Purpose of Indexing

and Bitmap Index

1

Indexes in general

- The purpose of indexes is to speedup data retrieval from a DBMS
- We have seen ISAM, B tree, B+ tree, and Hash indexes in detail. We will also discuss bitmaps as it is being used in the project
  - Some are static and others Dynamic (preferred)
  - Makes a difference in index management
  - Dynamic indexes need updates when data changes (insert/delete/modify)
  - On the other hand, static indexes require re-creating indexes periodically to account for changes
- Re-creation of indexes are expensive
- However, they do not need to be accessed during concurrent access!

2

Indexes in general

- For project Ia, we are asking YOU to choose indexes for each query
- However, choosing an index is non-trivial and DBMSs cannot put that burden on the end user!
- It is the job of the query optimizer to choose the 'best' index or indexes for each query, and
- It is the job of the DBA or a wizard to determine what indexes are useful (based on workload)!
- Wrong index use may increase response time
- Some queries may NOT be able to use indexes by their nature!

3

Choosing an Index for a SQL query

- It is non-trivial to determine which index to use and why
- How can one define the notion of 'best' indexes to use?
  - Remember more than one index can be used
- An SQL query can be complex and include conditions on multiple relations in addition to join, grouping (requires sort) and aggregation
  - Focus on WHEE clause
- It is done by the query optimizer using a 'cost' model
- Multiple query plans are generated – with different index choices – and each plan cost is estimated (using statistics and heuristics)
- Then the best cost plan is used for execution
  - Cost is measured in terms # of i/o's (page read and writes into the buffer)

4

Why are we asking you to Choose Index for a SQL query?

- To understand the relationship between SQL query processing and the role of indexes
- You are doing it intuitively rather than figuring out the cost of a plan
  - We will study this further in module IV on query optimization
  - We will come back to the SAME queries and look at their plans and costs for different use of indexes!
- To locate/retrieve tuples that satisfy a condition
  - You can either scan the entire relation (always works) **or**
  - You can use an index to search and retrieve only those tuples that satisfy a given condition (B+ tree index or bitmap index)
    - WHERE clause contains name = 'tom hanks'
    - WHERE clause contains year between '2000' and '2009'
    - WHERE clause contains genres in ('Comedy', 'Horror') // also LIKE operator
    - WHERE clause contains join on TCONST or NCONST from two relations
- Think how these select and join conditions can use the indexes available on relations in the query
  - Especially when multiple indexes exist
  - Some indexes may not be helpful

Issues with Indexes

- Definitely, there are advantages to using index provided they are present (i.e., created) and the optimizer knows which ones to use!
- Are there disadvantages?
  - Deciding what indexes to create on which relations
  - How many indexes to create
  - On which attributes (single or multiple)
  - Space occupied by the indexes (can be very large)
  - Most importantly,
    - Managing the indexes during insert/delete/update
    - Cost of updating indexes
    - Remember when a relation is modified, several/all indexes on that table may have to be modified
    - Concurrency control of indexes
- When an index is dropped, pre-compiled queries need to be updated
  - Managing query plan dependency on indexes!

Bitmap Index

- Bitmap is another type of index that was introduced for the warehouse data analysis
  - Data warehouses typically use star or snowflake schema
    - One very large table an multiple smaller satellite tables
  - Useful under certain conditions for speeding up identification of records/tuples
- Is different from the other indexes
- Is useful when data sizes are very large, and the number of distinct/unique key values are small
- Boolean and/or operations are faster as compared to set intersection and union used for other index types
  - Bit-wise operations are faster

Bitmap Index

- So, what is bitmap index?
- As the name indicates bitmaps (1 to indicate presence and 0 to indicate absence) are used rather than longer keys and disk pointers
- Actual records are not retrieved until they satisfy the condition
- Compound conditions can be efficiently processed if bitmaps are available for them
- Again, this index is useful only for equality and cannot be used for range searches

Bitmap Index intuition

- Suppose you are searching cars for specific color inside and outside combinations
- Assume there are millions of cars to search
- Assume there are only a few colors (cardinality of key)
- For each color there is a bit string indicating whether a record/car matches that color or not
  - For 1 Million cars, you need 1 M bits (for each key)
  - Instead of a large list of pointers pointing to each record containing that color
- With this representation, you retrieve records that have bit value 1 and position indicates the record id

Bitmap Index example

- For each color there is a bit string indicating whether a record matches that color or not
  - For 1 Million cars, you need 1 M bits (for each key)
  - Instead of a large list of pointers pointing to each record containing that color
- If you are looking for 'white' outer color and 'beige' inner color
  - You have two bit strings one for each
  - Their AND will satisfy both colors
- White 100010110001000111
- Beige 011101111001000000 will give 3 records
  11 1 to retrieve

Bitmap Index example

- When a query is run against a table with a bitmap index, the DBMS will use the bitmap to quickly identify which rows in the table match the search criteria. For example, consider the following query ?
- Select * from customers where gender = 'Female';
- To execute this query, the DBMS would use the bitmap index on the "gender" column to identify all of the rows in the table where the gender is male. It would do this by performing a bitwise AND operation on the "Male" bitmap and the bitmap for each row in the table. If the result of the AND operation is 1, it indicates that the row has the value "Female" for the "gender" column and should be included in the results.

Bitmap Index advantage

- When a query is run against a table with a bitmap index, the DBMS will use the bitmap to quickly identify which rows in the table match the search criteria. For example, consider the following query ?
- Select * from customers where gender = 'Female' and classification = 'sophomore';
- To execute this query, the DBMS would use the bitmap index on the "gender" column to identify all of the rows in the table where the gender is female. It would do a bitwise AND operation on the sophomore bitmap. If the result of the AND operation is 1, it indicates that the row has the value "Female" for the "gender" column and 'sophomore' for the classification.

Advantages of Bitmap Indexing

- Computation efficiency
- Space efficiency
- Developed in the context of data warehouses
- Useful for specific types of retrievals

Disadvantages of Bitmap Indexing

- Perhaps not as versatile as B+ tree indexing
- Not suited for high concurrency environments
  - Due to index updates for insert, delete etc.
- Not so good for small tables
  - Overhead outweighs benefits
- Not so good for columns with large number of unique/distinct values (Why?)
  - # of indexes will be large
  - Maintenance will have more overhead
  - E.g., course number, age, …

Bitmap Indexing

- In addition to a bitmap index on a single table, you can create a bitmap join index, which is a bitmap index for the join of two or more tables.
- A bitmap join index is a space efficient way of reducing the volume of data that must be joined by performing restrictions in advance.
- For each value in a column of a table, a bitmap join index stores the rowids of corresponding rows in one or more other tables.

Index sizes

- OWNER | TABLE_NAME | INDEX_NAME | INDEX_TYPE | COLUMNS_INDEXED | APPROX_MB_SIZE

- IMDB01 | NAME_BASICS | NB_BIRTHYEAR_BITMAP_IDX | BITMAP | BIRTHYEAR | 4.29
- IMDB01 | NAME_BASICS | NB_NCONST_BIRTHYEAR_IDX | NORMAL | NCONST,BIRTHYEAR | 307.59
  IMDB01 | NAME_BASICS | NB_NCONST_PRIMARYPROFESSION_IDX | NORMAL | NCONST,PRIMARYPROFESSION | 566.23
- IMDB01 | NAME_BASICS | NB_PRIMARYNAME_IDX | NORMAL | PRIMARYNAME | 498.59
  IMDB01 | NAME_BASICS | SYS_C00948177 | NORMAL | NCONST | 264.13
- IMDB01 | TITLE_BASICS | SYS_C00948176 | NORMAL | TCONST | 198.56
- IMDB01 | TITLE_BASICS | TB_GENRES_BITMAP_IDX | BITMAP | GENRES | 13.95
- IMDB01 | TITLE_BASICS | TB_STARTYEAR_GENRES_IDX | NORMAL | STARTYEAR,GENRES | 383.01
  IMDB01 | TITLE_BASICS | TB_TITLETYPE_BITMAP_IDX | BITMAP | TITLETYPE | 4.91
- IMDB01 | TITLE_BASICS | TB_TTYPE_GENRES_IDX | NORMAL | TITLETYPE,GENRES | 508.94
- IMDB01 | TITLE_PRINCIPALS | TP_CATEGORY_BITMAP_IDX | BITMAP | CATEGORY | 45.67
- IMDB01 | TITLE_PRINCIPALS | TP_NCONST_CAT_TCONST_IDX | NORMAL | NCONST,CATEGORY,TCONST | 2543.71
- IMDB01 | TITLE_PRINCIPALS | TP_TCONST_CAT_IDX | NORMAL | TCONST,CATEGORY | 1976.38
  IMDB01 | TITLE_PRINCIPALS | TP_TCONST_JOB_IDX | NORMAL | TCONST,JOB | 1614.41
- IMDB01 | TITLE_RATINGS | TR_AVERAGERATING_IDX | NORMAL | AVERAGERATING | 20.36
  IMDB01 | TITLE_RATINGS | TR_NUMVOTES_IDX | NORMAL | NUMVOTES | 19.49
- IMDB01 | TITLE_RATINGS | TR_TCONST_IDX | NORMAL | TCONST | 28.93
- IMDB01 | TITLE_RATINGS | TR_TCONST_NUMVOTES_IDX | NORMAL | TCONST,NUMVOTES | 33.35

- 18 rows selected.

Index sizes

- OWNER | TABLE_NAME | INDEX_NAME | INDEX_TYPE | COLUMNS_INDEXED | APPROX_MB_SIZE ----------IMDB01 | NAME_BASICS | NB_BIRTHYEAR_BITMAP_IDX | BITMAP | BIRTHYEAR | 4.29 IMDB01 | NAME_BASICS | NB_NCONST_BIRTHYEAR_IDX | NORMAL | NCONST,BIRTHYEAR | 307.59 IMDB01 | NAME_BASICS | NB_NCONST_PRIMARYPROFESSION_IDX | NORMAL | NCONST,PRIMARYPROFESSION | 566.23 IMDB01 | NAME_BASICS | NB_PRIMARYNAME_IDX | NORMAL | PRIMARYNAME | 498.59 IMDB01 | NAME_BASICS | SYS_C00948177 | NORMAL | NCONST | 264.13 IMDB01 | TITLE_BASICS | SYS_C00948176 | NORMAL | TCONST | 198.56 IMDB01 | TITLE_BASICS | TB_GENRES_BITMAP_IDX | BITMAP | GENRES | 13.95 IMDB01 | TITLE_BASICS | TB_STARTYEAR_GENRES_IDX | NORMAL | STARTYEAR,GENRES | 383.01 IMDB01 | TITLE_BASICS | TB_TITLETYPE_BITMAP_IDX | BITMAP | TITLETYPE | 4.91 IMDB01 | TITLE_BASICS | TB_TTYPE_GENRES_IDX | NORMAL | TITLETYPE,GENRES | 508.94 IMDB01 | TITLE_PRINCIPALS | TP_CATEGORY_BITMAP_IDX | BITMAP | CATEGORY | 45.67 OWNER | TABLE_NAME | INDEX_NAME | INDEX_TYPE | COLUMNS_INDEXED | APPROX_MB_SIZE ----------IMDB01 | TITLE_PRINCIPALS | TP_NCONST_CAT_TCONST_IDX | NORMAL | NCONST,CATEGORY,TCONST | 2543.71 IMDB01 | TITLE_PRINCIPALS | TP_TCONST_CAT_IDX | NORMAL | TCONST,CATEGORY | 1976.38 IMDB01 | TITLE_PRINCIPALS | TP_TCONST_JOB_IDX | NORMAL | TCONST,JOB | 1614.41 IMDB01 | TITLE_RATINGS | TR_AVERAGERATING_IDX | NORMAL |

17

Bitmap Indexing

- In a data warehousing environment, the join condition is an equi-inner join between the primary key column or columns of the dimension tables and the foreign key column or columns in the fact table.
- Bitmap join indexes are much more efficient in storage than materialized join views, an alternative for materializing joins in advance. This is because the materialized join views do not compress the rowids of the fact tables.
- Will discuss further in module IV, if time permits
- We are NOT created any bitmap join indexes for project Ia

18

Thank You !

19