POZNAN UNIVERSITY OF TECHNOLOGY

Data Warehouse Physical Design: Part II

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
- Partitioning
- MOLAP

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

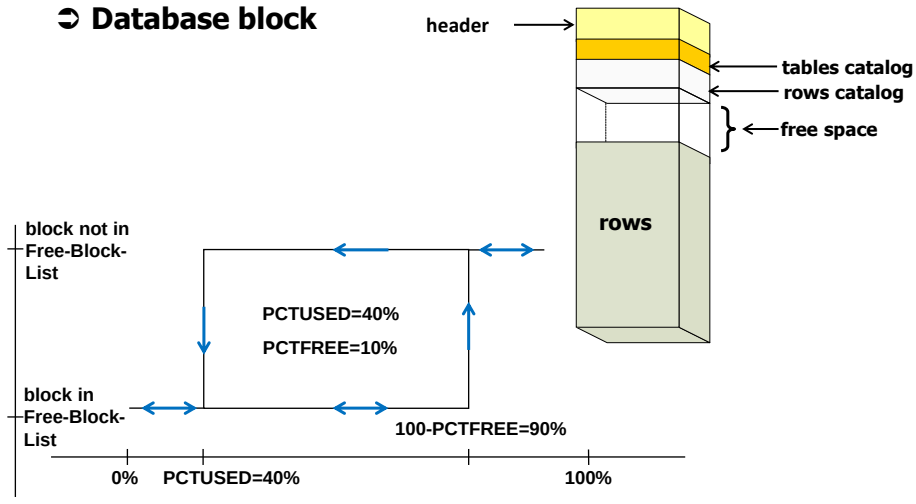
data blocks

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

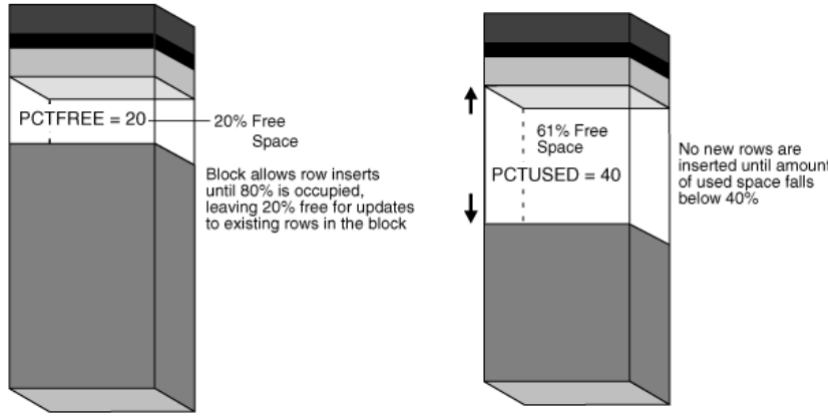
➲ **Rows identified by ROWIDs**

Row storage (2)

➲ **Database block**



Row storage (3)



Column storage (1)

- relational data model
- SQL interface
- every column stored and accessed separately

≠

- key:value data model
- no SQL interface
- columns are clustered ⇨ column family

Model 204 (projection index)
Sybase IQ (Sybase, Inc.)
SADAS (Advanced Systems)
C-Store/Vertica
MonetDB
Infobrigh
...

- HBase
- Hypertable
- Cassandra
- Bigtable
- ...

Column storage (2)

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

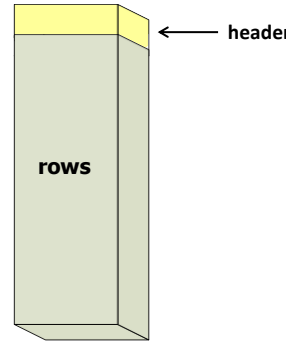
148,00	149,00	148,50	150,00	147,50	148,00	148,00	147,00	148,50
148,50	151,50	150,00	151,50	149,00	152,50	152,00	154,50	164,50
172,50	170,00	169,50	170,50	166,00				

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)

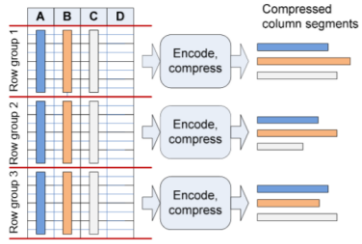
Column storage (3)

- Database block
  - no free space
  - better space utilization

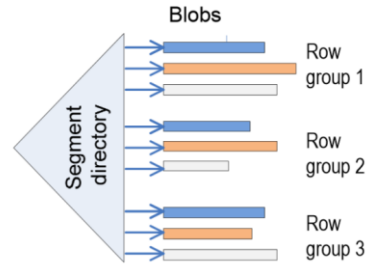


SQL Server

- Ver. 2012 and higher
- Divide rows into row groups of about one million rows each
- Compress each row group independently - dictionary compression for string columns
- Store each column segment as a separate BLOB
  - P.-A. Larson, E. N. Hanson, S. L. Price: Columnar Storage in SQL Server 2012. IEEE Data Eng. Bull. 35(1), 2012



(a) Convert rows to column segment

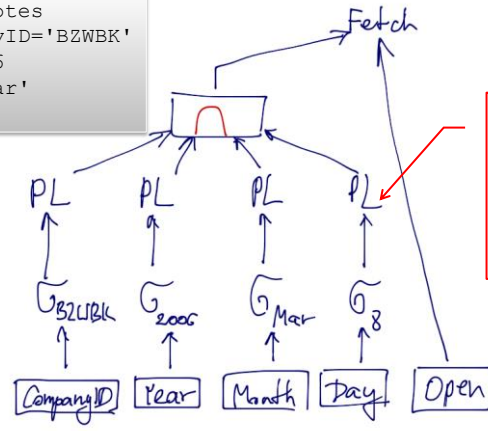


(b) Store segments in blobs

Query processing

- One table query

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

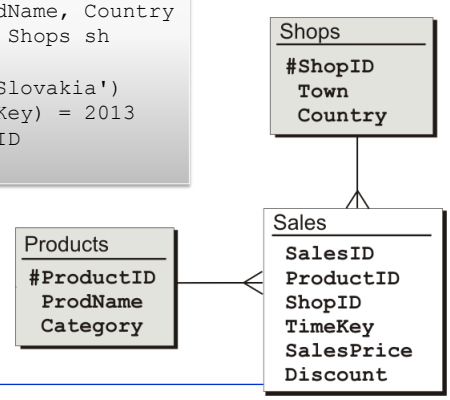


list of positions (slot numbers) of values fulfilling the predicate represented as:
- array or
- bit vector or
- set of position ranges

Query processing

- **Join query**
- **D.J. Abadi, S.R. Madden, N. Hachem: Column-stores vs. row-stores: how different are they really?. SIGMOD, 2008**

```
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**
  - **having ProductIDs and ShopIDs selected previously**
- **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

{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

{100, 200, 500}

Join query

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

2013

{S2, S3, S4, S5}

bitmap
0
1
1
1
1

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

'Poland' or 'Slovakia'

{10, 30}

bitmap
1
0
1
1
0

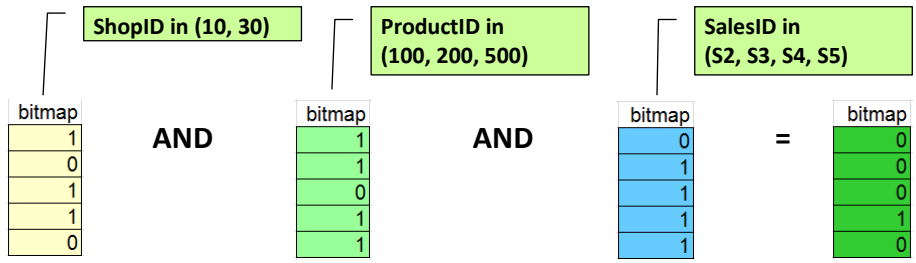
'cheese'

{100, 200, 500}

bitmap
1
1
0
1
1

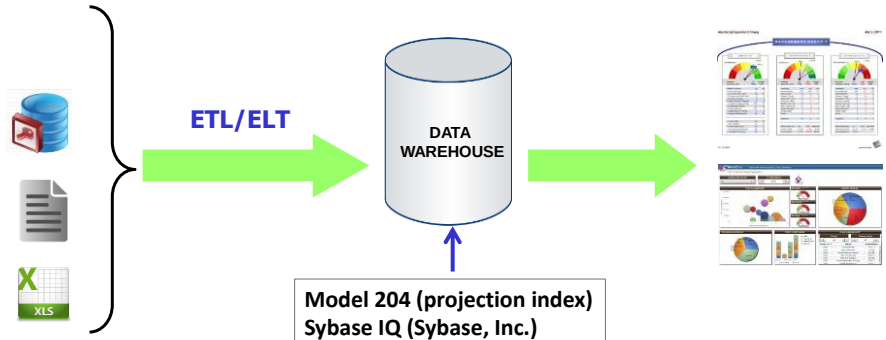
Join query

- Matching Sales rows are represented by bitmaps



- 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

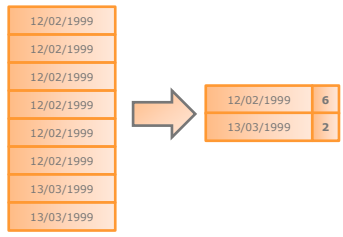
Column storage in DW architecture



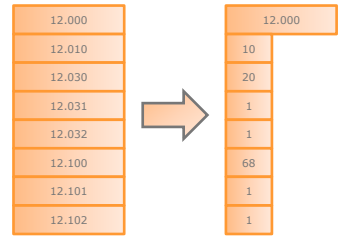
Model 204 (projection index)
Sybase IQ (Sybase, Inc.)
SADAS (Advanced Systems)
C-Store/Vertica
MonetDB
Infobrigh
Exasol
SQL Server 2012, 2014

CS - compression (1)

run-length encoding



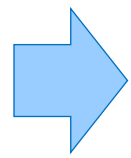
delta encoding



CS - compression (2)

➲ **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

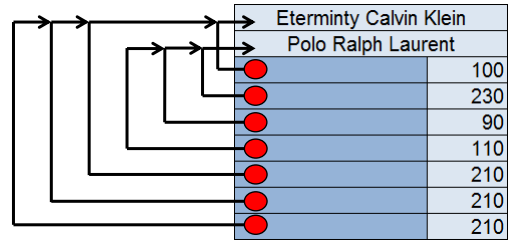
mapping table

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

RS - compression (1)

➲ **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



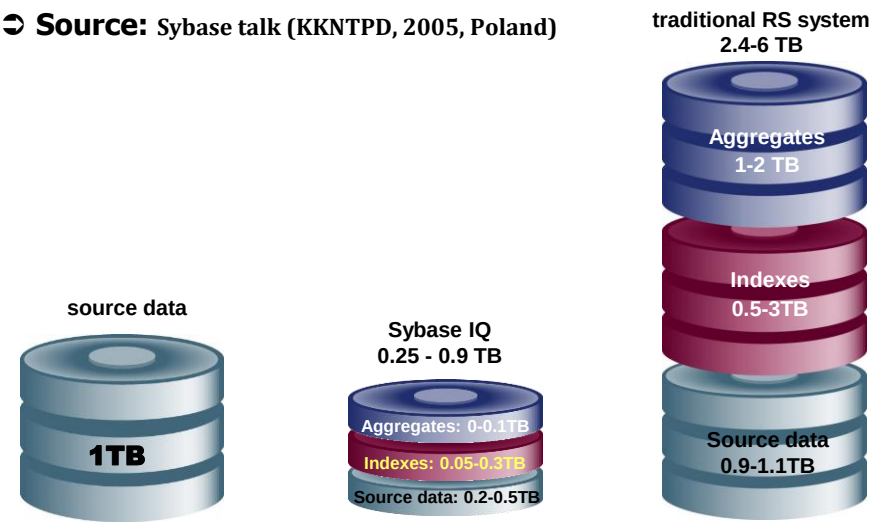
➲ **Dictionary compression (DB2)**

➲ **Like discrete domain encoding**

- **dictionary stored in**
  - **a dedicated table**
  - **data block header**

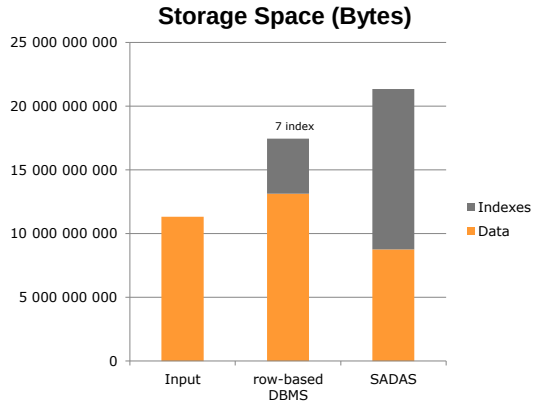
Performance comparison: CS-RS (1)

➲ **Source:** Sybase talk (KKNTPD, 2005, Poland)



Performance comparison: CS-RS (2)

➲ **Source:** presentation by Advanced Systems



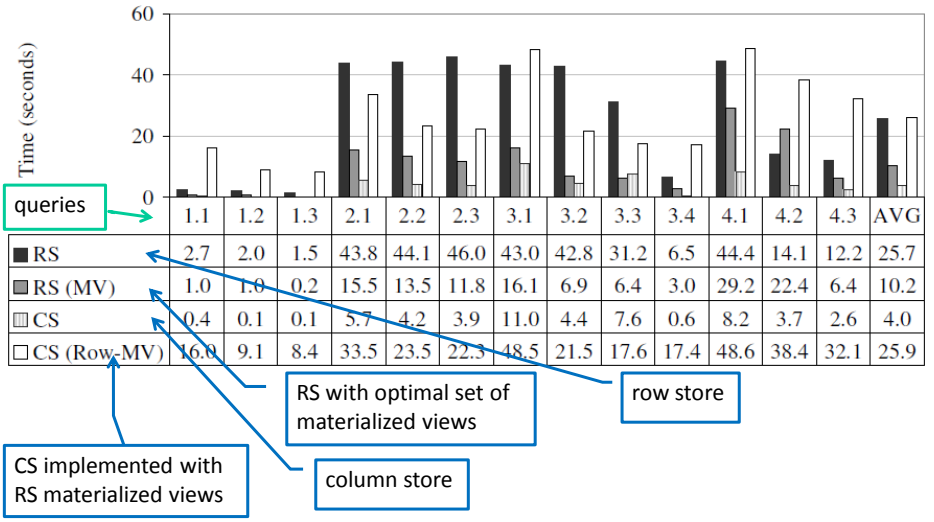
Performance comparison: CS-RS (3)

➲ **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

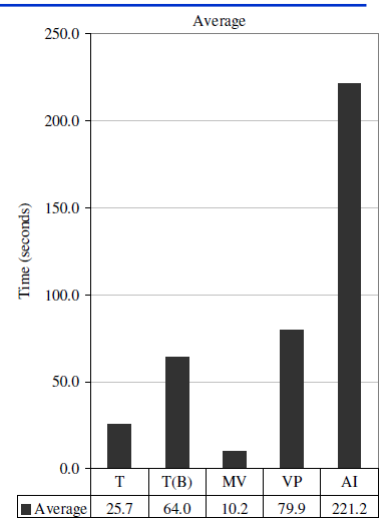
Performance comparison: CS-RS (4)



	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

Performance comparison: CS-RS (5)

- **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

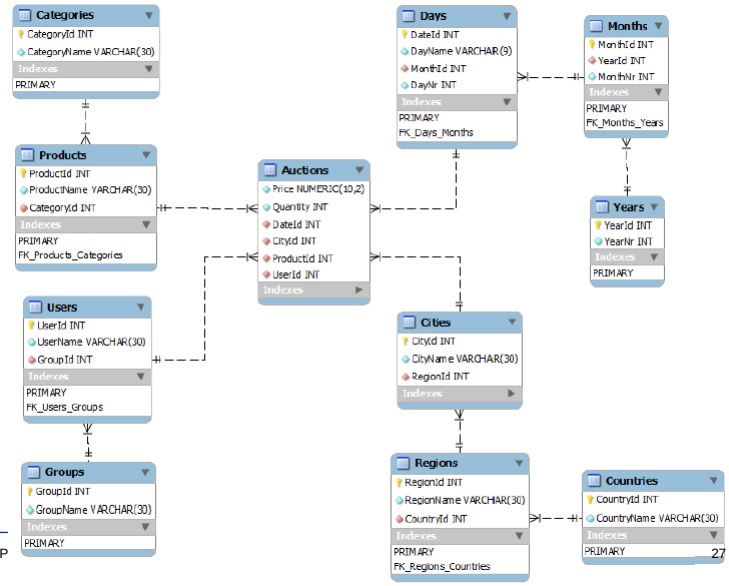
Our experiments (1)

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

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

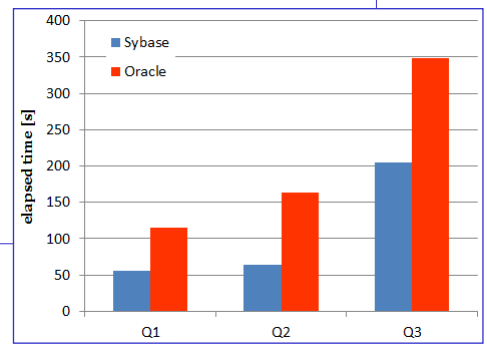
Our experiments (3)



Our experiments (4)

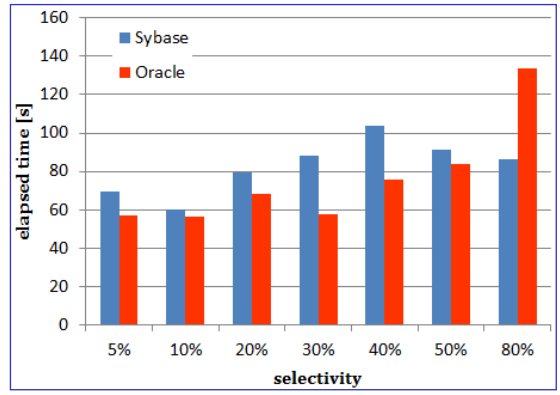
- **Q1: GROUP BY** (productName, regionName)
- **Q2: GROUP BY** (productName, regionName, monthNr)
- **Q3: GROUP BY, 4 dimensions**

```
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;
```



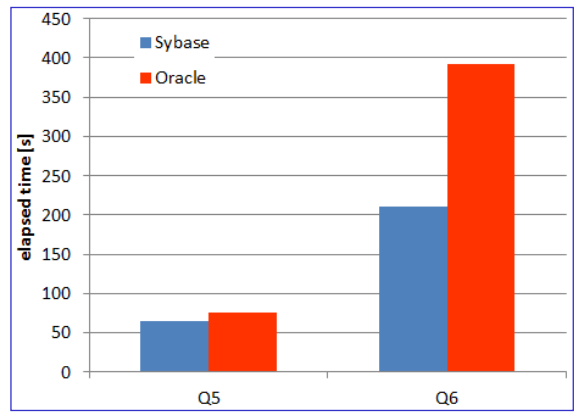
Our experiments (5)

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



Our experiments (6)

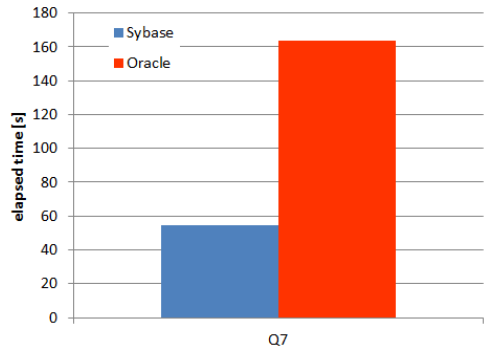
- **Q5: GROUP BY ROLLUP** (productName, regionName)
- **Q6: GROUP BY ROLLUP** (productName, countryName, monthNr)



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, 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
  - e.g. select ... from ... where TimeKey > '22-Mar-2006'

SMA bucket1 TimeKey max: 31.03.2006 TimeKey min: 27.03.2006 count: 5
SMA bucket2 TimeKey max: 24.03.2006 TimeKey min: 20.03.2006 count: 5
SMA bucket3 TimeKey max: 17.03.2006 TimeKey min: 13.03.2006 count: 5

Zone Map - IBM Netezza (1)

- ZM - Zone Maps
  - similar to SMA
  - data stored in extents (zones)
    - an extent is the smallest unit of disk allocation = 3MB
  - created automatically
  - by default created for columns of type integer, date, and timestamp
  - created automatically for columns used in the ORDER BY clause of a materialized view
  - for a given attribute ZMs store MIN and MAX value of the attribute in an extent
  - created for every extent
  - maintained automatically by the system

Zone Map - IBM Netezza (2)

ZM

ext1

Index	Date	Open	Close
BZWBK	21-03-2006	148,50	147,00
BZWBK	20-03-2006	148,50	147,00
BZWBK	17-03-2006	151,50	148,00
BZWBK	16-03-2006	150,00	149,50

Index		Date		Open		Close	
Min	Max	Min	Max	Min	Max	Min	Max
BZWBK	BZWBK	16-03-2006	21-03-2006	148,50	151,50	147,00	149,50

ext2

BZWBK	15-03-2006	151,50	149,00
BZWBK	14-03-2006	149,00	148,50
BZWBK	13-03-2006	152,50	146,00
BZWBK	10-03-2006	152,00	146,00
BZWBK	9-03-2006	154,50	154,00

Index		Date		Open		Close	
Min	Max	Min	Max	Min	Max	Min	Max
BZWBK	BZWBK	9-03-2006	15-03-2006	149,00	154,50	146,00	154,00

ext3

BZWBK	6-03-2006	170,00	168,00
BZWBK	5-03-2006	169,50	163,50
BZWBK	2-03-2006	170,50	168,00
BZWBK	1-03-2006	166,00	165,00

Index		Date		Open		Close	
Min	Max	Min	Max	Min	Max	Min	Max
BZWBK	BZWBK	1-03-2006	6-03-2006	166,00	170,50	163,50	168,00

Zone Filters (1)

- Zone filters, Bit vector filters, Zone indexes (G. Graefe, DAWAK, 2009)
- ZF - Zone Filter similar to SMA and ZM
  - ZF maintained for each zone and attribute
  - ZF stores m consecutive MIN and MAX values
    - if m=1 then ZF equivalent to ZM
    - if m=1 then either MIN or MAX can be NULL ⇨ not useful for filtering zones
    - if m=2 then in the presence of NULLs the second value of MIN or MAX is NOT NULL

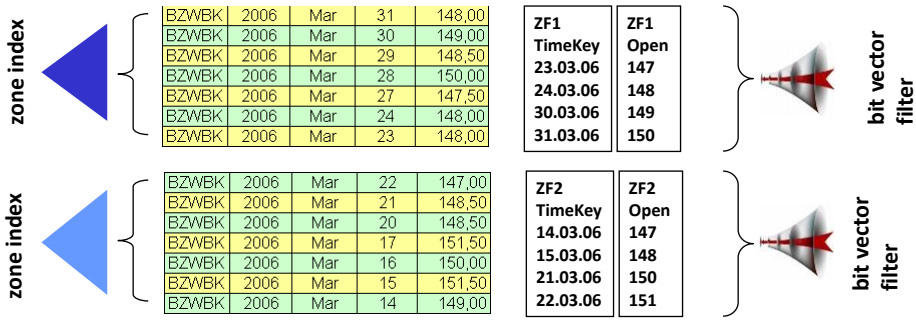
Zone Filters (2)

- Example
  - m=3
  - MIN (TimeKey)={'01-Feb-2013', '04-Feb-2013', '07-Feb-2013'}
  - query: WHERE TimeKey='02-Feb-2013' ⇨ the zone can be skipped

Zone Filters (3)

- BVF - Bit Vector Filter
  - maintained for each zone and for each column
  - provides a synopsis of the actual values
- ZI - Zone Index
  - supports searching within a zone
  - a dedicated ZI maintained for a zone

Materialization (6)

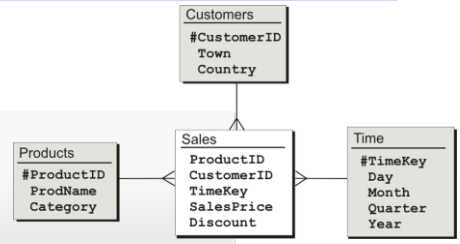


- **Application in queries**
  - **select zones by means of BVFs**
  - **find rows within zones by means of ZIs**

Materialized query

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

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 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; this refreshing 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
```


MV - example Netezza (1)

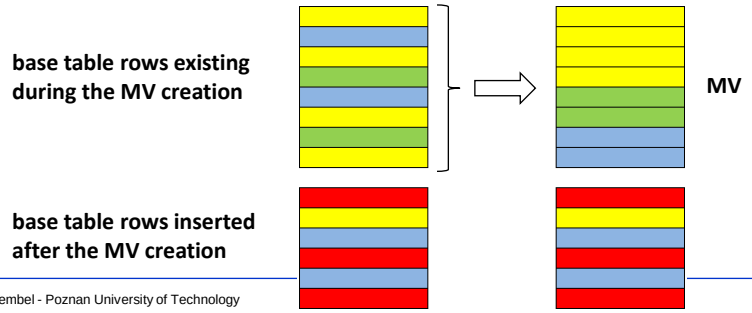
- Used for query rewriting
- Stored as a table
- Divided into data slices that are co-located on the same disk as the corresponding base table data slices

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

- Some restrictions
  - only one table in the FROM clause
  - the WHERE clause cannot be used
  - the columns in the projection list must be columns ⇨ not allowed expressions (aggregates, mathematical operators, SQL functions, DISTINCT, ...)
  - the columns in the optional ORDER BY clause must be one or more 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 by manual 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 YearlySalesMV2
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

MV refreshing

- Refreshing time
  - immediate
  - deferred
    - automatic (with a defined frequency)
    - manual
- Refreshing mode (A. Gupta, I.S. Mumick, MIT Press, 1999)
  - full
  - incremental
    - detecting changes in source tables
    - propagating the changes into a MV
- Querying MVs during their refreshing ⇨ assuring data consistency (Zhuge et. al., SIGMOD, 2005)
  - compensation algorithm
  - versions of data

MV design

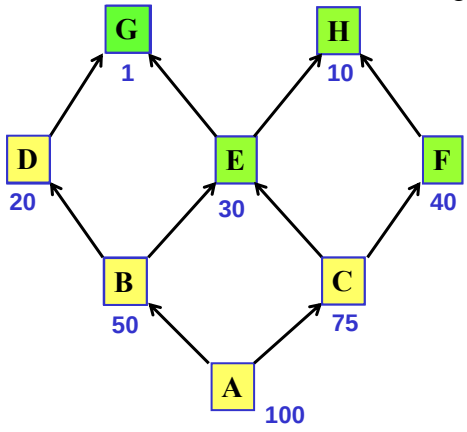
- Designing the optimal set of MVs
- Typically ⇨ for a given query workload
- Constraints
  - minimizing response time for the largest number of queries
  - minimizing response time for the most expensive queries
  - minimizing costs of refreshing MVs
  - minimizing disk space
- Physical design advisors

Greedy algorithm

- Harinarayan V., Rajaraman A., Ullman J.: Implementing Data Cubes Efficiently. SIGMOD, 1996
- **Assumptions**
  - the data size of every query is known (estimated)
  - a query execution cost is represented by the data volume (# of rows) that is read by the query
- **Goal**
  - minimize the data volume read by the queries with a given number of materialized views

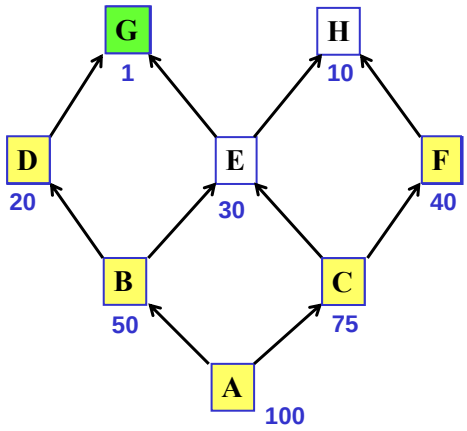
Execution (1)

- Node ⇨ query
- Arc ⇨ the possibility of computing an upper-level query based on a lower-level query



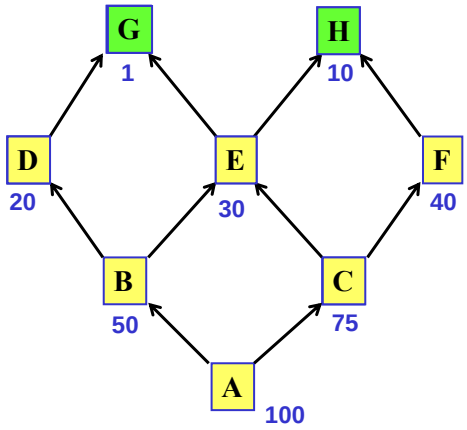
- **Run 1**
  - B: 50x5=250
  - C: 25x5=125
  - D: 80x2=160
  - E: 70x3=210
  - F: 60x2=120
  - G: 99x1
  - H: 90x1

Execution (2)



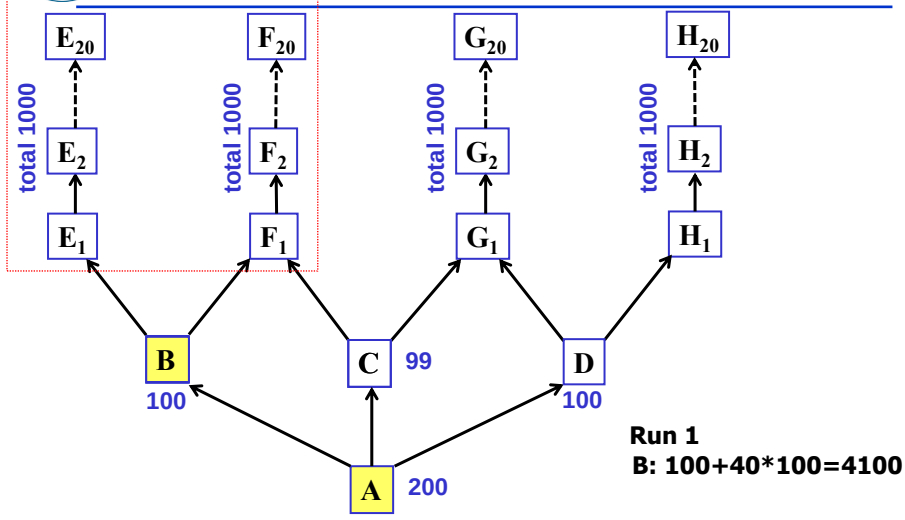
- Run 2
  - C: 25x2=50 (C and F profit; H, E, G computed from B)
  - D: 30x2=60
  - E: 20x3=60
  - F: 60+10 (B-F)=70
  - G: 49
  - H: 40 (H computed from B)

Execution (3)

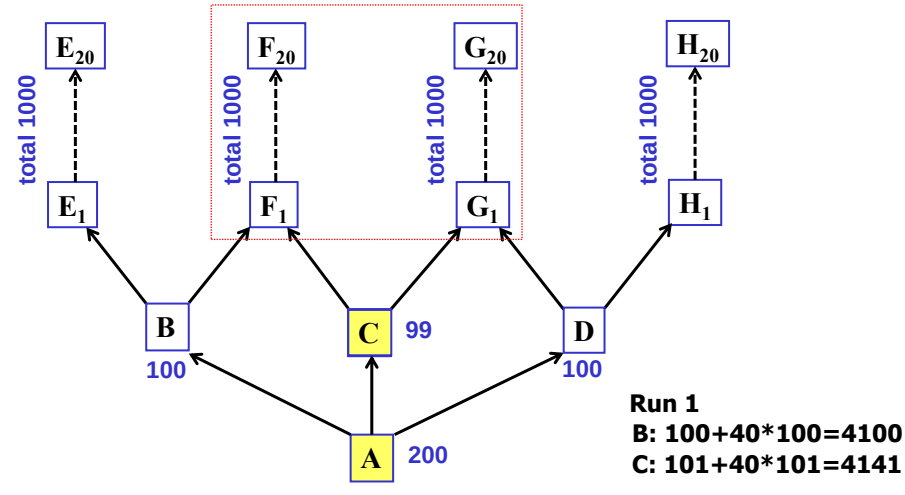


- Run 3
  - C: 25=25 (C profits; E, G computed from B)
  - D: 30x2=60
  - E: 20+20+10 (E-F)=50
  - G: 49
  - H: 30

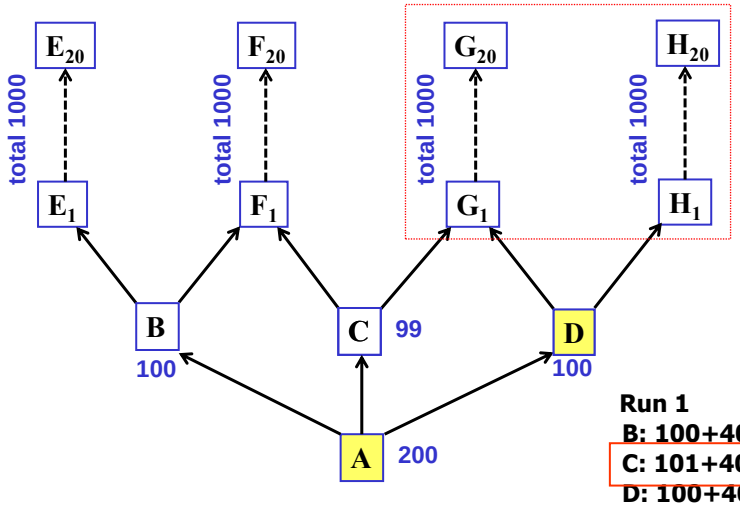
Execution (4)



Execution (5)

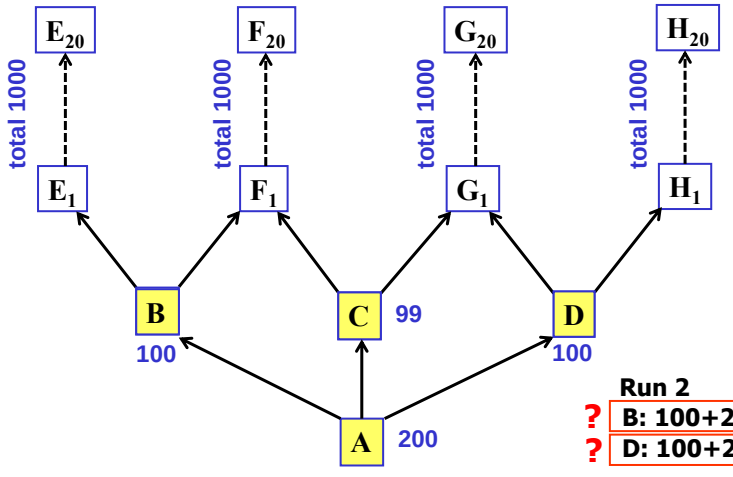


Execution (6)



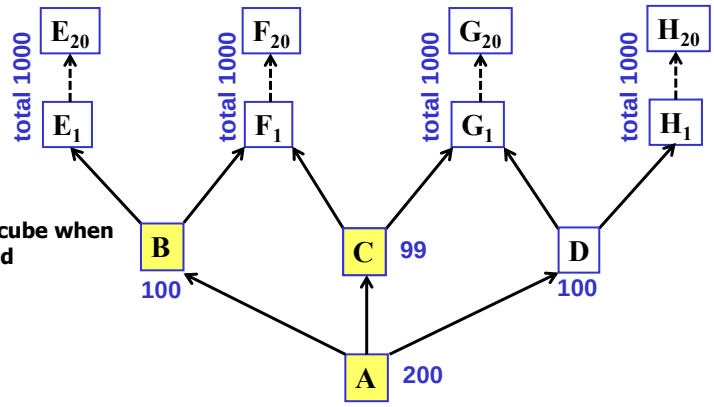
Run 1
B: 100+40*100=4100
C: 101+40*101=4141
D: 100+40*100=4100

Execution (7)



Run 2
? B: 100+20*100=2100
? D: 100+20*100=2100

Final solution



Reading the whole cube when ABC are materialized

A: 200

B: 100

C: 99

D: 200

E: 20*100

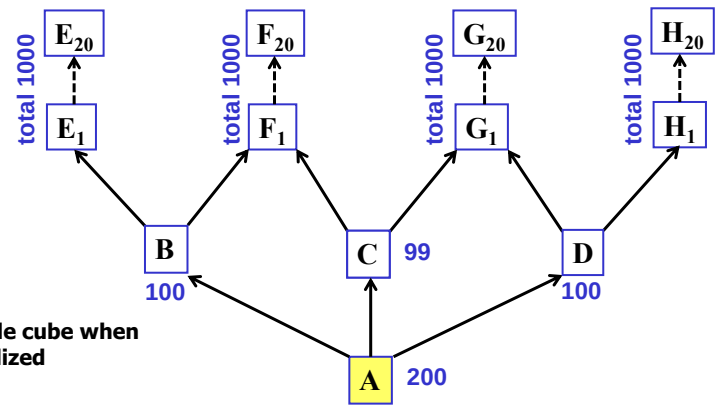
F: 20*99

G: 20*99

H: 20*200

$Cost_{MatABC}$=10559

Final solution is not optimal



Reading the whole cube when only A is materialized

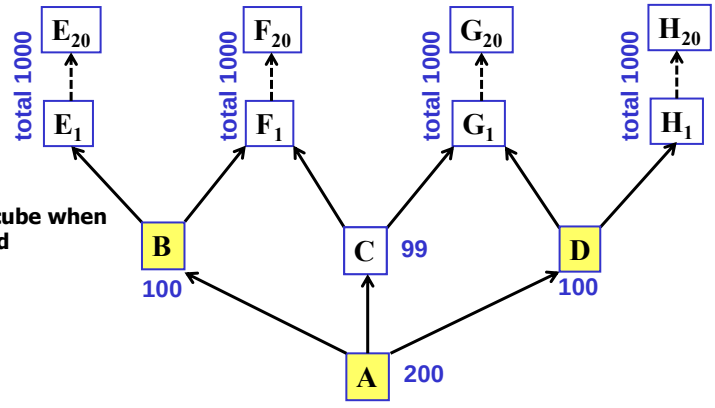
A: 200

B, C, D: 3*200

E, F, G, H: 4*20*200

$Cost_{MatA}$=16800

Final solution is not optimal



Reading the whole cube when ABD are materialized

A: 200
B: 100
C: 200
D: 100
E: 20*100
F: 20*100
G: 20*100
H: 20*100

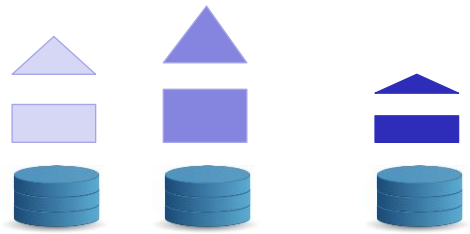
$Cost_{MatABD}=8600$

Greedy algorithm

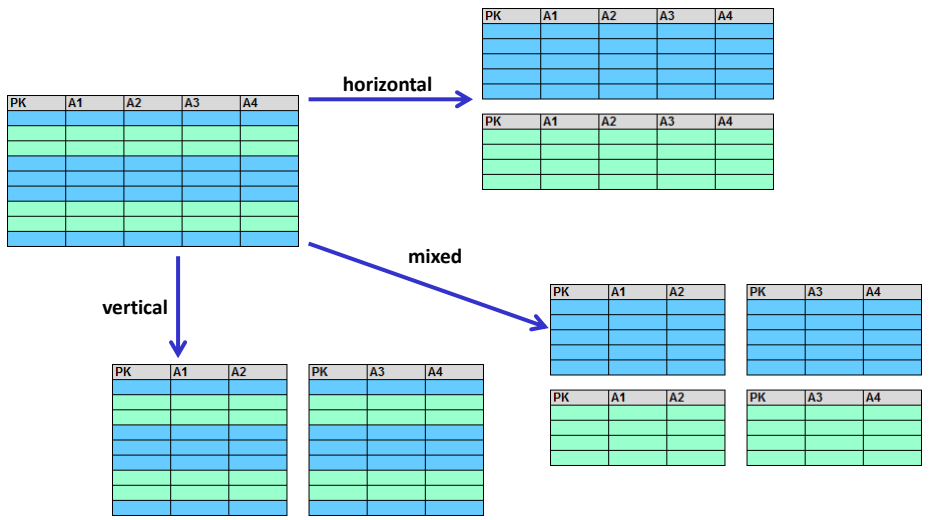
- **Open issues**
  - **which node to select if the materialization of N nodes offers the same greatest profit?**
  - **assumption that the probability of executing all queries in the lattice is the same**
  - **costs of refreshing materialized views were not considered**

Partitioning (1)

- A mechanism of dividing a table or index into smaller parts ⇨ partitions
- The most benefit from partitioning is achieved if every partition is stored on a separate disc ⇨ parallel disc scans



Partitioning (2)



Partitioning (3)

- Correctness criteria
- Completeness ⇨ when table T is partitioned into P_1, P_2, ..., P_n then every row from T or its fragment must be stored in one of these partitions
  - guarantees that after partitioning no data will disappear
- Disjointness ⇨ when table T is partitioned into P_1, P_2, ..., P_n then every row or its fragment from T must be stored in exactly one partition
  - guarantees that partitioning does not create data redundancy
- Reconstruction ⇨ there must be a mechanism of reconstructing original table T from its partitions

Partitioning (4)

- In horizontal partitioning rows are divided into subsets based on the value of partitioning attribute(s)
- Horizontal partitioning techniques
  - hash
  - range-based
  - set-based
  - round-robin

Example Oracle (1)

➲ **Range partitioning**

```
create table Sales_Range_TKey
(ProductID varchar2(8) not null references Products(ProductID),
 TimeKey date not null references time(TimeKey),
 CustomerID varchar2(10) not null references Customers(CustomerID),
 SalesPrice number(6,2))
PARTITION by RANGE (TimeKey)
(partition Sales_1Q_2009
   values less than (TO_DATE('01-04-2009', 'DD-MM-YYYY'))
   tablespace Data01,
 partition Sales_2Q_2009
   values less than (TO_DATE('01-07-2009', 'DD-MM-YYYY'))
   tablespace Data02,
 partition Sales_3Q_2009
   values less than (TO_DATE('01-10-2009', 'DD-MM-YYYY'))
   tablespace Data03,
 partition Sales_4Q_2009
   values less than (TO_DATE('01-01-2010', 'DD-MM-YYYY'))
   tablespace Data04,
 partition Sales_Others
   values less than (MAXVALUE) tablespace Data05);
```


Example Oracle (2)

➲ **List partitioning**

```
create table Sales_List_PayType
(ProductID varchar2(8) not null references Products(ProductID),
 TimeKey date not null references time(TimeKey),
 CustomerID varchar2(10) not null references Customers(CustomerID),
 SalesPrice number(6,2), PaymentType varchar(2))
PARTITION by LIST (PaymentType)
(partition Sales_Credit_Debit values ('Cr','De') tablespace Data01,
 partition Sales_Cash values ('Ca') tablespace Data02,
 partition Sales_Others values (DEFAULT) tablespace Data05);
```

➲ **Hash partitioning**

```
...
PARTITION by HASH (CustomerID)
(partition Cust1 tablespace Data01,
 partition Cust2 tablespace Data02));
```

Example Oracle (3)

- Other types of partitioning
  - virtual column ⇨ based on expression on partitioning attribute(s)
  - system ⇨ records placement in partitions controlled by an application
  - reference ⇨ partitioning FK tables according to a partitioning schema of their PK table
  - composite ⇨ partitions with subpartitions

Example DB2

- Range partitioning

```
create table Sales_Range_TKey
(ProductID varchar2(8) , ...)
PARTITION BY RANGE(TimeKey)
(partition Sales_1Q_2009 starting '01-01-2009',
 partition Sales_2Q_2009 starting '01-04-2009',
 partition Sales_3Q_2009 starting '01-07-2009',
 partition Sales_4Q_2009 starting '01-10-2009' ending '31-12-2009')
```

Example SQL Server

- Partition function ⇨ defines the number of partitions for a table and ranges of values for every partition
- Partition scheme ⇨ defines storage locations for table partitions

```
create PARTITION FUNCTION PF_Sales_Range_TKey (datetime)
as RANGE right for values
('20090401', '20090701', '20091001', '20100101');
```

date < 2009-04-01
2009-04-01 <= date <2009-07-01
2009-07-01 <= date <2009-10-01
2009-10-01 <= date <2010-01-01
date >=2010-01-01

```
create PARTITION SCHEME PS_Sales_Range_TKey
as partition PF_Sales_Range_TKey
to (Data01, Data02, Data03, Data04, Data05);
```

filegroup

```
create table T_Sales_Range_TKey
(ProductID varchar(8),
TimeKey datetime ...)
on PS_Sales_Range_TKey (TimeKey)
```


Part. tables in queries

```
CREATE TABLE Sales1
(...)
PARTITION by RANGE (TimeKey)
  (partition Sales_Jan2009
         values less than (TO_DATE('01-02-2009', 'DD-MM-YYYY')),
   partition Sales_Feb2009
         values less than (TO_DATE('01-03-2009', 'DD-MM-YYYY')),
   partition Sales_Mar2009
         values less than (TO_DATE('01-04-2009', 'DD-MM-YYYY')),
   partition Sales_Apr2009
         values less than (TO_DATE('01-05-2009', 'DD-MM-YYYY')));
```

```
select * from sales1
where TimeKey between to_date('01-01-2009', 'DD-MM-YYYY') and
to_date('31-01-2009', 'DD-MM-YYYY');
```

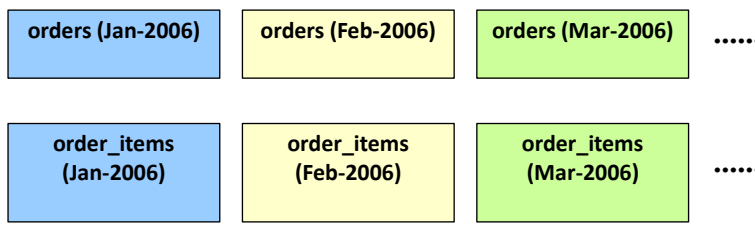
```
0 SELECT STATEMENT Optimizer=ALL_ROWS (Cost=17 Card=7505 Bytes =562875)
1 0  PARTITION RANGE (SINGLE) (Cost=17 Card=7505 Bytes=562875)
2 1     TABLE ACCESS (FULL) OF 'SALES1' (TABLE) (Cost=17 Card=7505 Bytes=562875)
```

Partition-wise join (1)

```
CREATE TABLE orders
(order_id NUMBER(12) NOT NULL PRIMARY KEY,
 order_date DATE NOT NULL, ...)
PARTITION BY RANGE (order_date)
(PARTITION p_2006_jan VALUES LESS THAN (TO_DATE('01-FEB-2006','dd-MON-yyyy')),
 PARTITION p_2006_feb VALUES LESS THAN (TO_DATE('01-MAR-2006','dd-MON-yyyy')),
 PARTITION p_2006_mar VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy')),
 PARTITION p_2006_apr VALUES LESS THAN (TO_DATE('01-MAY-2006','dd-MON-yyyy')),
 .....
 PARTITION p_2006_dec VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy')))
```

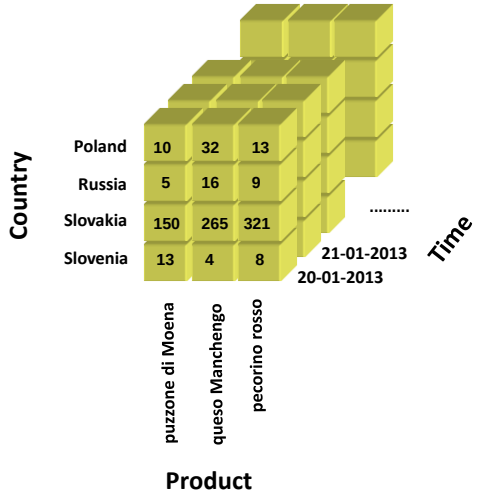
```
CREATE TABLE order_items
(order_id NUMBER(12) NOT NULL, ...,
 CONSTRAINT order_items_orders_fk FOREIGN KEY (order_id)
            REFERENCES orders(order_id))
PARTITION BY REFERENCE (order_items_orders_fk)
```


Partition-wise join (2)




```
SELECT o.order_date , sum(oi.sales_amount) sum_sales
FROM orders o , order_items oi
WHERE o.order_id = oi.order_id
AND o.order_date BETWEEN TO_DATE('01-FEB-2006','DD-MON-YYYY')
AND TO_DATE('31-MAR-2006','DD-MON-YYYY')
GROUP BY o.order_id , o.order_date
ORDER BY o.order_date;
```

MOLAP (1)



- **Operations**
  - drill-down / roll-up
  - slice, dice
  - rotate (pivot)
  - drill-accross
  - drill-through

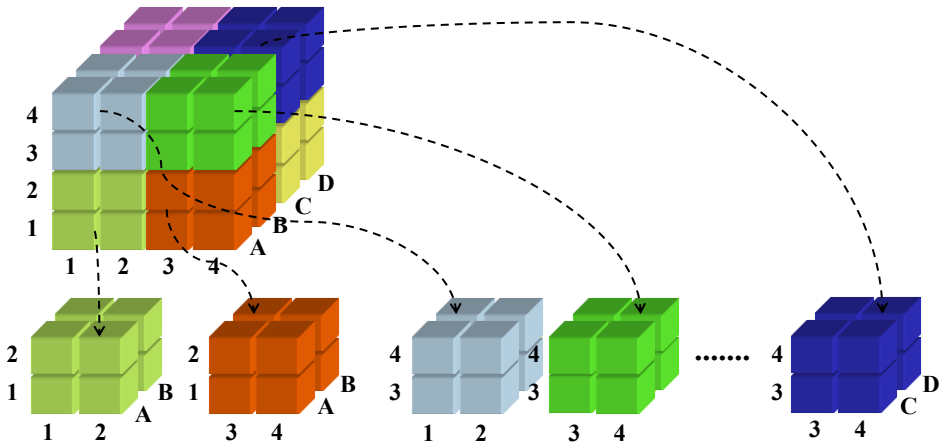
MOLAP (2)

➲ **Implementations**
- N-dim array
- Hash table (SQL Server)
- BLOB (Oracle)
- Quad-tree
- K-D-tree

➲ **Systems**
- Cognos PowerPlay (IBM)
- Oracle OLAP DML
- Hyperion Essbase (Oracle)
- MicroStrategy
- MS Analysis Services (Microsoft)
- SAS OLAP Server

MOLAP (3)

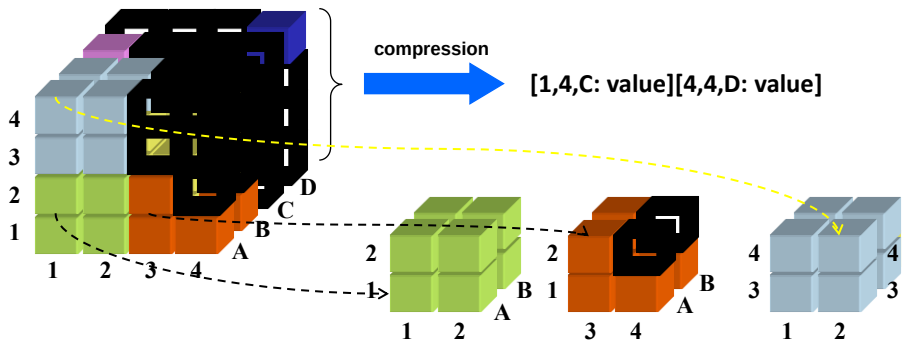
➲ **Cube chunking**



MOLAP (4)

➲ **Compression - store cells that contain NOT NULL values**

- **compress when % of NULL cells reaches a given threshold (e.g., 40%)**



MOLAP (5)

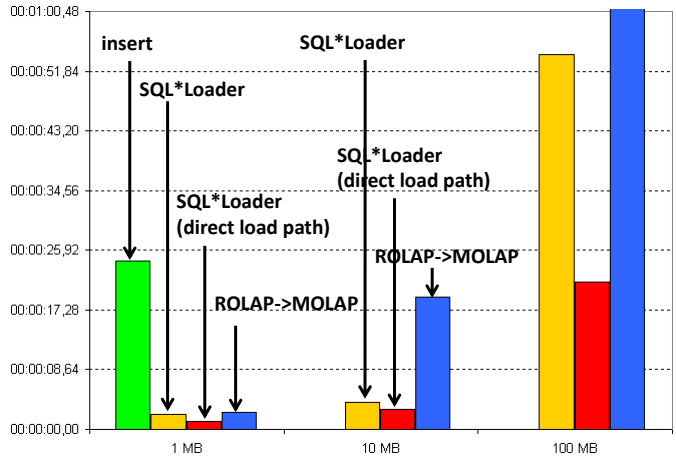
- More efficient than ROLAP for aggregate computing
- Efficient when a cube contains a few dimension
- Loading less efficient than ROLAP

Our experiment (1)

- Oracle
- ROLAP
  - TPC-H benchmark
  - B-tree indexes on PKs and FKs
- MOLAP
  - 4 cubes
  - Discounts(Parts, Orders, ShipTime)
  - Discounts(Parts, Orders, ReceipTime)
  - Quantitites(Parts, Orders, ShipTime)
  - Prices(Parts, Orders, ShipTime)

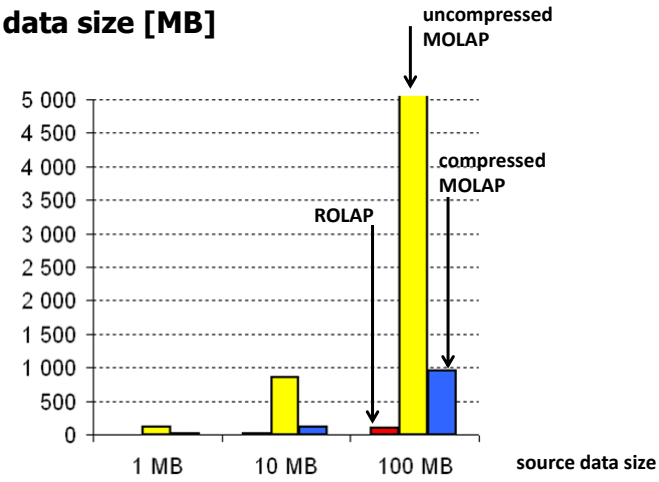
Our experiment (2)

➲ **Load time [hh:mi:ss]**

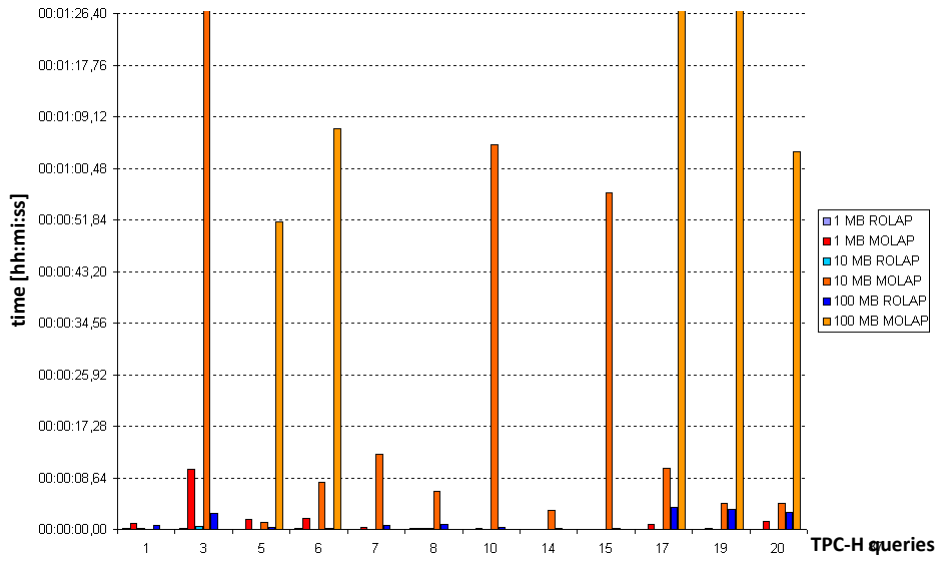


Our experiment (3)

➲ **MOLAP data size [MB]**



Our experiment (4)



Our experiment (5)

- **Remarks**
  - **MOLAP may not always be more efficient than ROLAP**
  - **ROLAP may take advantage of**
    - **bitmap indexes, bitmap join indexes, materialized views, partitioning**
    - **sophisticated query optimizer**
  - **the presented results shouldn't be generalized ⇨ they show the characteristics of MOLAP implementation in a particular version of the system**

Other DW technologies

- Parallel and distributed DWs
  - data (partitions, MVs, indexes) allocation in nodes
  - load balancing and data redistribution
- In-memory/main-memory DWs
  - optimization of memory usage
  - compression
- DWs in a cloud
  - assuring scalability
  - load balancing and data redistribution
  - high availability
  - building DWs functionalities on Hadoop/Map Reduce
  - benchmarking

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

Materialized views

- S. Ceri, J. Widom: Deriving Production Rules for Incremental View Maintenance. VLDB, 1991
- A. Gupta, I.S. Mumick: Materialized Views: Techniques, Implementations, and Applications. MIT Press, 1999
- A. Gupta, I.S. Mumick, V.S. Subrahmanian: Maintaining Views Incrementally. SIGMOD, 1993
- S. Kulkarni, M. Mohania.: Concurrent Maintenance of Views Using Multiple Versions. IDEAS, 1999
- D. Quass, J. Widom: On-line Warehouse View Maintenance. SIGMOD, 1997
- M. Teschke, A. Ulbrich: Concurrent Warehouse Maintenance Without Compromising Session Consistency. DEXA, 1998
- S. Samtani, V. Kumar, M. Mohania: Self Maintenance of Multiple Views in Data Warehousing. CIKM, 1999
- H. Wang, M. Orlowska, W. Liang: Efficient Refreshment of Materialized Views With Multiple Sources. CIKM, 1999
- Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom: View Maintenance in Warehousing Environment. SIGMOD, 1995
- Y. Zhuge, H. Garcia-Molina, J. Wiener: The Strobe Algorithms for Multi-Source Warehouse Consistency. PDIS, 1996

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

Partitioning

- P. Furtado: Experimental evidence on partitioning in parallel data warehouses. DOLAP 2004
- P. Furtado: Algorithms for Efficient Processing of Complex Queries in Node-Partitioned Data Warehouses. IDEAS 2004
- P. Furtado: A Survey of Parallel and Distributed Data Warehouses. IJDWM 5(2), 2009
- L. Bellatreche, R. Bouchakri, A. Cuzzocrea, S. Maabout: Horizontal partitioning of very-large data warehouses under dynamically-changing query workloads via incremental algorithms. SAC, 2013