Indexes

- B-tree index
- Bitmapped index
- Bitmapped join index

All DBMSs provide variations of b-trees for indexing

A data warehousing DBMS will likely provide these, or variations, on these

B-tree structures

- Most used access structure in database systems.
- There are b-trees, b+-trees, b*-trees, etc.
- B-trees and their variations are:
  - balanced
  - very good in environments with mixed reading and writing operations, concurrency, exact searches and range searches
  - provide excellent performance when used to find a few rows in big tables
  - are studied in ACS-3902

Bitmapped index

Consider the following Customer table

Cust_id	gender	province	phone
22	M	Ab	(403) 444-1234
44	M	Mb	(204) 777-6789
77	F	Sk	(306) 384-8474
88	F	Sk	(306) 384-6721
99	M	Mb	(204) 456-1234

Province:

Mb … rows 2, 5

Sk … rows 3, 4

Ab … row 1

Gender:

M … rows 1, 2, 5

F … rows 3, 4

Bitmapped index

- Suppose for a relation R the cardinality of attribute A is *c* and so we can represent the values existing for A as a_1, a_2, … a_c
- Then, if we have a bitmap index for R on attribute A there are ***c*** bit arrays, b_1, b_2, … b_c , one for each value of attribute A: b_i is the bit array corresponding to value a_i
- Consider b_k

  if the i^{th} row of R contains the value a_K for attribute A, then the i^{th} bit of b_k is 1

  otherwise the i^{th} bit of b_k is 0

Bitmapped index

- If we construct a bitmapped index for Customer on Gender we would have two bit arrays of 5 bits each

m	1	1	0	0	1
f	0	0	1	1	0

or:

m	f
1	0
1	0
0	1
0	1
1	0

Bitmapped index

- If we construct a bitmapped index for Customer on Province we would have three bit arrays of 5 bits each:

Ab	1	0	0	0	0
Mb	0	1	0	0	1
Sk	.	.	.	.	.

What values appear in this vector?

Bitmapped index

- Consider a query

```
Select Customer.name, Sum(s.amount)
From Sales s Inner Join Customer c On ( … )
where c.gender =M
and c.province = Mb
Group by Customer.name
```

How could the query access plan utilize bit map indexes?

Bitmapped index

- A *query* tree (*ACS-4902*) for

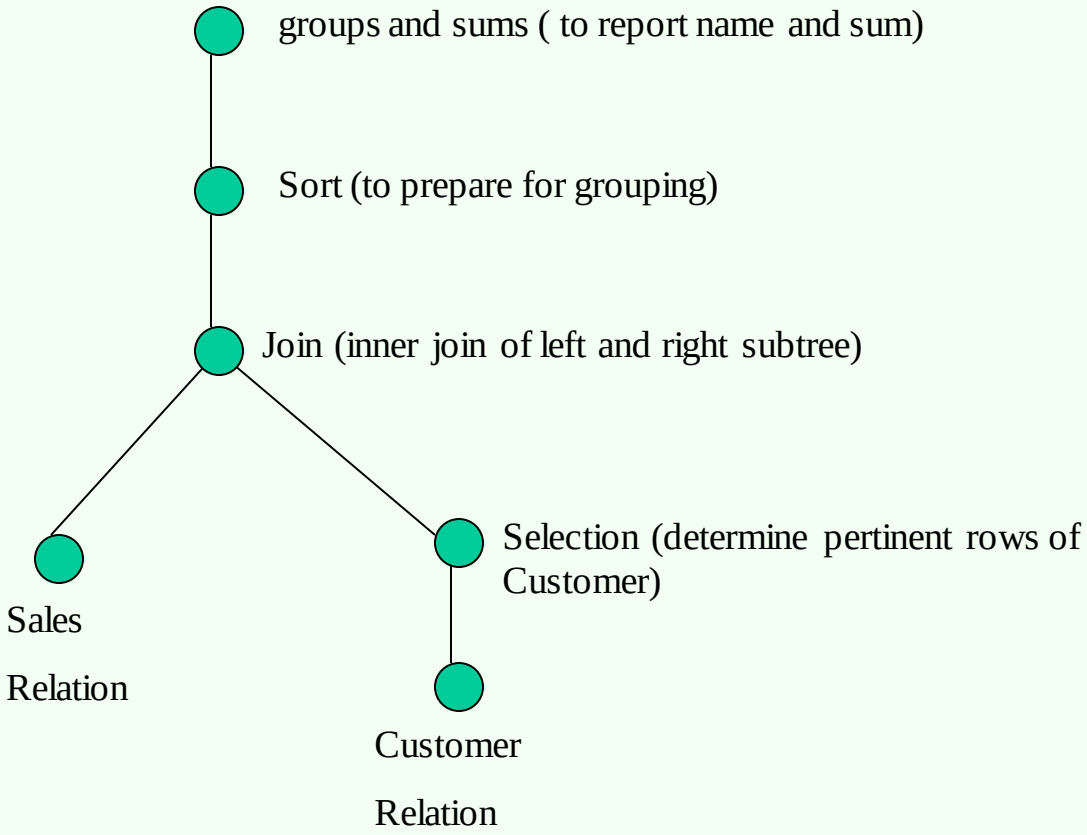
  Select Customer.name, Sum(s.amount)

  From Sales s Inner Join Customer c On (…)

  where c.gender =M

  and c.province = Mb

  Group by Customer.name



Bitmapped index

- Consider the where clause that selects rows of Customer

  c.gender =M

  and c.province = Mb

By *and*ing the two bit arrays for gender=M and province=Mb, the dbms knows which rows of Customer to join to Sales

M 1 1 0 0 1

Mb 0 1 0 0 1

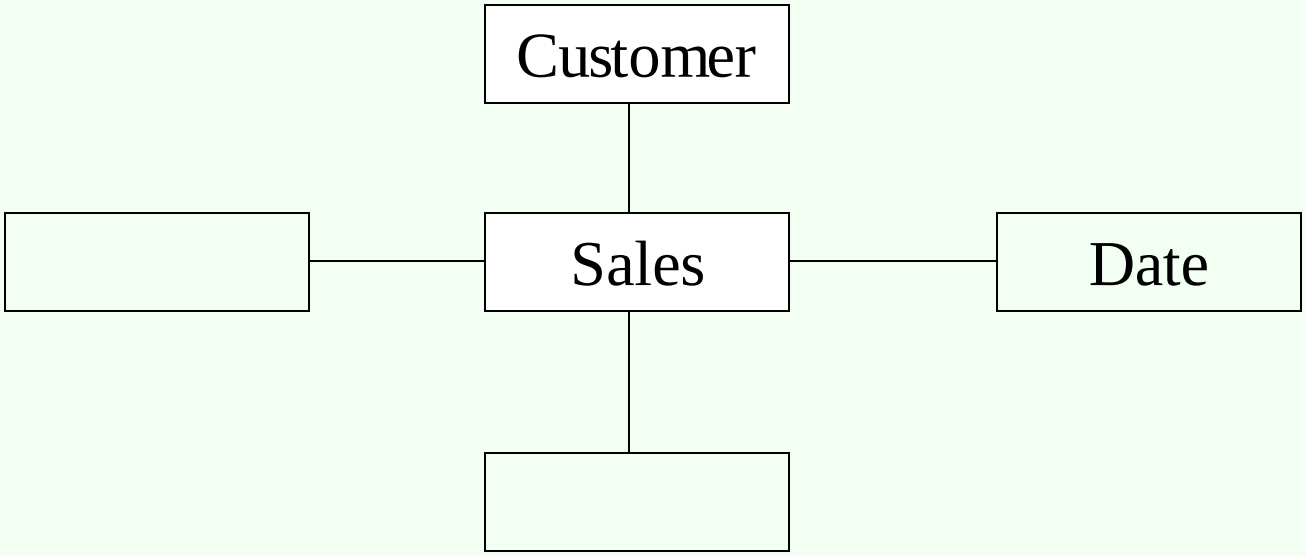
"and" ➔ 0 1 0 0 1

In our case, two rows of Customer are involved instead of the whole Customer table.

Bitmapped Join Index

In general, a join index is a structure containing index entries (attribute value, row pointers), where the attribute values are in one table, and the row pointers are to related rows in another table

Consider



Bitmapped Join Index

Customer

Cust_id	gender	province	phone
22	M	Ab	(403) 444-1234
44	M	Mb	(204) 777-6789
77	F	Sk	(306) 384-8474
88	F	Sk	(306) 384-6721
99	M	Mb	(204) 456-1234

Sales

row	**Cust_id**	**Store_id**	**Date_id**	Amount
1	22	1	90	100
2	44	2	7	150
3	22	2	33	50
4	44	3	55	50
5	99	3	55	25

Bitmapped Join Index

Example (Oracle): to create a bitmapped join index on Sales using the province attribute of Customer:

```
CREATE BITMAP INDEX cust_sales_bji
ON Sales(Customer.province)
FROM Sales, Customer
WHERE Sales.cust_id = Customer.cust_id;
```

Bitmapped Join Index

There are three province values in Customer. The join index will have three entries where each has a province value and a bitmap:

Mb	0 1 0 1 1
Ab	1 0 1 0 0
Sk	0 0 0 0 0

The bitmap join index shows that rows 2, 4, 5 of the Sales fact table are rows for customers with province = Mb

The bitmap join index shows that rows 1, 3 of the Sales fact table are rows for customers with province = Ab

The bitmap join index shows that no rows of the Sales fact table are rows for customers with province = Sk

Bitmapped Join Index

A bitmap join index could be used to evaluate the following query. In this query, the CUSTOMER table does not need to be accessed; the query can be executed using only the bitmap join index and the Sales table.

```
SELECT SUM(Sales.dollar_amount)
FROM Sales, Customer
WHERE Sales.cust_id = Customer.cust_id
AND Customer.province = Mb;
```

The bitmap index will show that rows 2, 4, 5 of the Sales fact table are rows for customers with province = Mb

HOBI and Time-HOBI

Reference:

Time-HOBI: Indexing Dimension Hierarchies by Means of Hierarchically Organized Bitmaps; Chmiel, Morzy, Wrembel; DOLAP ’10; October 30, 2010; Toronto, Ontario, Canada

See sections 3, 4 & 5.1, 5.2

Bit-Sliced Index (ignore till further notice)

Consider a numeric attribute c of a relation R.

Suppose n is the number of bits needed in the binary coding of values of c.

Suppose R has m tuples.

Let B be a bit matrix of n columns and m rows where $b_{i,j}$ is 1 if the coding of c in the i^{th} tuple has the j^{th} bit on.

Each column of B is stored separately.

Bit-Sliced Index

Consider the following Customer table

Cust_id	age	province	phone
22	20	Ab	(403) 444-1234
44	21	Mb	(204) 777-6789
77	22	Sk	(306) 384-8474
88	23	Sk	(306) 384-6721
99	40	Mb	(204) 456-1234

A bit-sliced index on age for Customer:

00010100
00010101
00010110
00010111
00101000

Bit-Sliced Index

Consider the following Customer table

Cust_id	age	province	phone
22	20	Ab	(403) 444-1234
44	21	Mb	(204) 777-6789
77	22	Sk	(306) 384-8474
88	23	Sk	(306) 384-6721
99	40	Mb	(204) 456-1234

A bit-sliced index on age for Customer:

00010100
00010101
00010110
00010111
00010100

Calculate the average age without using the Customer dimension:

Bit-Sliced Index

Similarly suppose there is a bit-sliced index on Sales based on the quantity attribute.

- To find the total sales quantity without going to the data (i.e. using the index only)
  - Examine the columns one by one… Accumulate a sum over the columns B_i, i=0, 1, 2, …:
    - For column i, count number of bits on, multiply by 2^i
- To find those sales where the quantity is > 63
  - Examine column B_6 to determine if the bit is on