A Modern Approach to Dimensional Modelling – In a Columnar Database

Truls Bergersen
Data + AI Summit by Databricks
San Francisco, June 13th 2024

Okeanós
TITANS OF DATA

About the presenter

- Truls Bergersen
- Lives in Oslo, Norway
- 23 years of experience in data warehousing and business intelligence
- Data modeling and data integration expert
- Background from row based relational databases
- Working with Azure Databricks for 1.5 years
- Founder of start-up company Okeanos AS
- (Contracted as) Lead architect of the Norwegian Tax Administration's data warehouse

Okeanós
TITANS OF DATA

Disclaimer

- This presentation is a compilation of my personal thoughts on the future of dimensional modelling.
- Examples are simplified.
- There is no silver bullet, so one method will not fit all purposes.
- 40 minutes is only enough to scratch the surface of this topic.

Agenda

1. Recap of dimensional modelling
2. OBTs
3. The way forward using star schemas

Intro

- Dimensional modelling originates from a joint research project conducted by General Mills and Dartmouth University in the 1960s.[1]
- Used in the 1970s by both AC Nielsen and IRI. [1]
- In 1996 the book *The Data Warehouse Toolkit by* Ralph Kimball is published.

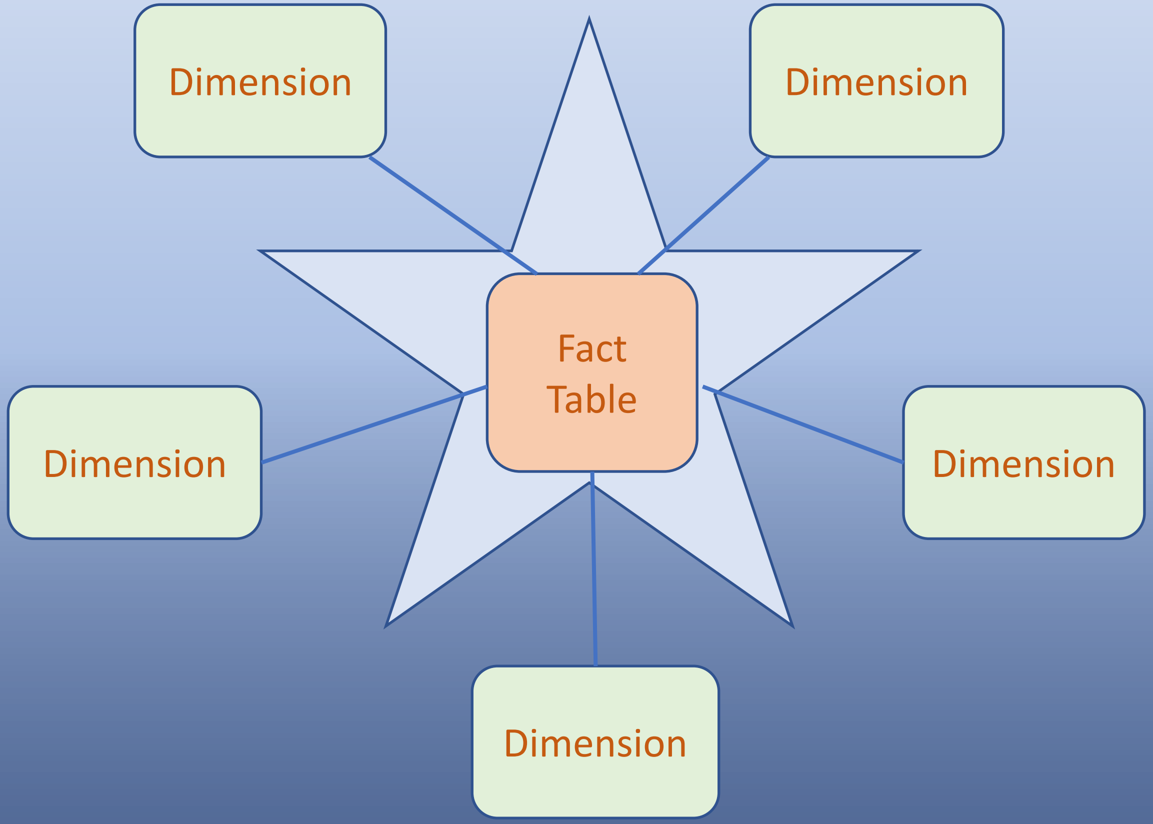
[1] The Data Warehouse Toolkit, 3rd edition, p15.

Star Schemas

Star schemas are implementations of dimensional models in a relational database.

They consist of:

- Fact tables
- Dimension tables



Fact Tables

- Store the performance measures
  - I.e. aggregable numbers such as
    - Quantity
    - Amount
- Reference to dimension tables via foreign keys
- Usually contains many rows and few columns
- Four types of fact tables:
  - Transactional
  - Accumulative snapshot
  - Periodic snapshot
  - Hybrid

FACT

Measure #1
Measure #2
Measure #n

Foreign key to Dimension 1
Foreign key to Dimension 2
Foreign key to Dimension n

Dimension tables

- Descriptive data giving context to the «facts».
- Has a primary key that is linked to from the foreign keys of the fact table.
- Usually contains few rows and many columns.
- May or may not contain history: 8 types of «Slowly Changing Dimensions».
- Some special dimension types, such as:
  - Junk
  - Degenerate
  - Outrigger

DIMENSION

Primary key
Business key

Attribute 1
Attribute 2
Attribute n

Optionally columns to handle history

Star Schemas

A very simple star schema might look something like this:

Star Schemas

```
CREATE TABLE dim_date (
 PK    bigint not null
,date_ date   not null
);
ALTER TABLE dim_date ADD CONSTRAINT dim_date_pk PRIMARY KEY(PK);

CREATE TABLE dim_customer (
 PK               bigint not null
,customer_no      bigint not null
,customer_name    string
,customer_address string
);
ALTER TABLE dim_customer ADD CONSTRAINT dim_customer_pk PRIMARY KEY(PK);

CREATE TABLE dim_organization (
 PK            bigint not null
,employee_no   bigint not null
,employee_name string
,department    string
);
ALTER TABLE dim_organization ADD CONSTRAINT dim_organization_pk PRIMARY
KEY(PK);
```

```
CREATE TABLE fak_orders (
 quantiy             bigint not null
,amount              double not null
,order_no            bigint not null
,dim_date_fk         bigint not null
,dim_customer_fk     bigint not null
,dim_organization_fk bigint not null
);

ALTER TABLE fak_orders ADD CONSTRAINT
fak_orders_dim_date_fk FOREIGN KEY(dim_date_fk)
REFERENCES dim_date;

ALTER TABLE fak_orders ADD CONSTRAINT
fak_orders_dim_customer_fk
FOREIGN KEY(dim_customer_fk) REFERENCES dim_customer;

ALTER TABLE fak_orders ADD CONSTRAINT
fak_orders_dim_organization_fk
FOREIGN KEY(dim_organization_fk) REFERENCES
dim_organization;
```

Slowly Changing Dimension Type 0 and 1

- Type 0:
  - No history
  - and no attributes are updated
    even if the values change in the source.

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Never updated

- Type 1:
  - No history
  - Attribute *are* updated
    if the values change in the source.
  - So always the current values

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Can be updated

Slowly Changing Dimension Type 2

- Stores history
- A new row per change in the source
- One row represents a time period
- From Date (Effective Date)
- To Date (Expiration Date)

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Never updated – new row is added when new value
From Date	Never updated
To Date	Updated when a new row is added for the same NK
Current Row Flag	Updated when a new row is added for the same NK

Slowly Changing Dimension Type 3

- Stores some history
- However only one row per natural key
- The history is kept in dedicated columns usually containing the previous or original value

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Can be updated
Attribute 1..n Historic Value	Can be updated
Attribute 1..n Effective Date	Can be updated

Slowly Changing Dimension 4

- Like a type 1, 2 or 3, but split in two:
  - One with the attributes that do not change very often
  - One with the attributes that change frequently
  - The fact table has two foreign keys

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Rarely updated

Fact table

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Can be updated – frequently

Slowly Changing Dimension Type 5

- Hybrid/combination of 1 and 4
- The extra Type 1 is modelled as an outrigger from the main dimension.

Fact table

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Rarely updated
FK to outrigger	Updated frequently

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Updated – frequently

Slowly Changing Dimension Type 6

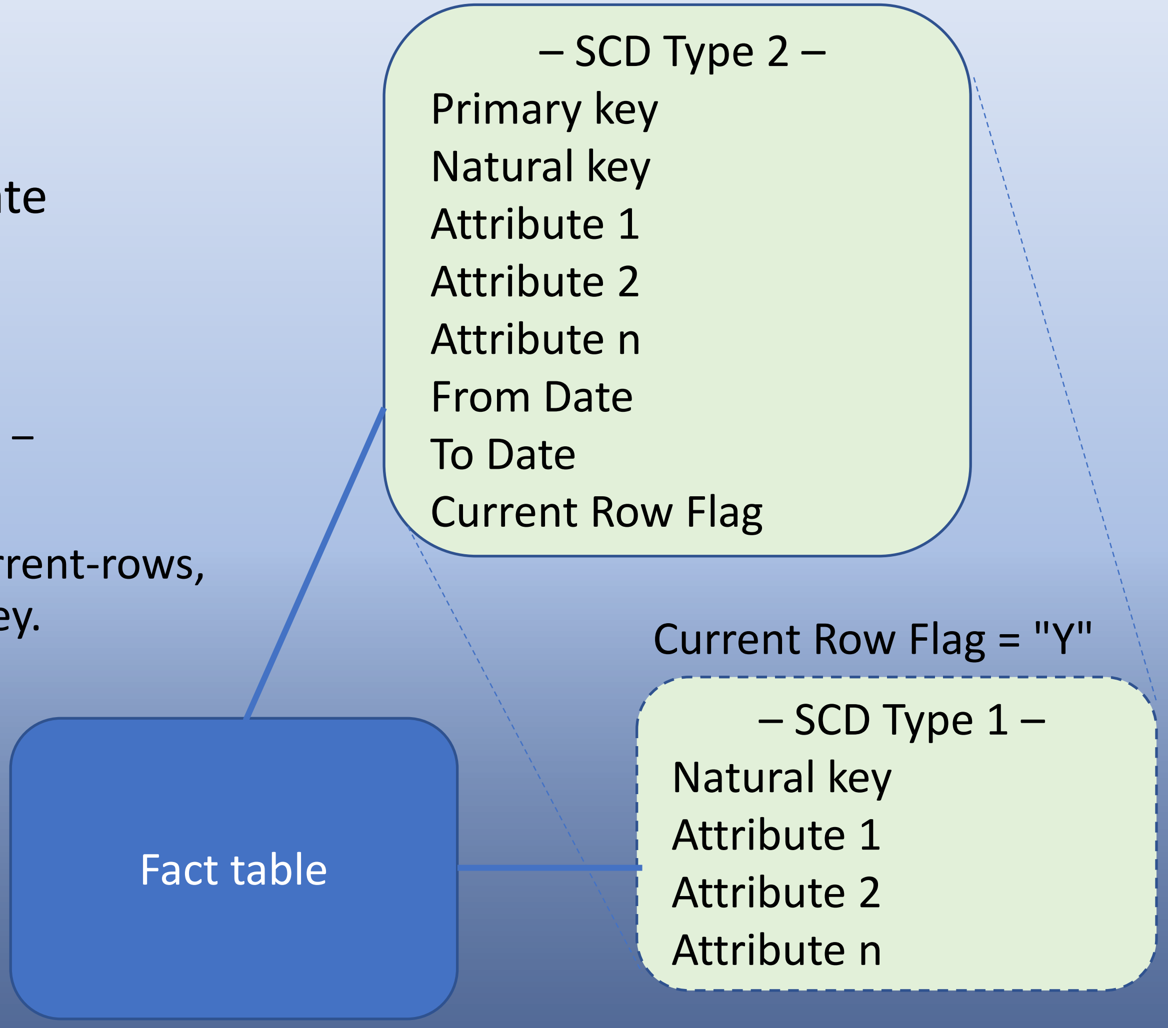
- Hybrid/combination of 1, 2 and 3
- Modelled as a Type 2, with one or more extra columns that is updated with the current value in all (historic) rows.

Fact table

Primary key	Never updated
Natural key	Never updated
Attribute 1..n	Never updated – new row is added when new value
Attribute 1..n Current value	Updated when a new row is added for the same NK
From Date	Never updated
To Date	Updated when a new row is added for the same NK
Current Row Flag	Updated when a new row is added for the same NK

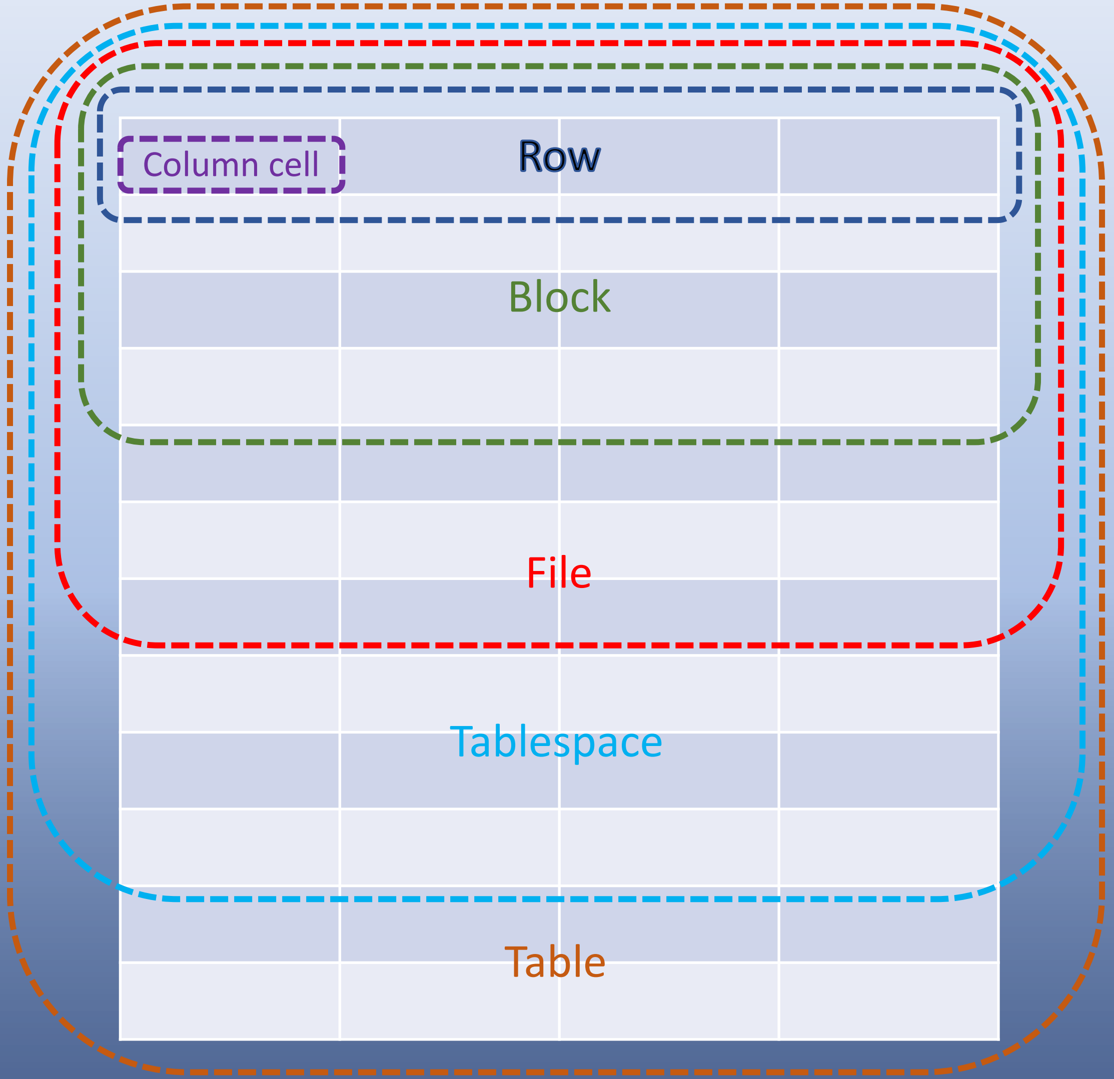
Slowly Changing Dimension Type 7

- Like Type 6, but:
  - The current-values are treated as a separate dimension, but having two foreign keys to the same dimension in the fact table:
    - One to the SCD Type 2 dimension, and
    - One to an SCD Type 1-version of the dimension –
    - either by a separate physical table
    - or though a view that returns only the current-rows, and using the natural key as the foreign key.



Star Schemas in Row-based Databases

- Query performance comes down to the number of i/o read and processed.
- A query is usually limited to a small number of the total columns available.
- In a row-based database the whole row must be read, even if you only query one of the columns
- **=> The i/o of the whole block that these rows are stored must be read.***
- If your query have to access all the rows, then the total bytes of the *whole* table must be read.



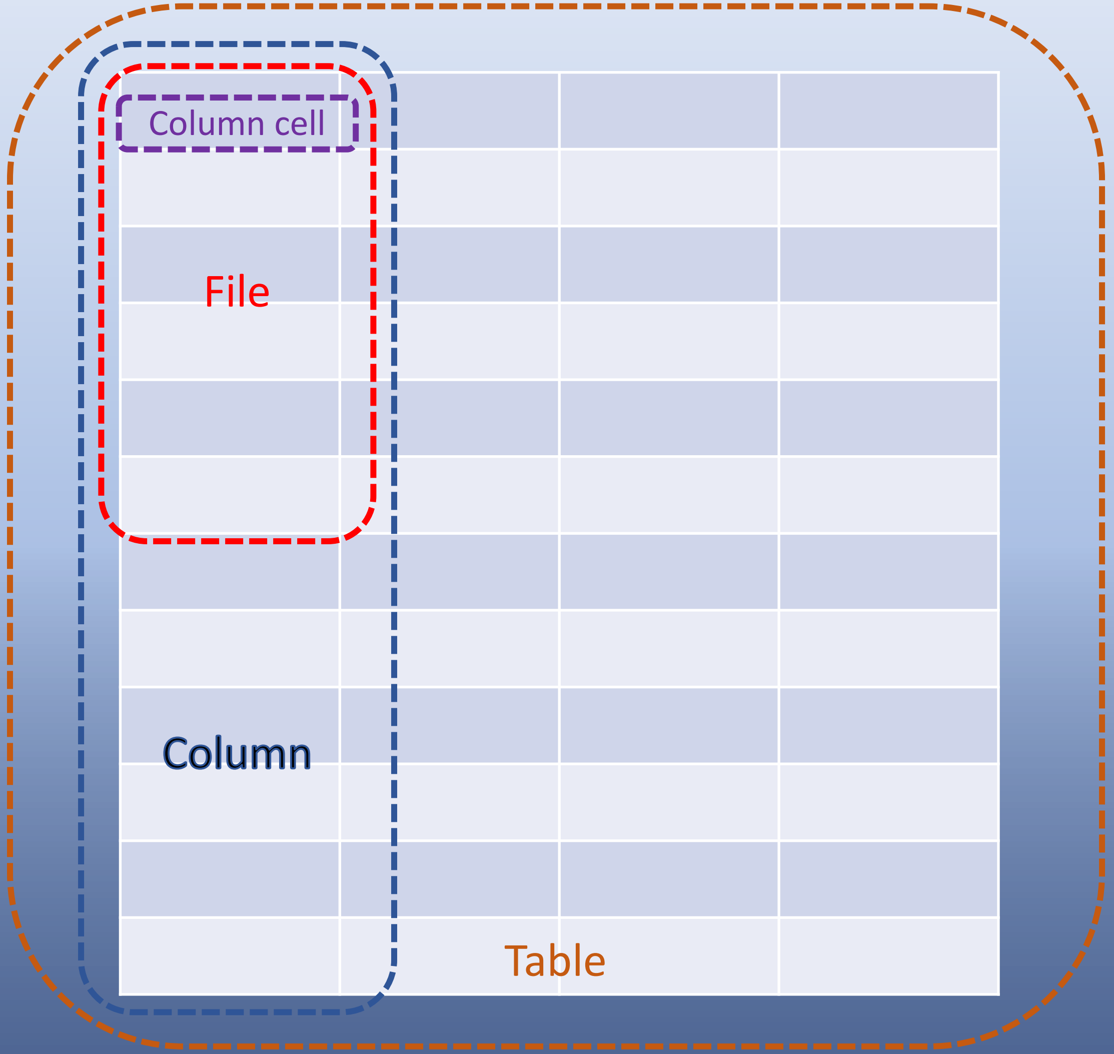
* At least in an Oracle database

Star Schemas in Row-based Databases

- A star schema is a compromise between
  - a completely denormalized table (e.g. Excel spreadsheet) and
  - a completely normalized model (e.g. 3NF)
- By normalizing the textual and descriptive attributes into dimensions, the fact table is made narrow.
  - So even a when querying the whole fact table, the number of bytes is relatively low.
- By keeping the dimension tables *de*normalized, the number of joins are kept to a minimum.
  - Joins are CPU-costly.
  - Dimensions usually contain few rows, so a full table scan is normally cheaper than a join.

Star Schemas in Databricks

- The same applies as for row-based databases:
  - => Query performance comes down to the number of i/o read and processed.
- Databricks is a columnar database using parquet/delta files.
- Only the columns of the query have to be read – not the entire row.
  - The i/o read is the size of the files accessed.



Star Schemas in Databricks

Then why not denormalize everything?

«One Big Table»

- Denormalizing everything into one big table is referred to as OBT, Wide table, fat table etc.

Order #	Order Date	Customer #	Customer Name	Customer Address	Sales Person #	Sales Person Name	Sales Person Department	Order Units	Order Amount
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales	5	$869
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing	8	$1736
.	.	.	.	.	.	.		.	.
..	..	..	..	..	..	..		..	..

- Such tables can get *very* wide.
- Most of the columns will be dimensional attributes, with repeating values.

«One Big Table»

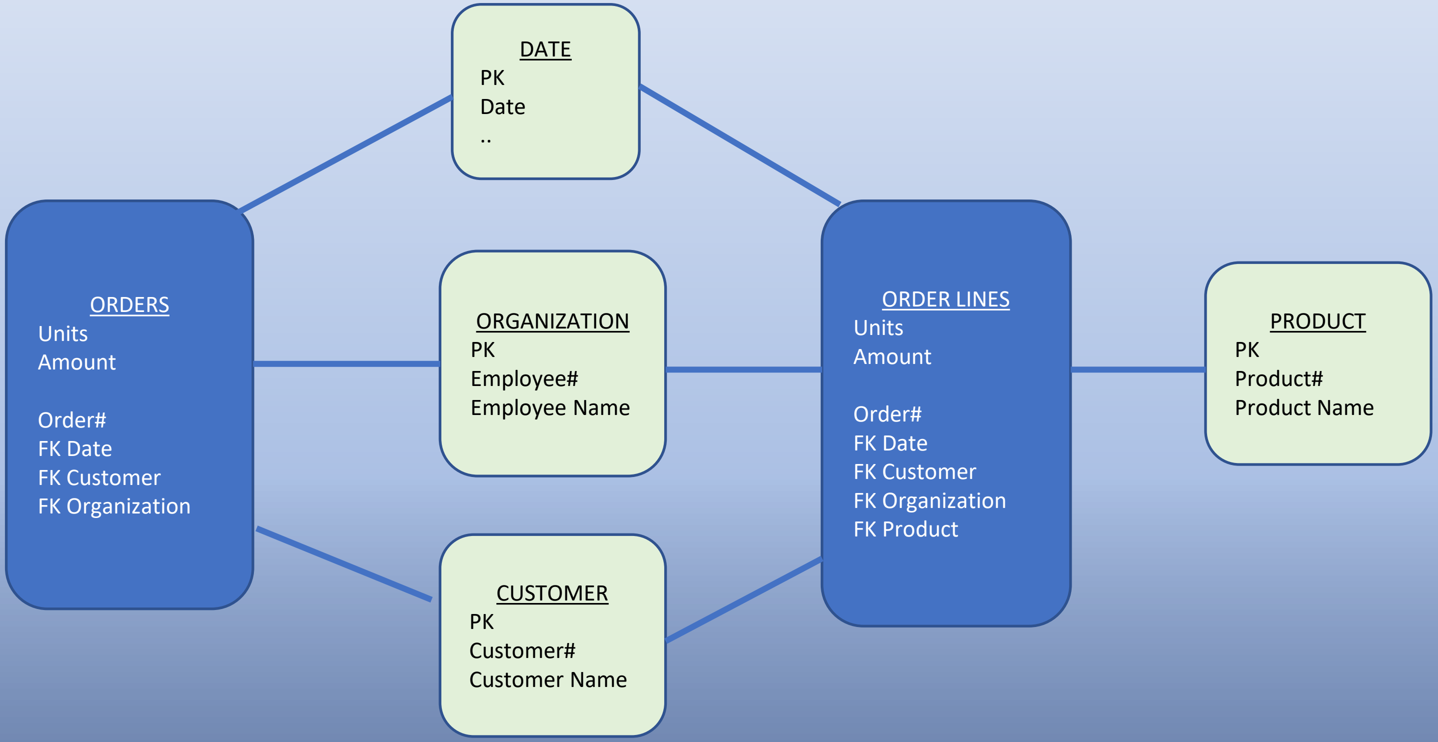
- Voices in the community claim that star schemas are a thing of the past, and that in columnar databases OBTs is the best query-performing modelling technique.

- I disagree:
  - Test of 1TB star schemas I have been involved with show no performance difference. Rather star schemas out-performing OBTs slightly.
  - Benchmark* on columnar analytical engine VertiPaq: https://www.sqlbi.com/articles/power-bi-star-schema-or-single-table/
  - BI tools (e.g. Power BI) need to access all OBT rows to create list-of-values.

* April 12th 2021 by Marco Russo and Alberto Ferrari

Drill-Across Star Schema

- A «drill across» is querying two fact tables through their common conform dimensions.
- A query have to aggregate the measures and group by dimension attributes, separately per fact table. Then join the results on the dimension attributes.
- A BI-tool such as Power BI or Oracle Analytics Server will generate the query based on the semantic model.



Drill-Across Star Schema SQL

```
WITH f1 AS (
 SELECT d.date_, c.customer_name, e.employee_name, sum(units) AS units, sum(amount) AS amount
 FROM   fak_order f
 JOIN   dim_date d         ON d.PK = f.FK_date
 JOIN   dim_customer c     ON c.PK = f.FK_customer
 JOIN   dim_organization e ON e.PK = f.FK_organization
 GROUP BY all
)
, f2 AS (
 SELECT d.date_, c.customer_name, e.employee_name, count(distinct p.product_name) AS num_products
 FROM   fak_order_lines f
 JOIN   dim_date d         ON d.PK = f.FK_date
 JOIN   dim_customer c     ON c.PK = f.FK_customer
 JOIN   dim_organization e ON e.PK = f.FK_organization
 JOIN   dim_product p      ON p.PK = f.FK_product
GROUP BY all
)
SELECT f1.date_, f1.customer_name, f1.employee_name, f1.units, f1.amount, f2.num_products
FROM   f1
JOIN   f2 ON (f1.date_=f2.date_
          AND f1.customer_name = f2.customer_name
          AND f1.employee_name = f2.employee_name);
```

"Drill-Across" Wide Table

- In a «OBT» analytic environment, the wide tables must be set up for each drill-across scenarios.
- Common conform dimension attribute values are filled in (thus repeated) for all rows.
- Dimension values for dimensions that are not common will be null for the fact rows that is not connected.
- The measure column will only have a value when the row represents that “fact table”.
- Queries are simple, because there are no joins.

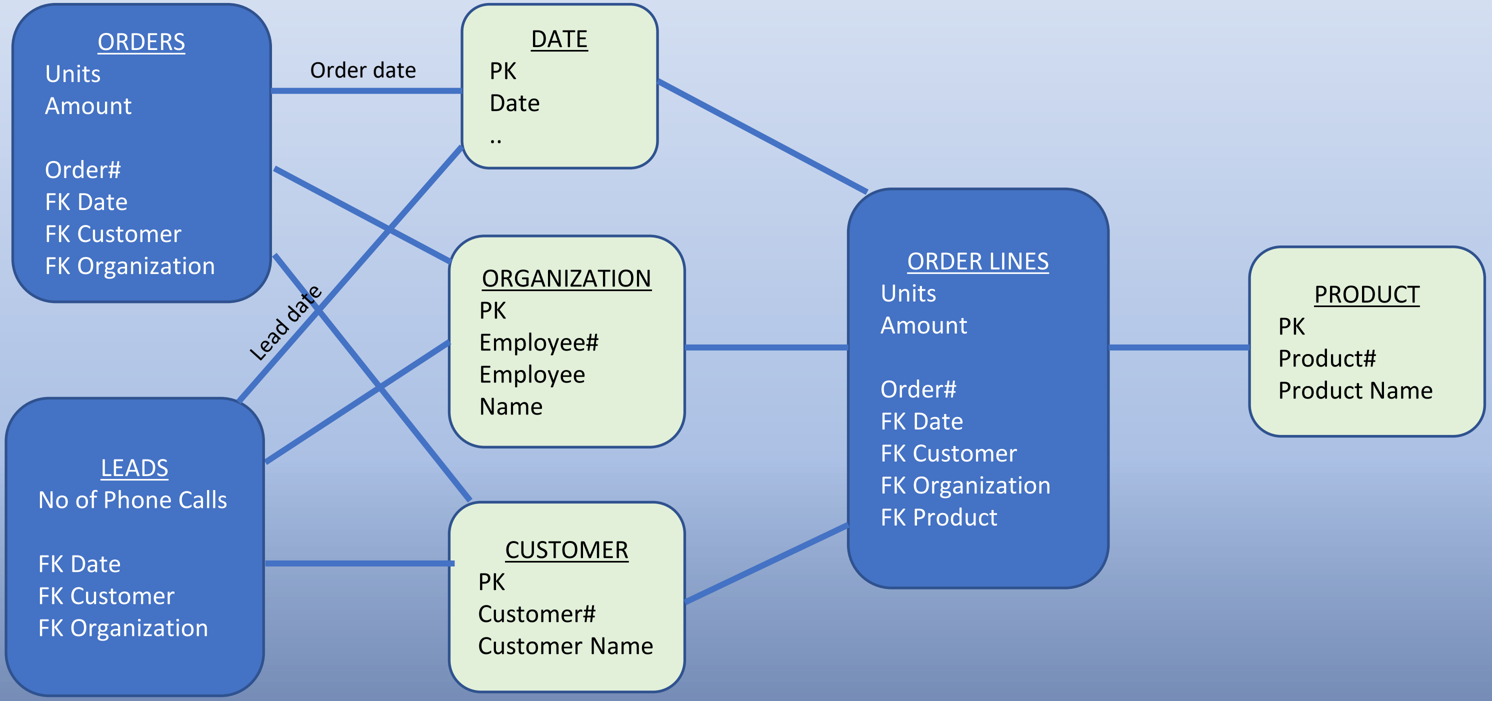
Order #	Order Date	Customer #	Customer Name	Customer Address	Sales Person #	Sales Person Name	Sales Person Department	Order Units	Order Amount	Product #	Product Name	Product Units	Product Amount
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales	5	$869				
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales			200	Screen	2	$656
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales			287	Ink	3	$213
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing	8	$1736				
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing			150	Printer	1	$1097
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing			287	Ink	10	$639

Drill-Across Wide Table SQL

```
SELECT w.date_
      ,w.customer_name
      ,w.employee_name
      ,sum(w.units) AS units
      ,sum(w.amount) AS amount
      ,count(distinct w.product_name) AS num_products
FROM   wide_table w
GROUP BY all;
```

Drill-Across Star Schema - Extended

- The dimensional model can easily be extended with new fact tables.
- A report can combine these three fact tables in any way – through the conformed dimensions.
- The BI-tool will auto-generate the SQL.



"Drill-Across" Wide Table - Extended

- In the «OBT» analytic environment we can extend the existing wide table:
  - Add new rows for the new "fact table".
  - Add new columns for:
    - the new measures
    - any new dimension – including new roles/uses of dimensions.
- The alternative to keep on adding new stars to the same wide table,
  is to split in more than one wide table.
  E.g. one table per common drill-across requirement.

Order #	Order Date	Customer #	Customer Name	Customer Address	Sales Person #	Sales Person Name	Sales Person Department	Order Units	Order Amount	Product #	Product Name	Product Units	Product Amount	Lead Date	No of Phone Calls
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales	5	$869						
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales			200	Screen	2	$656		
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales			287	Ink	3	$213		
		99	Acme Inc.	33 1st Street	4	George	Sales							2024.03.20	1
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing	8	$1736						
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing			150	Printer	1	$1097		
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing			287	Ink	10	$639		
		84	Okeanos Inc.	42 Galaxy Rd.	4	George								2024.03.25	1

How many fact tables do you have in your data warehouse?

Do you know all the users’ current and future desired drill-across paths?

How many OBTs would you need, and how tall would these be?

The OBT dilemma

- If you create few wide tables to handle many drill-across scenarios, you will get a very tall and very wide table.
  - This will reduce performance
- If you split into many wide tables, performance will be better, but:
  - You will never be able to cover all user requirements

Other OBT issues

- Dimensions are duplicated:
  - Data governance becomes more difficult
  - Lose “one version of the truth”.

- Updating dimension attribute values require update of many rows.

The SCD Type 2 Problem

- At some point in April, John changes position from the Marketing department to the Sales department

Order #	Order Date	Customer #	Customer Name	Customer Address	Sales Person #	Sales Person Name	Sales Person Department	Order Units	Order Amount
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales	5	$869
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing	8	$1736
.	.	.	.	.	.	.	.	.	.
1099	2024.05.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Sales	6	$1275
..	..	..	..	..	..	..	..	..	..

- We can easily write an SQL to give us number of orders per department.
- However, it is not so straight forward to write an SQL that gives us number of orders per sales person, and also showing the current department of that person.

The SCD Type 2 Problem

- A different scenario: The Marketing department changes its name to «Communications»

Order #	Order Date	Customer #	Customer Name	Customer Address	Sales Person #	Sales Person Name	Sales Person Department	Order Units	Order Amount
1000	2024.04.01	99	Acme Inc.	33 1st Street	4	George	Sales	5	$869
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Marketing	8	$1736
.	.	.	.	.	.	.	.	.	.
1099	2024.05.01	84	Okeanos Inc.	42 Galaxy Rd.	5	John	Communications	6	$1275
..	..	..	..	..	..	..	..	..	..

- Now we can *not* easily write an SQL to give us number of orders per department, because «Marketing» and «Communications» is the same department.

SCD Type 6/7 in OBT

- Let’s introduce «current value» columns in the wide table.
- Since the transaction date:
  - One customer has changed address
  - One sales person has changed department

Order #	Order Date	Customer #	Customer Name	Customer Address	Cust Adr Current	Sales Person #	Sales Person Name	Sales Person Department	Sales Person Dept Current	Order Units	Order Amt	Product #	Product Name	Product Units	Product Amount
1000	2024.04.01	99	Acme Inc.	33 1st Street	22 Main Street	4	George	Sales	Sales	5	$869				
1000	2024.04.01	99	Acme Inc.	33 1st Street	22 Main Street	4	George	Sales	Sales			200	Screen	2	$656
1000	2024.04.01	99	Acme Inc.	33 1st Street	22 Main Street	4	George	Sales	Sales			287	Ink	3	$213
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	42 Galaxy Rd.	5	John	Marketing	Sales	8	$1736				
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	42 Galaxy Rd.	5	John	Marketing	Sales			150	Printer	1	$1097
1001	2024.04.01	84	Okeanos Inc.	42 Galaxy Rd.	42 Galaxy Rd.	5	John	Marketing	Sales			287	Ink	10	$639

SCD Type 6/7 in OBT

- All the rows of the table for the given natural keys must be updated whenever a «dimension» is updated.
- Databricks is «append only»
  => Updates create completely new files, thus increasing storage.
- Very timely operation.
- Difficult to ensure that all the current values of every natural key is equal in every wide table?

Pros and cons for «One Big Table»

Pros

- No joins needed.
  => Simple SQLs
- The users don’t have to search in more than one table for all the columns needed.

Cons

- Necessary to read all rows to find only a few distinct values.
- Columns not «categorized» in tables, so it might be difficult for user to find the columns he is looking for.
- Necessary to update many rows to update a single dimensional attribute value.
- Not conformed dimensions => Siloed data
- Poor data governance and not “one version of the truth”
- Data redundancy
- Require more storage than star schemas.

The Future is...

- Experience show that the majority of reports only need current attribute values.
  - SCD Type 1 is sufficient
- Only some reports require historic attribute values
  - And even then, SCD Type 2 is probably only needed for one of the report's dimensions
- The SCD-type that solve both 1 and 2 are 6 and 7.

The Future is...

- Remember, SCD Type 6 is one table with columns for current attribute values for all historic rows.
    - There could be many such columns.
    - The values have to be updated constantly for all the historic rows.

- Remember, SCD Type 7 is two foreign keys pointing to the same dimension:
    - FK1 points to the current row – an SCD Type 1-view of the dimension
      FK1 is a natural key
    - FK2 points to the historic row of the SCD Type 2
      FK2 is a surrogate key
    - No extra columns in the dimension
    - Only update of last row's To Date and Current Row Flag when a new row is added.

- Therefore: Start using Type 1 and 7 only!

Use of SCD Type 7

Date Dimension

Date SK	Date	Day of Week	Etc
20240401	2024.04.01	Monday	..
20240402	2024.04.02	Tuesday	..
..	..	..	..
20240501	2024.05.01	Wednesday	..

Orders Fact Table

Date SK	Cust SK	Cust NK	Emp SK	Emp NK	Units	Amount
20240401	1	99	1	4	5	869
20240402	2	84	2	5	8	1736
..	..	..	..			
20240501	9	99	8	5	7	1265

Customer Dimension – SCD Type 1

Customer #	Name	Address
84	Okeanos Inc.	42 Galaxy Rd.
99	Acme Inc.	22 Main Street

Customer Dimension – SCD Type 2

Cust SK	Customer #	Name	Address	From Date	To Date	Current Row
1	99	Acme Inc.	33 1st Street	2021.10.14	2024.04.30	N
2	84	Okeanos Inc.	42 Galaxy Rd.	2023.06.10	9999.12.31	Y
..	..	..	..			
9	99	Acme Inc.	22 Main Street	2024.05.01	9999.12.31	Y

Employee Dimension – SCD Type 1

Employee #	Name	Dept
4	George	Sales
5	John	Sales

Employee Dimension – SCD Type 2

Emp SK	Employee #	Name	Dept	From Date	To Date	Current Row
1	4	George	Sales	2022.02.01	9999.12.31	Y
2	5	John	Marketing	2023.07.01	2024.04.30	N
..	..	..	..			
8	5	John	Sales	2024.05.01	9999.12.31	Y

Use of SCD Type 7

To make a report that groups on the current values of the dimensions it is necessary to join the fact table with the dimension on the natural key (NK) + the «current row» flag.

```
SELECT d.dept, sum(amount) AS amount
FROM   fact f
JOIN   employee d
ON    (d.emp_nk = f.emp_nk AND
       d.current_row = "Y")
GROUP BY all;
```

Use of SCD Type 7

- Some BI tools cannot perform complex joins / joins on composite keys. The join should therefore be with a view…

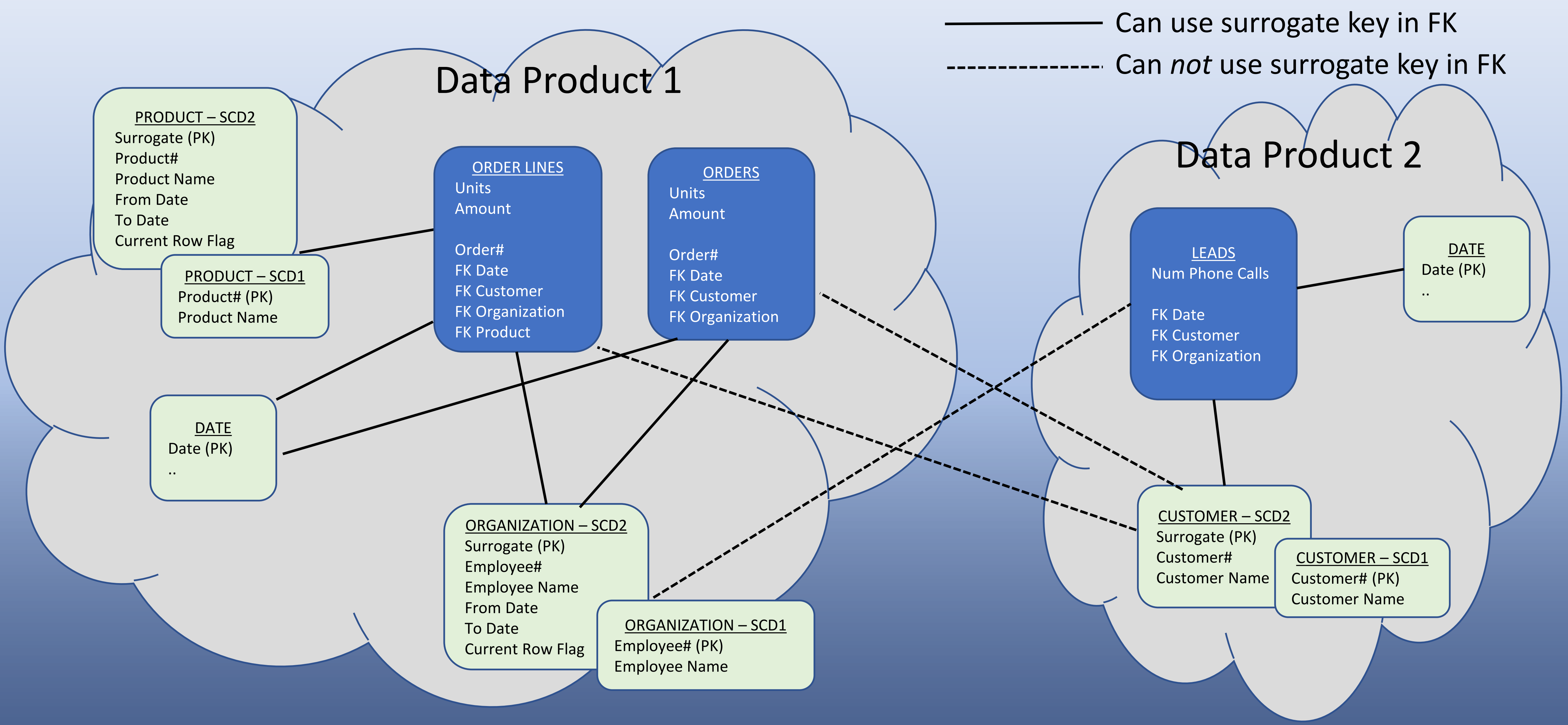
  … preferably materialized by a delta live table.
  (As a DLT can be given a primary key.*)

```
CREATE VIEW employee_current AS
SELECT *
FROM   employee
WHERE  current_row = "Y";
```

```
SELECT d.dept, sum(amount) AS amount
FROM   fact f
JOIN   employee_current d
ON    (d.emp_nk = f.emp_nk)
GROUP BY all;
```

* In public preview

Star Schemas in a Data Mesh



Star Schemas in a Data Mesh

- Fact tables can have foreign keys to surrogate keys only for dimensions within the same data product.
- Fact tables with foreign keys to dimensions in other data products must use the natural key.
    - For SCD Type 1 the NK is unique, so the the FK on the NK is enough.
    - For SCD Type 2 the NK is not unique, so the transaction date must be used together with the NK.

- Thus:
    - For SCD Type 1 consider to *always* add the primary key to the natural key and *not* to a surrogate key column.
      (The join performance will be slightly reduced, but in most cases good enough.)
    - For the SCD Type 2 table of your Type 7 dimensions, add the primary key to the surrogate key column.

- Some BI tools cannot handle complex joins (join on composite keys).
    - This is a problem for the FKs from fact tables to Type 2 dimensions in other data products.
    - Solution proposition: Denormalize the *required* historical dimension attributes into your fact table, as done for an OBT. (This has to be done by ETL.)

Summary of recommendations

- Continue to use Star Schemas!
- Maximize the use of SCD Type 1
- Use SCD Type 7 when history is *required*
- If your star schema is split into many data products in a data mesh, and when referencing a dimension in another DP, then:
  - use the natural key for Type 1 dimensions.
  - If history is required – and your BI tool cannot handle complex joins – then denormalize the historic attribute values in the fact table ETL-time.