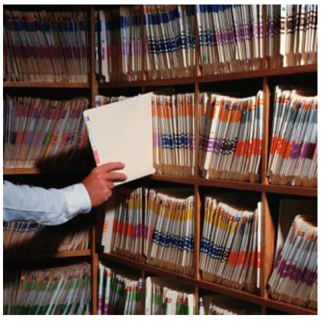
POZNAN UNIVERSITY OF TECHNOLOGY

Data Warehouse Physical Design: Part I

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

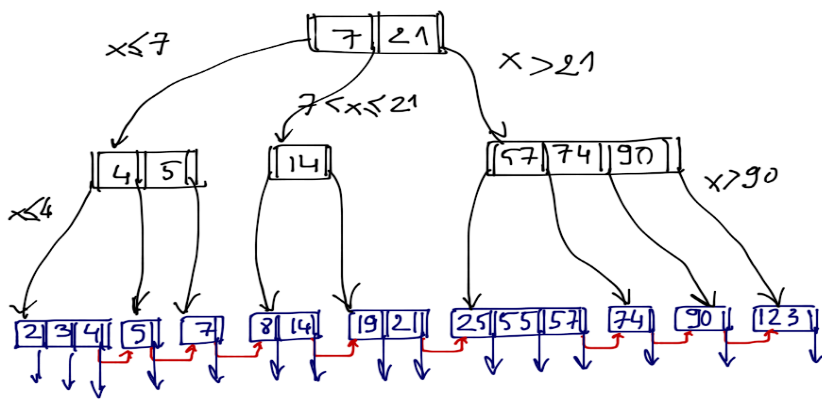
Lecture outline

➲ **Basic index structures**
- **B-tree**
- **bitmap**
- **join**
- **bitmap join (Oracle)**
- **clustered (DB2)**
- **multidimensional cluster (DB2)**
- **indexing dimensions**



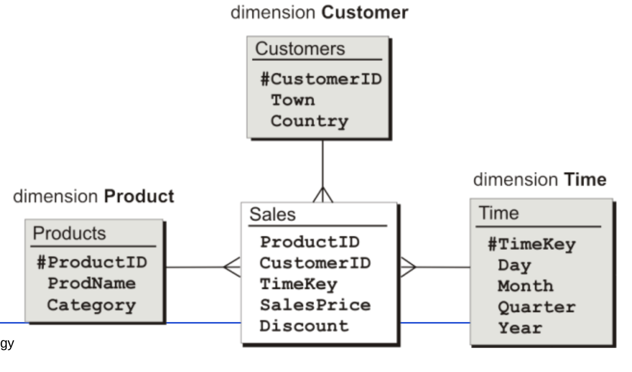
B-tree

➲ **Foundation for other indexes (join, bitmap, bitmap join, clustering, MDC)**



Star schema and queries

```
select sum(SalesPrice), ProdName, Country, Year
from Sales s, Products p, Customers c, Time t
where s.ProductID=p.ProductID
and s.CustomerID=c.CustomerID
and s.TimeKey=t.TimeKey
and p.Category in ('electronics')
and t.Year in (2009, 2010)
group by ProdName, Country, Year;
```



Join index

- Materialized join of 2 tables (typically fact and dimension(s))

Products			
ROWID	productID	prodName	category
BFF1	100	HP Pavillon	electronics
BFF2	230	Dell Inspiron	electronics
BFF3	300	Acer Ferrari	electronics

Sales			
ROWID	salesPrice	discount	productID
0AA0	...	5	100
0AA1	...	15	230
0AA2	...	5	100
0AA3	...	10	300
0AA4	...	10	300
0AA5	...	15	230

Products.ROWID	Sales.ROWID
BFF1	0AA0
BFF1	0AA2
BFF2	0AA1
BFF2	0AA5
BFF3	0AA3
BFF3	0AA4

Join index

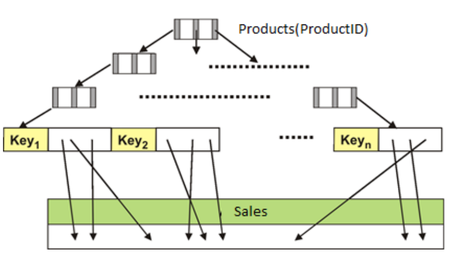
- In order to make searching the join index faster, the join index is physically ordered (clustered) by one of the attributes
- Alternatively, the access to the join index can be organized by means of a B-tree or a hash index

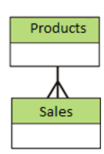
prodID=100

prodID=230

prodID=300

Products.ROWID	Sales.ROWID
BFF1	0AA0
BFF1	0AA2
BFF2	0AA1
BFF2	0AA5
BFF3	0AA3
BFF3	0AA4





DW queries

- DW queries typically are not selective
  - select dozens % of rows in a fact table
- B-tree indexes are efficient up to 10% selectivity of a query
- Other types of indexes are needed

Bitmap index - definitions

- Attribute cardinality ⇨ domain size
  - card(make)=4, card(color)=4
- Bitmap ⇨ vector of bits
  - a bit corresponds to a row in a table

B: red
1
0
0
0
0
0
1
1
0
0
1
0

CarSales

make	...	color
Subaru		red
Mercedes		green
Mercedes		blue
Subaru		blue
BMW		blue
BMW		green
BMW		red
Audi		red
Audi		black
BMW		black
Subaru		red
Mercedes		green

Bitmap index - definitions

- **Bitmap index**
  - **collection of bitmaps**
  - **one bitmap for one value from attribute domain**
- **Organizing bitmaps**
  - **2-dimensional table**
  - **B-tree index**
  - **...**

CarSales

make	...	color
Subaru		red
Mercedes		green
Mercedes		blue
Subaru		blue
BMW		blue
BMW		green
BMW		red
Audi		red
Audi		black
BMW		black
Subaru		red
Mercedes		green

B: red	B: green	B: blue	B: black
1	0	0	0
0	1	0	0
0	0	1	0
0	0	1	0
0	0	1	0
0	1	0	0
1	0	0	0
1	0	0	0
0	0	0	1
0	0	0	1
1	0	0	0
0	1	0	0

Bitmap index in queries

```
select count(*) from CarSales
where  make in ('Audi', 'Mercedes', 'BMW')
and    type = 'sport'
and    color ='red';
```

B:Audi	OR	B:Mercedes	OR	B: BMW	AND	B:sport	AND	B: red	=	
0		0		0		1		1		0
0		1		0		0		0		0
0		1		0		0		0		0
0		0		0		0		0		0
0		0		1		1		0		0
0		0		1		1		0		0
0		0		1		1		1		1
1		0		0		1		1		1
1		0		0		0		0		0
0		0		1		0		0		0
0		0		0		1		1		0
0		1		0		1		0		0

Mapping bits to ROWIDs

- **Easy solution: fixed number of rows per DB block - rpb**

block 1

1
2
3
4
5

block 2

6
7
8
9
10

block 3

11
12
13
14
15

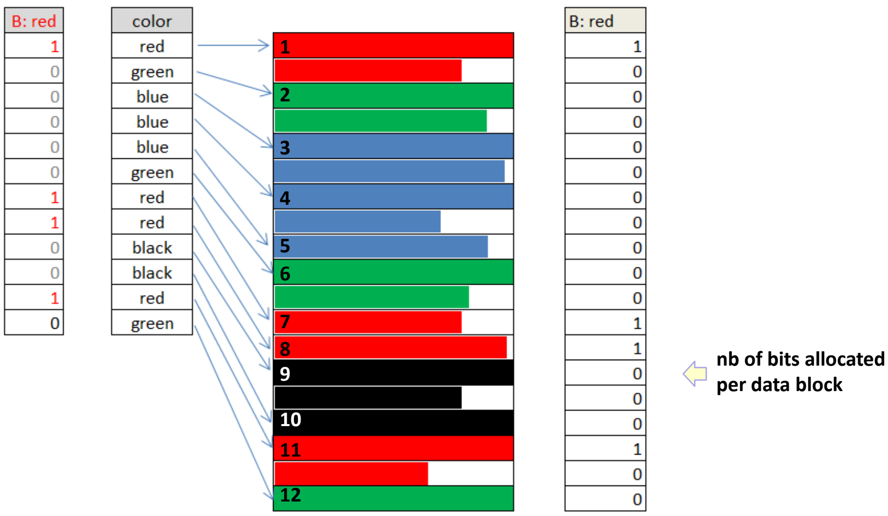
$$pgNo = \left\lceil \frac{bitNo}{rpb} \right\rceil$$

$$slotNo = bitNo\ MOD\ rpb$$

Mapping bits to ROWIDs

- **Real approach**
  - **estimate the average length L of a row in a table**
  - **compute the number of slots in a DB block: BSize/L**
  - **allocate more slots than BSize/L**
  - **real row length differs from L ⇨ some slots are wasted ⇨ BI is larger than it could be**

Mapping bits to ROWIDs



Mapping bits to ROWIDs: Oracle

```
create table TEST_BI1
(id number(7),
 no number(7),
 txt1 varchar2(20),
 txt2 varchar2(20),
 txt3 varchar2(20),
 txt4 varchar2(20),
 txt5 varchar2(20));
```

```
create table TEST_BI2
(id number(7) not null,
 no number(7) not null,
 txt1 varchar2(20) not null,
 txt2 varchar2(20) not null,
 txt3 varchar2(20) not null,
 txt4 varchar2(20) not null,
 txt5 varchar2(20) not null);
```

```
select o.name, o.obj#, t.spare1
from sys.obj$ o, sys.tab$ t
where o.obj#=t.obj# and o.name in ('TEST_BI1', 'TEST_BI2')

NAME                                 OBJ#     SPARE1
------------------------------ ---------- ----------
TEST_BI1                            73450        736
TEST_BI2                            73452        506
```

max number of rows per data block

Mapping bits to ROWIDs: Oracle

- Populating table TEST_BI1 with 10^6 of rows of lenght 113B
- Computing column statistics

```
create bitmap index NO_BI_INDX on TEST_BI1(NO) pctfree 0;
```

```
select index_name, leaf_blocks
from dba_indexes
where index_name = 'NO_BI_INDX';

INDEX_NAME     LEAF_BLOCKS
-------------- -----------
NO_BI_INDX             350
```


Mapping bits to ROWIDs: Oracle

```
drop index NO_BI_INDX;
```

find the actual maximum number of rows in a data block in TEST_BI1

```
alter table TEST_BI1 minimize records_per_block;
```

```
create bitmap index NO_BI_INDX on TEST_BI1(NO) pctfree 0;
```

```
select index_name, leaf_blocks
from dba_indexes
where index_name = 'NO_BI_INDX';

INDEX_NAME     LEAF_BLOCKS
-------------- -----------
NO_BI_INDX             150
```

BI characteristics (1)

- **Reasonably small size** for attributes of low cardinality
- **Example**
  - #rows = 1 000 000
  - card(A) = 4
  - ROWID = 10B (Oracle)
  - bitmap index on attribute A
    - 4 bitmaps: 4 x (1 000 000 / 8) = 4 x 125kB = 500kB
  - B^+-tree on attribute A
    - 1 000 000 x 10B = 10MB

BI characteristics (2)

- **Efficient processing** of bitmaps
  - logical operations AND, OR, NOT, COUNT
  - 64 bits processed in one CPU clock cycle
- **Size**
  - small index ⇨ small #I/O
  - processing in RAM
- **Not applicable to LIKE**

BI characteristics (3)

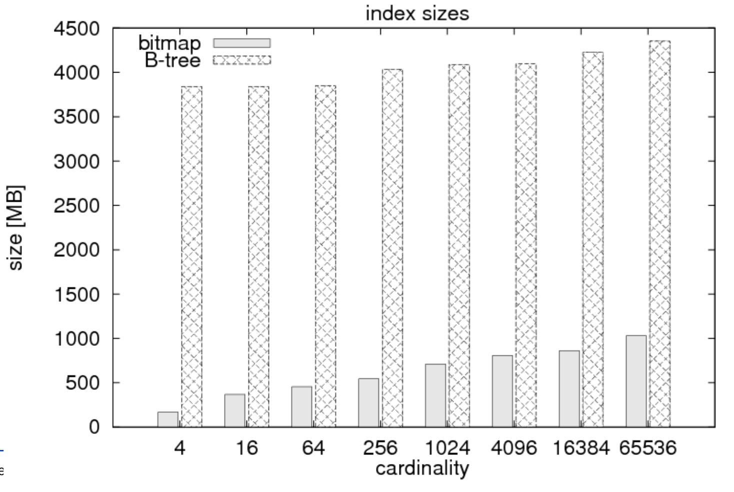
- ➲ **Large size for attributes of large cardinality**
- ➲ **Example**
  - **#rows = 1 000 000**
  - **card(A) = 1024**
  - **ROWID = 10B (Oracle)**
  - **bitmap index on attribute A**
    - **1024 bitmaps: 1024 x (1 000 000 / 8) = 1024 x 125kB = 128MB**
  - **B^+-tree on A**
    - **1 000 000 x 10B = 10MB**

BI characteristics (3)

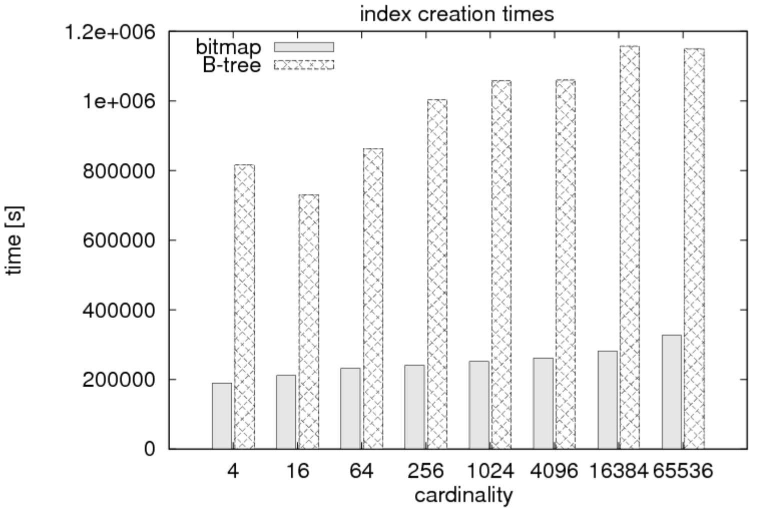
- ➲ **Index maintenance**
  - **inserting rows ⇨ increasing length of bitmaps**
  - **updating rows ⇨ updating 2 bitmaps**
  - **deleting rows ⇨**
    - **decreasing length of bitmaps**
    - **OR bitmap of deleted rows**
  - **locking contiguous segments of bitmaps ⇨ concurrency decreasing**

Experiment (1)

- **Oracle11g**
  - **data cache: 1.7GB, SGA: 3.4GB**
  - **#rows: 250 000 000 (DB size: 10.8GB)**

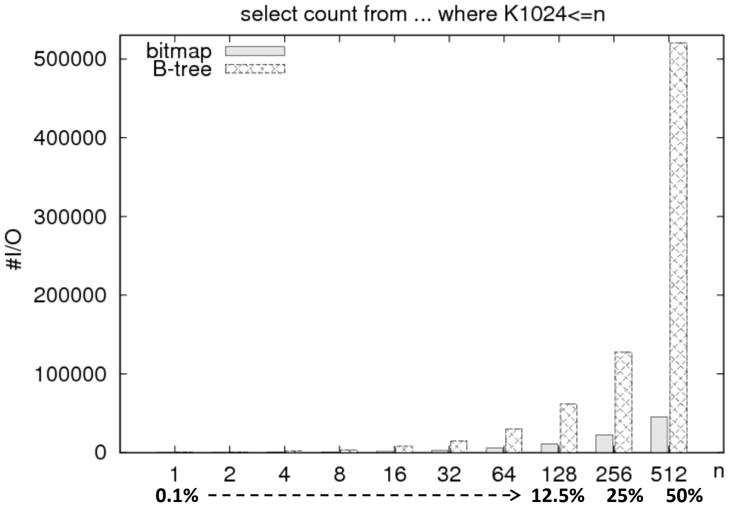


Experiment (2)



Experiment (3)

- cardinality of indexed attribute: 1024



Experiment (4)

WHERE K1024 <= 128

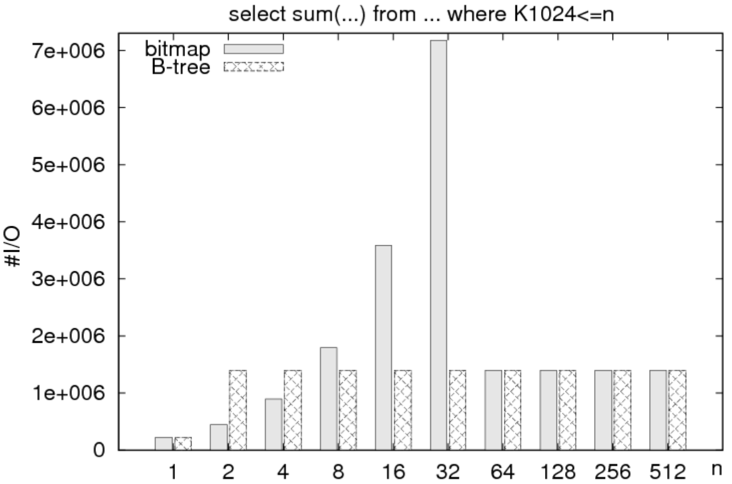
bitmap index

```
---------------------------------------------------------------------------------------
| Id  | Operation                 | Name        | Rows  | Bytes | Cost  (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |             |     1 |     4 | 11315    (1)| 00:02:16 |
|   1 |  SORT AGGREGATE           |             |     1 |     4 |             |          |
|   2 |   BITMAP CONVERSION COUNT |             |   31M|   119M| 11315    (1)| 00:02:16 |
|*  3 |    BITMAP INDEX RANGE SCAN| BMP_5K1024  |       |       |             |          |
---------------------------------------------------------------------------------------
```

B*-tree index

```
--------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost  (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |             |     1 |     4 | 65441    (1)| 00:13:06 |
|   1 |  SORT AGGREGATE    |             |     1 |     4 |             |          |
|*  2 |   INDEX RANGE SCAN| BMP_5K1024  |   31M|   119M| 65441    (1)| 00:13:06 |
--------------------------------------------------------------------------------
```

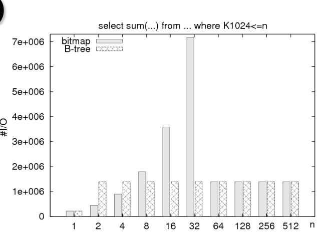
Experiment (5)



0.1% ------------------------------------> 50%

Experiment (7)

WHERE K1024 <= 4



bitmap index

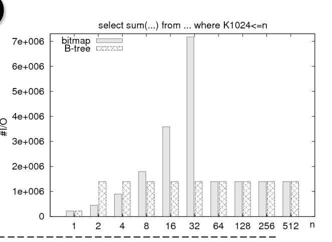
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	149K (1)	00:29:51
1	SORT AGGREGATE		1	10		
2	TABLE ACCESS BY INDEX ROWID	BMTEST	977K	9543K	149K (1)	00:29:51
3	BITMAP CONVERSION TO ROWIDS					
* 4	BITMAP INDEX RANGE SCAN	BMP_5K1024				

B*-tree index

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	387K (2)	01:17:26
1	SORT AGGREGATE		1	10		
* 2	TABLE ACCESS FULL	BMTEST	977K	9543K	387K (2)	01:17:26

Experiment (8)

WHERE K1024 <= 32



bitmap index

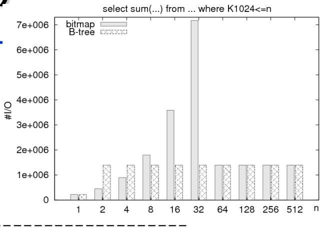
Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	344K (1)	01:08:57
1	SORT AGGREGATE		1	10		
2	TABLE ACCESS BY INDEX ROWID	BMTEST	7819K	74M	344K (1)	01:08:57
3	BITMAP CONVERSION TO ROWIDS					
* 4	BITMAP INDEX RANGE SCAN	BMP_5K1024				

B*-tree index

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	387K (2)	01:17:26
1	SORT AGGREGATE		1	10		
* 2	TABLE ACCESS FULL	BMTEST	7819K	74M	387K (2)	01:17:26

Experiment (8)

WHERE K1024 <= 64



bitmap index

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	387K (2)	01:17:26
1	SORT AGGREGATE		1	10		
* 2	TABLE ACCESS FULL	BMTEST	15M	149M	387K (2)	01:17:26

B*-tree index

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	387K (2)	01:17:26
1	SORT AGGREGATE		1	10		
* 2	TABLE ACCESS FULL	BMTEST	15M	149M	387K (2)	01:17:26

Decreasing size of BI

- ➲ **Range-based bitmap index**
- ➲ **Encoding**
- ➲ **Compression**

Range-based BI (1)

- ➲ **Domain of indexed attribute is divided into ranges**
  - **e.g., temperature: <0, 20), <20, 40), <40, 60), <60, 80), <80, 100)**

indexed attribute

tempC
21
39.6
51.3
12
98.8
71
68.8
50.4
40

B4	B3	B2	B1	B0	bitmap No
(100, 80>	(80, 60>	(60, 40>	(40, 20>	(20, 0>	bitmap range
0	0	0	1	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	0	0	1	
1	0	0	0	0	
0	1	0	0	0	
0	1	0	0	0	
0	0	1	0	0	
0	0	1	0	0	

- ➲ **query: count records for which 10<=temp<45**

Range-based BI (2)

- Bitmaps can represent also sets of values
  - e.g., B1: {yellow, orange, red}, B2: {light blue, blue, navy blue}
- Characteristics
  - the number of bitmaps depends less on the attribute cardinality ⇨ depends on the range/set width
  - border bitmaps may point to rows that do not fulfill selection criteria ⇨ additional row filtering after fetching

Encoding (1)

- Replacing the value of an indexed attribute by another value whose bitmap representation is more compact
- Example
  - card(productName): 50000 ⇨ typical number of products in a supermarket
  - standard bitmap index ⇨ 50000 bitmaps
  - 50000 distinct values can be encoded on 16 bits
    - $\lceil log_2 50000 \rceil = 16$
  - a mapping data structure is required for mapping the encoded values into their real values

Encoding (2)

- query: select * from T where product = 'pecorino d'Abruzzo'
- apply mask: ¬B15 ... ¬B4 B3 ¬B2 ¬B1 ¬B0

indexed attribute — mapping table

product	B15	B14	...	B3	B2	B1	B0
queso Manchengo	0	0	0	0	0	0	1
queso de Burgos	0	0	0	0	0	1	0
queso Cerrato	0	0	0	0	0	1	1
queso Serrat	0	0	0	0	1	0	0
tupi	0	0	0	0	1	0	1
queso de Urbasa	0	0	0	0	1	1	0
pecorino baccellone	0	0	0	0	1	1	1
pecorino d'Abruzzo	0	0	0	1	0	0	0
pecorino dei Berici	0	0	0	1	0	0	1
pecorino di Farindola	0	0	0	1	0	1	0
pecorino lucano	0	0	0	1	0	1	1
pecorino rosso	0	0	0	1	1	0	0
pecorino sardo	0	0	0	1	1	0	1
pecorino sense	0	0	0	1	1	1	0
...	0	0	0	1	1	1	1

Compression (1)

- Byte-aligned Bitmap Compression (BBC)
- Word-Aligned Hybrid (WAH)
- Run Length Huffman
- Based on the run-length encoding
  - homogeneous vectors of bits are replaced with a bit value (0 or 1) and the vector length
  - 0000000 1111111111 000 ⇨ 07 110 03
- A bitmap is divided into words
  - BBC uses 8-bit words
  - WAH uses 31-bit words
  - RLH uses n-bit words (n - parameter)

Compression (2)

- **WAH-compressed bitmaps are larger than BBC-compressed ones**
- **Operations on WAH-compressed bitmaps are faster than on BBC-compressed ones**
  - Wu, K. and Otoo, E. J. and Shoshani, A.: Compressing Bitmap Indexes for Faster Search Operations, SSBDM, 2002
  - Wu, K. and Otoo, E. J. and Shoshani, A.: On the performance of bitmap indices for high cardinality attributes, 2004, VLDB
- **Types of words in BBC and WAH**
  - **fill word ⇨ represents a compressed segment of a bitmap (composed either of all 0s or all 1s)**
  - **tail word ⇨ represents non-compressable segment of a bitmap (composed of interchanged 0 and 1 bits)**

WAH (1)

- **Example: 32-bit processor, bitmap composed of 5456 bits**
  - **taken from** Stockinger K., Wu K.: Bitmap Indices for Data Warehouses. In Wrembel R. and Koncilia C. (eds.): Data Warehouses and OLAP: Concepts, Architectures and Solutions. IGI Global, 2007

100000....000000111000011100000000000000000000..........0000000000000111111111111.....1111011111

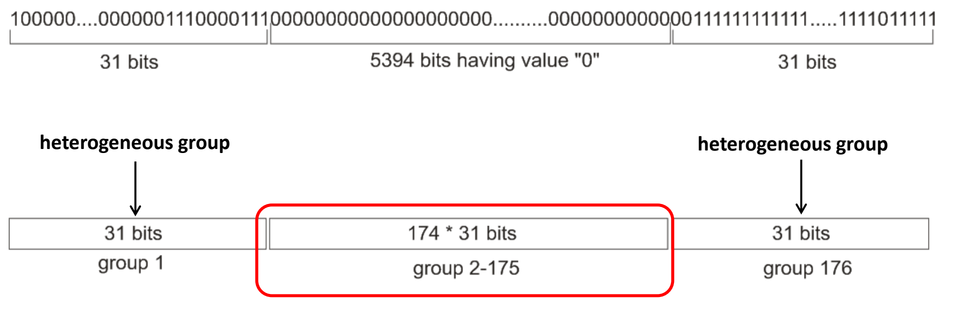
31 bits	5394 bits having value "0"	31 bits

- **Step 1: divide the bitmap into groups including 31 bits each**

31 bits	31 bits	....	31 bits
group 1	group 2		group 176

WAH (2)

➲ **Step 2: merge adjacent homogeneous groups (having the same values of all bits, i.e., groups 2-175)**

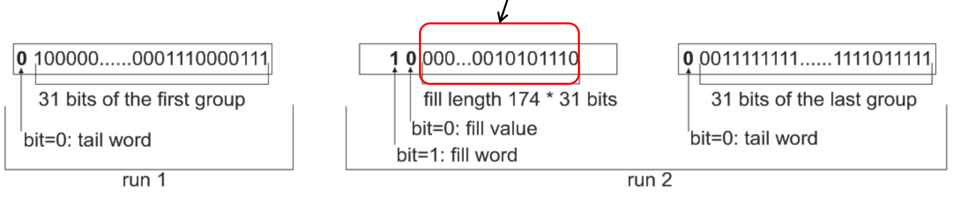


WAH (3)

➲ **Step 3: group encoding**

- **run: fill + tail**
- **run 1: tail**
- **run 2: fill + tail**
- **run 3: tail**

the number of 31-bit groups



WAH (4)

- Unsorted data
- For low cardinality attributes bitmaps are dense
  - many homogeneous 31-bit words filled with 1
- For high cardinality attributes bitmaps are sparse
  - many homogeneous 31-bit words filled with 0
- For medium cardinality attributes
  - the number of homogeneous 31-bit words is lower

RLH

- **RLH** - the Run-Length Huffman Compression (M. Stabno and R. Wrembel. Information Systems, 34(4-5), 2009)
- Based on
  - the Huffman encoding
  - a modified run-length encoding

Huffman Encoding (1)

➲ **Concept**

- **original symbols from a compressed file are replaced with bit strings**
- **the more frequently a given symbol appears in the compressed file the shorter bit string for representing the symbol**
- **encoded symbols and their corresponding bit strings are represented as a Huffman tree**
- **the Huffman tree is used for both compressing and decompressing**

Huffman Encoding (2)

➲ **Example: encoding text "this_is_a_test"**

➲ **Step 1: frequencies of the symbols in the encoded string**

symbol	t	s	_	i	h	e	a
frequency	3	3	3	2	1	1	1

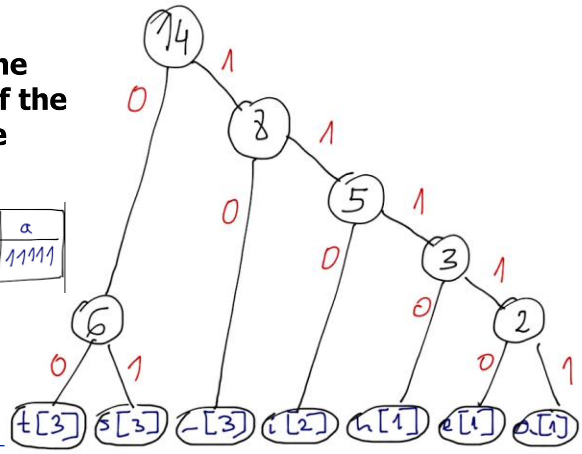
t[3] s[3] _[3] i[2] h[1] e[1] a[1]

Huffman Encoding (3)

- **Step 2: building Huffman tree**
  - **start with nodes of the lowest frequency**

- **Step 3: getting the Huffman codes of the symbols from the tree**

t	s	_	i	h	e	a
00	01	10	110	1110	11110	11111



Huffman Encoding (4)

- **Step 4: replacing original symbols with their Huffman codes**
  - **orignal text: 14B**
  - **compressed text: 37b ⇨ 5B**

t	h	i	s	_	i	s	_	a	_	t	e	s	t
00	1110	110	01	10	110	01	10	11111	10	00	11110	01	00

RLH (1)

➲ **Modified run-length encoding**

- **encodes distances between bits of value 1**

```
 1   0  0    3     0    3    0 0  1  0 0 0
1 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1
```

Clients		bitmap index	
ID	sex	female	male
1	male	0	1
2	female	1	0
3	female	1	0
4	female	1	0
5	male	0	1
6	male	0	1
7	male	0	1
8	female	1	0
9	female	1	0
10	male	0	1
11	male	0	1
12	male	0	1
13	female	1	0
14	female	1	0
15	female	1	0
16	male	0	1
17	female	1	0
18	female	1	0
19	female	1	0

female: 100303001000

male: 030020033

RLH (2)

➲ **Huffman encoding**

- **step1: computing frequencies of symbols (distances) in encoded bitmaps**

female: 100303001000

male: 030020033

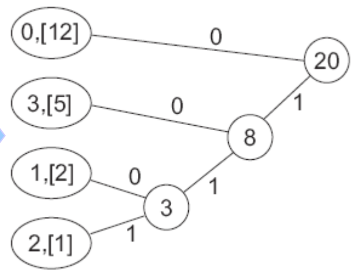
distance	frequency
0	12
3	5
1	2
2	1

RLH (3)

➲ **Huffman encoding**

- **step2: building a Huffman tree**

distance	frequency
0	12
3	5
1	2
2	1



distance	code
0	0
3	10
1	110
2	111

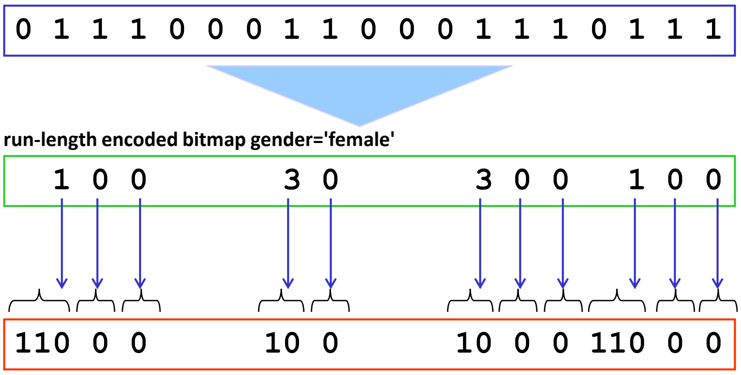
- **an encoded symbol is represented by a path from the root to a leaf**

RLH (4)

➲ **Huffman encoding**

- **step3: replacing distances with their Huffman codes**

original bitmap gender='female'



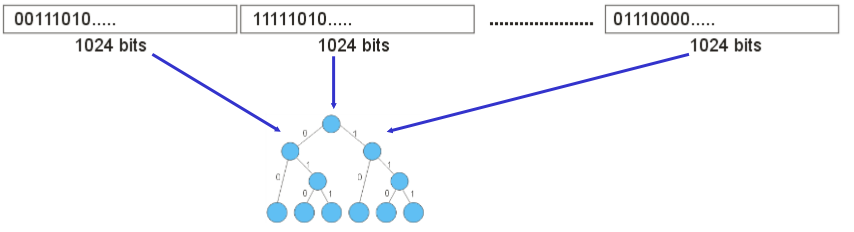
distance	code
0	0
3	10
1	110
2	111

run-length encoded bitmap gender='female'

RLH-compressed bitmap gender='female'

RLH-N

- **Dividing a bitmap into N-bit sections**
  - constructing one Huffman tree based on frequencies of distances from all N-bit sections



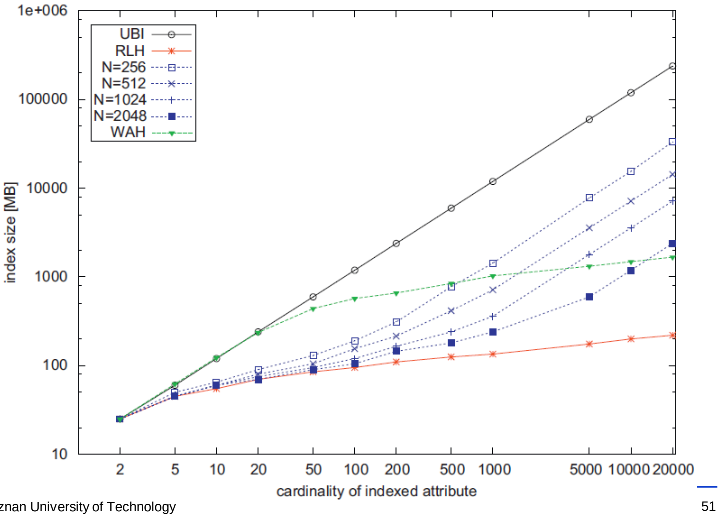
- **Including in the HT all possible distances that may appear in a N-bit section**
  - non-existing distances have assigned the frequency of 1

Experimental Evaluation

- Comparing RLH, WAH, and uncompressed bitmaps (UBI) with respect to
  - bitmap sizes
  - query response times
- Implementation in Java
  - data and bitmap indexes stored on disk in OS files
- Experiments run on
  - PC, AMD Athlon XP 2500+; 768 MB RAM; Windows XP
- Data
  - 100 000 000 indexed rows
  - indexed attribute of type integer
    - cardinality from 2 to 20 000
    - randomly distributed values

WAH and RLH: index sizes

- RLH, RLH-N, WAH, and UBI with respect to the size of a bitmap index (N = {256, 512, 1024, 2048} for RLH-N)

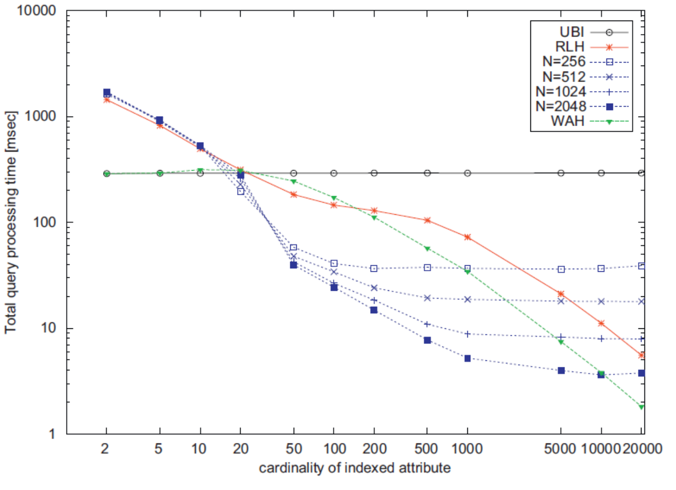


WAH and RLH: response times

- **Query:**

```
select ... from ...
where ind_attribute in (v1, v2, ..., v100)
```

- **Randomly ordered rows wrt. the value of the indexed attribute**

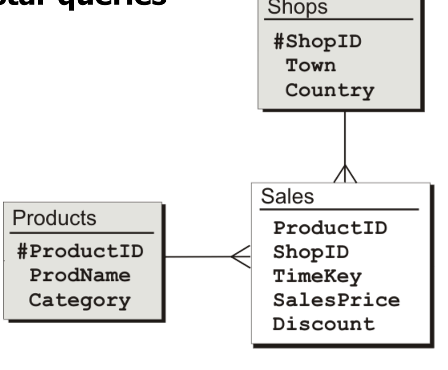


Updating RLH Bitmaps

- Costly process
  - decompressing the whole bitmap
  - modifying the bitmap
  - compressing the bitmap
  - changes frequencies of distances between 1 bits
  - creates new distances between 1 bits
- In a DW environment index structures
  - are dropped before loading a DW
  - are recreated after loading is finished

BIs in Oracle

- Defined explicitly by DBA
- Compressed automaticaly
- Bitmap join index available
- Used for optimizing star queries



Bitmap Join Index (1)

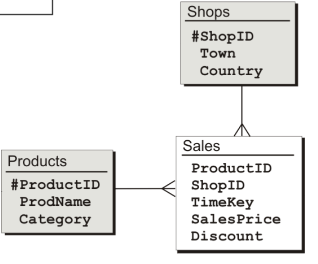
Products

ProductID	ProdName	...
100	queso Manchengo	
200	queso de Burgos	
300	queso Cerrato	
400	queso de Urbasa	
500	pecorino baccellone	

Sales

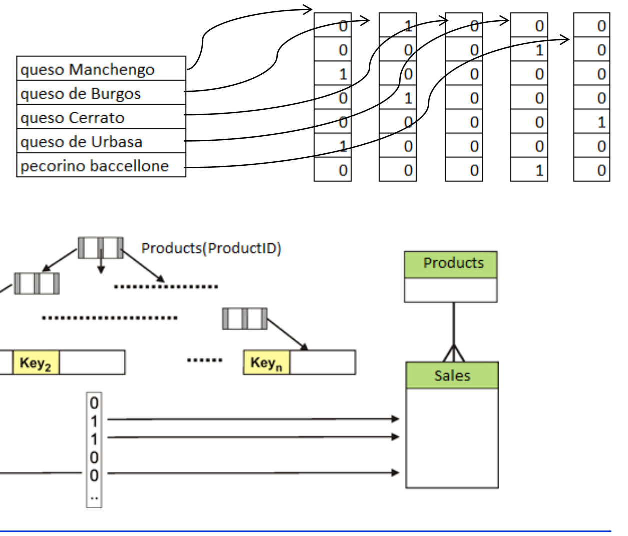
ProductID	SalesPrice	...
200	45	
400	50	
100	40	
200	55	
500	75	
100	65	
400	70	

```
create bitmap index Sales_JBI
on Sales(Products.ProdName)
from Sales s, Products p
where s.ProductID=p.ProductID;
```



BJI (2)

bitmap join index on Products.ProdName

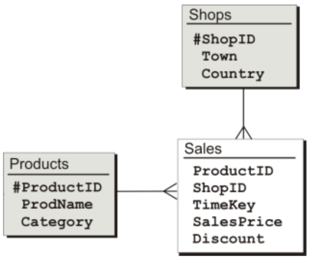


BJI (3)

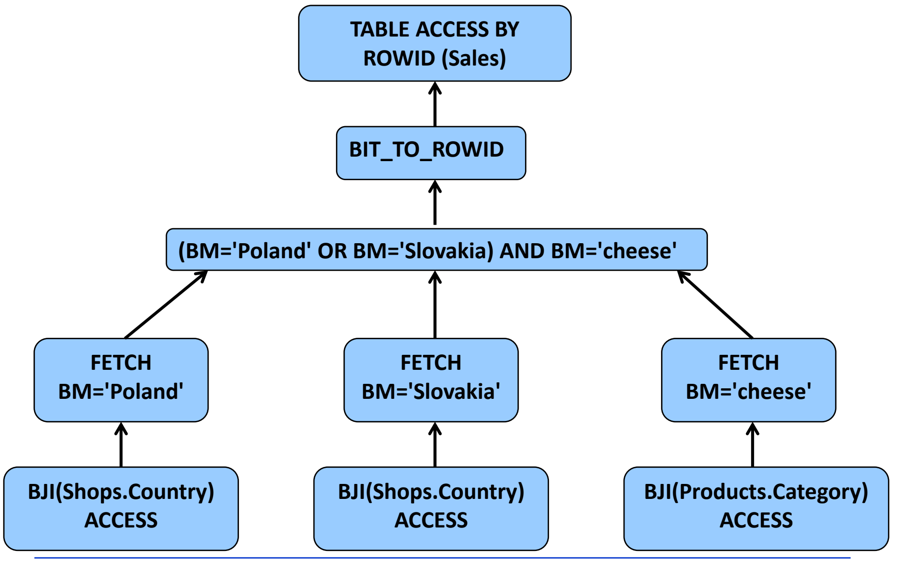
➲ **Star query optimization with the suport of BJI**

```
select sum(sa.SalesPrice), p.ProdName, sh.ShopID
from Sales sa, Shops sh, Products p
where sh.country in ('Poland', 'Slovakia')
and p.Category='cheese'
and sa.ShopID=sh.ShopID
and sa.ProductID=p.ProductID
group by p.ProdName, sh.ShopID;
```

➲ **BJIs defined on attributes**
- `Shops.Country`
- `Products.Category`

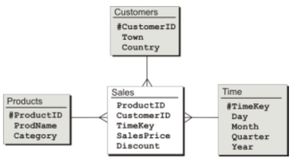


BJI (4)



BJI (5)

The Oracle case




```
select sum(SalesPrice)
from Sales, Products, Customers, Time
where Sales.ProductID=Products.ProductID
and Sales.CustomerID=Customers.CustomerID
and Sales.TimeKey=Time.TimeKey
and ProdName in
    ('ThinkPad Edge', 'Sony Vaio', 'Dell Vostro')
and Town='London'
and Year=2009;
```

```
create bitmap index BI_Pr_Sales
on Sales(Products.ProdName)
from Sales s, Products p
where s.ProductID=p.ProductID;

create bitmap index BI_Cu_Sales
on Sales(Customers.Town)
from Sales s, Customers c
where s.CustomerID=c.CustomerID;
```

```
create bitmap index BI_Ti_Sales
on Sales(Time.Year)
from Sales s, Time t
where s.TimeKey=t.TimeKey;
```


BJI (6)

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	58	13 (8)	00:00:01
1	SORT AGGREGATE		1	58		
2	NESTED LOOPS		21	1218	13 (8)	00:00:01
3	HASH JOIN		22	1012	12 (9)	00:00:01
4	TABLE ACCESS FULL	PRODUCTS	3	51	3 (0)	00:00:01
5	TABLE ACCESS BY INDEX ROWID	SALES	1155	33495	8 (0)	00:00:01
6	BITMAP CONVERSION TO ROWIDS					
7	BITMAP AND					
8	BITMAP INDEX SINGLE VALUE	BI_CU_SALES				
9	BITMAP OR					
10	BITMAP INDEX SINGLE VALUE	BI_PR_SALES				
11	BITMAP INDEX SINGLE VALUE	BI_PR_SALES				
12	BITMAP INDEX SINGLE VALUE	BI_PR_SALES				
13	TABLE ACCESS BY INDEX ROWID	TIME	1	12	1 (0)	00:00:01
14	INDEX UNIQUE SCAN	PK_TIME	1		0 (0)	00:00:01

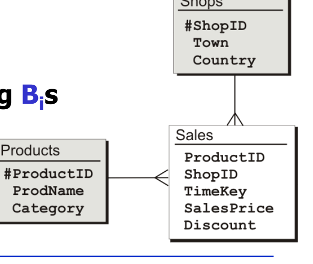
BJI (7)

```
create bitmap index BI_Pr_Cu_Ti_Sales
on Sales(Products.ProdName, Customers.Town, Time.Year)
from Sales, Products, Customers, Time
where Sales.ProductID=Products.ProductID
and Sales.CustomerID=Customers.CustomerID
and Sales.TimeKey=Time.TimeKey;
```

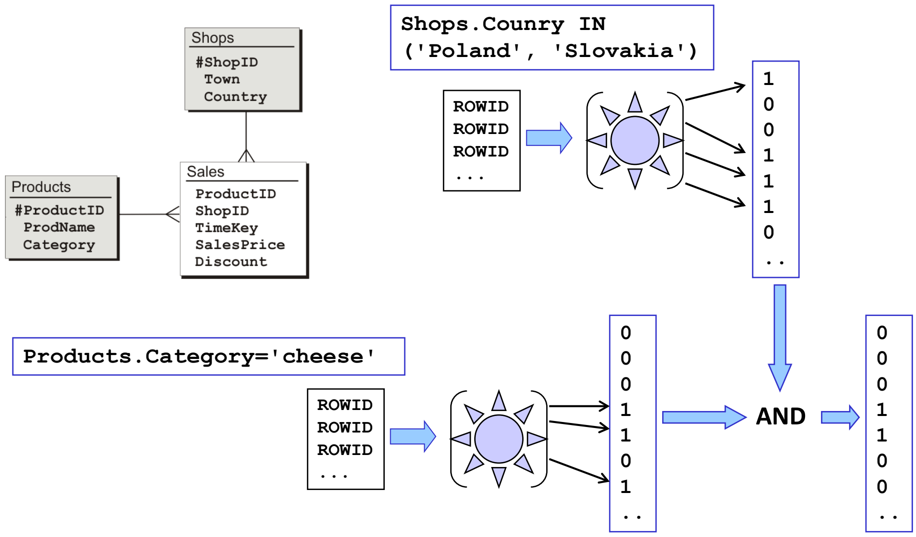
```
-----------------------------------------------------------------------------------------
|Id | Operation                      |Name              |Rows |Bytes|Cost(%CPU)|Time    |
-----------------------------------------------------------------------------------------
| 0 | SELECT STATEMENT               |                  |    1|   29|    7  (0)|00:00:01 |
| 1 |  SORT AGGREGATE                |                  |    1|   29|          |         |
| 2 |   INLIST ITERATOR              |                  |     |     |          |         |
| 3 |    TABLE ACCESS BY INDEX ROWID |SALES             |   22|  638|    7  (0)|00:00:01 |
| 4 |     BITMAP CONVERSION TO ROWIDS|                  |     |     |          |         |
| 5 |      BITMAP INDEX SINGLE VALUE |BI_PR_CU_TI_SALES |     |     |          |         |
-----------------------------------------------------------------------------------------
```


BIs in DB2 (1)

- Created and managed implicitly by the system
- Applied to join optimization
  - Every dim table is independently semi-joined with a fact table
  - The semi-joins use B-trees on foreign keys
  - ROWIDs of every semi-join result are transformed into a separate bitmap
  - Bitmaps B_i are constructed by means of a hash function on ROWID
    - the hash value points to a bit in B_i
  - Final bitmap is computed by AND-ing B_is



BIs in DB2 (2)

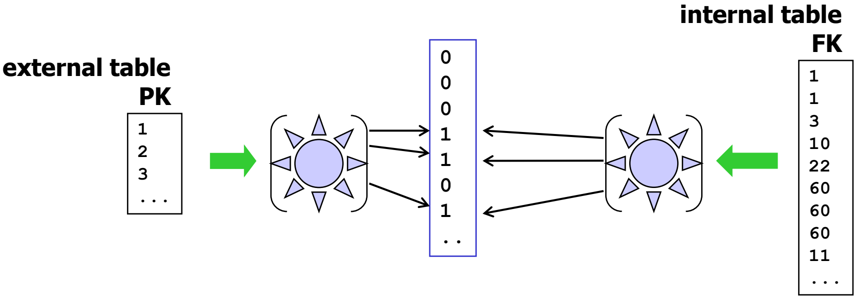


BIs in SQL Server (1)

- Created and managed implicitly by the system
- Applied to join optimization
  - join of a dim table with a fact table by means of hash join
  - table with a PK (dim table) ⇨ external table
  - table with a FK (fact table) ⇨ internal table

BIs in SQL Server (2)

- Hashing PK values into a bitmap
  - HashFunction(PK) → bit no of value 1
- Hashing FK values into a bitmap
  - HashFunction(PK) → bit no of value 1
- The rows from both tables that hash to the same bit ⇨ join result

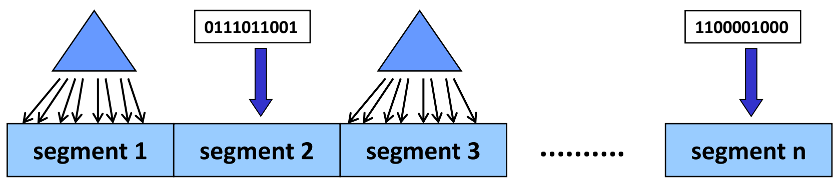


BIs in SybaseIQ

- Defined explicitly by DBA
- Low Fast ⇨ for attributes of low cardinalities
  - max cardinality: 10 000
  - the highest performance for cardinality up to 1000
- High Non Group ⇨ for attributes of high cardinalities
  - for aggregate queries with range predicates

BIs in SAS

- SAS Scalable Performance Data (SPD) Server
  - hybrid index
  - table is divided into segments (e.g., 8192 rows)
  - every segment is indexed independently by
    - bitmap index or
    - B-tree
  - index type selected automatically by the system taking into account the distribution of values in a segment



DB2: Clustering index (1)

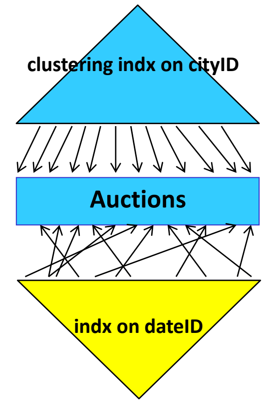
- **Clustering index determines how rows are physically ordered (clustered) on disk**
- **After defining the index, rows are inserted in the order determined by the index**
- **Only one index can be a clustering index (one physical order of rows on disk)**
- **By default the first index created is the clustering one** (unless you explicitly define another index to be the clustering index)

DB2: Clustering index (2)

```
CREATE INDEX cityID_Indx ON Auctions(cityID) CLUSTER
```

cityID	dateID	quantity	...
POZ	03-07-2013	10	
WRO	03-07-2013	13	
WAW	03-07-2013	7	
WRO	05-07-2013	8	
POZ	05-07-2013	9	
WRO	05-07-2013	21	
WRO	07-07-2013	14	
GDA	08-07-2013	17	
POZ	08-07-2013	2	
GDA	08-07-2013	7	
POZ	08-07-2013	19	
GDA	10-07-2013	17	
WRO	10-07-2013	22	
GDA	10-07-2013	11	
WRO	10-07-2013	4	

cityID	dateID	quantity	...
GDA	08-07-2013	17	
GDA	08-07-2013	7	
GDA	10-07-2013	17	
GDA	10-07-2013	11	
POZ	03-07-2013	10	
POZ	05-07-2013	9	
POZ	08-07-2013	2	
POZ	08-07-2013	19	
WAW	03-07-2013	7	
WRO	03-07-2013	13	
WRO	05-07-2013	8	
WRO	05-07-2013	21	
WRO	07-07-2013	14	
WRO	10-07-2013	22	
WRO	10-07-2013	4	



DB2: Clustering index (3)

- **Eliminates sorting**
- **Operations that benefit from clustering indexes include:**
  - **grouping**
  - **ordering**
  - **comparisons other than equal**
  - **distinct**

DB2: MDC (1)

- **MultiDimensional Cluster - MDC**
  - **groups data based on values of multiple dimension attributes**
  - **a physical region (block) is associated with each unique combination of dimension attribute values**
  - **a block stores records with the same values of dimension attributes**
- **Block Map: a structure that stores information about block states** (in use, free, loaded, ...)

DB2: MDC (2)

```
CREATE TABLE Auctions
(... cityID VARCHAR(4), dateID DATE,
  quantity INT, ...)
ORGANIZE BY (cityID, dateID);
```

orygìnal table

cityID	dateID	quantity	...
POZ	03-07-2013	10	
WRO	03-07-2013	13	
WAW	03-07-2013	7	
WRO	05-07-2013	8	
POZ	05-07-2013	9	
WRO	05-07-2013	21	
WRO	07-07-2013	14	
GDA	08-07-2013	17	
POZ	08-07-2013	2	
GDA	08-07-2013	7	
POZ	08-07-2013	19	
GDA	10-07-2013	17	
WRO	10-07-2013	22	
GDA	10-07-2013	11	
WRO	10-07-2013	4	

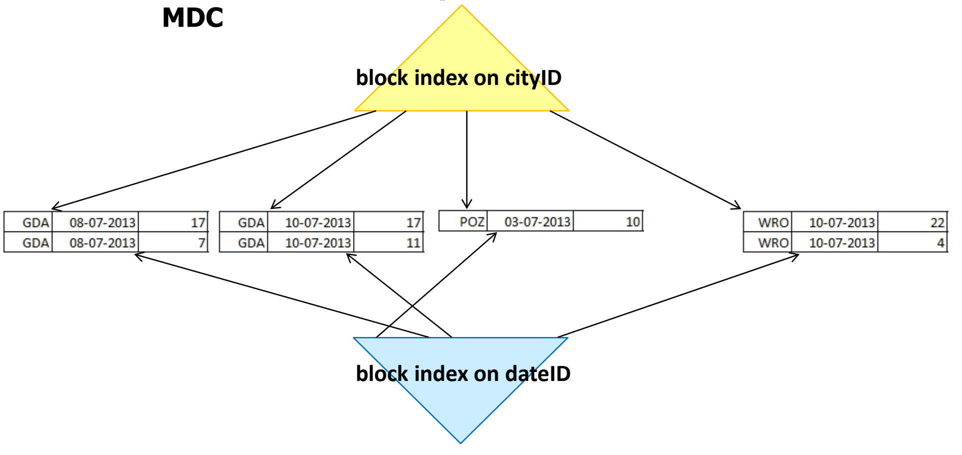
data block

MDC

cityID	dateID	quantity	...
GDA	08-07-2013	17	
GDA	08-07-2013	7	
GDA	10-07-2013	17	
GDA	10-07-2013	11	
POZ	03-07-2013	10	
POZ	05-07-2013	9	
POZ	08-07-2013	2	
POZ	08-07-2013	19	
WAW	03-07-2013	7	
WRO	03-07-2013	13	
WRO	05-07-2013	8	
WRO	05-07-2013	21	
WRO	07-07-2013	14	
WRO	10-07-2013	22	
WRO	10-07-2013	4	

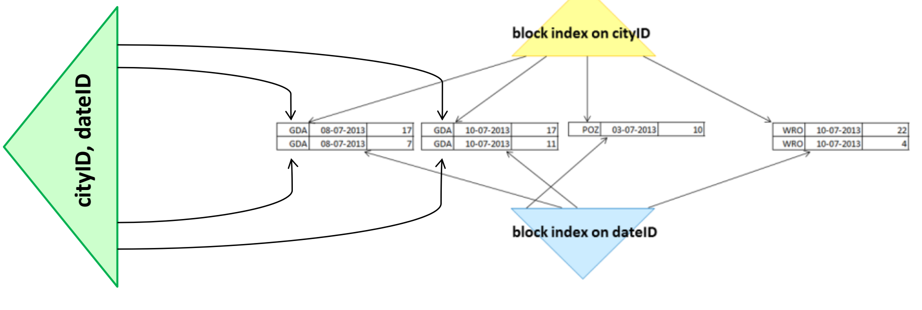
DB2: MDC (3)

➲ Block index: B-tree based, points to blocks
- created automatically for each of the dimensions in MDC



DB2: MDC (4)

➲ Composite block index: includes all dimension key columns
- used for insert, update, delete



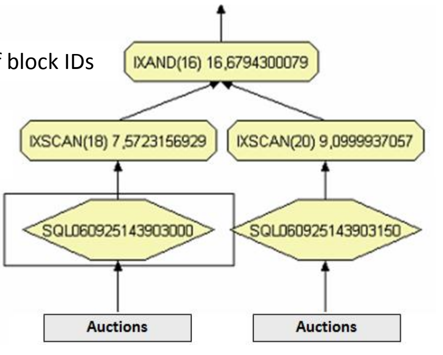
MDC in queries

```
SELECT SUM(quantity), cityID, dateID
FROM Auctions
WHERE cityID ='GDA' AND dateID='10-07-2013'
group by cityID, dateID;
```

intersection of block IDs

block IDs from block index on cityID

block IDs from block index on dateID



MDC in queries

```
SELECT SUM(quantity), cityID, dateID
FROM Auctions
WHERE cityID ='GDA' OR dateID='10-07-2013'
group by cityID, dateID
```

➲ **{block IDs with cityID='GDA'} UNION {block IDs with dateID='10-07-2013'}**

MDC

- Candidates as dimensions in MDC
  - attributes used in predicates: range, =, IN
  - dimension foreign keys in fact table
  - attributes used in GROUP BY
  - attributes used in ORDER BY
- Summary
  - Data ordered on disk ⇨ less I/O
  - Block index points to a data block ⇨ inserting, updating, deleting may not affect the index structure

MDC case study

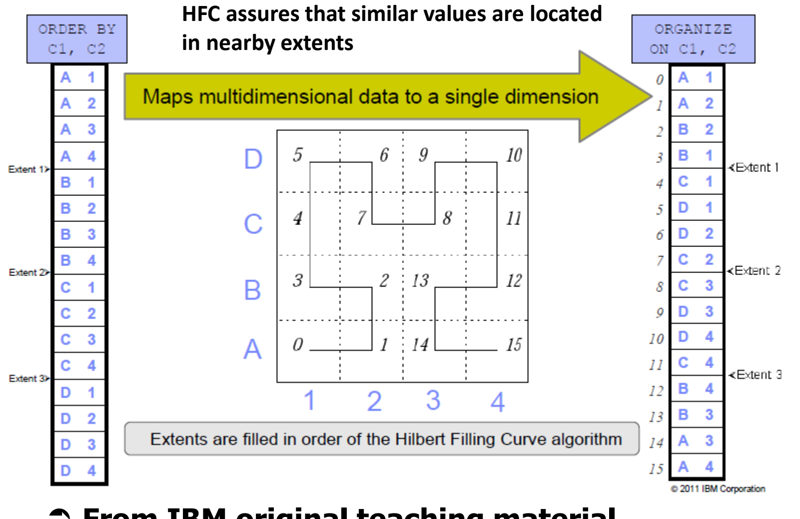
- Source: IBM presentation
- Mobile network operator in USA
- Characteristic
  - 10 billion transactions daily
  - 32 TB raw data
  - thousands concurrent users
  - up to 37000 queries daily
  - DW loading: over 1 billion rows daily (max 1.6 billion)
  - DB2 DWE, 16 x 8 CPU P5 pSeries
- MDC
  - deletion faster by 80% of time
  - I/O lower by 43%

Clustered Based Table (Netezza)

- Clustered Base Table (CBT) ⇨ data are organized by 1 to 4 attributes (organizing keys)
- Data stored in extents (zones)
  - an extent is the smallest unit of disk allocation = 3MB
- Organizing keys are used to group records within the table (store them in one or more nearby extents)
- Netezza creates zone maps for the organizing keys
- Materialized views cannot be build on CBTs

```
CREATE TABLE tab-name
(...)
[ORGANIZE ON (org-key1, ...)]
```

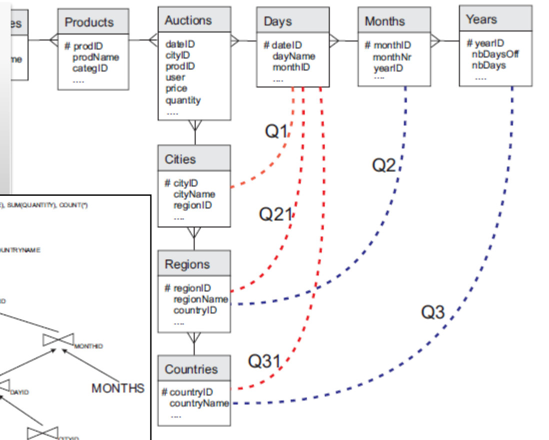

CBT



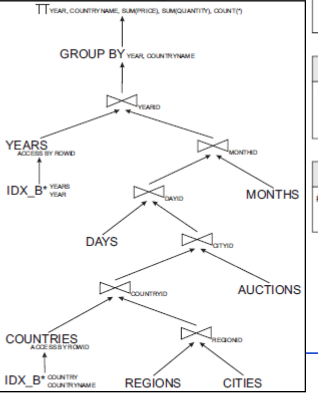
- From IBM original teaching material

Indexing dimensions

```
CREATE BITMAP INDEX bmi_a_years_countries
ON auctions(yr.year, cr.countryName)
FROM
Auctions a,
Days d, Months m, Years yr,
Cities ci, Regions r, Countries cr
WHERE
a.cityid=ci.cityid AND
r.regionid=ci.regionid AND
r.countryid=cr.countryid AND
a.dateid=d.dateid AND
d.monthid=m.monthid AND
m.yearid=yr.yearid
```



exec. plan of Q3



Indexing dimensions

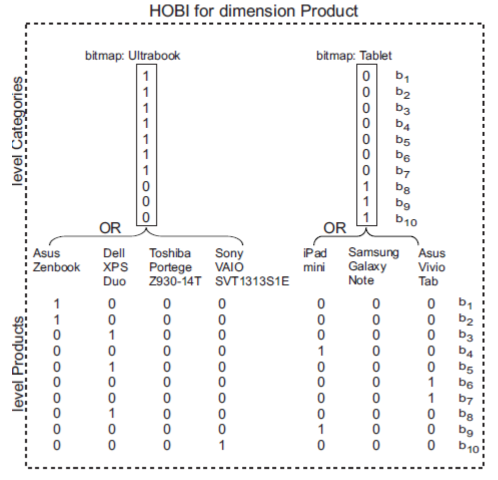
- Indexes do not reflect a hierarchical structure of dimensions
- Time dimension is used in most of star queries ⇨ joins
- Time-HOBI (T. Morzy, R. Wrembel, J. Chmiel, A. Wojciechowski. Information Systems, 37:(5), 2012)
  - HOBI ⇨ Hierarchically Organized Bitmap Index
  - TI ⇨ Time Index
  - PA ⇨ Partial Aggregates

HOBI

- **HOBI is composed of bitmap indexes created for every level of a dimension hierarchy**

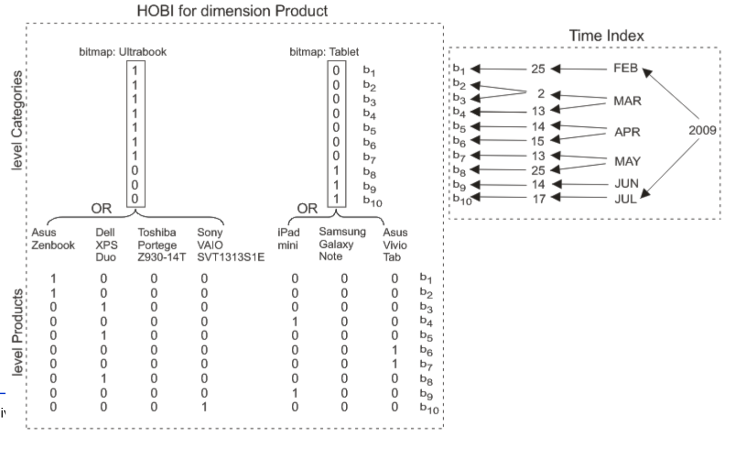
- **BI are also organized in a hierarchy**

- **The BI hierarchy reflects the dimension hierarchy, such that a bitmap index at an upper level aggregates bitmap indexes from a lower level**



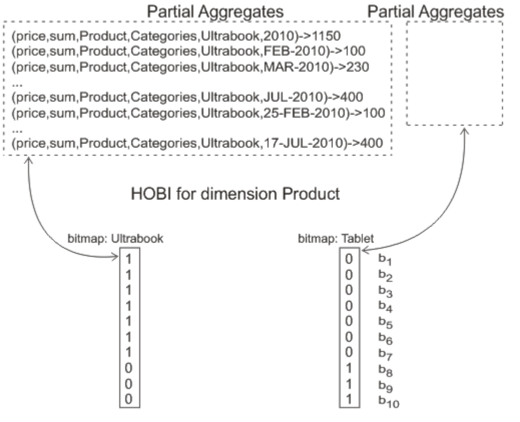
Time Index

- **Assumption: data ordered on disk by time**
- **TI ⇨ encodes time in other dimensions**
- **Shared by all dimensions**
- **No joins with Time required**



Partial Aggregates

- PA are computed for:
  - selected measures
  - selected aggregation functions
  - a given dimension
  - a given dimension level
  - a given dimension level instances
  - a given time interval

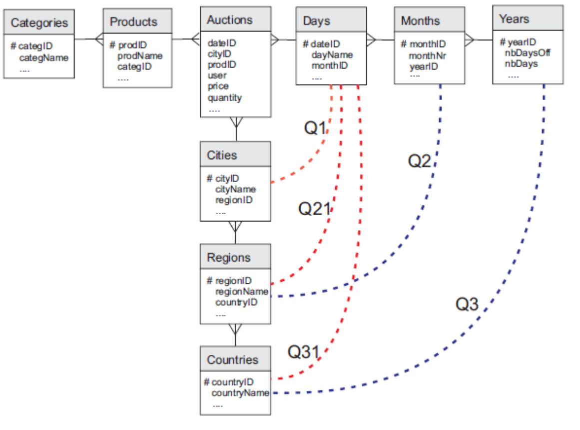


Time-HOBI

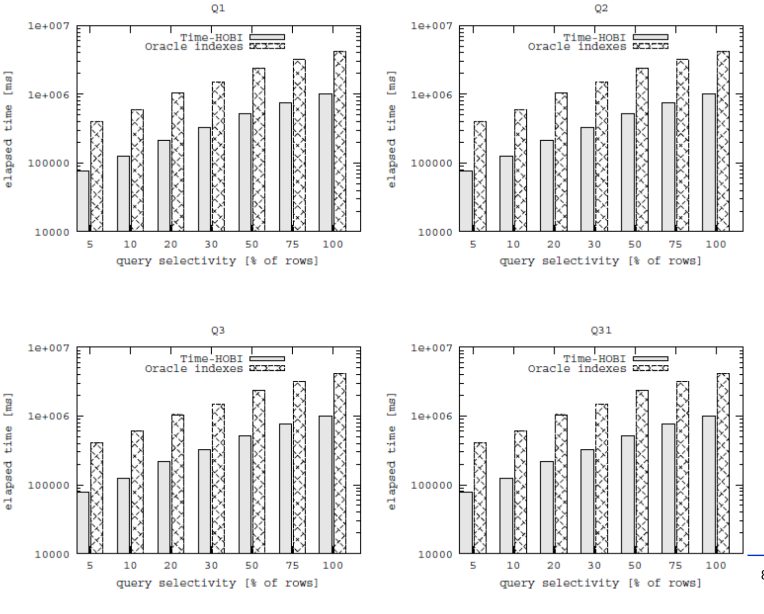
- Eliminates joins of a fact table with the Time dimension ⇨ Time Index
- Eliminates joins of a fact table with other dimensions ⇨ HOBI
- Eliminates or reduces the costs of computing aggregates of measures at various levels of a dimension hierarchy ⇨ Partial Aggregates

Experiments (1)

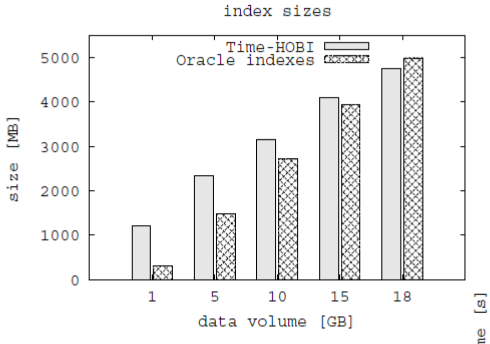
- **Auctions: 500 000 000, 30GB**
- **Competitor: Oracle indexes**
  - concatenated BJI on year and countryName that joined tables Years, Months, Days, Countries, Regions, Cities, and Auctions
  - BI on year
  - BI on countryName

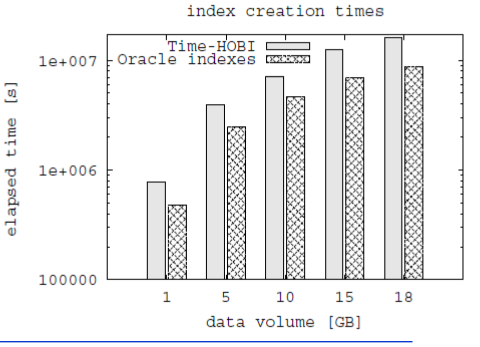


Experiments (2)



Experiments (3)





Time-HOBI vs. MV

- **Time-HOBI takes advantage of materialized partial aggregates ⇨ functionality of MV**
- **Unlike MV, Time-HOBI may be used to optimize queries that either do not compute aggregates or compute aggregates that have not been materialized in the index ⇨ more flexible structure**

References

- **Various types of indexes**
  - J. Dyke: Bitmap Index Internals. juliandyke.com
  - N. Koudas. Space efficient bitmap indexing. CIKM, 2000
  - P. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. SIGMOD Record, 1995
  - P. O'Neil and D. Quass. Improved query performance with variant indexes. SIGMOD, 1997
  - R. Bouchakri, L. Bellatreche, K.W. Hidouci: Static and Incremental Selection of Multi-table Indexes for Very Large Join Queries. ADBIS, 2012
  - R. Bouchakri, L. Bellatreche: On Simplifying Integrated Physical Database Design. ADBIS, 2011
  - T. Morzy, R. Wrembel, J. Chmiel, A. Wojciechowski: Time-HOBI: Index for optimizing star queries. Information Systems, 37:(5), 2012

References

- **Indexes in DB2**
  - S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, M. Huras: Multi-Dimensional Clustering: A New Data Layout Scheme in DB2. SIGMOD, 2003
  - P. Bates, P. Zikopoulos: Multidimensional Clustering (MDC) Tables in DB2 LUW. Talk, DB2 Night Show, Jan 2011
  - IBM Netezza System Administrator's Guide. IBM Netezza 7.0 and Later, Oct 2012
  - Clustering indexes. DB2 technical doc http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=%2Fcom.ibm.db2z10.doc.intro%2Fsrc%2Ftpc%2Fdb2z_clusteringindexes.htm

References

Binning

- Koudas N.: Space Efficient Bitmap Indexing. CIKM, 2000
- Stockinger K., Wu K., Shoshani A.: Evaluation Strategies for Bitmap Indices with Binning. DEXA, 2004
- Rotem D., Stockinger K., Wu K.: Optimizing Candidate Check Costs for Bitmap Indices. CIKM, 2005

Encoding

- Wu M., Buchmann A.P.: Encoded Bitmap Indexing for Data Warehouses. ICDE, 1998
- Chan C.Y., Ioannidis Y.E.: An Efficient Bitmap Encoding Scheme for Selection Queries. SIGMOD, 1999

Compressing

- BBC
  - Antoshenkov G., Ziauddin M.: Query Processing and Optimization in ORACLE RDB. VLDB Journal, 1996

References

Compressing

- WAH
  - Stockinger K., Wu K., Shoshani A.: Strategies for Processing ad hoc Queries on Large Data Sets. DOLAP, 2002
  - Wu K., Otoo E.J., Shoshani A. (2004): On the Performance of Bitmap Indices for High Cardinality Attributes. VLDB, 2004
  - Stockinger K., Wu K.: Bitmap Indices for Data Warehouses. In Wrembel R. and Koncilia C. (eds.): Data Warehouses and OLAP: Concepts, Architectures and Solutions. IGI Global, 2007
- PL-WAH
  - F. Deliège and T. B. Pedersen. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. EDBT, 2010
- Approximate Compression with Bloom Filters
  - Apaydin T., Canahuate G., Ferhatosmanoglu H., Tosun, A. S.: Approximate encoding for direct access and query processing over compressed bitmaps. VLDB, 2006

References

- **Compressing**
  - Reordering
    - Johnson D., Krishnan S., Chhugani J., Kumar S., Venkatasubramanian S.: Compressing Large Boolean Matrices Using Reordering Techniques. VLDB, 2004
    - Pinar A., Tao T., Ferhatosmanoglu H.: Compressing Bitmap Indices by Data Reorganization. ICDE, 2005
  - WAH vs. BBC
    - Stockinger K., Wu K., Shoshani A.: Strategies for processing ad hoc queries on large data warehouses. DOLAP, 2002
    - Wu K., Otoo E.J., Shoshani A.: Compressing bitmap indexes for faster search operations. SSDBM, 2002
    - Wu K., Otoo E.J., Shoshani A.: On the Performance of Bitmap Indices for High Cardinality Attributes. VLDB, 2004
  - RLH
    - M. Stabno and R. Wrembel. RLH: Bitmap compression technique based on runlength and Hufman encoding. Information Systems, 34(4-5), 2009

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

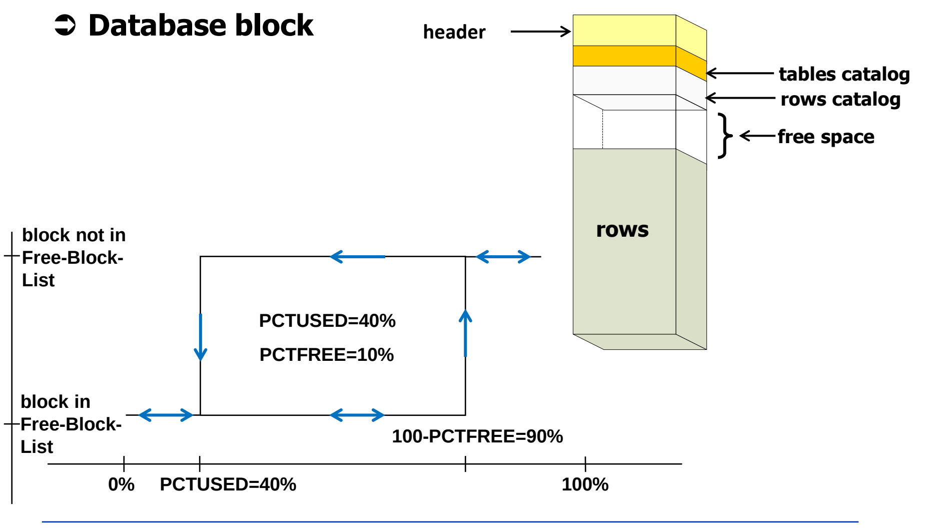
⋮

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**



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

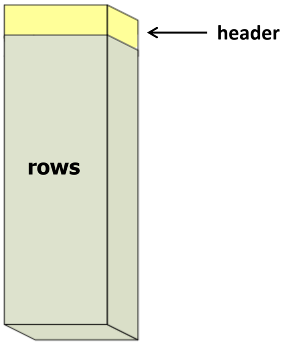
148,00	149,00	148,50	150,00	147,50	148,00	148,00	147,00	148,50
148,50	151,50	150,00	151,50	149,00	152,50	152,00	154,50	164,50
172,50	170,00	169,50	170,50	166,00				

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)**

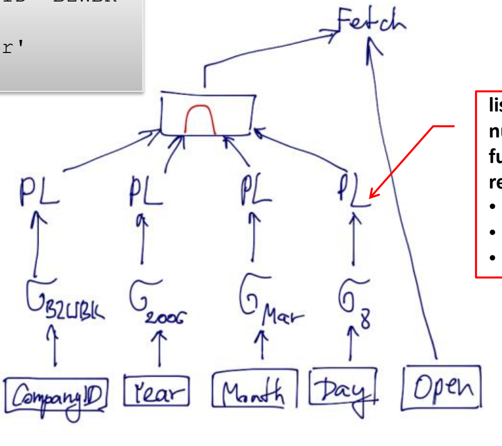
Column storage (3)

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



Query processing (1)

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



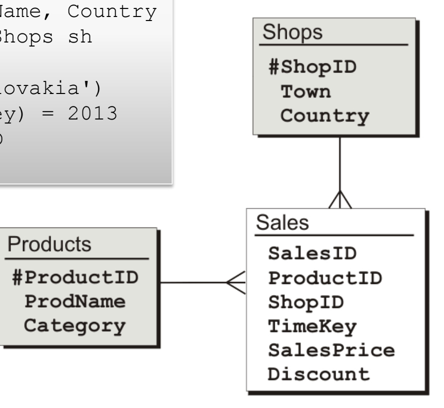
list of positions (slot numbers) of values fulfilling the predicate represented as:

- **array or**
- **bit vector or**
- **set of position ranges**

Query processing (2)

➲ **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;
```



Query processing (3)

- ➲ **Get Product.ProductIDs that satisfy:**
  - **Category='cheese'**
- ➲ **Get Shops.CountryIDs that satisfy: country in**
  - **('Poland', 'Slovakia')**
- ➲ **Get Sales.SalesIDs that satisfy:**
  - **extract(year from TimeKey)=2013**

Query processing (4)

hash tables

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

Poland or Slovakia

hashing

10
30

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

cheese

hashing

100
500
200

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

hashing

S2
S3
S4
S5

Query processing (5)

- **Hashing Sales rows**

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

hash table ShopID

hashing (ShopID)

10
30

hash table ProductID

hashing (ProductID)

100
500
200

- **Matching Sales rows are represented by bitmaps**

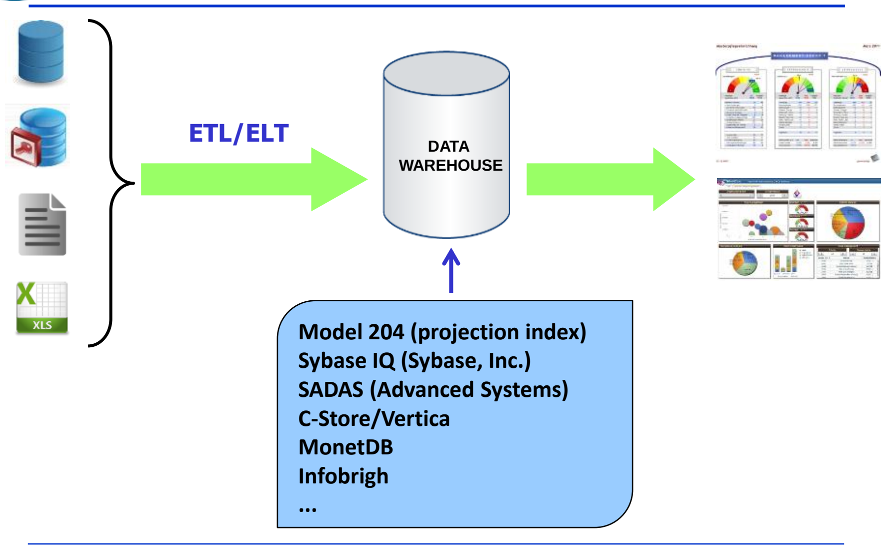
ShopID in (10, 30)		ProductID in (102, 500, 200)		SalesID in (S2, S3, S4, S5)		
1	AND	1	AND	0	=	0
0		1		1		0
1		0		1		0
1		1		1		**1**
0		1		1		0

Query processing (6)

- Get ProductIDs using the final bitmap
- Join with Products to get 'prodName' using the ProductIDs

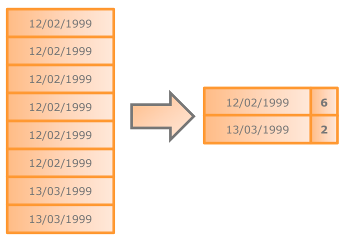
- Get ShopIDs using the final bitmap
- Join with Shops to get 'Country' using the ShopIDs

Column storage in DW architecture

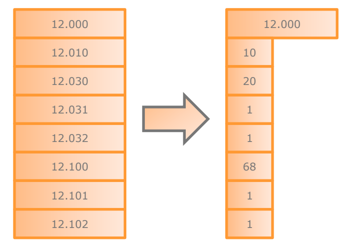


CS - compression (1)

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

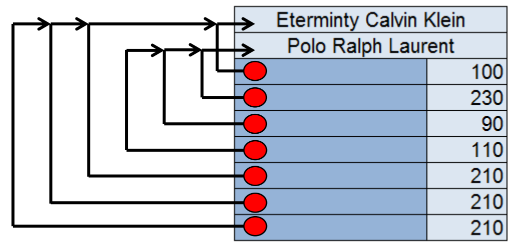
mapping table

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

RS - compression (1)

- **Oracle**

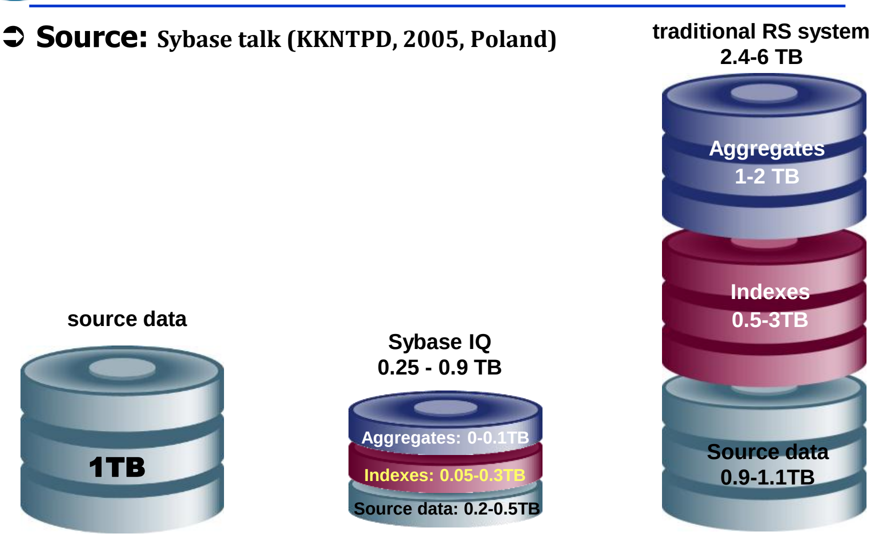
Eternity Calvin Klein	100
Polo Ralph Laurent	230
Polo Ralph Laurent	90
Polo Ralph Laurent	110
Eternity Calvin Klein	210
Eternity Calvin Klein	210
Eternity Calvin Klein	210



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

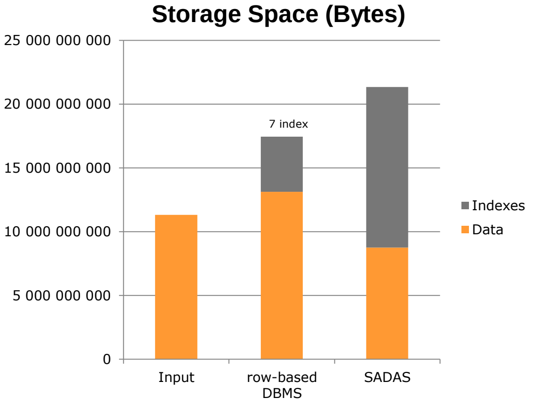
Performance comparison: CS-RS (1)

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



Performance comparison: CS-RS (2)

➲ **Source:** presentation by Advanced Systems



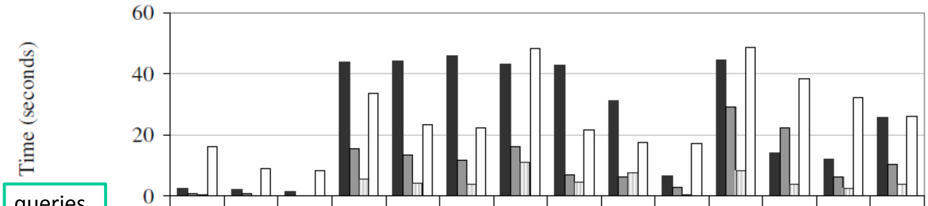
Performance comparison: CS-RS (3)

➲ **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)



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

RS with optimal set of materialized views

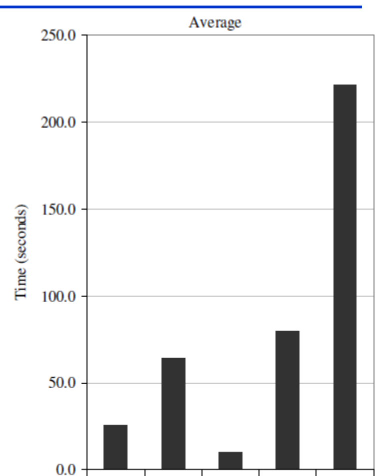
row store

CS implemented with RW materialized views

column store

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

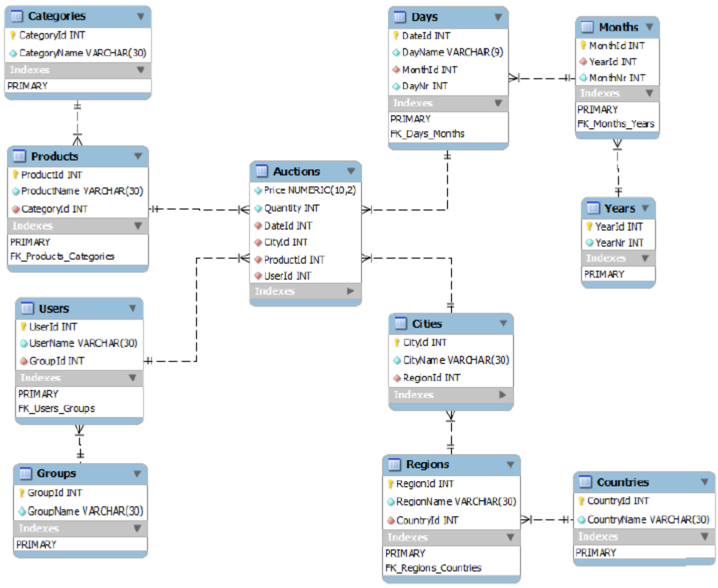
Our experiments (1)

- ➲ **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**

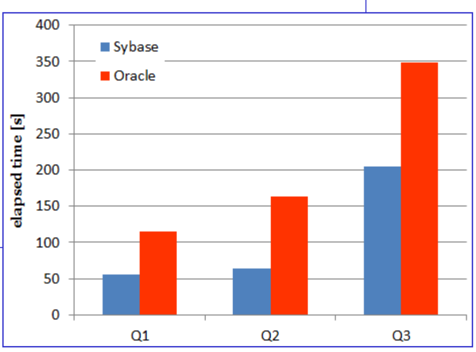
Our experiments (3)



Our experiments (4)

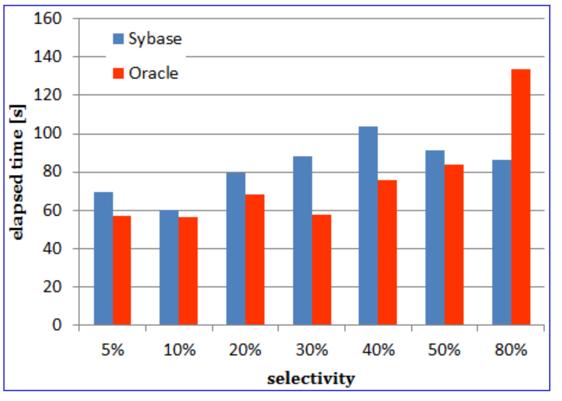
- **Q1: GROUP BY** (productName, regionName)
- **Q2: GROUP BY** (productName, regionName, monthNr)
- **Q3: GROUP BY, 4 dimensions**

```
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;
```



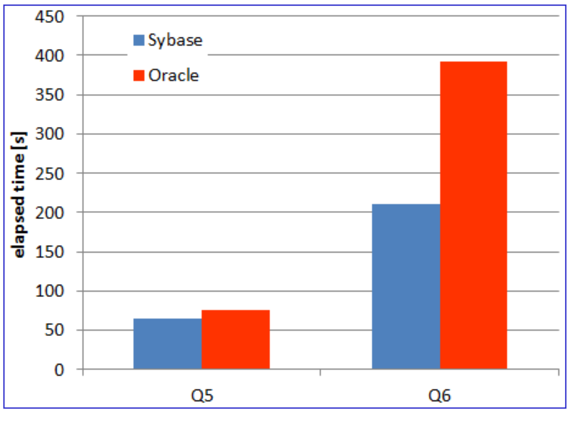
Our experiments (5)

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



Our experiments (6)

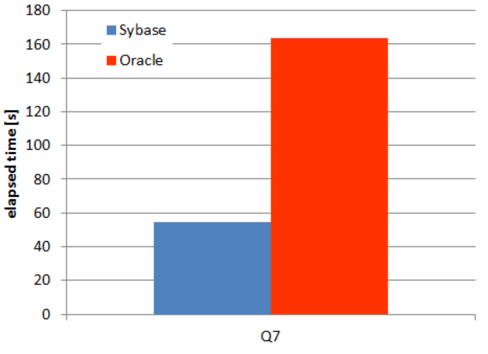
- **Q5: GROUP BY ROLLUP** (productName, regionName)
- **Q6: GROUP BY ROLLUP** (productName, countryName, monthNr)



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 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 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)

	Index	Date	Open	Close
ext1	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
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
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

ZM

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

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

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 combines SMA i 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 (depending on NULLS LAST/FIRST) ⇨ 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**
- **the m MIN and MAX values are not represented in the BVF**

➲ **ZI - Zone Index**

- **supports searching within a zone**
- **a dedicated ZI maintained for a zone**

Materialization (6)

zone index

BZWBK	2006	Mar	31	148,00
BZWBK	2006	Mar	30	149,00
BZWBK	2006	Mar	29	148,50
BZWBK	2006	Mar	28	150,00
BZWBK	2006	Mar	27	147,50
BZWBK	2006	Mar	24	148,00
BZWBK	2006	Mar	23	148,00

ZF1 TimeKey	ZF1 Open
23.03.06	147
24.03.06	148
30.03.06	149
31.03.06	150

bit vector filter

zone index

BZWBK	2006	Mar	22	147,00
BZWBK	2006	Mar	21	148,50
BZWBK	2006	Mar	20	148,50
BZWBK	2006	Mar	17	151,50
BZWBK	2006	Mar	16	150,00
BZWBK	2006	Mar	15	151,50
BZWBK	2006	Mar	14	149,00

ZF2 TimeKey	ZF2 Open
14.03.06	147
15.03.06	148
21.03.06	150
22.03.06	151

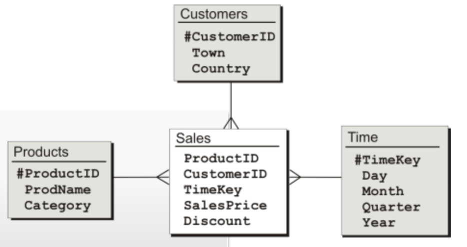
bit vector filter

- **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)

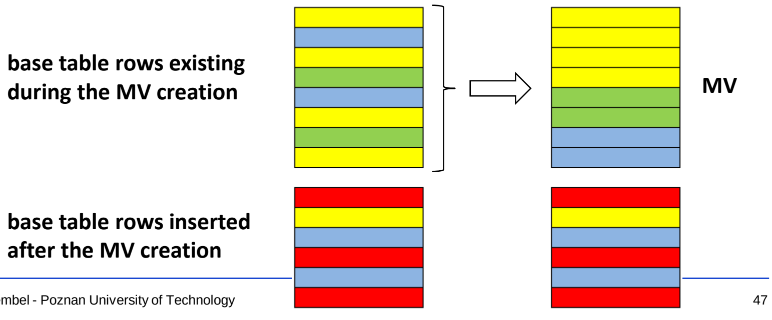
- **Used for query rewriting**
- **Stored as a table**
- **Divided into data slices that are co-located on the same data slices as the corresponding base table data slices**

```
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 - expressions (aggregates, mathematical operators, SQL functions, DISTINCT, ...) are not allowed**
  - **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 thresholds allows to refresh all the materialized views associated with a base table**
    - **the threshold specifies the percentage of unsorted data in the materialized view, value from 1 to 99 (default 20)**

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

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

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

```
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
```

```
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

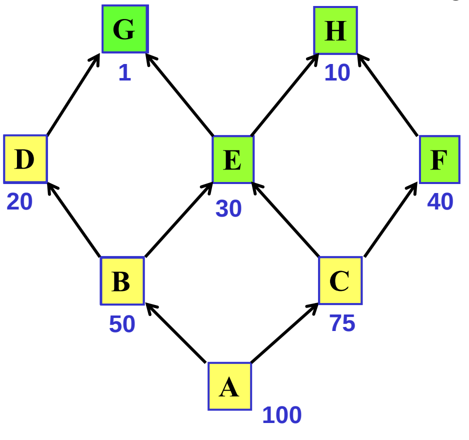
MV design

- 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 that is read by the query
- **Goal**
  - minimize the data volume read by the queries with a given number of materialized views

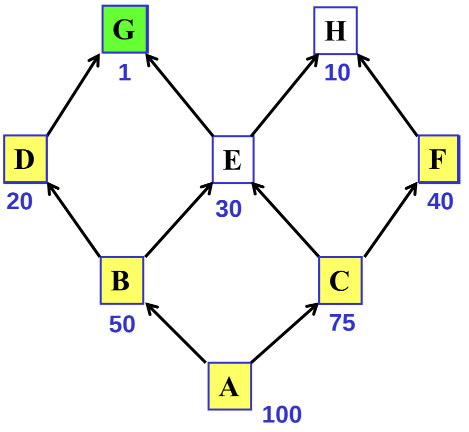
Execution (1)



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

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

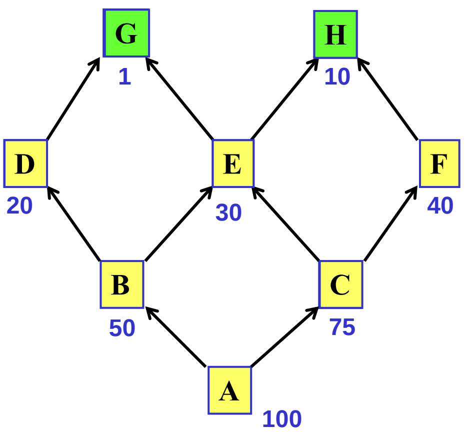
Execution (2)



➲ **Run 2**

- **C: 25x2=50 (C and F; 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)

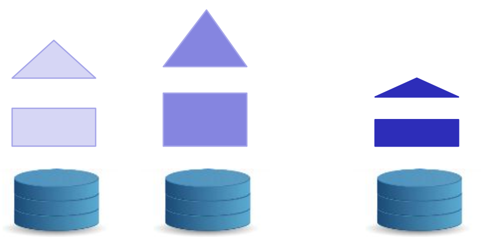


➲ **Run 3**

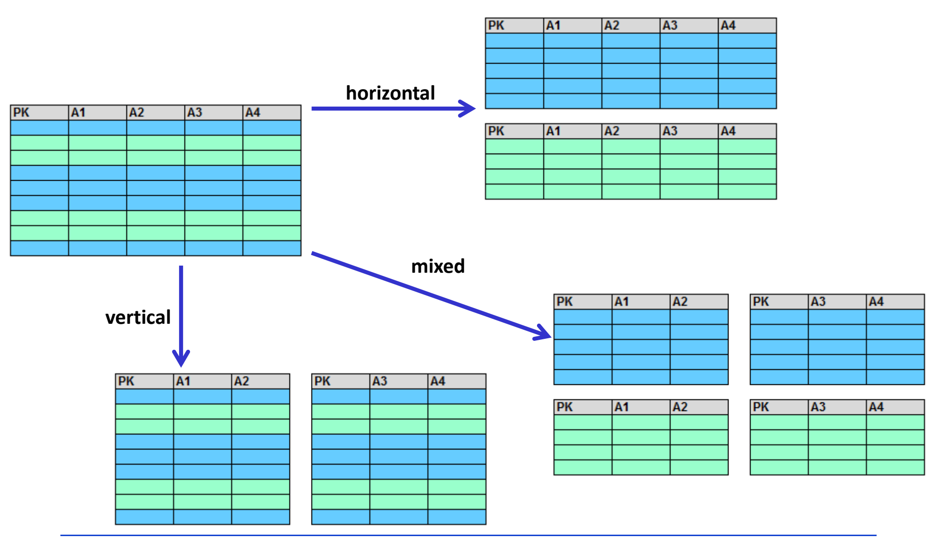
- **C: 25=25 (C; E, G computed from B)**
- **D: 30x2=60**
- **E: 20+20+10 (E-F)=50**
- **G: 49**
- **H: 30**

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 [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 Sales_Range_TKey
to (Data01, Data02, Data03, Data04, Data05);
```

filegroup

```
create table 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')))
```

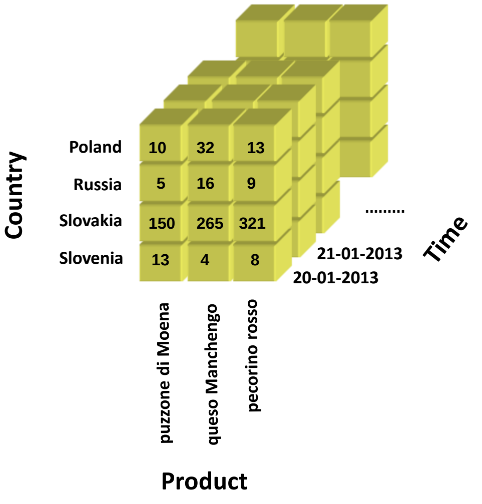
```
CREATE TABLE order_items
(order_id NUMBER(12) NOT NULL, ...,
 CONSTRAINT order_items_orders_fk FOREIGN KEY (order_id)
            REFERENCES orders(order_id))
PARTITION BY REFERENCE (order_items_orders_fk)
```


Partition-wise join (2)

orders (Jan-2006)	orders (Feb-2006)	orders (Mar-2006)	......
order_items (Jan-2006)	order_items (Feb-2006)	order_items (Mar-2006)	......

```
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

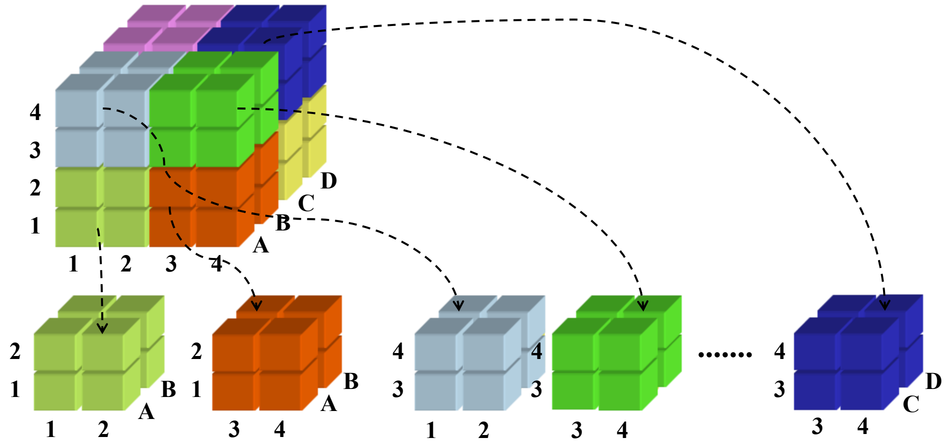
MOLAP (2)

➲ 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)

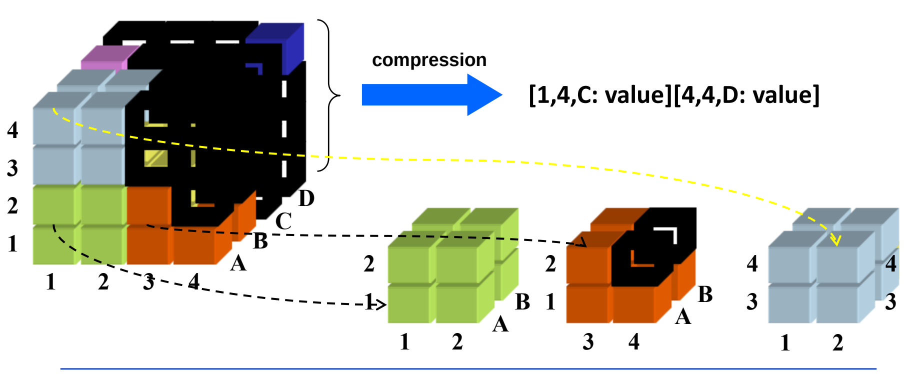
➲ **Cube chunking**



MOLAP (4)

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

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



MOLAP (5)

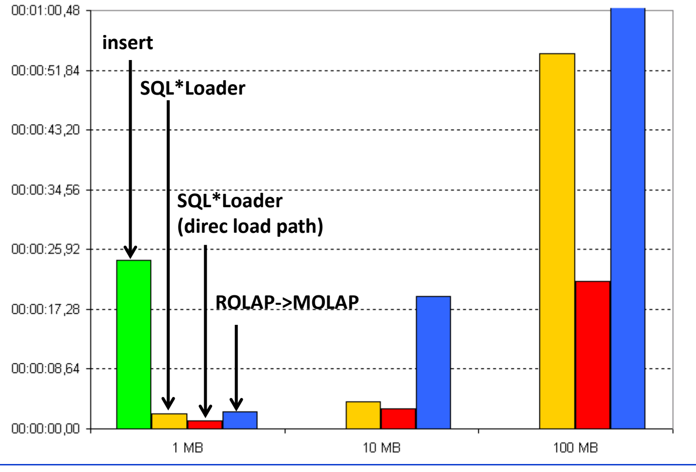
- ➲ More efficient than ROLAP for aggregate computing
- ➲ Efficient when a cube contains a few dimension
- ➲ Loading less efficient than ROLAP

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)

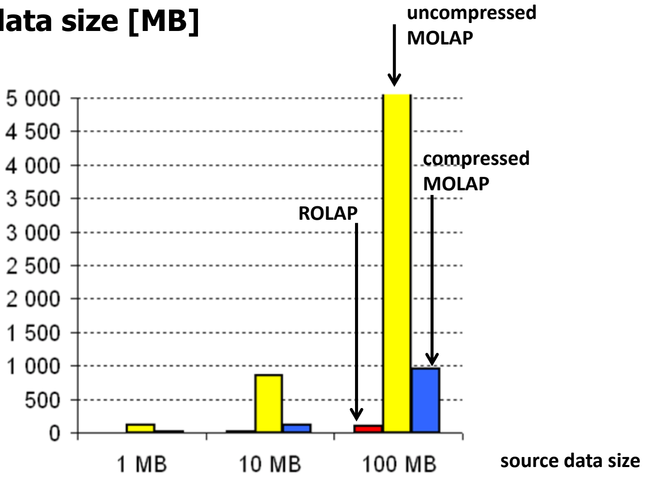
Our experiment (2)

➲ **Load time [hh:mi:ss]**

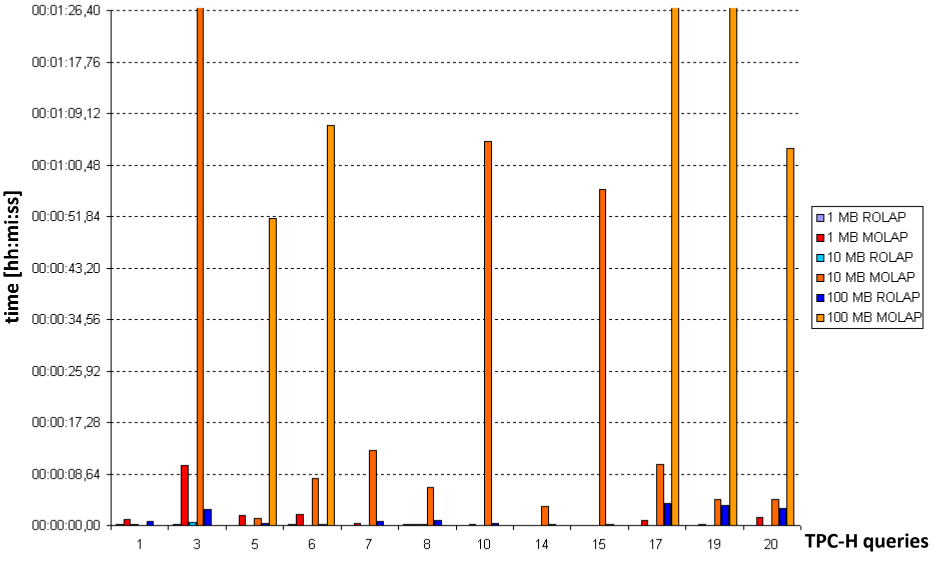


Our experiment (3)

➲ **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 indxs, bitmap join indxs, materialized views, partitioning**
  - **sophisticated query optimizer**
- **the presented results shuldn't be generalized ⇨ they show the characteristics of MOLAP impementation in a particular version of the system**

Other directions

- 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