<app> Reporting Database Example Services Engagement

Louis Mugnano
Consulting Services

Mugnano Data Consulting

Integrity Matters, Data and Personal

Disclaimer

- ✓ The following slides show the concepts and design strategies for building a Relational OLAP Data Warehouse that we have helped other clients, across multiple business sectors, implement
- ✓ There is nothing inherent in the concepts and design that dictate any product, these concepts are not product specific or focused and transcend relational database technologies
- ✓ The specific tools presented and many of the physical design characteristics are specialized for Greenplum or Cloudberry and have only been implemented for Greenplum customers (Tools work on Postgres as well

Disclaimer

- ✓ The deck provides an example of a Data Services engagement but, for Proprietary reasons and ease of learning, the examples shown utilizes a generic tiny data model and example

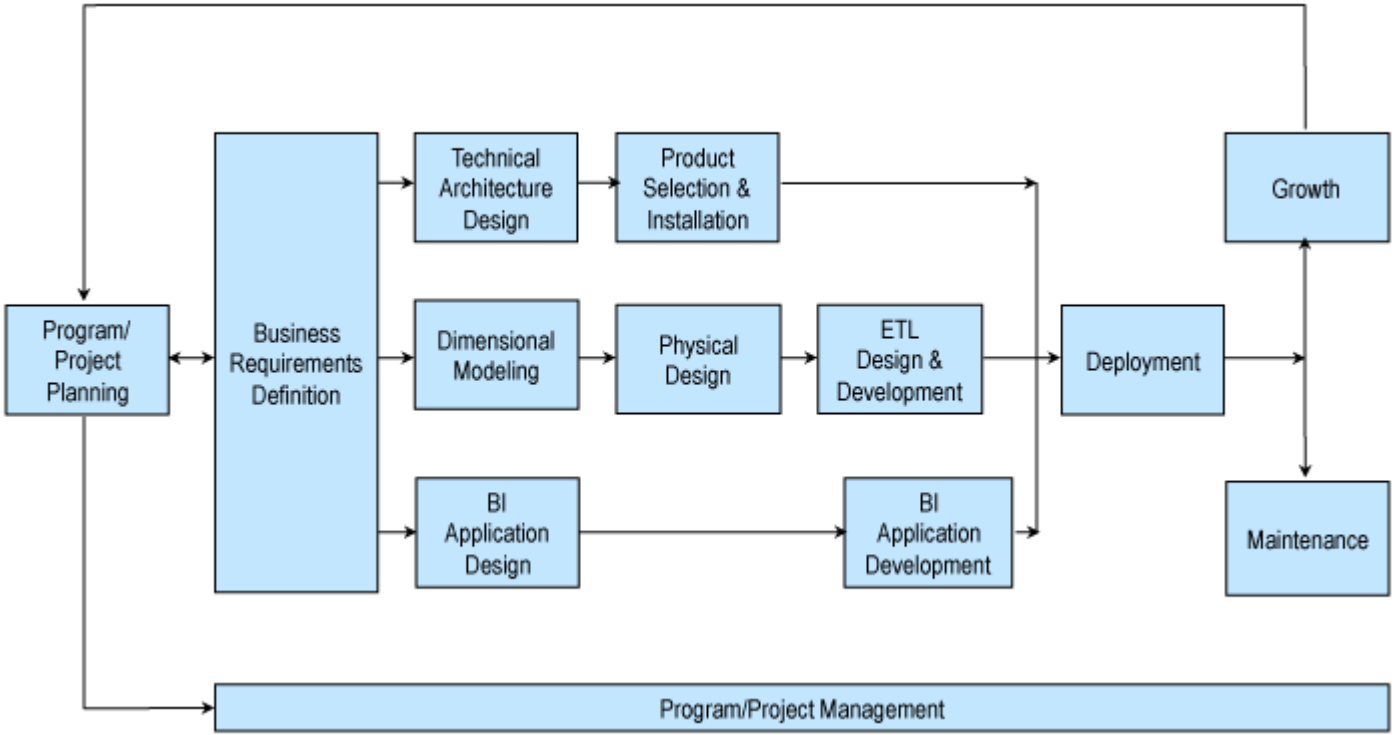
Example Project Characteristics

- ✓ Data was already in Greenplum as an ODS in the same database that the reporting schema(s) would reside (1000's of tables and 10's of schemas)
- ✓ ODS data was direct copy of source feeds (generally mainframe data) with minor enhancements of the data
- ✓ Extensive development effort was required for reporting and inconsistent results were provided depending on which developer answered the question (inconsistent interpretation of data)

Project Goals for the Example Project

- ✓ Provide a schema structured for reporting use cases that simplifies the business logic that is currently required in all upstream extract and reporting applications
- ✓ Provide data versioning such that clients can see the data as it was at the time of the transaction and the data as it is currently
- ✓ Incorporate the business logic into the ETL so that extracts and reporting tools will have a consistent view of the data
- ✓ Give end users ability to do ad-hoc reporting with consistent results

Kimball Lifecycle Approach – High level tasks



Kimball Bus Architecture

Model the Business, not your source data

	COMMON DIMENSIONS						
BUSINESS PROCESSES	Date	Product	Warehouse	Store	Promotion	Customer	Employee
Issue Purchase Orders	X	X	X				
Receive Warehouse Deliveries	X	X	X				X
Warehouse Inventory	X	X	X				
Receive Store Deliveries	X	X	X	X			X
Store Inventory	X	X		X			
Retail Sales	X	X		X	X	X	X
Retail Sales Forecast	X	X		X			
Retail Promotion Tracking	X	X		X	X		
Customer Returns	X	X		X	X	X	X
Returns to Vendor	X	X		X			X
Frequent Shopper Sign-Ups	X			X		X	X

Full coverage of the Kimball Data Warehouse Bus Architecture is available in *The Data Warehouse Toolkit, Third Edition*, including sample bus matrices for 12 industry case studies.

What is a dimension?

It's all about the 'W's and sometimes a H ☺

Who – Usually multiple "Who's" like customer, sales rep, technician, etc

What – Usually a product

Where – Location information, sometimes multiple locations

When – Almost always multiple "when's". Critical component of physical design is determining which business date is the most critical and most frequently used (Retention and Partitioning)

Why – Not as common

How – Not as common

Kimball: dimensional-modeling-techniques/dimensions-for-context

Dimension tables are sometimes called the "soul" of the data warehouse because they contain the entry points and descriptive labels that enable the DW/BI system to be leveraged for business analysis. A disproportionate amount of effort is put into the data governance and development of dimension tables because they are the drivers of the user's BI experience.

“Common” dimension = Conformed Dimension

How are Conformed Dimensions used? Cognos Stitch query example:

https://blogs.perficient.com/2012/05/17/cognos-concepts-stitch-query-in-cognos-reporting-explained

So what exactly does it mean when Cognos performs a *stitch query*? Let’s start with a business scenario and question:

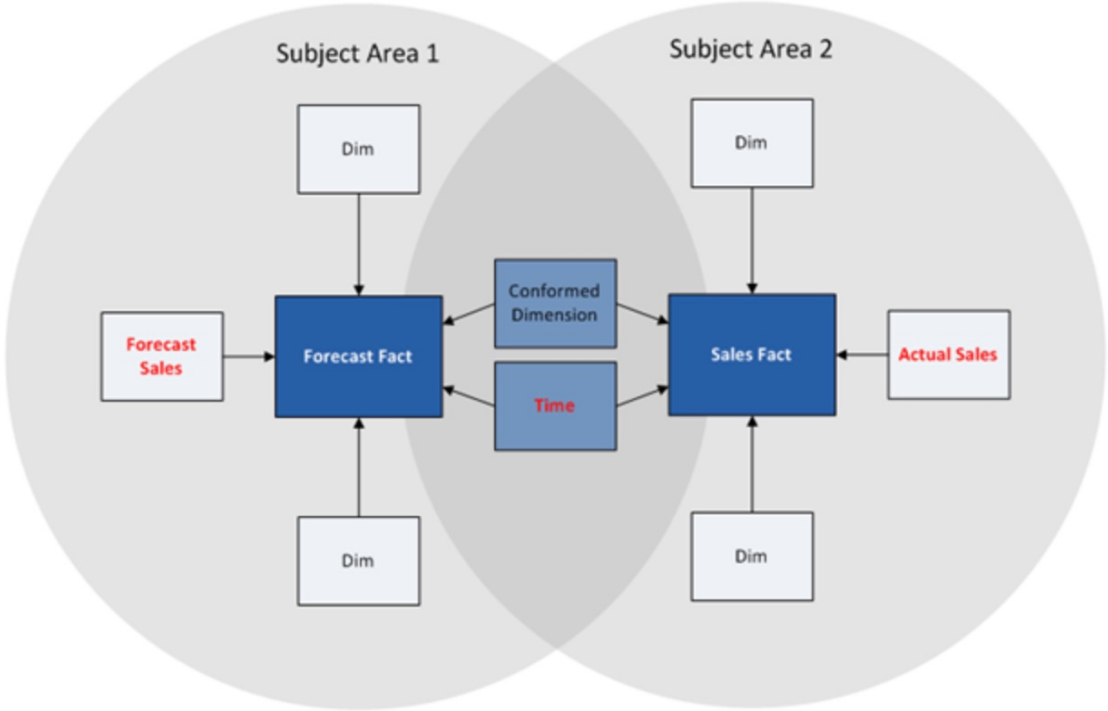
> *A Manager asks a Report Developer to provide a report showing a comparison between Actual Sales and Forecast Sales for any years we have data.*

But these measures come from two completely separate facts: the Forecast Fact and Sales Fact. How would Cognos understand their relation to each other? To achieve this Cognos uses a conformed dimension(s) (dimensions that occur in both facts) as a common data point. Since the Manager has requested the attribute Year as the granularity, we will use the dimension which Year is a part of: Time. The example below shows how the Time dimension for each fact is used to bridge the two subject areas:

“Common” dimension = Conformed Dimension

How are Conformed Dimensions used? Cognos Stitch query example:

https://blogs.perficient.com/2012/05/17/cognos-concepts-stitch-query-in-cognos-reporting-explained



Data Modeling = Start of Development

1. The Data Model should **not** just be documentation, it should be a development effort that, at the very least, generates the complete DDL with full physical design elements. It can also be used for generating DAO (Data Access Objects) and additional information that can be persisted and used for code generation

2. Similarly, any ”Data Governance” tool should not just be documentation, it should provide API’s that allow developers to utilize the information for generating repetitive code

WARNING: Data Modeling Tools and/or Data Governance tools only leveraged for documentation will inevitably become out of sync and their usefulness will degrade over time

Greenplum Partner GAP: Many Data Modeling tools do not support Greenplum as a target for forward engineering which is paramount in achieving bullets 1 and 2 above. Many tools however can be “extended” in it’s forward engineering engine to achieve. This has been done in the field in Erwin, pgModeler and DbSchema.

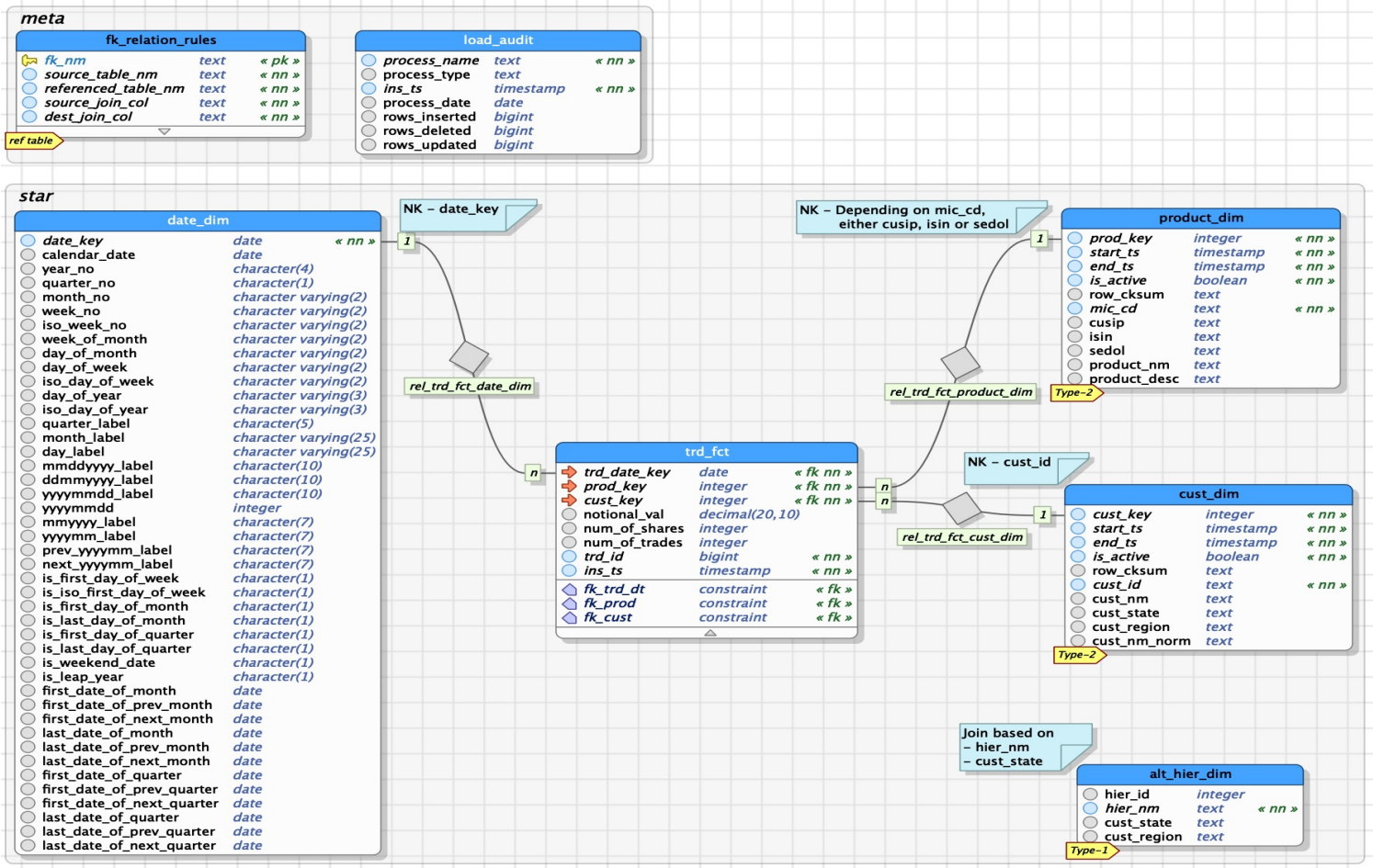
Data Modeling Tool used in example engagement

- Data Modeling tool used is a slightly altered version of an open source Postgres modeling tool called PgModeler (https://pgmodeler.io/)
- The alterations were done in the code that generates the DDL in order to allow the tool to generate DDL appropriate for Greenplum and to also insert FK relationship rows into a metadata table that is used by the attribution engine. If you purchase this PgModeler tool ($26 a year) let me know and I'll send you the files I altered
- Model follows a Kimball based OLAP model very closely and much of the concepts can be found online (https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/)
- All dimensions contain a consistent set of fields that the utility procs described in this presentation expect:
    - <name>_key – This is the primary key (surrogate key) of the table which is an incremental numeric value
    - start_ts - The timestamp when this row was inserted
    - end_ts – The timestamp when this row became inactive
    - is_active – Boolean indicating if the dimension row is the active (current) row or not
    - <name>_crc – This is a md5 checksum of the business fields in the dimension which is used to determine if any data was updated when subsequent loads are performed

Data Modeling used in example engagement

- The physical attributes of the table (AO, compression, partitioning, etc) are in the custom sql section of the model for each table in the Append Sql section. This code will be generated into the DDL right after the generated Create Table
- **Every table and every column in the main model (not Staging) has comments** that indicate the source of the data and a brief description of the table/column. This information is loaded as comments via the generated DDL and is available in the database by querying the v_dict view under <dw> schema. This view essentially returns a data dictionary for the tables
- The model uses “tags” to indicate if the table is a type2 dimension, type2-nohist dimension, junk dimension, fact table or summary table
- A textual note exists above each dimension table to indicate what the natural key is for the dimension
- The model generates the DDL for almost everything created in the <dw> schema. There is a post_ddl.sql script that does create additional objects that are generated from the objects in the model. This includes the dimensions current view table, the dimensions current view and the dimension diff view. The code to generate all these objects are defined in the dimension_util_pkg.sql file

Sample Data Model – Primary reporting tables



Data Reporting Notes

Document the aspects of the model that reporting teams need to understand, Examples include:

- Fact to dimension joins are all inner joins on the fact tables FK attribute. If a dimension key in the fact table is “null” it will be set to -1 which is a standard row in every dimension representing null
- Dimension joins to “alternate” dimensions (Ex. Cust_dim to alter_hier_dim) should be outer joins since every customer doesn’t have an alternate location hierarchy. These joins should also always filter by hier_nm to avoid double counting issues.
- Fact to fact joins (None in this example model) would be outer joins and the joins would join through the conformed dimensions
- Fact to dimension joins are ALWAYS on the fact tables FK attribute regardless of the join wanting the “as-was” vs “as-is” view of the dimension data. The standard _dim table is the ”as-was” view of the data and the _dim_curr table is the “as-is” view of the data but the join to these tables is the same.

Dimension Processing

- ✓ Dimension “ETL” will be slightly different based on the dimension “type”
- ✓ Tools like Informatica have extremely standard design patterns for type-2 dimensional processing (I believe now much of the mapping is generated)
- ✓ This deck describes a Greenplum/Postgres generic UDF that “mimics” the same design pattern for dimension processing (Type-2 and Type-1)

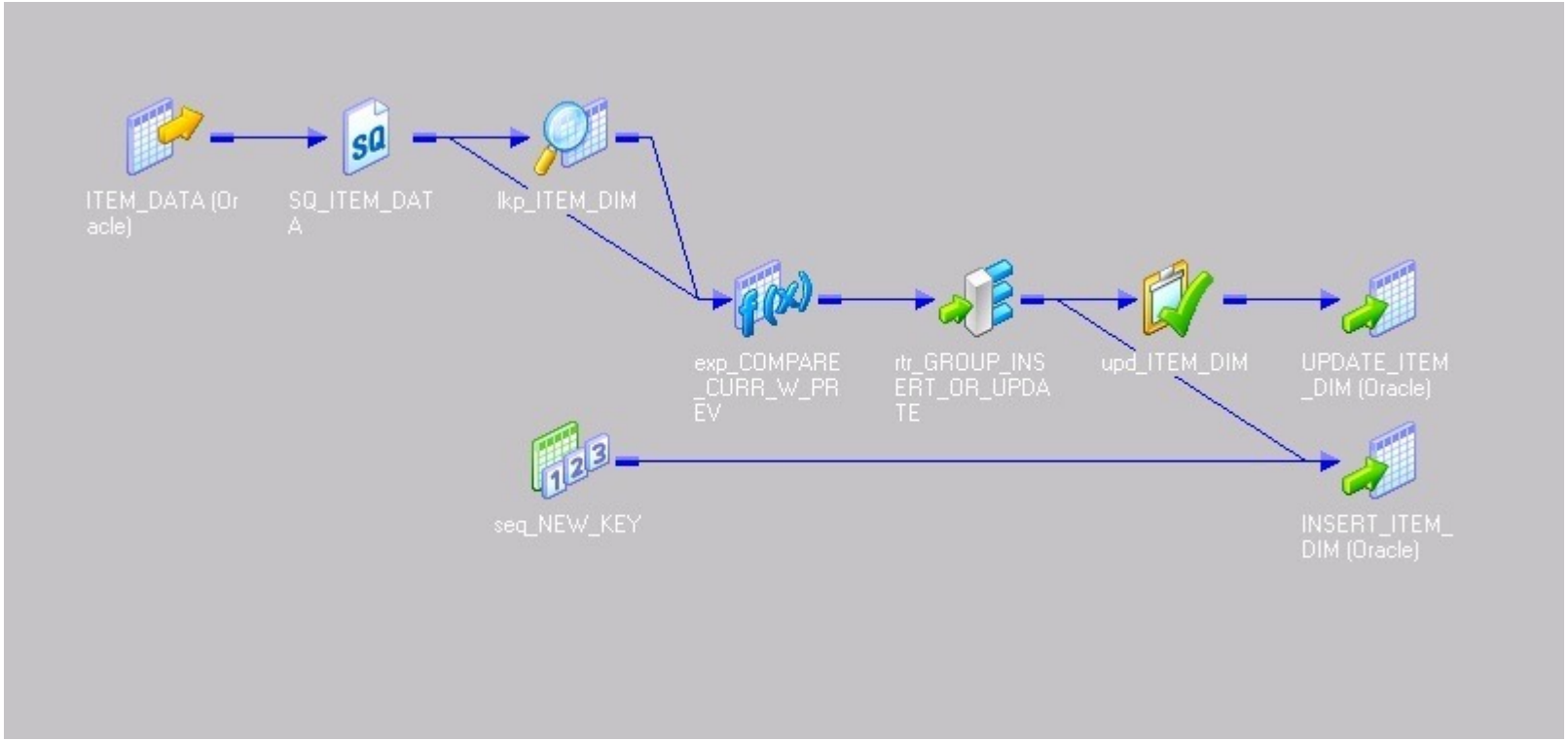
Dimension Processing

Aspects of different approaches

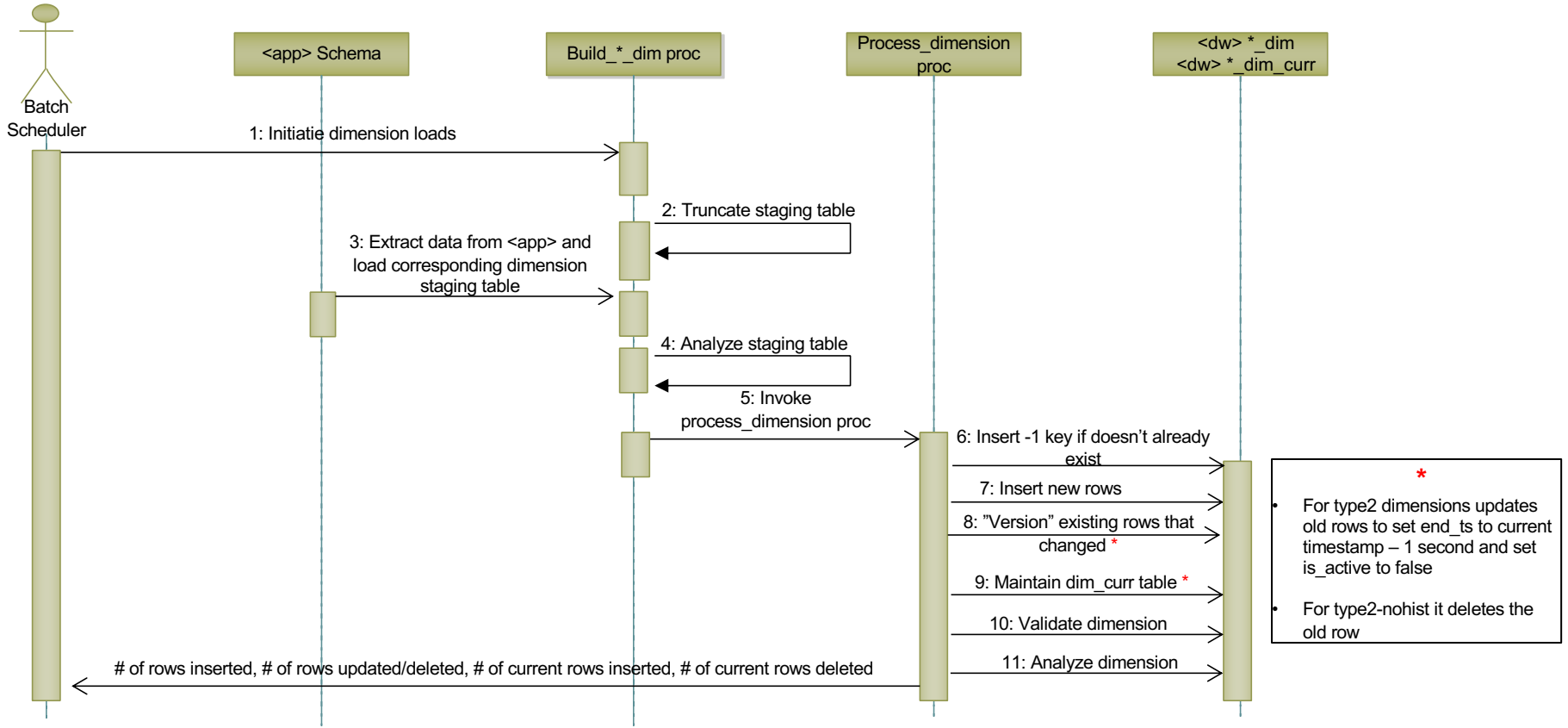
Informatica	UDF
Visual Interface	Procedural Dynamic SQL (procedural code)
1 Mapping per Dimension (little if/any reuse)	Generic procedure that works with any dimension
Potentially “cheep” ETL dev team that can scale (can throw bodies at the effort)	Solid knowledge of SQL and pg/plsql required for 1 or 2 developers maintaining the UDF
Initial development might be quick assuming Informatica generates the mappings	No new development for the dimensional processing logic
Very Tedious maintenance if/when initial mappings need to change (ie. Add new columns)	No logic changes required to add new columns, everything is dynamic based on the contents of the dimension table being loaded and the staging table being loaded from
Can be slow depending on the size of the dimension (million row lookup table will spill to disk and be very slow)	Very performant since everything is “set-based” processing leveraging the MPP nature of GPDB

Dimension Processing in Informatica example

(This is conceptually what the process_dimension proc is doing)



High level dimension processing (UDF based)



Cust_dim Dimension Processing – Seq 1-5

- All dimensions have a stored procedure with pattern build_<dimension name>. This is the proc you will invoke.
- The build proc performs the following:
    - Truncates the staging table. Pattern of all table names is the dimension name with the suffix _dim replaced with _stg
    - Queries the source <app> tables to get the data and populate the staging table. In this example all the dimensions are built to do a full pull from the source tables (instead of relying on any kind of maintenance date)
    - Analyze the staging table
    - Invoke the process_dimension utility proc passing the parameters:

        Dimension name

        Staging table name

        An array containing the list of natural keys (as documented in the data model)

        The dimensions crc column

        The dimensions primary key

Ex: select star.process_dimension('star.cust_dim','star.cust_stg','{cust_id}','row_cksum','cust_key');

Cust_dim Dimension Processing – Seq 6

- The process_dimension utility proc performs the following:
  - Validate that the dimension has a single column PK which is a requirement for this proc. In this case the ”PK” is cust_key
  - Query the information_schema.columns table for all not null columns in the dimension table and generate the insert of the “-1” (null value) row. This row will be inserted if it doesn’t already exist:

Null Key Insert Example:

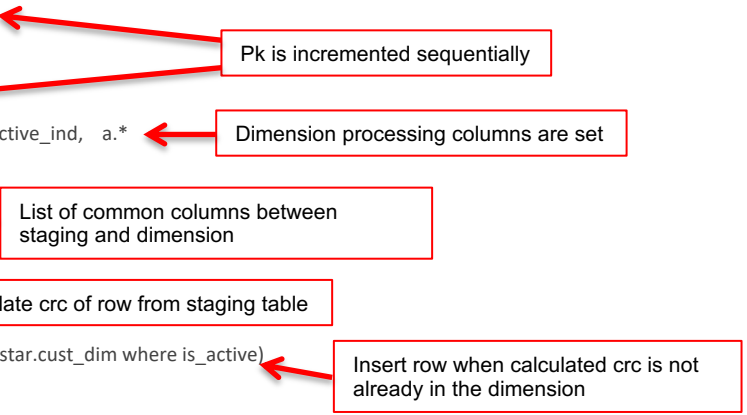
```
insert into star.cust_dim
(cust_key, start_ts, end_ts, is_active, cust_id)
select
   '-1'::int4,
   '1900-01-01 00:00:00'::timestamp,
   '9999-12-31 23:59:59'::timestamp,
   true::bool,
   '-1’::text
where not exists (select 1 from star.cust_dim where cust_key = -1)
```

Cust_dim Dimension Processing – Seq 7

- Query the information_schema.columns table for the list of columns that are common (by column name) between the staging table and dimension table. This list of columns is used to generate the remaining dml
- Generate an insert statement to insert any new rows from the staging table that don't already exist in the dimension or have changed in some way (determined by comparing the crc column in the dimension against the calculated crc of the row from the staging table):

Dimension Insert:

```
insert into star.cust_dim (cust_key, start_ts, end_ts, is_active, cust_id,cust_nm,cust_state,cust_region,cust_nm_norm,row_cksum)
with seq as (    select coalesce(max(cust_key),1) as key from star.cust_dim)
select
(seq.key::bigint + row_number() over()) as cust_key,
'2023-03-23 18:11:51' as start_ts, '9999-12-31 23:59:59' as end_ts,    true as active_ind,    a.*
from (
   select cust_id,cust_nm,cust_state,cust_region,cust_nm_norm,
          md5(textin(record_out(s.*))) as row_cksum
   from star.cust_stg s
) a, seq where a.row_cksum not in (    select coalesce(row_cksum ,'-1')   from star.cust_dim where is_active)
```



Cust_dim Dimension Processing – Seq 8

- If the type of dimension is type2 (maintain history) then generate an update statement to update any existing rows with the same natural key as the row just inserted. To version out a row we set the end_ts to the current timestamp – 1 second and set the is_active flag to false:

Dimension Update:

```
update star.cust_dim a
set end_ts = to_timestamp('2023-03-23 18:11:51','YYYY-MM-DD HH24:MI:SS') - interval '1 second',
    is_active = false
from (
  select cust_id, md5(textin(record_out(s1.*))) as crc
  from star.cust_stg s1
) s
where coalesce(a.cust_id,'-1') = coalesce(s.cust_id,'-1')  and a.is_active and a.row_cksum != s.crc
```

Subselect gets the natural key(s) and the calculate crc column for the staging table row

Where natural keys match

Dimension row is the currently active one

Dimension row crc doesn't match the calculated staging crc (The row we just inserted would be active and match the staging crc)

Dimension Processing – Seq 9

- If the type of dimension is type-2 (maintain history) then generate code to maintain the corresponding "current" table. Naming pattern for the current table is <dimension name>_curr:

Dimension Current Delete:

```
delete from star.cust_dim_curr a
using (
   select cust_key,cust_id
   from star.cust_dim
   where start_ts = '2023-03-23 18:11:51'
) s where s.cust_key is not null and coalesce(a.cust_id,'-1') = coalesce(s.cust_id,'-1')
```

Subselect gets the PK and natural key(s)

Get the dimension rows we've just inserted. We can determine this by the start_ts of the rows

Delete where natural keys match

Dimension Current Insert:

```
insert into star.cust_dim_curr (cust_key,start_ts,end_ts,is_active,cust_id,cust_nm,cust_state,cust_region,cust_nm_norm,row_cksum)
select cust_key,start_ts,end_ts,is_active,cust_id,cust_nm,cust_state,cust_region,cust_nm_norm,row_cksum
from star.v_cust_dim_curr_wrk
where cust_key not in (select cust_key from star.cust_dim_curr)
```

List of common columns between dimension current table and dimension

The _wrk view is generated during DDL object creation (next slide will show what this looks like)

Insert rows that don't already exist in dimension current table

Dimension Processing – Seq 9

- The proc performs a sanity check that the number of rows inserted into the _curr table is >= the number of rows that were added to the dimension table during this invocation. If it's not the proc will raise an error to review the _wrk view.
- The v_<dimension_name>_curr_wrk view is generated by the dimension util proc called build_dimension_current_view. These views are created during DDL object creation.

View definition:

```
SELECT
    d1.cust_key,
    d2.start_ts, d2.end_ts, d2.is_active,
    d2.cust_nm, d2.cust_state, d2.cust_region,
    d2.cust_nm_norm, d2.row_cksum
FROM star.cust_dim d1, star.cust_dim d2
WHERE
    d2.is_active
    AND
    COALESCE(d1.cust_id, (-1)::text) = COALESCE(d2.cust_id, (-1)::text);
```

Primary key of dimension

Values of columns from dimension row that is currently the active version

Self join of dimension

D2 is the active row (current row)

Join on natural key(s)

Dimension Processing – Seq 10

- As part of the process_dimension proc and whole transaction are validations to confirm that the dimension processing has left the dimension in a valid state. The following queries are performed to validate the dimension

Dimension END_TS/START_TS Recon:

```
select count(*)   from star.cust_dim  where end_ts < start_ts
```

This should only happen if duplicates by natural key exist in the staging table

Dimension NK Recon:

```
select count(*) from (
        select cust_id, count(*)
        from star.cust_dim
        where is_active and cust_id is not null
        group by cust_id  having count(*) > 1 ) a
```

Looks for any duplicates by natural key. This would cause issues during the fact table load when the fact rows join to dimension to get the key (would cause duplicates in fact table and double counting

Dimension PK Recon:

```
select count(*) from (
        select cust_key, count(*)
        from star.cust_dim
        group by cust_key  having count(*) > 1 ) a
```

Looks for any duplicates by primary key. Since some of the dimensions are AO tables we can't have GPDB enforce the uniqueness so this will check that the primary key is unique

Additional Dimension Utilities

- ✓ The dimension_util_pkg contains other utility procs that can be utilized to enrich the usage of the dimensions or are used by the main process_dimension process described in the preceding slides
- ✓ The next set of slides will describe these additional utility procs

Additional Dimension Utility Procs

- Build_dimension_objects is a wrapper function that invokes all the dimension utility procs described on the following slides. After invoking this proc you'll have generated the following objects in the database:
  - <Dimension name>_stg: Staging table
  - <Dimension_name>_curr: Dimension Current table
  - v_<Dimension_name>_curr_wrk: Dimension Current Work view
  - v_<Dimension_name>_diff: Dimension diff view
  - v_<Dimension_name>_what_changed: Dimension "what changed" view

Additional Dimension Utility Procs

- Build_dimension_current_view generates DDL to create a view called v_<dimension name>_curr_wrk. This view is used by internal processing to maintain the dimension current table. The view returns the currently active data on the dimension over all the available primary keys of that dimension. View is used in Dimension sequence step 9 and sql is explained in that slide.

- Gen_dimension_asof_obj generates DDL to create a view (named v_<dimension name>_asof_<user specified suffix>) or temp table (named tmp_<dimension name>_asof_<user specified suffix>). This object is used to show the dimensional values as they looked at a specific point in time (as-of view of the data).

Additional Dimension Utility Procs

- Build_dimension_current_table generates DDL to create a table called <dimension name>_curr. This table can be used by reports/extracts that want to retrieve the view of the dimension data that is currently active (the dimension row where is_active = true). Joins to this table are identical to joins against the normal “as-was” dimension so you would still join by FK off the fact table inner joined to primary key on the dimension. Examples of this table being built can be found in the preceding sequence diagram Seq 9 slides.
    - Table is created with same columns as it’s corresponding dimension
    - Same physical storage parameters (distribution, with-clause)
    - Same defaults
    - Same constraints
    - Same permissions
    - Same indexes

Additional Dimension Utility Procs

- Build_dimension_staging_table generates DDL to create a table called <dimension name>_stg. This table is the staging table for the dimension. It has all the same columns of the dimension table with the following exceptions:
    - Table doesn’t contain the dimension surrogate key
    - Table doesn’t have the dimensional processing columns (start_ts, end_ts, is_active, row_cksum)
    - Same distribution for MPP databases
    - Created as Heap table regardless of the physical design of the dimension

Additional Dimension Utility Procs

- Build_dimension_diff_view generates DDL to create 2 views called v_<dimension name>_diff and v_<dimension name>_what_changed. These views can be used to report current/previous values for all dimensional fields through the version chain and provide insight on the columns that are changing and resulting in versioning of dimensional records.

- Example of diff view:
    - Update to customer staging table changed the customer name and customer state for one of our customers:

        update cust_stg set cust_nm = 'Mrs. Stacy Jones Smith', cust_state = 'NJ' where cust_id = '1965C';

    - Process_dimension was executed with this updated record creating a versioned record in the dimension:

    - Next slide shows what the v_cust_dim_diff view results look like for one of the office rows that received these updates

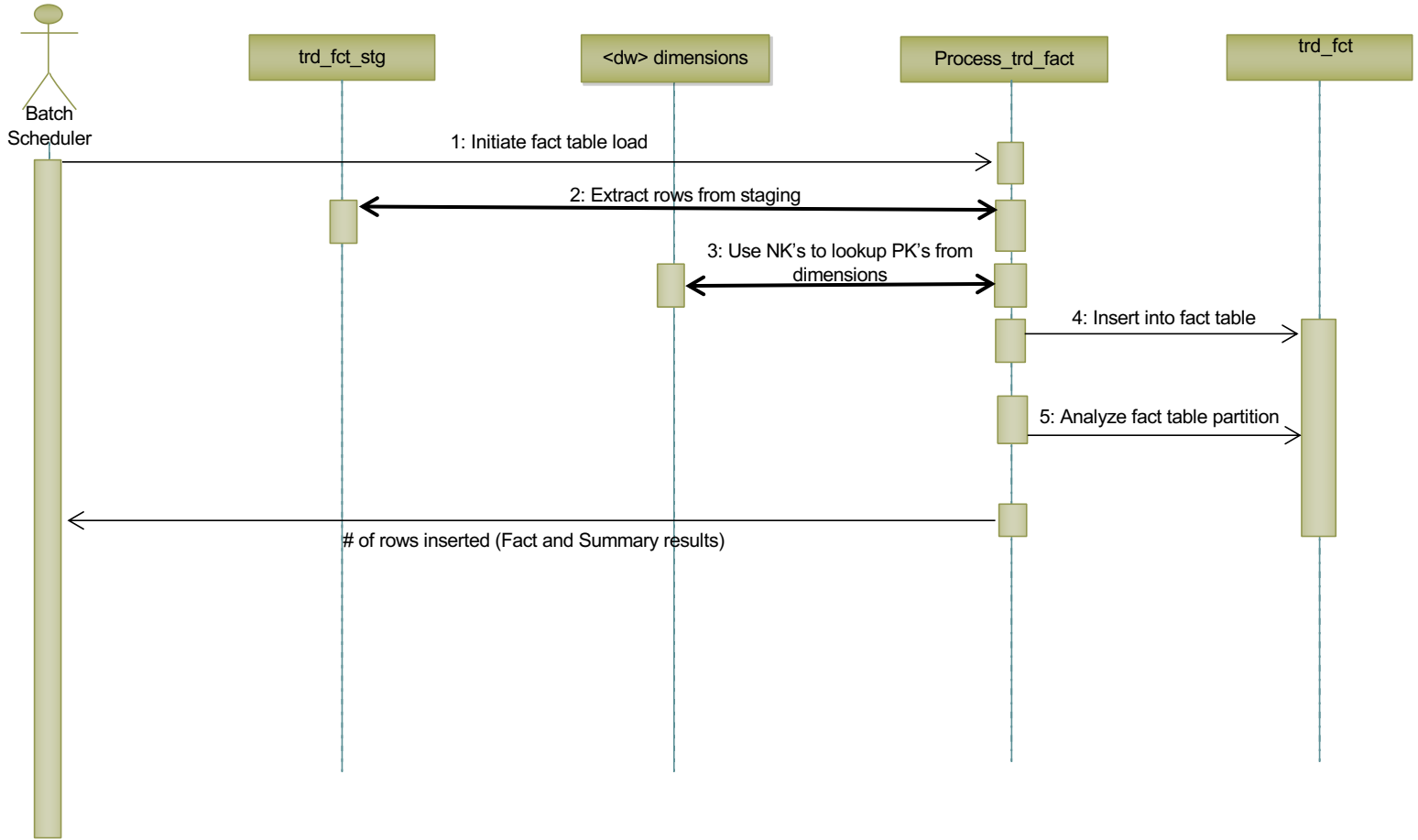
Additional Dimension Utility Procs

```
star_demo=# select * from star.v_cust_dim_what_changed order by 1;
-[ RECORD 1 ]---+--------------------
cust_id         | 1965C
start_ts        | 2024-07-01 14:43:13
changed_columns | cust_nm, cust_state

star_demo=#
star_demo=# select * from star.v_cust_dim_diff where cust_id = '1965C' order by 1;
-[ RECORD 1 ]-----+----------------------
vers_order        | 1
curr_start_ts     | 2024-07-01 14:43:13
prev_start_ts     | 2024-07-01 14:43:10
cust_id           | 1965C
curr_cust_nm      | Mrs. Stacy Jones Smith
prev_cust_nm      | Ms. Stacy Jones
curr_cust_state   | NJ
prev_cust_state   | NY
curr_cust_region  | NE
prev_cust_region  | NE
curr_cust_nm_norm |
prev_cust_nm_norm |
```

Fact Table Processing

High level fact processing



Fact Table Recon

- Recon_fact_table is a proc that is called after the load of the fact table staging and fact table. The recon proc should perform the sanity checks of the data to ensure your data load was clean. Examples include:

    1. Confirm we have no duplicates by trd_id (the NK of the trd_fct table). This would happen if, for some reason, a lookup to one of the dimensions matched more than 1 row in the dimension.
    2. Confirm that all the trd_ids that were pulled into staging were loaded into the fact table. A trd_id may not load if one or more of the lookups to dimensions that are processed as a inner join didn't work (dimension value NK from staging area doesn't exist in dimension)
    3. In cases where the dimension values shouldn't be null, confirm none of the FK's in the fact table are "null" where "null" is defined as the FK having the -1 key

- Results of the recons (success or fail) are stored in a table to make replay of the records easier

Fact Table Recon Reports

- When possible, compare results of summary “reports” between the warehouse and existing application reports. As an example, the following 2 summary reports can be compared:
    1. Compare report that summarizes number of shares by cust_region
    2. Compare report that summarized number of shares by prod_desc
- If either of the results don’t match exactly between the application query and warehouse query result then the procedure will raise an exception and provide the query that showed the difference.

Demo

- Model/DDL Creation with physical design incorporated
  - Show model in PGModeler
  - Generate DDL from PGModeler and save to file
  - Create schema from generated DDL model
  - Show post_ddl processing that creates additional objects derived from the objects in the model

- Dimension processing
  - Populate initial staging tables
  - Run process_dimension proc
  - Run process_dimension proc a second time with no changes to staging
  - Update some values in the staging table
  - Run process_dimension proc a third time
  - Show resulting dimension and dimension current tables
  - Show dimension diff view

- Reporting examples
  - Populate example data in dimension and fact tables
  - Show report using both the dimension (as-was) and dimension current (as-is) tables to show the difference
  - Show example of using an alternate hierarchy based on some user defined hierarchy definition
  - Show screenshots of Tableau reporting tool model and same reports

Tableau – Data Source Model

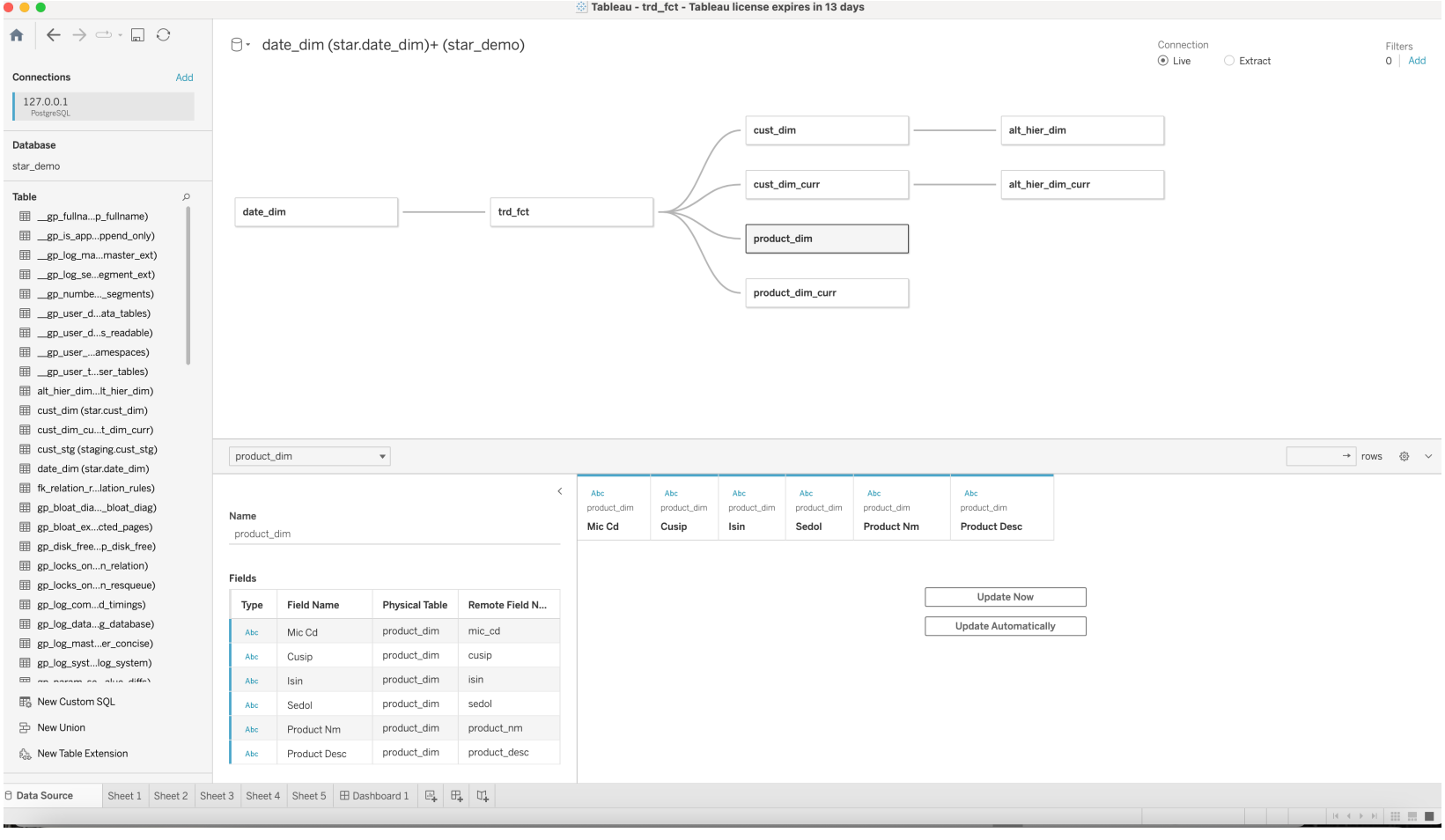
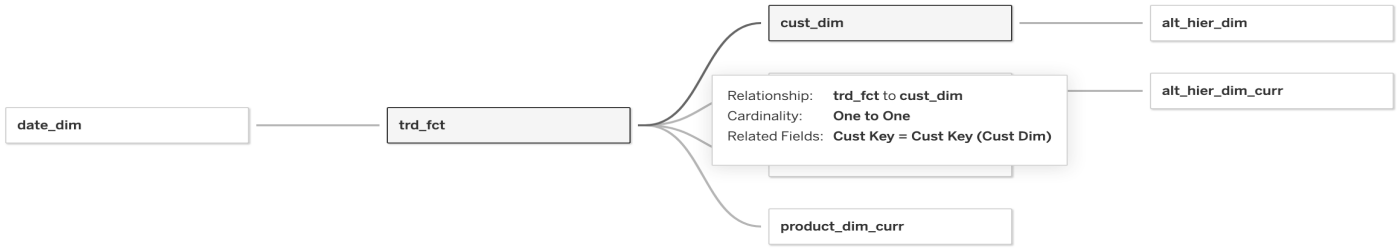


Tableau – Data Source Model



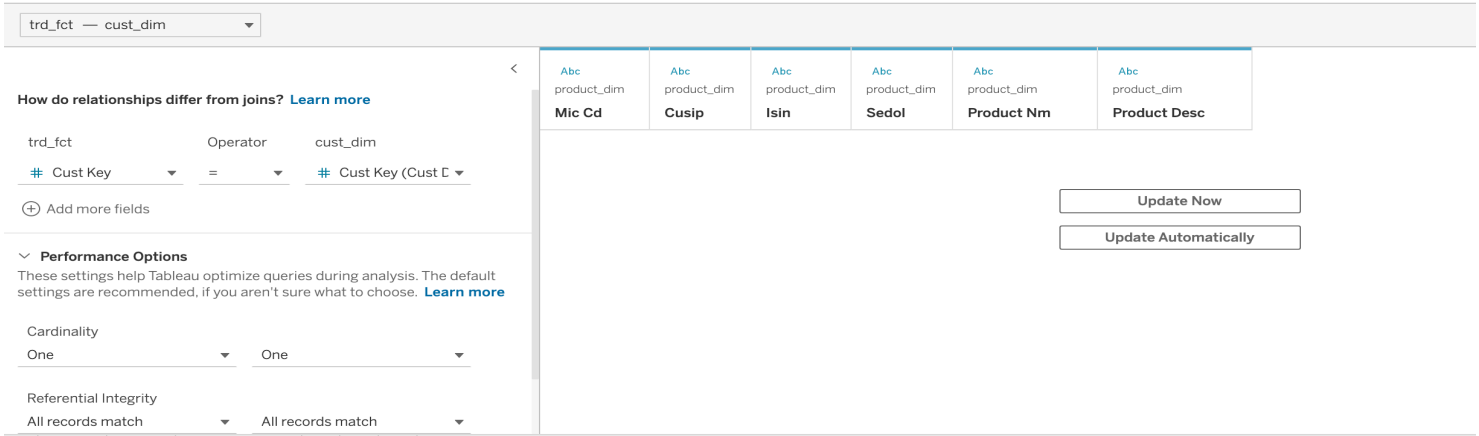


Tableau – Data Source Model

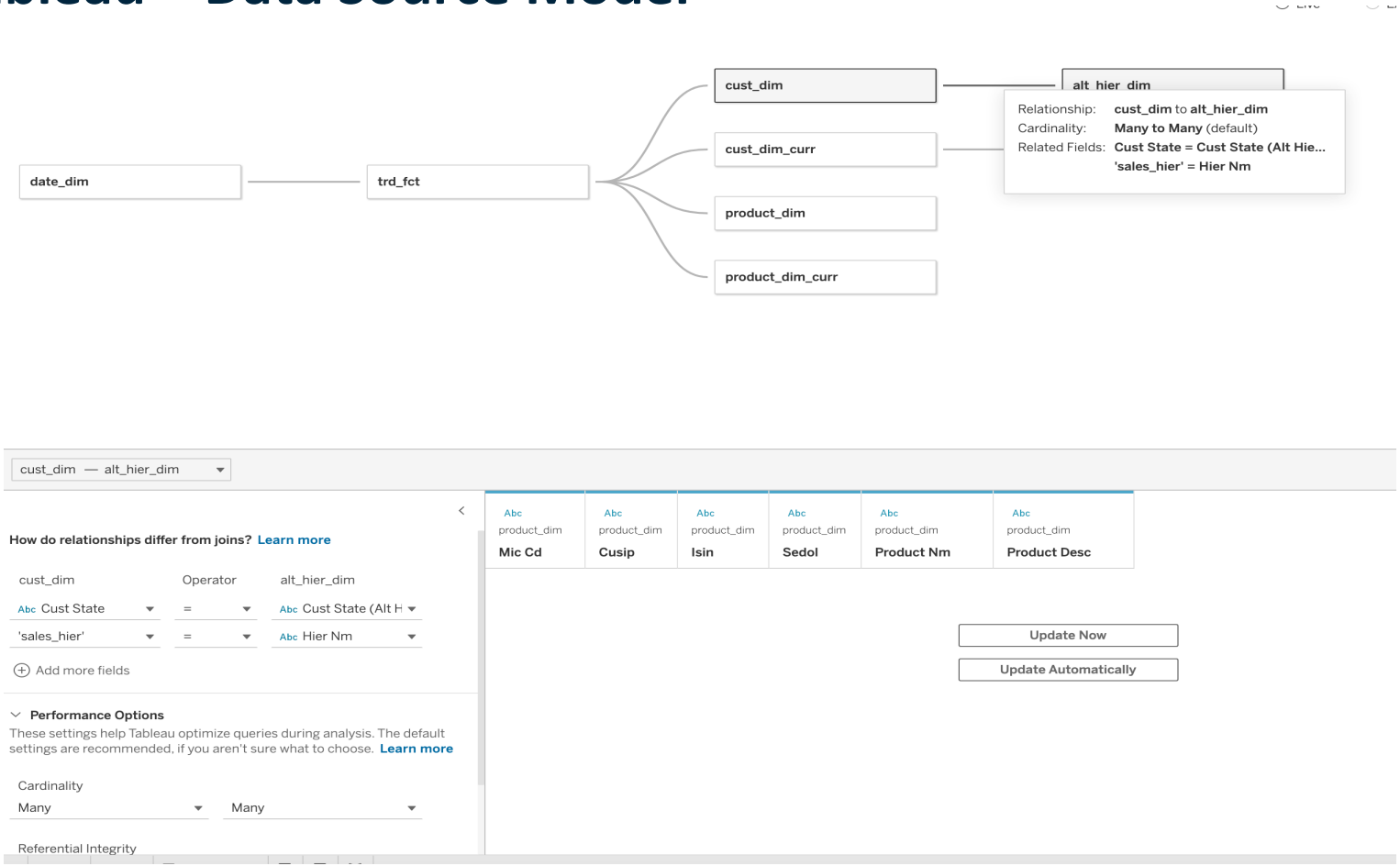


Tableau – Reports

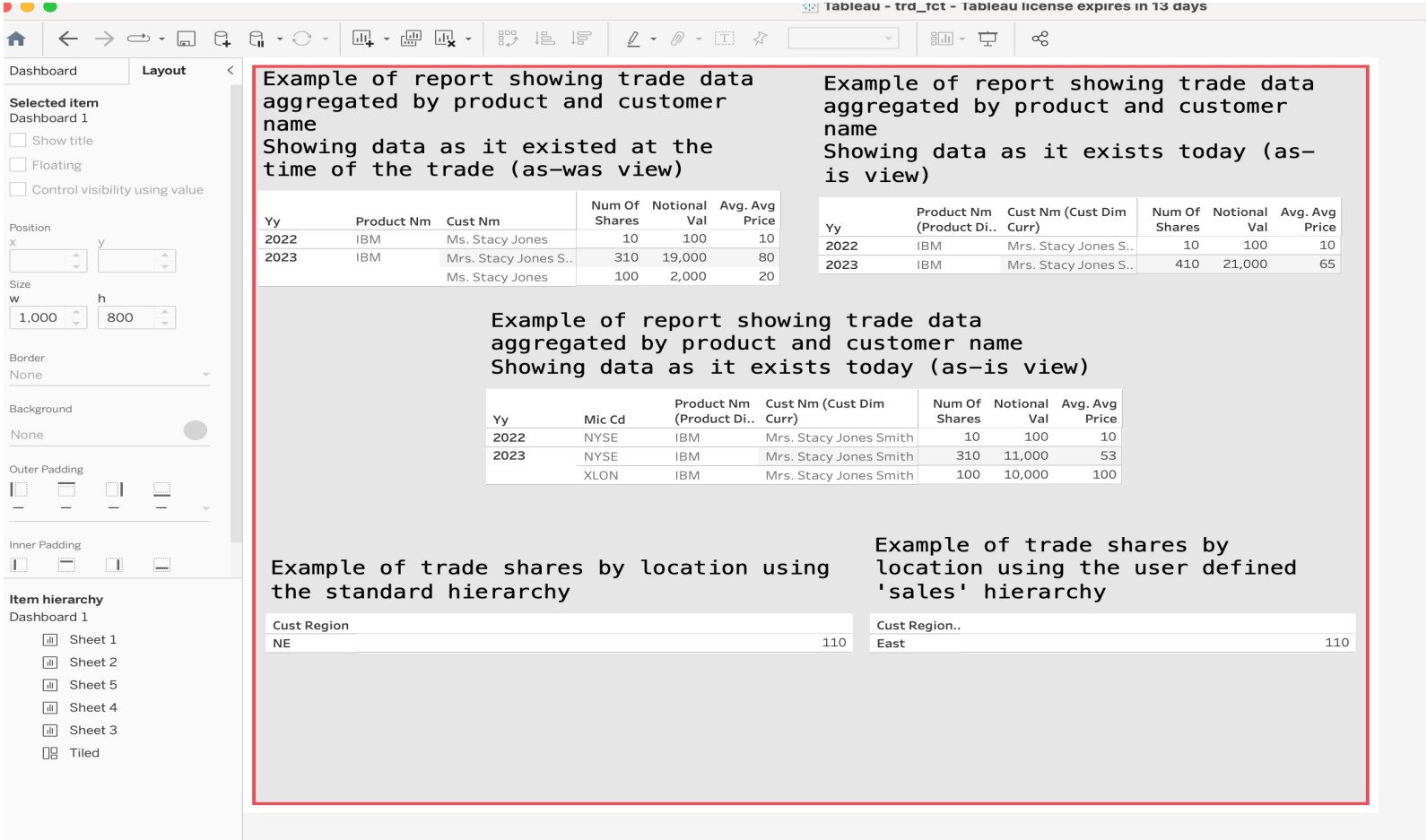
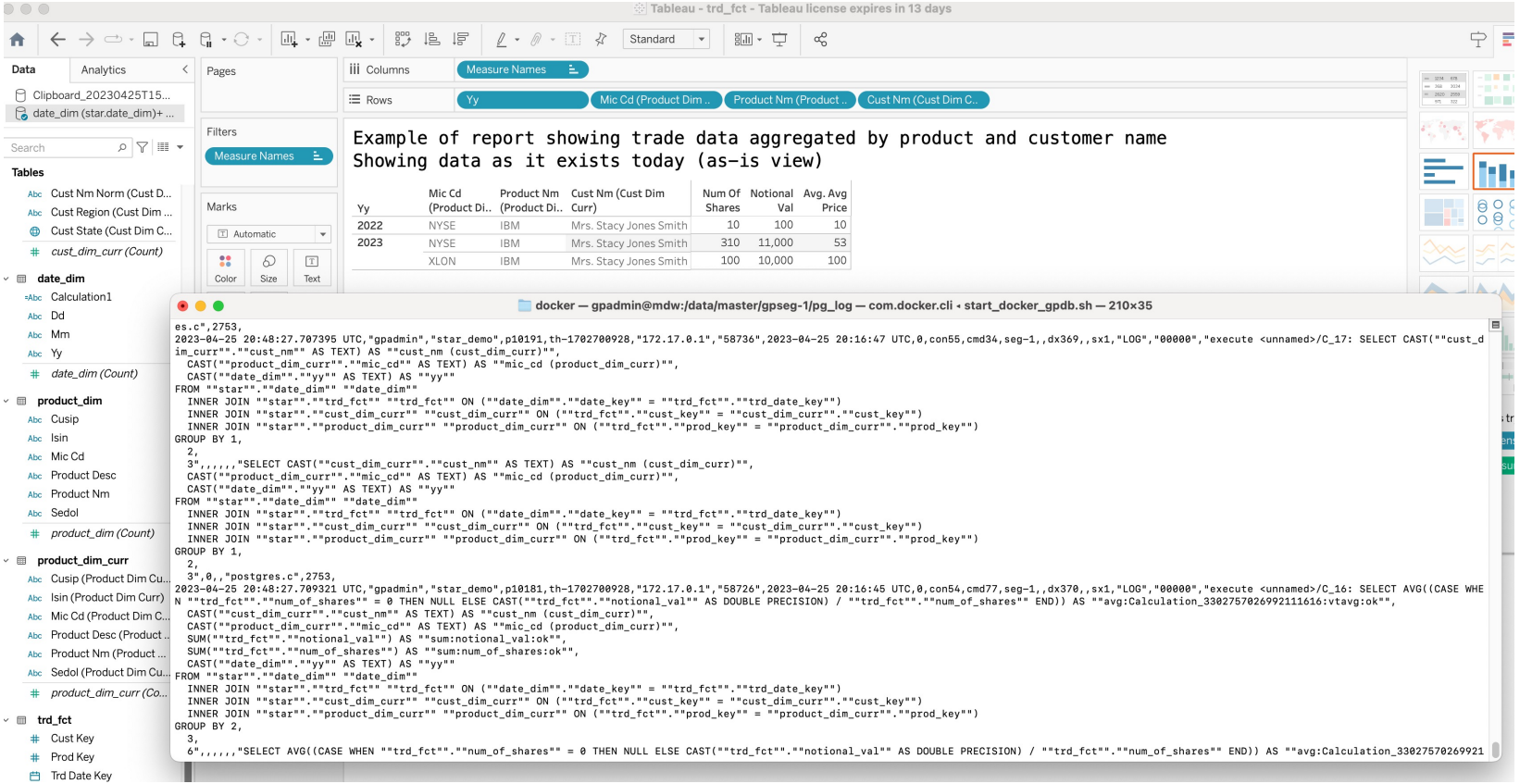


Tableau – Reports (Generated SQL in GPDB)




Thank
You