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

- **Index structures**
  - **compression techniques for bitmap indexes**
  - **join index**
  - **bitmap join index (Oracle)**
  - **clustered index (DB2)**
  - **multidimensional cluster MDC (DB2)**

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 / **bitmap No** / **bitmap range**

	B4	B3	B2	B1	B0
tempC	(100, 80>	(80, 60>	(60, 40>	(40, 20>	(20, 0>
21	0	0	0	1	0
39.6	0	0	0	1	0
51.3	0	0	1	0	0
12	0	0	0	0	1
98.8	1	0	0	0	0
71	0	1	0	0	0
68.8	0	1	0	0	0
50.4	0	0	1	0	0
40	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

- **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 hipermarket**
  - **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

- query: select * from Products
  where product = 'pecorino d'Abruzzo'
- apply mask: 00...1000

dimension Products

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

Encoding

```
select sum(quantity) from Sales
where product = 'pecorino d'Abruzzo'
```

```
where B0=0 and B1=0 and B2=0 and B3=1 and ...
```

Sales

...	quantity	product	B15	B14	...	B3	B2	B1	B0
...	2	queso Manchengo	0	0	0	0	0	0	1
...	3	queso de Burgos	0	0	0	0	0	1	0
...	1	queso Manchengo	0	0	0	0	0	0	1
...	**4**	**pecorino d'Abruzzo**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
...	1	queso Manchengo	0	0	0	0	0	0	1
...	5	queso de Urbasa	0	0	0	0	1	1	0
...	2	pecorino baccellone	0	0	0	0	1	1	1
...	**3**	**pecorino d'Abruzzo**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
...	**2**	**pecorino d'Abruzzo**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
...	1	pecorino di Farindola	0	0	0	1	0	1	0
...	**1**	**pecorino d'Abruzzo**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
...	2	pecorino rosso	0	0	0	1	1	0	0
...	**2**	**pecorino d'Abruzzo**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
...	1	pecorino sense	0	0	0	1	1	1	0
...	...	...	...	...	...	...	...	...	...

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)

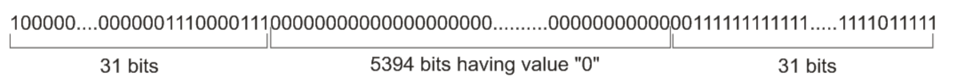
Compression (2)

- 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**

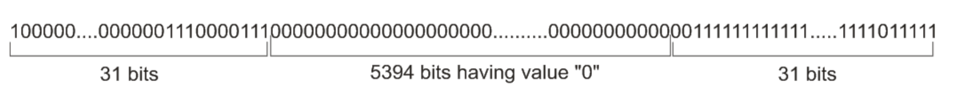
- **taken from** Stockinger K., Wu K.: Bitmap Indices for Data Warehouses. In Wrembel R. and Koncilia C. (eds.): Data Warehouses and OLAP: Concepts, Architectures and Solutions. IGI Global, 2007

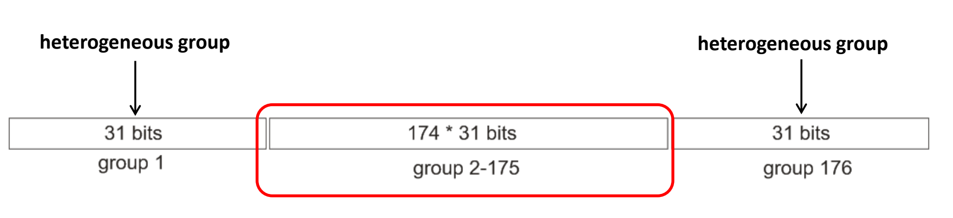


➲ **Step 1: divide the bitmap into groups including 31 bits each**

WAH (2)

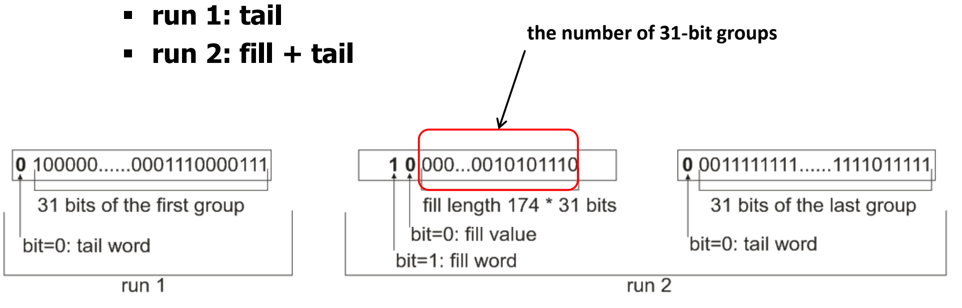
➲ **Step 2: merge adjacent homogeneous groups (having the same values of all bits, i.e., groups 2-175)**





WAH (3)

- **Step 3: group encoding**
  - **run: fill + tail**
  - **run 1: tail**
  - **run 2: fill + tail**



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**

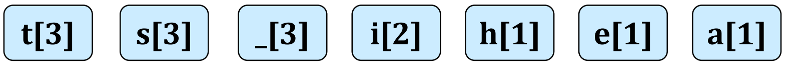
Huffman Encoding

- **Concept**
  - **original symbols from a file being compressed 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

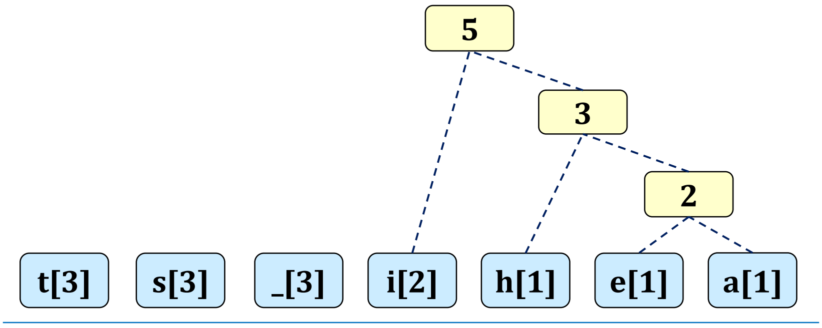
- **Example: encoding text "this_is_a_test"**
- **Step 1: frequencies of the symbols in the encoded string**

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



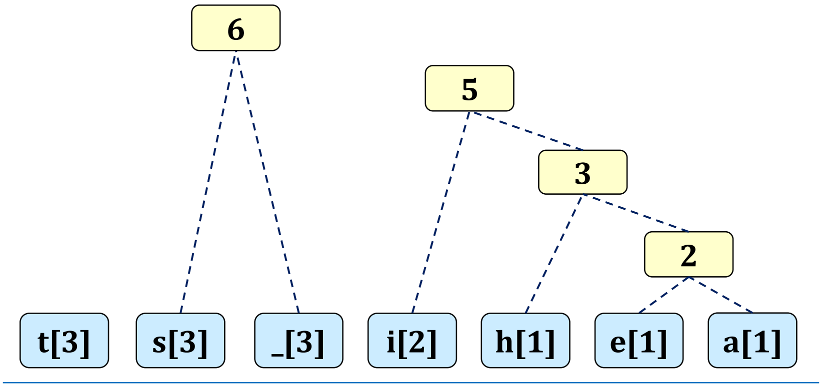
Huffman Encoding

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



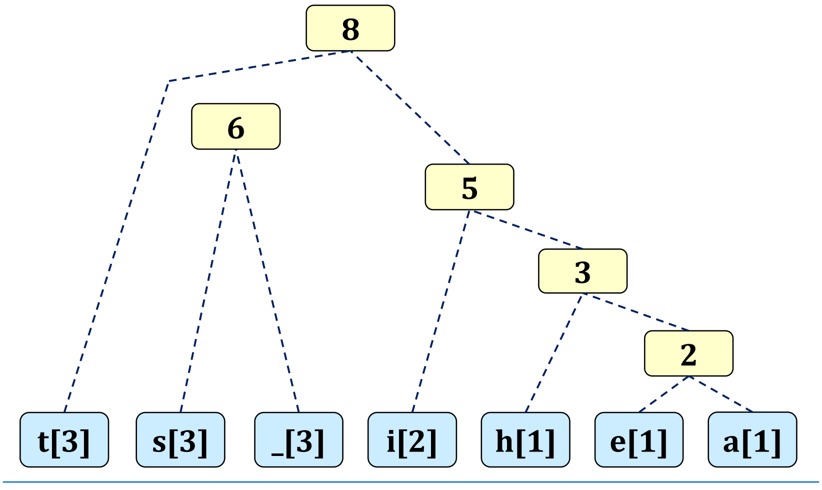
Huffman Encoding

➲ **Step 2: building Huffman tree**
- **merge nodes of the lowest frequency**



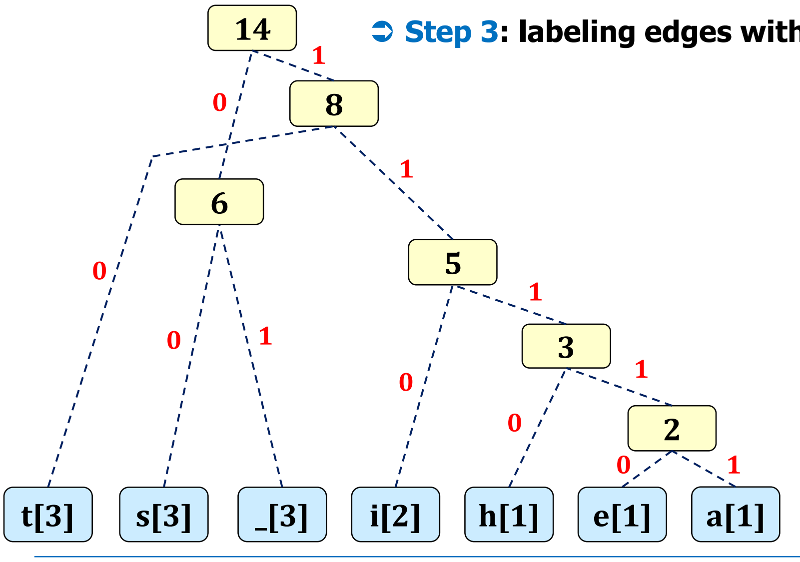
Huffman Encoding

➲ **Step 2: building Huffman tree**



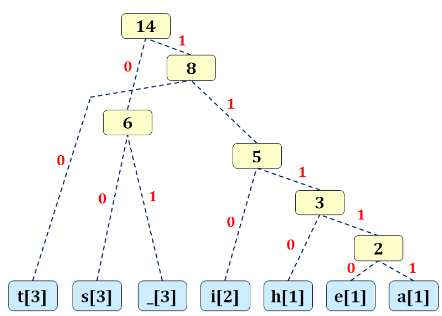
Huffman Encoding

➲ **Step 3: labeling edges with 0 or 1**



Huffman Encoding

➲ **Step 4: getting the codes of the symbols from the Huffman tree**



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

codes of symbols

Huffman Encoding

➲ **Step 5: replacing original symbols with their codes**
- **orignal text: 14B**
- **compressed text: 38b ⇨ 5B**

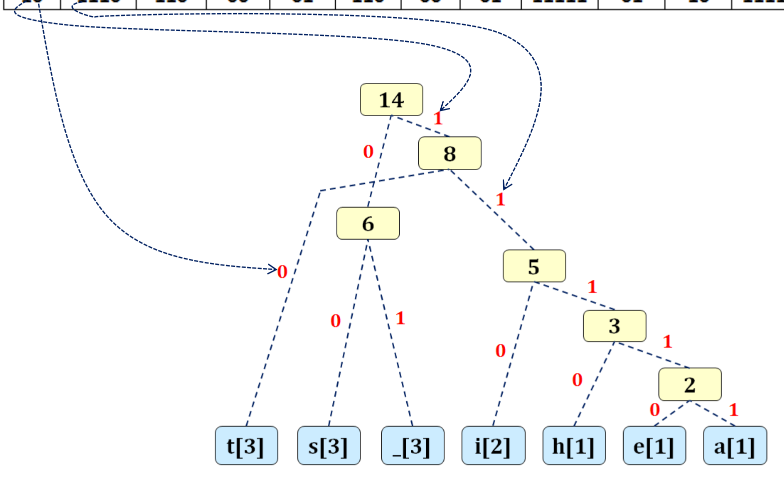
t	h	i	s	_	i	s	_	a	_	t	e	s	t
10	1110	110	00	01	110	00	01	11111	01	10	11110	00	10

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

codes of symbols

Decoding

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

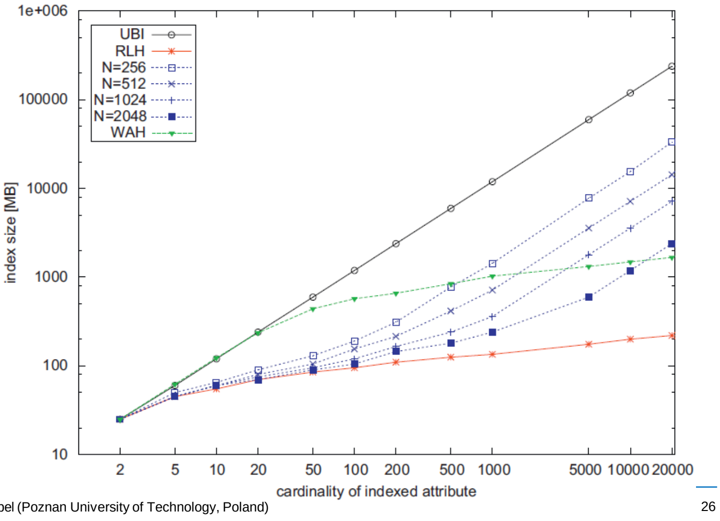


Experimental Evaluation

- 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)

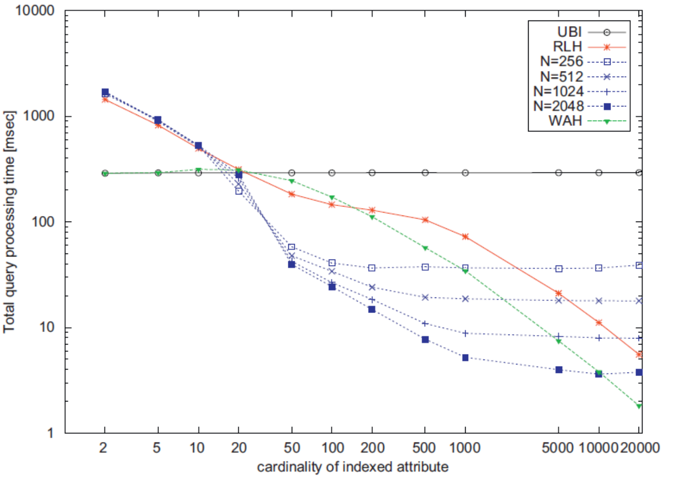


WAH and RLH: response times

- **Query:**

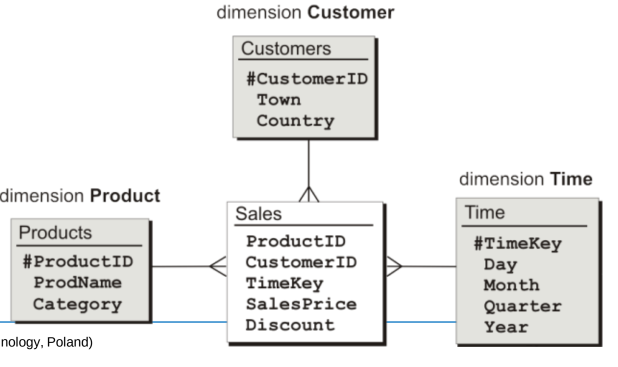
```
select ... from ...
where ind_attribute in (v1, v2, ..., v100)
```

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



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	salesID	salesPrice	discount	productID
0AA0	1	...	5	100
0AA1	2	...	15	230
0AA2	3	...	5	100
0AA3	4	...	10	300
0AA4	5	...	10	300
0AA5	6	...	15	230

P.productID	P.ROWID	S.ROWID	S.salesID
100	BFF1	0AA0	1
100	BFF1	0AA2	3
230	BFF2	0AA1	2
230	BFF2	0AA5	6
300	BFF3	0AA3	4
300	BFF3	0AA4	5

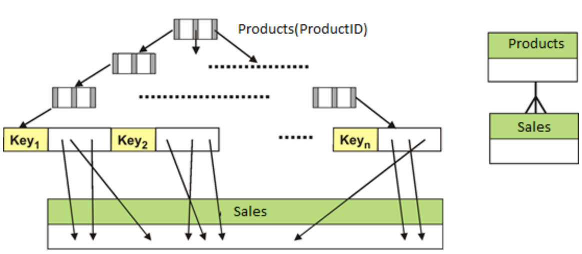
Join index

➲ **In order to make searching the join index faster, the join index is physically ordered (clustered) by one of the attributes (simple approach)**

➲ **The access to the join index can be organized by means of a B-tree or a hash index**

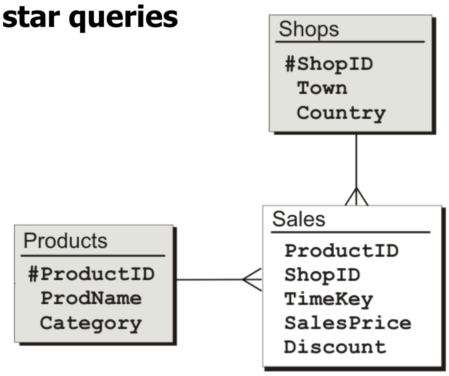
access technique
B-tree or hash

P.productID	P.ROWID	S.ROWID	S.salesID
100	BFF1	0AA0	1
100	BFF1	0AA2	3
230	BFF2	0AA1	2
230	BFF2	0AA5	6
300	BFF3	0AA3	4
300	BFF3	0AA4	5



BIs in Oracle

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



Bitmap Join Index (1)

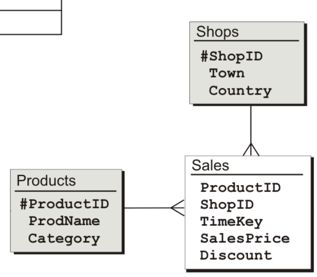
Products

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

Sales

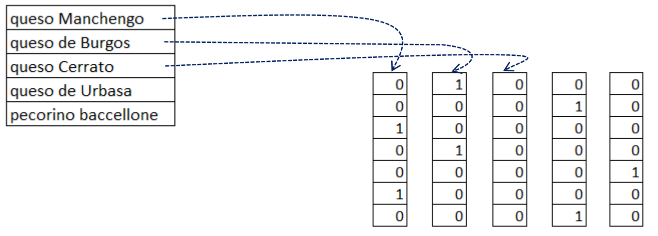
ProductID	SalesPrice	...
200	45	
400	50	
100	40	
200	55	
500	75	
100	65	
400	70	

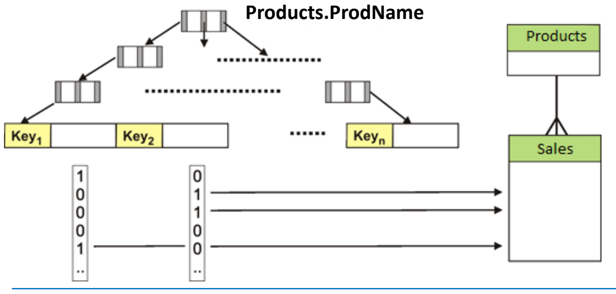
```
create bitmap index Sales_JBI
on Sales(Products.ProdName)
from Sales s, Products p
where s.ProductID=p.ProductID;
```



BJI (2)

bitmap join index on Products.ProdName



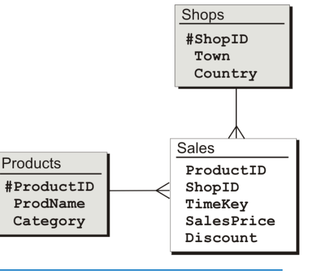


BJI (3)

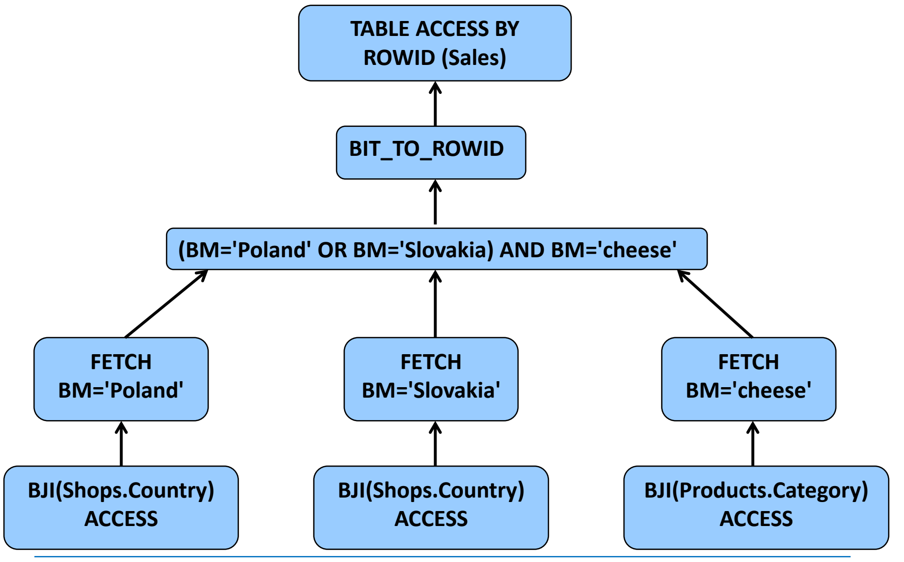
- **Star query optimization with the suport of BJI**

```
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`

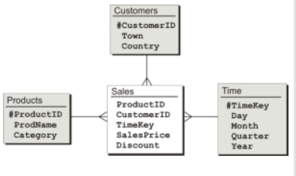


BJI (4)



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|
-----------------------------------------------------------------------------------------
```

on Customers.Town ('London')

on Products.ProdName ('ThinkPad Edge', 'Sony Vaio', 'Dell Vostro')

hash join Products - Sales

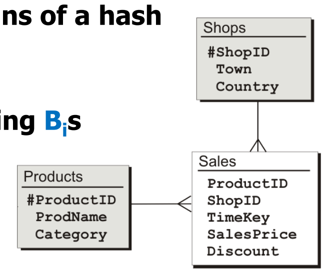
BJI (7)

```
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;
```

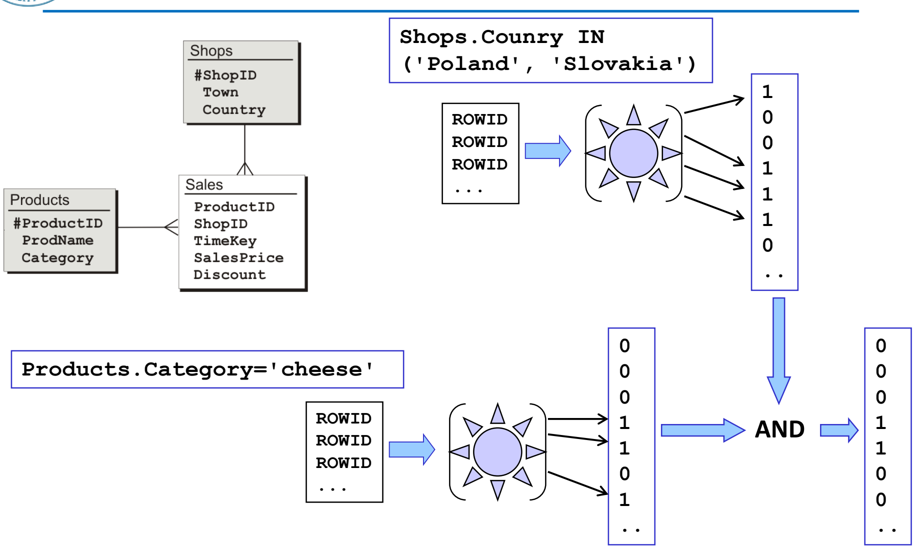
```
-----------------------------------------------------------------------------------------------
|Id | Operation                      |Name               |Rows |Bytes|Cost(%CPU)|Time      |
-----------------------------------------------------------------------------------------------
| 0 | SELECT 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)

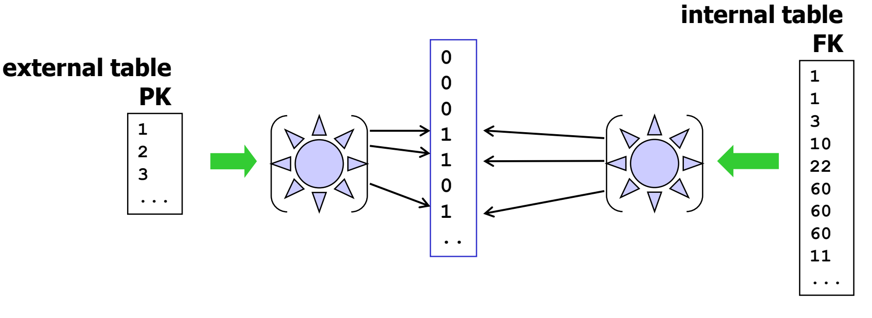


BIs in SQL Server (1)

- 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
  - a bitmap is used to check if a foreign key value joins with a primary key value

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



DB2: Clustering index (1)

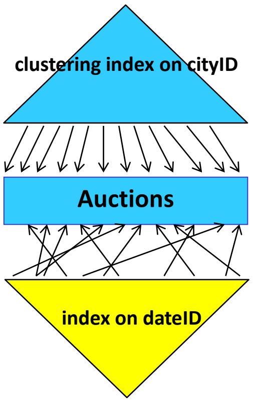
- **Clustering index determines how rows are physically ordered (clustered) on disk**
- **After defining the index, rows are inserted in the order determined by the index**
- **Only one index can be a clustering index (one physical order of rows on disk)**
- **By default the first index created is the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

oryginal 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	

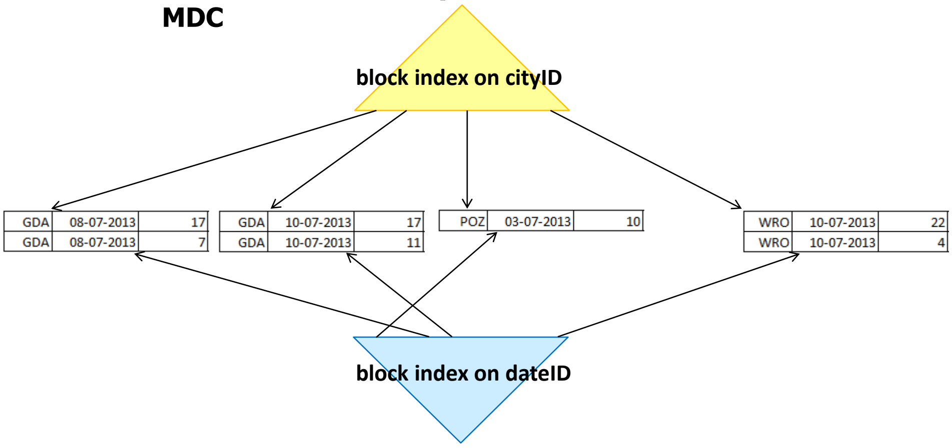
data block

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	

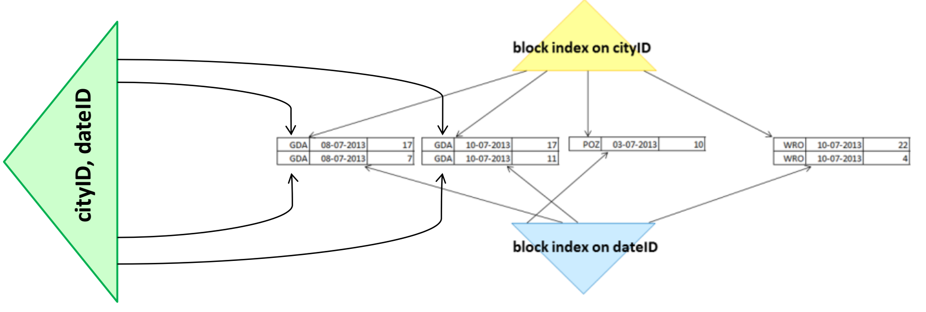
DB2: MDC (3)

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



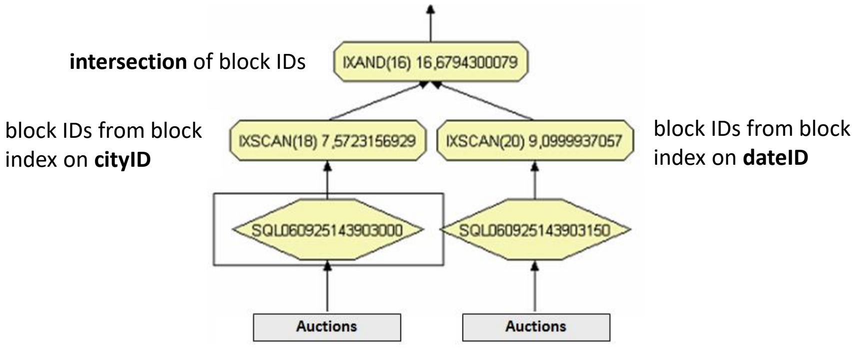
DB2: MDC (4)

- **Composite block index: includes all dimension key columns**



MDC in queries

```
SELECT SUM(quantity), cityID, dateID
FROM Auctions
WHERE cityID ='GDA' AND dateID='10-07-2013'
group by cityID, 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**
    - **B-tree indexes on single attributes in a MDC**
    - **B-tree concatenated index on all attributes in a MDC**
  - **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**

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