

POS Explorer

Martin Faust

Retail Promotions

- ☐ **Promotions** are important for retailers
 - competition, price pressure, increasing frequency of promotions
 - negotiations with vendors (vendor funds), need for quick assessment of consequences
 - attract new customers
 - increase revenue

- ☐ Share of revenue from promotions is increasing

UMSATZANTEIL VON PREISPROMOTIONS AN FMCG NACH VERTRIEBSLINIEN

LEH gesamt, FMCG ohne Frische, Angaben in Prozent



Quelle: GfK ConsumerScan

© 2011 GfK + SAP

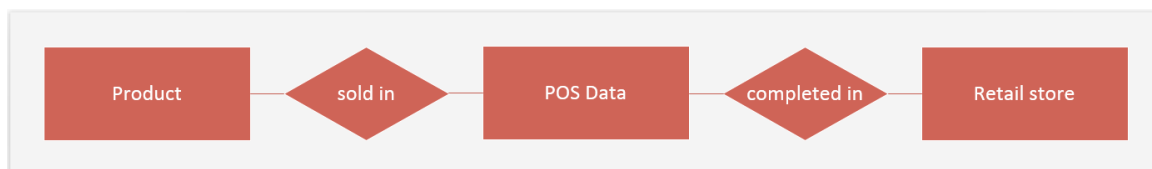
Page 3

Use Case

- You want to initiate a **new promotion**
- Our tool helps you to **explore your sales data** and to find the product you should promote
- Everything needs to be **sub second** so that you can **explore every product combination** you like
- User may want to apply filters that prohibit pre-computed aggregates or data cubes

Point-of-Sales Data

- ❑ We got POS data of 2.5 years from **about 5000 stores**
- ❑ Table sizes (rows):
 - Products: 500,000
 - Retail stores: 5,000
 - **POS data: 8 billion**
 - partitioned by receipt id
 - 100 evenly sized partitions



Page 5

Point-of-Sales Data II

500.000 Products:

- Product ID
- **Main Product Group**
- **General Product Group**
- **Specific Product Group**
- Supplier
- Type of Brand
- Product Attribute 1
- Product Attribute 2

8 bn POS data items:

- Receipt ID
- Product ID
- Store ID
- Sales Unit
- Currency
- Date
- Time
- **Promotion Flag**
- TIX (Transaction Index)
- **Price**
- **Quantity**
- **Revenue**
- **Total Receipt Revenue**
- Product Group Level 1 to 3

5000 Retail stores:

- Store ID
- Wholesale ID
- Type of Business
- Postal Code
- Distribution Channel
- Region
- Store Size

Page 6

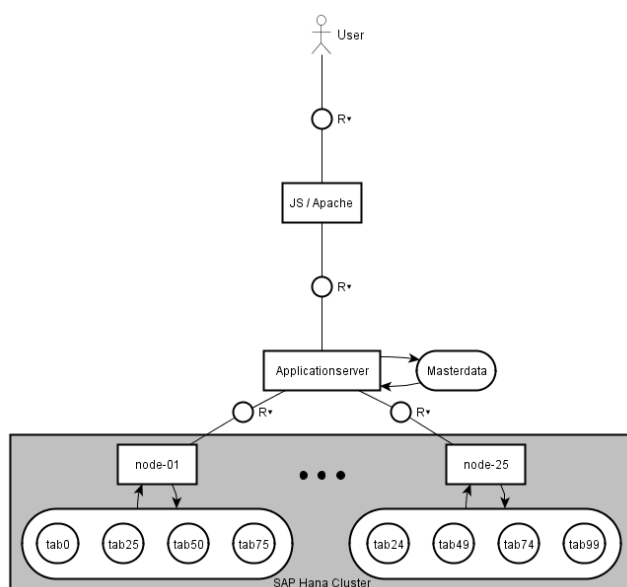
Infrastructure

- ❑ 1000 Core Cluster (25 Nodes) with 25 TB RAM and 75 TB SSD
- ❑ SAP's In-Memory Database HANA

Record distribution:

- ❑ Records equally distributed over 100 partitions on 25 nodes (4 per node) having partition-ID = receipt id mod 100
- ❑ All items of a receipt are in the same partition

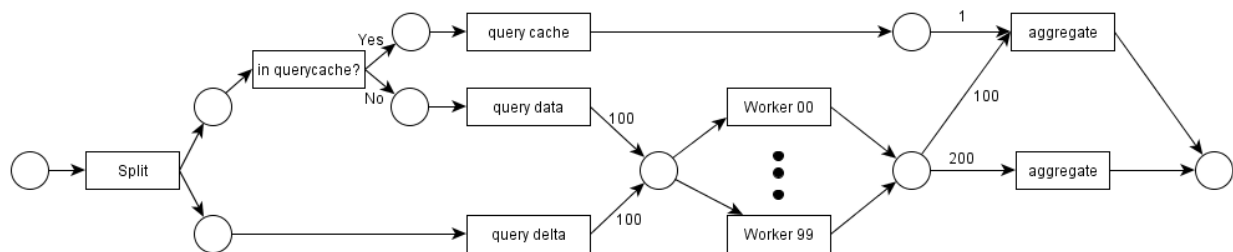
Architecture



- ❑ Frontend powered by Apache webserver
- ❑ For each query a AJAX request is sent to the application server
- ❑ Application server looks up the requested query in the query dictionary
- ❑ Query gets distributed over 100 threads and query is sent to 100 tables on 25 nodes
- ❑ 100 resultsets are aggregated by the application server and result is sent back to the frontend in JSON format

Query Caching

- ❑ If the application server receives a request it checks for the query results in the query cache and splits the query into a main and delta query
- ❑ Cache-Hit: get results from query cache
- ❑ Cache-Miss: get results from data tables
- ❑ **Delta** and data queries get queued in query pool from which the 100 threads fetch the next query when they finished their last query
- ❑ Aggregate query results
- ❑ Save result to query cache
- ❑ Return result



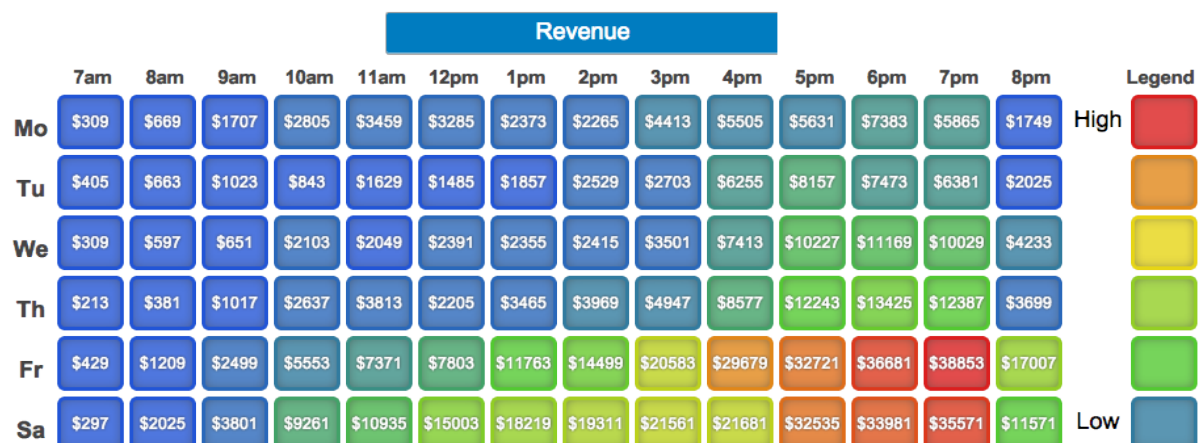
Key Figures

- ❑ **market share**
market share of products compared to their **product groups**
- ❑ **basket value**
the average revenue generated by **other products** in baskets where the product was sold
- ❑ **revenue**
total revenue generated of the products

Apriori Analysis

- Apriori algorithm:
 - If Cola is in the basket, what is the probability that chips are bought as well?
- **products** that are most commonly **sold together**
- support value:
 - transactions containing **both products relative to the total number of transactions**
- confidence value:
 - transactions containing **both products relative to the number of transactions containing product A**

Demo



Classic Implementation of Apriori Analysis

- ❑ Transform receipts to vectorized representation
- ❑ Columns are essentially bitmap indices for ProductID
- ❑ **Example:** Select count(*) from T where Cola=1 and Chips=1
- ❑ Problems: How to find commonly sold together products?
- ❑ How to extent schema for other attributes like *price*, *promotion flag*, or *number of items sold*?

Receipt ID	Cola	Pepsi	Chips	> 100.000 products
0	0	1	0	...
1	1	0	0	...
2	1	1	1	...
3	0	1	1	...
...				...
n				...

POS Data Queries

Query

```
SELECT B.PRODUCT_ID, COUNT(A.RECEIPT_ID) AS ABTOTAL
FROM
  "POS_DATA" AS A
JOIN
  "POS_DATA" AS B
  ON A.PRODUCT_ID <> B.PRODUCT_ID
  AND A.RECEIPT_ID = B.RECEIPT_ID
WHERE
  A.PRODUCT_ID = 123456789
GROUP BY PRODUCT_ID
ORDER BY ABTOTAL DESC
LIMIT 500
```

Apriori on POS Data Schema

8 billion records

Receipt ID	Product Id
0	p2
1	p1
2	p1
2	p2
2	p3
3	p2
3	p3



8 billion records

Receipt ID	Product Id
0	p2
1	p1
2	p1
2	p2
2	p3
3	p2
3	p3

ON A.ReceiptID = B.Receipt.ID
AND B.ProductID <> B.ProductID
WHERE A.ProductID=p2

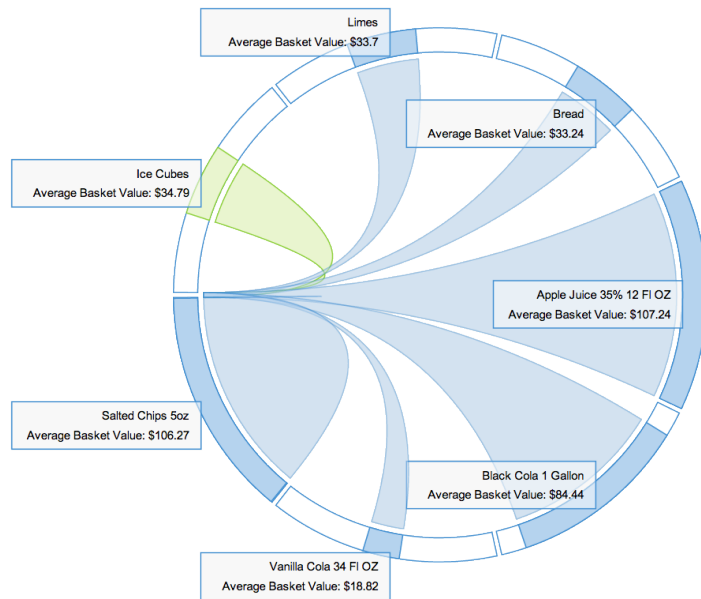
Receipt ID	Product Id A	Product Id B
2	p2	p1
2	p2	p3
3	p2	p3

Apriori on POS Data Schema II

- ❑ **Advantages:** Multi-Purpose Schema
- ❑ Flexibility, no extra storage costs for specialized basket analysis table, single source of truth
- ❑ **Disadvantages:** More complex queries, huge self-joins.

Visualization Techniques

- Get rid of listviews
- 1:n relationship
- Size of chords to represent basket value

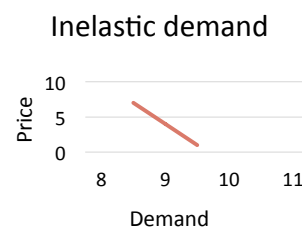
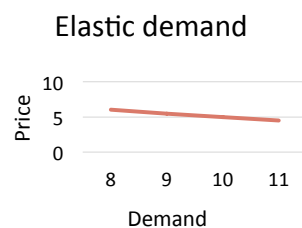


Page 17

Excuse: Price Elasticity

Price elasticity measures the rate of response of quantity demanded due to a price change. [1]

- Value indicates how sales of a product react to price changes
- **Important key figure**



Thank you

- Questions?