

## Improved Query Performance with Variant Indexes

Presenter: Manish Singh

10/30/09

EECS 584, Fall 2009

1

## Outline

- Motivation
- Problem Definition
- Definitions
  - Traditional Value-List & Bitmap Indexes
  - Projection & Bit-Sliced Indexes
- Algorithms
  - Aggregate queries, and
  - Queries having where clause
- Optimized Index strategy for OLAP queries
- Conclusions

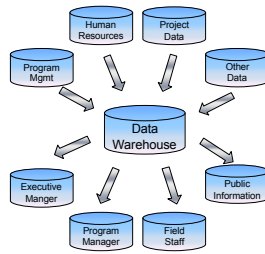
10/30/09

EECS 584, Fall 2009

2

## Motivation

- Application to **Data warehouses**
- Use for analyzing trends and anomalies
- Characteristics:
  - Periodic batch updates
  - No concurrent updates during queries
  - Expected set of queries may be known in advance



10/30/09

EECS 584, Fall 2009

3

## Problem Definition

- **Given:** A Data Warehouse type of environment
- How to optimize SQL queries, such as:
  - Aggregates,
  - Predicate evaluation, and
  - Group by type of queries(OLAP).
- **Goal:** Use the no concurrent update and query condition, by:
  - Reorganizing data and index to new optimal clustered form, and
  - Introducing specialized indexes.

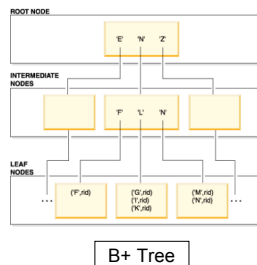
10/30/09

EECS 584, Fall 2009

4

## Traditional Value-List index

- RID: Row Identifier



10/30/09

EECS 584, Fall 2009

5

## Bitmap Indexes

- An alternative representation for RID-lists.
- Space efficient
  - when number of key values is low i.e., dense bitmaps.
- CPU-efficient
  - AND, OR, and NOT operations.

partno	color	size	weight
1	GREEN	MED	96.1
2	RED	MED	124.1
3	RED	SMALL	100.1
4	BLUE	LARGE	54.9
5	RED	MED	124.1
6	GREEN	SMALL	60.1
...	...	...	...

Bitmap Index on color

```

color = 'BLUE'  0 0 0 1 0 0 ...
color = 'RED'   0 1 1 0 1 0 ...
color = 'GREEN' 1 0 0 0 0 1 ...
    
```

Part number 1 2 3 4 5 6

SELECT partno from PARTStable where colour = 'RED' and size = 'LARGE'

Take AND of bitmaps corresponding to color = 'RED' and size = 'LARGE'

10/30/09

EECS 584, Fall 2009

6

## Projection Index

RowId	Store	Date	TransactionId	Dollar_sales
1	ABC	10/12/09	ABC_13	1100
2	XYZ	10/14/09	XYZ_19	2000
3	XYZ	10/19/09	XCY_20	2500
4	ABC	10/21/09	ABC_14	3000

■ Most queries retrieve only a few columns.  
 - Acts as an auxiliary aid  
 ■ Reduces number of disk page access.

Dollar_sales
1100
2000
2500
3000

10/30/09 EECS 584, Fall 2009 7

## Bit-Sliced Index

RowId	Store	Date	TransactionId	Dollar_sales
1	ABC	10/12/09	ABC_13	1100
2	XYZ	10/14/09	XYZ_19	2000
3	XYZ	10/19/09	XCY_20	2500
4	ABC	10/21/09	ABC_14	3000

■ Bitmap index not suitable  
 ■ Improves efficiency:  
 - Allows bit operations even in Value-List index.

$B_{i=1}$

Dollar_sales
010001001100
011111010000
1001110000100
101110111000

To find Dollar\_sales > 1023  
 → Need to consider only bitmaps  $B_{i=11}$  and  $B_{i=12}$

10/30/09 EECS 584, Fall 2009 8

## Comparison of Indexes (for Single Column Sum Aggregate)

■ **Experiment:**

- **Table:** SALES(RowId, Store, Date, TransactionId, Dollar\_sales, ...)
- **Query:** SELECT SUM (Dollar\_sales) FROM SALES WHERE condition;

■ SALES table has 100 million rows

- Each tuple of size 200 bytes
- Page size 4 KB
- The query result has 2 million rows

10/30/09 EECS 584, Fall 2009 9

## Plan 1: Direct Access

- Each page has 20 tuples.
- SALES table occupies 5,000,000 disk pages.
- Query result has 2,000,000 rows.
  - Required # of disk page access =  $5,000,000(1 - e^{-2,000,000/5,000,000}) = 1,648,400$  pages

10/30/09 EECS 584, Fall 2009 10

## Plan 2: Using Projection Index

- Each disk page can contain 1000 (4 byte each) column values.
  - Projection Index require: 100,000 disk pages
  - Only 100,000 disk accesses.

10/30/09 EECS 584, Fall 2009 11

## Plan 3: Value List (Bitmap) Index

- $B_i$ : Bitmap representing query result
  - Bits set to 1 for rows which satisfy the where condition
- $B_{nm}$ : Bitmap for non-null values in a column

**Algorithm**

```

if (COUNT (Bi AND Bnm) = 0)
  Return null;
SUM = 0.0;
for each non-null value v in the index for C {
  Designate the set of rows with value v as Bv
  SUM += v * COUNT(Bi AND Bv);
}
Return SUM;

```

- $B_i$ : 100,000,000 bits = 12,500,000B → 3125 pages
- $B_v$ : 100,000,000 RIDs of 4 bytes each → 100,000 pages
- Requires 103,125 page accesses.

10/30/09 EECS 584, Fall 2009 12

## Plan 4: Using Bit-Sliced Index

**Algorithm**

```

if (COUNT(Bi AND Bm) = 0)
  Return null;
SUM = 0;
for i = 0 to N {
  SUM += 2i * COUNT(Bi AND Bj)
}
Return SUM;

```

Dollar_sales
010001001100
011111010000
1001110000100
101110111000

- N: Max number of bits required to represent all possible values in the column. N= 19
- B<sub>i</sub> and B<sub>m</sub>: 100,000,000 bits each = 12,500,000B → 3125 pages
- B<sub>j</sub>: 100,000,000 bits each → 3125 pages
- Requires 21\*3125 = 65,625 page accesses.

10/30/09 EECS 584, Fall 2009 13

## Comparison of Algorithm Performance

Method	I/O	CPU contribution
Add from Rows	1,341 K	I/O + 2M*(25 ins)
Projection index	100K	I/O + 2M*(10 ins)
Value-List index	103K	I/O + 100M*(10 ins)
Bit-Sliced index	69K	I/O + 197M*(1 ins)

10/30/09 EECS 584, Fall 2009 14

## Comparison for Other Aggregate Queries

Aggregate	Value-List Index	Projection Index	Bit-Sliced Index
COUNT	Not needed	Not needed	Not needed
SUM	Not bad	Good	Best
AVG	Not bad	Good	Best
MAX/MIN	Best	Slow	Slow
MEDIAN,N-Tile	Usually Best	Not Useful	Sometimes Best
Column-Product	Very Slow	Best	Very Slow

10/30/09 EECS 584, Fall 2009 15

## Comparison For Range Queries

Range Evaluation	Value-List Index	Projection Index	Bit-Sliced Index
Narrow Range	Best	Good	Good
Wide Range	Not Bad	Good	Best

10/30/09 EECS 584, Fall 2009 16

## Online Analytical Processing, or OLAP

- OLAP is an approach to answer analytical queries that are multi-dimensional in nature.
- The goal is to do efficient computation of some expected set of queries.
- OLAP addresses GROUP BY queries on different combination of columns, known as **dimensions**.

10/30/09 EECS 584, Fall 2009 17

## Star Schema

Product (Dimension Table)			
PID	name	grade	brand
P1	Apple	A1	B1
P2	Plum	A5	B2
....	....	....	....

CUSTOMER (Dimension Table)			
CID	name	address	city
C3	Amy	Cram Pl	Ann Arbor
C4	Ben	Bishop	Ann Arbor
C5	Carl	Bishop	Ann Arbor

```

Select Sum(Dollar_sales)
From Sales S
Natural Join
Product P
Natural Join
Customer C
Where P.Brand='B1' AND
C.city='Ann Arbor'

```

OID	CID	PID	Qty	Dollar_sales
100	C3	P1	1	12
102	C3	P2	2	17
105	C5	P1	5	13
....	....	....	....	....

SALES(Fact Table)

10/30/09 EECS 584, Fall 2009 18

## Summary Tables

- Pre-calculate results of frequent GROUP BY queries and store them as summary table.

brand	city	Dollar_Sales
B1	Ann Arbor	1000
B1	Ypsilanti	2000
...	...	...

- Size of summary table grows as the product of number of values in independent dimensions.
- Cannot handle non-dimensional restrictions and dimensions not foreseen in advance.

10/30/09

EECS 584, Fall 2009

19

## Join Index

CID	name	address	city
C3	Amy	Cram Pl	Ann Arbor
C4	Ben	Bishop	Ann Arbor
C5	Carl	Bishop	Ann Arbor

OID	CID	PID	Qty	Dollar_sales
100	C3	P1	1	12
102	C3	P2	2	17
105	C5	P1	5	13
....	....	....	....	....

city	PID
Ann Arbor	P1
Ann Arbor	P2
...	...

City where various products are purchased

- Avoid actual join.
- # of join index would be exponential to the number of independent columns.

10/30/09

EECS 584, Fall 2009

20

## Bitmap Join Index

Index on Table T based on a single column of table S, where S commonly joins with T in a specified way.

PID	name	grade	brand
P1	Apple	A1	B1
P2	Plum	A5	B2
P3	Pears	A1	B1
P4	Guava	A3	B3
P5	Grapes	A7	B2
...	...	...	...

S (Dimension Table)

T (Fact Table)

OID	CID	PID	Qty	Dollar_sales
100	C3	P1	1	12
102	C3	P2	2	17
105	C5	P1	5	13
110	C11	P3	5	25
....	....	....	....	....

B1	B2	B3
1	0	0
0	1	0
1	0	0
1	0	0
....	....	....

Grows linearly with number of additional important columns

10/30/09

EECS 584, Fall 2009

21

## Bitmap Join Index

Select Sum(Dollar\_sales) From Sales S Natural Join Product P Natural Join Customer C Where P.Brand='B1' AND C.city='Ann Arbor'

Ann Arbor	Ypsilanti
1	0
1	0
1	0
0	1
....	....

B1	B2	B3
1	0	0
0	1	0
1	0	0
1	0	0
....	....	....

1
0
1
0
....

Use the Projection Index on column Dollar\_Sales in table Sales (No Join!)

10/30/09

EECS 584, Fall 2009

22

## Calculating Groupset Aggregates

- Aggregation Query:

- F is fact table
- D<sub>i</sub>'s are dimension table

```
Select Sum(Sales.Dollar_sales)
From Sales S, Product P, Customer C
Where Condition
GROUP BY Product, City
Select Sum(F.A)
From F, D1, D2, D3
Where Condition
GROUP BY D1.d1, D1.d2, D3.d3
```

- Using Projection Index

- Bring the relevant values of F.A in memory and create groups by looking at dimensions.

- Compute groups using Value-List Index

- Can be used if grouping not possible in memory using projection index.

10/30/09

EECS 584, Fall 2009

23

Thank you!

10/30/09

EECS 584, Fall 2009

24