existing canned queries.

**DIMENSIONS THAT VARY OVER TIME:**

Some dimensions vary overtime, particularly true for dimensions that use hierarchies or networks. As business can be changed in future so chances that dimension may change over time.

e.g shirts can move from menswear to , baked bean could move from canned