POZNAN UNIVERSITY OF TECHNOLOGY

Data Warehouse Physical Design: Part III

Robert Wrembel
Poznan University of Technology
Institute of Computing Science
Robert.Wrembel@cs.put.poznan.pl
www.cs.put.poznan.pl/rwrembel

Lecture outline

- Row storage vs. Column storage
- Data compression
- Materialization
  - Small summary data
  - Materialized views and query rewriting

Row storage (standard)

CompanyID	Year	Month	Day	Open	Min	Max	Close	Change
BZWBK	2006	Mar	31	148,00	147,00	148,00	148,00	0,00
BZWBK	2006	Mar	30	149,00	147,50	150,50	148,00	0,34
BZWBK	2006	Mar	29	148,50	146,50	148,50	147,50	-1,01
BZWBK	2006	Mar	28	150,00	148,00	150,00	149,00	-0,67
BZWBK	2006	Mar	27	147,50	147,50	150,00	150,00	2,74
BZWBK	2006	Mar	24	148,00	142,00	150,00	146,00	-1,02
BZWBK	2006	Mar	23	148,00	147,50	150,00	147,50	0,00
BZWBK	2006	Mar	22	147,00	145,50	150,00	147,50	0,34
BZWBK	2006	Mar	21	148,50	147,00	150,00	147,00	-2,00
BZWBK	2006	Mar	20	148,50	147,00	151,50	150,00	1,01
BZWBK	2006	Mar	17	151,50	148,00	152,00	148,50	-1,00
BZWBK	2006	Mar	16	150,00	149,50	152,50	150,00	0,67
BZWBK	2006	Mar	15	151,50	149,00	152,00	149,00	-0,67
BZWBK	2006	Mar	14	149,00	148,50	151,50	150,00	0,00
BZWBK	2006	Mar	13	152,50	146,00	152,50	150,00	-0,33
BZWBK	2006	Mar	10	152,00	146,00	154,50	150,50	-2,27
BZWBK	2006	Mar	9	154,50	154,00	156,50	154,00	0,33
BZWBK	2006	Mar	8	164,50	153,50	165,00	153,50	-6,97
BZWBK	2006	Mar	7	172,50	165,00	172,50	165,00	-4,62
BZWBK	2006	Mar	6	170,00	168,00	173,00	173,00	1,76
BZWBK	2006	Mar	5	169,50	163,50	171,50	170,00	0,29
BZWBK	2006	Mar	2	170,50	168,00	171,50	169,50	-0,88
BZWBK	2006	Mar	1	166,00	165,00	173,50	171,00	4,27

data blocks

BZWBK	2006	Mar	31	148,00	147,00	148,00	148,00	0,00
BZWBK	2006	Mar	30	149,00	147,50	150,50	148,00	0,34
BZWBK	2006	Mar	29	148,50	146,50	148,50	147,50	-1,01
BZWBK	2006	Mar	28	150,00	148,00	150,00	149,00	-0,67

BZWBK	2006	Mar	27	147,50	147,50	150,00	150,00	2,74
BZWBK	2006	Mar	24	148,00	142,00	150,00	146,00	-1,02
BZWBK	2006	Mar	23	148,00	147,50	150,00	147,50	0,00
BZWBK	2006	Mar	22	147,00	145,50	150,00	147,50	0,34

BZWBK	2006	Mar	6	170,00	168,00	173,00	173,00	1,76
BZWBK	2006	Mar	5	169,50	163,50	171,50	170,00	0,29
BZWBK	2006	Mar	2	170,50	168,00	171,50	169,50	-0,88
BZWBK	2006	Mar	1	166,00	165,00	173,50	171,00	4,27

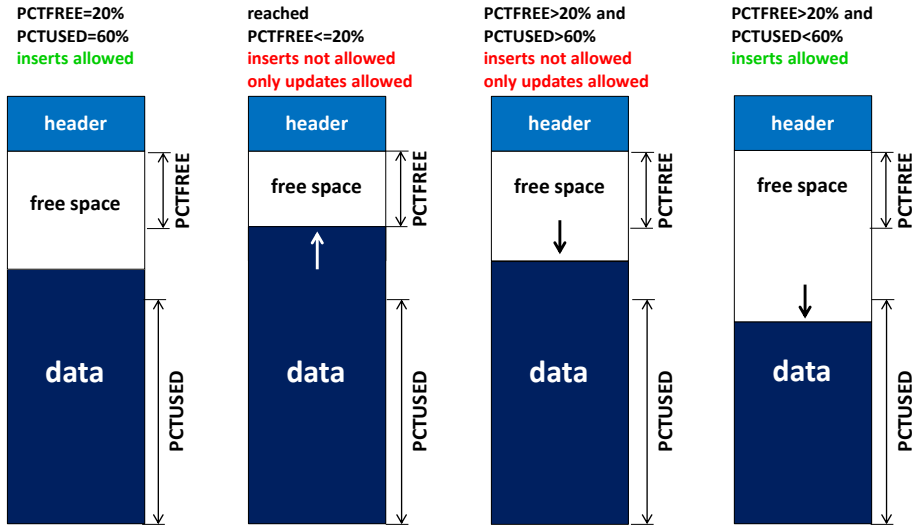
➲ **Rows identified by ROWIDs**

Row storage

➲ **Database block**

blockID
available for inserts flag
available for updates flag
transactions modifying data
locks
tables catalog
rows catalog

Row storage



Column storage

CompanyID	Year	Month	Day	Open	Min	Max	Close	Change
BZWBK	2006	Mar	31	148,00	147,00	148,00	148,00	0,00
BZWBK	2006	Mar	30	149,00	147,50	150,50	148,00	0,34
BZWBK	2006	Mar	29	148,50	146,50	148,50	147,50	-1,01
BZWBK	2006	Mar	28	150,00	148,00	150,00	149,00	-0,67
BZWBK	2006	Mar	27	147,50	147,50	150,00	150,00	2,74
BZWBK	2006	Mar	24	148,00	142,00	150,00	146,00	-1,02
BZWBK	2006	Mar	23	148,00	147,50	150,00	147,50	0,00
BZWBK	2006	Mar	22	147,00	145,50	150,00	147,50	0,34
BZWBK	2006	Mar	21	148,50	147,00	150,00	147,00	-2,00
BZWBK	2006	Mar	20	148,50	147,00	151,50	150,00	1,01
BZWBK	2006	Mar	17	151,50	148,00	152,00	148,50	-1,00
BZWBK	2006	Mar	16	150,00	149,50	152,50	150,00	0,67
BZWBK	2006	Mar	15	151,50	149,00	152,00	149,00	-0,67
BZWBK	2006	Mar	14	149,00	148,50	151,50	150,00	0,00
BZWBK	2006	Mar	13	152,50	146,00	152,50	150,00	-0,33
BZWBK	2006	Mar	10	152,00	146,00	154,50	150,50	-2,27
BZWBK	2006	Mar	9	154,50	154,00	156,50	154,00	0,33
BZWBK	2006	Mar	8	164,50	153,50	165,00	153,50	-6,97
BZWBK	2006	Mar	7	172,50	165,00	172,50	165,00	-4,62
BZWBK	2006	Mar	6	170,00	168,00	173,00	173,00	1,76
BZWBK	2006	Mar	5	169,50	163,50	171,50	170,00	0,29
BZWBK	2006	Mar	2	170,50	168,00	171,50	169,50	-0,88
BZWBK	2006	Mar	1	166,00	165,00	173,50	171,00	4,27

slot 1

data blocks

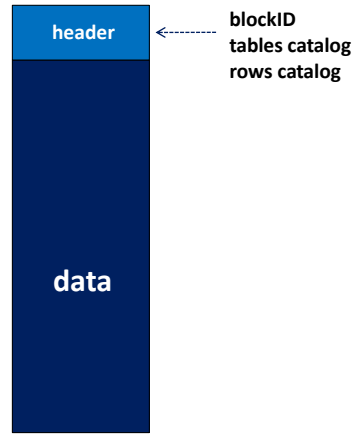
148,00	149,00	148,50	150,00	147,50	148,00	148,00	147,00	148,50
148,50	151,50	150,00	151,50	149,00	152,50	152,00	154,50	164,50
172,50	170,00	169,50	170,50	166,00				

147,00	147,50	146,50	148,00	147,50	142,00	147,50	145,50	147,00
147,00	148,00	149,50	149,00	148,50	146,00	146,00	154,00	153,50
165,00	168,00	163,50	168,00	165,00				

➲ **Rows identified by slot numbers (in a block)**

Column storage

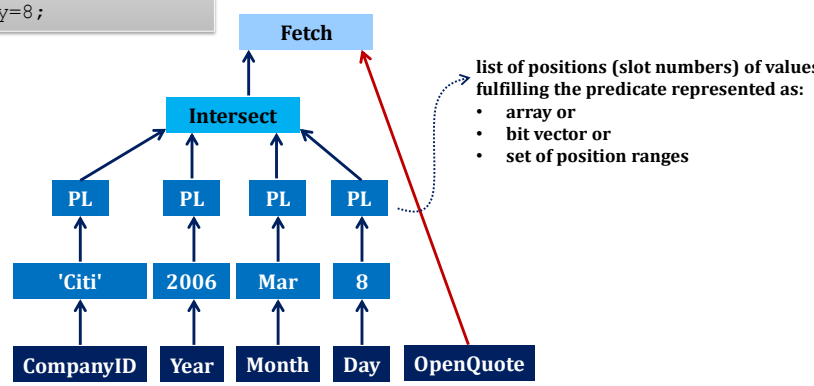
- **Database block**
  - **no free space**
  - **better space utilization**



Query processing example

```
select OpenQuote
from StockQuotes
where CompanyID='Citi'
and Year=2006
and Month='Mar'
and Day=8;
```

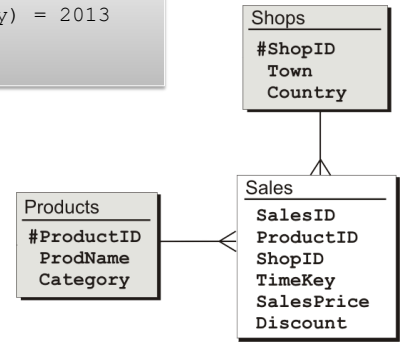
- **One table query in column storage**



Query processing

- **Join query**

```
select sum(SalesPrice), ProdName, Country
from Sales sa, Products pr, Shops sh
where Category='cheese'
and Country in ('Poland', 'Slovakia')
and extract (year from TimeKey) = 2013
and sa.ProductID=pr.ProductID
and sa.ShopID=sh.ShopID;
```



Join query

- **Get Products.ProductIDs that satisfy:**
  - **Products.Category='cheese',**
- **Get Shops.ShopIDs that satisfy:**
  - **Shops.country in ('Poland', 'Slovakia')**
- **Get Sales.SalesIDs that satisfy:**
  - **extract(year from Sales.TimeKey)=2013**
- **How to do this?**

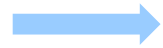
```
select sum(SalesPrice), ProdName, Country
from Sales sa, Products pr, Shops sh
where Category='cheese'
and Country in ('Poland', 'Slovakia')
and extract (year from TimeKey) = 2013
and sa.ProductID=pr.ProductID
and sa.ShopID=sh.ShopID;
```

Join query

- Get Shops.ShopIDs that satisfy:
  - country in ('Poland', 'Slovakia')

Shops

ShopID	Town	Country
10	Poznan	Poland
20	Moscou	Russia
30	Bratislava	Slovakia
40	Lublana	Slovenia

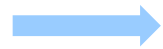


ShopID: {10, 30}

- Get Product.ProductIDs that satisfy:
  - Category='cheese'

Products

ProductID	ProdName	Category
100	queso Manchengo	cheese
500	pecorino baccellone	cheese
200	queso de Burgos	cheese
700	Rioja reserva	wine



ProductID: {100, 200, 500}

Join query

- Get Sales.SalesIDs that satisfy:
  - extract(year from TimeKey)=2013

Sales

SalesID	ProductID	ShopID	TimeKey	SalesPrice
S1	500	10	30-Apr-2012	55
S2	100	20	20-Mar-2013	50
S3	700	10	22-Mar-2013	30
S4	200	30	01-Apr-2013	75
S5	200	40	04-May-2013	45

2013

{S2, S3, S4, S5}

bitmap
0
1
1
1
1

Join query

- Represent slot numbers as bitmaps
  - 2013: TimeKey in {S2, S3, S4, S5}
  - 'Poland' or 'Slovakia': ShopID in {10, 30}
  - 'cheese': ProductID in {100, 200, 500}
- Find an intersection (slot numbers) of the bitmaps

Sales

SalesID	ProductID	ShopID	TimeKey	SalesPrice
S1	500	10	30-Apr-2012	55
S2	100	20	20-Mar-2013	50
S3	700	10	22-Mar-2013	30
S4	200	30	01-Apr-2013	75
S5	200	40	04-May-2013	45

2013 {S2, S3, S4, S5} → bitmap: 0, 1, 1, 1, 1

'Poland' or 'Slovakia' {10, 30} → bitmap: 1, 0, 1, 1, 0

'cheese' {100, 200, 500} → bitmap: 1, 1, 0, 1, 1

Join query

- Matching Sales rows are represented by bitmaps

ShopID in (10, 30) bitmap: 1, 0, 1, 1, 0

AND

ProductID in (100, 200, 500) bitmap: 1, 1, 0, 1, 1

AND

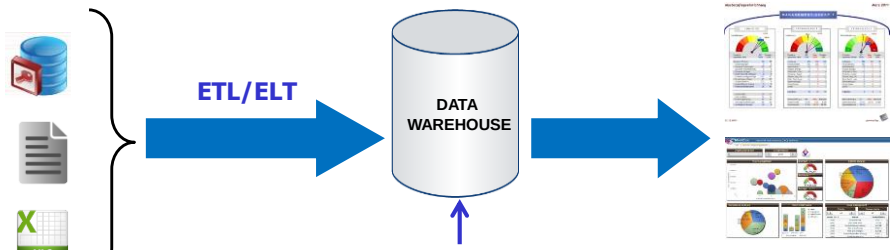
SalesID in (S2, S3, S4, S5) bitmap: 0, 1, 1, 1, 1

=

bitmap: 0, 0, 0, 1, 0

- Get Sales.ProductIDs using the final bitmap
- Join with Products to get 'prodName' using the ProductIDs
- Join with Shops to get 'Country' using the ShopIDs

Column storage in DW architecture



Model 204 (projection index)
Sybase IQ (Sybase, Inc.)
SADAS (Advanced Systems)
C-Store/Vertica
MonetDB
Infobrigh
Exasol
SQL Server since ver. 2012

CS - compression

run-length encoding

15.05.2020
15.05.2020
15.05.2020
15.05.2020
16.05.2020
16.05.2020
16.05.2020
16.05.2020
16.05.2020
16.05.2020
16.05.2020

15.05.2020	4
16.05.2020	7

delta encoding

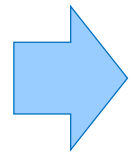
2000
2002
2002
2004
2005
2030
2040
2045
2050

2000
2
0
2
1
25
10
5
5

Data compression

➲ **Discrete domain encoding**

Shop	Date	Price
Alma Old Brewery	01.2006	56 000
Alma Citi Park	01.2006	13 000
Alma Focus Park	01.2006	24 000
Alma Old Brewery	02.2006	52 000
Alma Citi Park	02.2006	18 600
Alma Focus Park	02.2006	21 100
Alma Old Brewery	03.2006	43 200
Alma Citi Park	03.2006	25 700
Alma Focus Park	03.2006	14 700
Alma Old Brewery	04.2006	32 400
Alma Citi Park	04.2006	19 500
Alma Focus Park	04.2006	15 000



Shop	Date	Price
1	01.2006	56 000
2	01.2006	13 000
3	01.2006	24 000
1	02.2006	52 000
2	02.2006	18 600
3	02.2006	21 100
1	03.2006	43 200
2	03.2006	25 700
3	03.2006	14 700
1	04.2006	32 400
2	04.2006	19 500
3	04.2006	15 000

mapping table

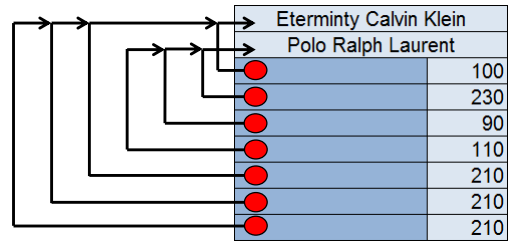
Shop	Code
Alma Old Brewery	1
Alma Citi Park	2
Alma Focus Park	3

Data compression

➲ **Dictionary compression (DB2)**
- **like discrete domain encoding**
- **mapping table (dictionary) stored in**
  - **a dedicated table**
  - **data block header**

➲ **Block compression (Oracle)**

Eternity Calvin Klein	100
Polo Ralph Laurent	230
Polo Ralph Laurent	90
Polo Ralph Laurent	110
Eternity Calvin Klein	210
Eternity Calvin Klein	210
Eternity Calvin Klein	210



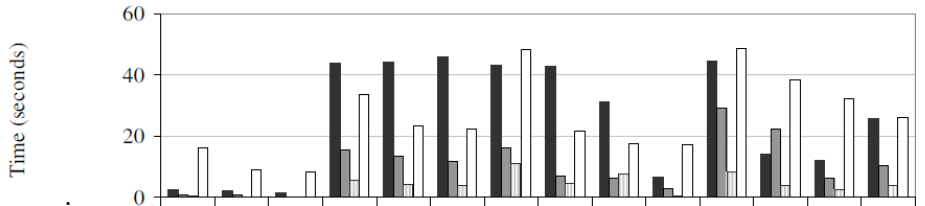
RS-CS comparison

- DB size
  - source data file: 1TB
  - RS DB size > 1TB
  - CS DB size < 1TB
- Performance of group by queries
  - RS: lower → more data needs to be read (whole rows)
  - CS: higher → less data needs to be read (only columns in group by and aggreg. functions)
  - RS: more indexes and materialized views need to be created

Performance comparison: CS-RS

- Source
  - D.J. Abadi, S.R. Madden, N. Hachem: Column-stores vs. row-stores: how different are they really?. SIGMOD, 2008
- Experimental setup:
  - Star Schema Benchmark ⇨ DW benchmark derived from TPC-H (pure star schema)
  - 13 queries divided into 4 categories

Performance comparison: CS-RS



queries	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	3.4	4.1	4.2	4.3	AVG
RS	2.7	2.0	1.5	43.8	44.1	46.0	43.0	42.8	31.2	6.5	44.4	14.1	12.2	25.7
RS (MV)	1.0	1.0	0.2	15.5	13.5	11.8	16.1	6.9	6.4	3.0	29.2	22.4	6.4	10.2
CS	0.4	0.1	0.1	5.7	4.2	3.9	11.0	4.4	7.6	0.6	8.2	3.7	2.6	4.0
CS (Row-MV)	16.0	9.1	8.4	33.5	23.5	22.3	48.5	21.5	17.6	17.4	48.6	38.4	32.1	25.9

row store

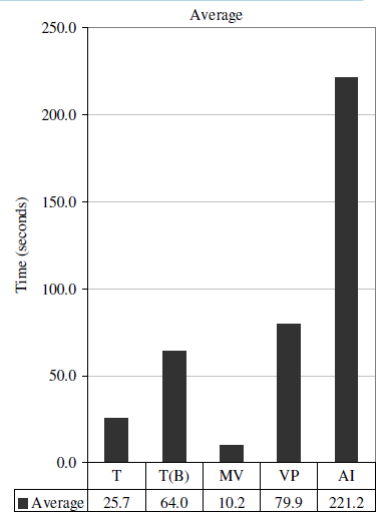
column store

RS with optimal set of materialized views

CS implemented by standard RS materialized views

Performance comparison: CS-RS

- RS with various data structures
  - T: traditional RS
  - T(B): traditional + bitmap indexes
  - MV: optimal set of mat. views
  - VP: vertical partitioning (simulated - each column in its own table)
  - AI: B+-tree on each column



	T	T(B)	MV	VP	AI
Average	25.7	64.0	10.2	79.9	221.2

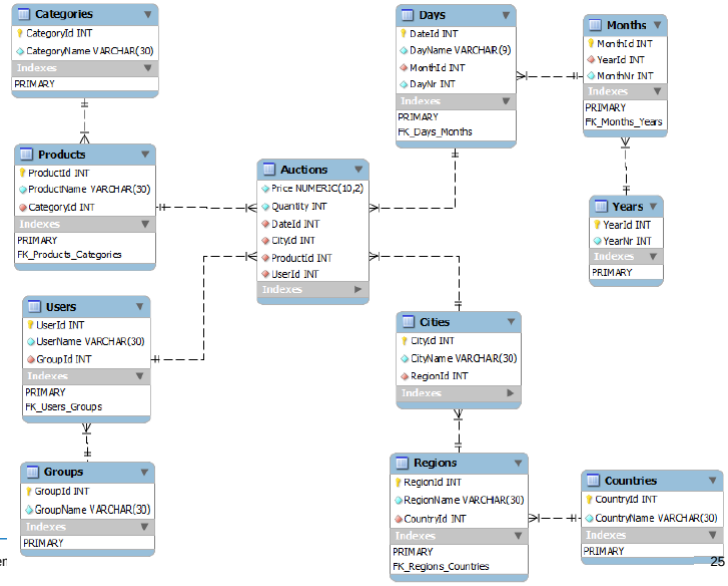
Our experiments

- Oracle11g and SybaseIQ 15.4
- DB size 3GB
- Cache
  - Sybase: 1024MB (main cache size)
  - Oracle: 1024MB (data cache)
- Intel Core 2 Duo P8400 2,27 GHz, 4GB RAM, disc Hitachi Travelstar 5K250 HTS542525K9SA00

Our experiments (2)

- Indexes
  - SybaseIQ
    - Fast Projection (default) ⇨ on all columns for projection optimization
    - High Group (default) ⇨ on UNIQUE, PRIMARY KEY, FOREIGN KEY
  - Oracle
    - PRIMARY KEY (default)
    - FOREIGN KEY

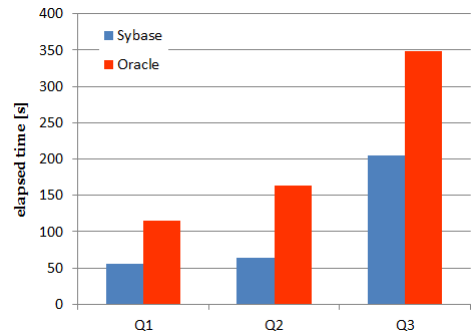
Our experiments (3)



Our experiments (4)

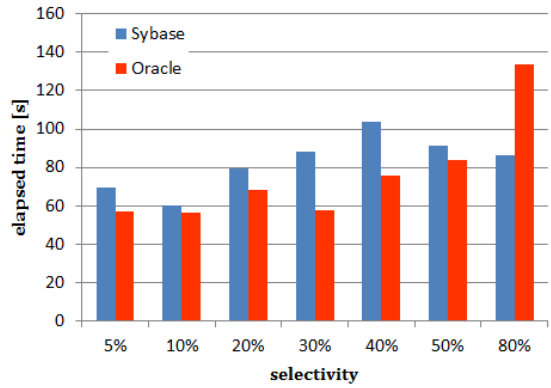
- **Q1: GROUP BY** (productName, regionName)
- **Q2: GROUP BY** (productName, regionName, monthNr)
- **Q3: GROUP BY, 4 dimensions**

```
SELECT sum(a.price), p.productName, r.regionName, m.monthNr, u.userId
FROM   auctions a, products p, cities c, regions r,
       days d, months m, users u
WHERE a.productId = p.productId
AND a.cityId = c.cityId
AND c.regionId = r.regionId
AND a.dateId = d.dateId
AND d.monthId = m.monthId
AND a.userId = u.userId
GROUP BY p.productName,
         r.regionName,
         m.monthNr,
         u.userId;
```



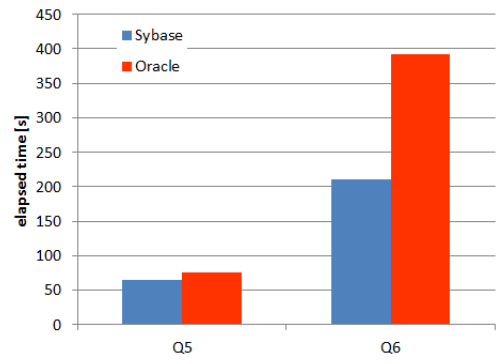
Our experiments (5)

- **Q4: GROUP BY** (productName, regionName, monthNr)
  - **variable selectivity: 5, 10, 20, 30, 40, 50, 80%**, on attribute City and Date



Our experiments (6)

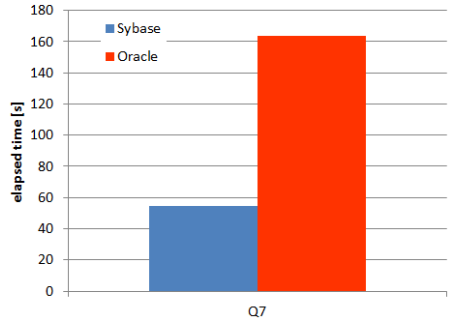
- **Q5: GROUP BY ROLLUP** (productName, regionName)
- **Q6: GROUP BY ROLLUP** (productName, countryName, monthNr)



Our experiments (7)

- **Q7: one table query**

```
SELECT sum(a.price), a.productId, a.cityId, a.dateId
FROM Auctions a
GROUP BY productId, cityId, dateId;
```



Materialization - SMA

➲ SMA - Small Materialized Aggregates

- G. Moerkotte: Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. VLDB, 1998
- **disk data are divided into buckets**
- **every bucket has associated SMA**

BZWBK	2006	Mar	31	148,00	147,00	148,00
BZWBK	2006	Mar	30	149,00	147,50	150,50
BZWBK	2006	Mar	29	148,50	146,50	148,50
BZWBK	2006	Mar	28	150,00	148,00	150,00
BZWBK	2006	Mar	27	147,50	147,50	150,00

SMA bucket1
TimeKey max: 31.03.2006
TimeKey min: 27.03.2006
count: 5

BZWBK	2006	Mar	24	148,00	142,00	150,00
BZWBK	2006	Mar	23	148,00	147,50	150,00
BZWBK	2006	Mar	22	147,00	145,50	150,00
BZWBK	2006	Mar	21	148,50	147,00	150,00
BZWBK	2006	Mar	20	148,50	147,00	151,50

SMA bucket2
TimeKey max: 24.03.2006
TimeKey min: 20.03.2006
count: 5

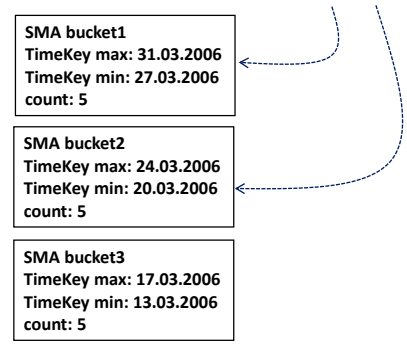
BZWBK	2006	Mar	17	151,50	148,00	152,00
BZWBK	2006	Mar	16	150,00	149,50	152,50
BZWBK	2006	Mar	15	151,50	149,00	152,00
BZWBK	2006	Mar	14	149,00	148,50	151,50
BZWBK	2006	Mar	13	152,50	146,00	152,50

SMA bucket3
TimeKey max: 17.03.2006
TimeKey min: 13.03.2006
count: 5

SMA

- SMA
  - defined on an ordering attribute
  - used for filtering buckets

```
SELECT... FROM... WHERE TimeKey > '22-Mar-2006'
```



Zone Map - IBM Netezza

- ZM - Zone Maps
  - similar to SMA
  - data stored in extents (zones)
  - for a given attribute, ZM stores MIN and MAX value of an attribute in an extent
  - created and maintained automatically for every extent

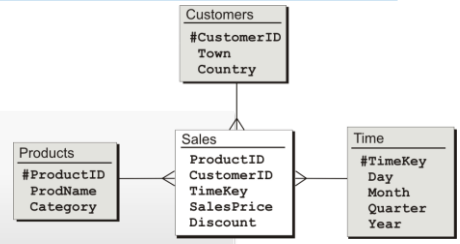
Materialized query

- The result of a query persistently stored in a database
  - table (naive approach)
  - additional functionality
    - refreshing
    - query rewriting
  - other names
    - materialized view (Oracle, IBM Netezza)
    - materialized query table/ summary table (DB2)
    - indexed view (SQL Server)

MV refreshing

- Refreshing moment
  - immediate
  - deferred
    - automatic (with a defined frequency)
    - manual
- Refreshing mode
  - full
  - incremental
    - detecting changes in source tables
    - propagating the changes into a MV

MV - example Oracle (1)




```
create materialized view SalesMV1
build immediate
refresh force
with rowid
as
select ProdName, Category, Country, Month, Quarter,
       sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by ProdName, Category, Country, Month, Quarter;
```

```
alter materialized view SalesMV1 enable query rewrite;
```


MV - example Oracle (2)

```
create materialized view SalesMV1
...
select ProdName, Category, Country, Month, Quarter, Year,
       sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by ProdName, Category, Country, Month, Quarter, Year;
```

```
select Category, Country, Quarter, sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by Category, Country, Quarter;
```

```
select Category, Country, Quarter, sum(SumSales)
from salesMV1
group by Category, Country, Quarter;
```

MV - example Oracle (3)

```
select Category, Country, Quarter, sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by Category, Country, Quarter;
```

query rewriting

```
0   SELECT STATEMENT Optimizer=ALL_ROWS (Cost=4 Card=170 Bytes=7 140)
1 0   HASH (GROUP BY) (Cost=4 Card=170 Bytes=7140)
2 1   MAT_VIEW REWRITE ACCESS (FULL) OF 'SALESMV1' (MAT_VIEW REWRITE)
                                        (Cost=3 Card=170 Bytes=7140)
```


MV - example Oracle (4)

```
create materialized view SalesMV2
...
select ProductID, Category, Country, Month, Quarter, Year,
       sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by ProductID, Category, Country, Month, Quarter, Year;
```

```
select ProdName, Category, Country, Year, sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by Category, Country, Year;
```

ProductID → ProdName

query rewriting
join-back

```
0 SELECT STATEMENT Optimizer=ALL_ROWS (Cost=8 Card=175 Bytes=1 2425)
1 0 HASH (GROUP BY) (Cost=8 Card=175 Bytes=12425)
2 1   HASH JOIN (Cost=7 Card=175 Bytes =12425)
3 2     TABLE ACCESS (FULL) OF 'PRODUCTS' (TABLE) (Cost=3 Card=162 Bytes=4374)
4 2     MAT_VIEW REWRITE ACCESS (FULL) OF 'SALESMV2' (MAT_VIEW REWRITE)
                    (Cost=3 Card=175 Bytes=7700)
```

MV - example Oracle (5)

- For incremental refreshing MV
  - MV log on all tables used in the definition of the MV

```
create materialized view SalesMV1
build immediate
refresh force
with rowid
as
select ProdName, Category, Country, Month, Quarter, sum(SalesPrice) as SumSales
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
group by ProdName, Category, Country, Month, Quarter;
```

```
create materialized view log on Sales
with rowid, primary key
including new values;
```

...

```
create materialized view log on Products
with rowid, primary key
including new values;
```


MV example DB2 (1)

- Maintained by
  - user: maintained by user clause
  - system (default): maintained by system clause
    - either automatic or non-automatic refreshing mode is available
    - automatic mode (refresh immediate clause) ⇨ a MQT is refreshed automatically as the result of changes in the content of its base tables
      - automatic mode requires that a unique key from each base table is included the MQT
    - non-automatic (refresh deferred clause) ⇨ a MQT has to be refreshed by explicit execution of:

```
refresh table TableName {incremental|not incremental}
```

MV - example DB2 (2)

```
create table YearlySalesMV2
as
(select ProdID, ProdName, Year,
        sum(salesPrice) as SumSales
from Sales s, Products p, Time t
where s.ProductID=p.ProductID
and s.TimeKey=t.TimeKey
and t.Year=2009
group by ProdID, ProdName, Year)
data initially immediate
refresh immediate
maintained by system
enable query optimization;
```


MV - example DB2 (3)

- **For incremental refreshing**
  - **MV log ⇨ staging table**

```
create table YearlySalesMV3_ST for YearlySalesMV3
propagate immediate
set integrity for YearlySalesMV3
      staging
      immediate unchecked
```

allows to contol how integrity constraints are checked for a MV

integrity constraints are not checked after refreshing the MV

MV - example Netezza (1)

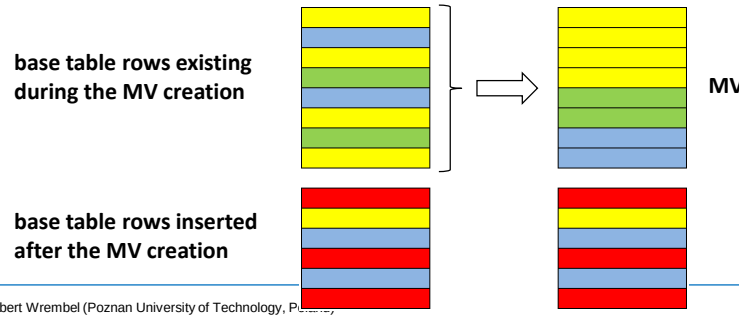
- Used for query rewriting
- Stored as a table

```
CREATE MATERIALIZED VIEW v-name AS
SELECT ... FROM tab-name [ORDER BY ...]
```

- Some restrictions
  - only one table in the FROM clause
  - no WHERE clause
  - the columns in the projection list must be columns ⇨ not allowed expressions (aggregates, mathematical operators, SQL functions, DISTINCT, ...)
  - if ORDER BY is used, then it must use columns in the projection list

MV - example Netezza (2)

- Inserting rows into a base table
  - new rows are appended to the MV ⇨ two areas in the MV:
    - the sorted records generated when the view was created
    - the unsorted records that have been inserted into the base table after the MV was created
  - resorting while refreshing



MV - example Netezza (3)

- Suspending MV ⇨ making it inactive
- Refreshing MV
  - manually ⇨ the REFRESH option
  - automatically ⇨ setting a refresh threshold
    - the threshold specifies the percentage of unsorted data in the materialized view, value from 1 to 99 (default 20)
    - the thresholds allows to refresh all the materialized views associated with a base table

```
ALTER VIEW MV-name MATERIALIZE {REFRESH | SUSPEND}
```

- The system creates zone maps for all columns in a MV that have data types integer, date, or timestamp

MV example - SQL Server

- MV is created by creating a unique clustered index on a view (clustering data by the value of the indexed column)
- The index causes that the view is materialized

```
create view YearlySalesMV
with schemabinding
as
select ProdID, ProdName, Year, sum(salesPrice) as SumSales
from Sales s, Products p, Time t
where s.ProductID=p.ProductID
and s.TimeKey=t.TimeKey
and t.Year=2009
group by ProdID, ProdName, Year
```

prevents from modifying base tables' schemas as long as the view exists

```
create unique clustered index Indx_ProdID
on YearlySalesMV(ProdID, ProdName, Year)
```

MV - SQL Server

- Query rewriting: MV must be explicitly referenced in a query with noexpand

```
select Column1, Column2, ...
from Table, IndexedView with (noexpand)
where ...
```

- Refreshing: immediate and incremental

References

- Column storage
  - D.J. Abadi, S.R. Madden, M. Ferreira: Integrating compression and execution in column-oriented database systems. SIGMOD, 2006
  - D.J. Abadi, S.R. Madden, N. Hachem: Column-stores vs. row-stores: how different are they really?. SIGMOD, 2008
  - A. Albano: SADAS - An Innovative Column-Oriented DBMS for Business Intelligence Applications. SADAS Manual
  - S. Harizopoulos, D. Abadi, P. Boncz: Column-Oriented Database Systems. VLDB tutorial, 2009
  - M. Stonebraker, D.J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S.R. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, S. Zdonik: C-store: a column-oriented DBMS. VLDB, 2005

References

- **Small summary data**
  - **G. Graefe: Fast loads and fast queries. DaWaK, 2009**
  - **G. Moerkotte: Small materialized aggregates: A light weight index structure for data warehousing. VLDB, 1998**
  - **IBM Netezza Database User's Guide. IBM Netezza 7.0.x, Oct 2012**
  - **Netezza underground: Zone maps and data power.** https://www.ibm.com/developerworks/community/blogs/Netezza/entry/zone_maps_and_data_power20?lang=en