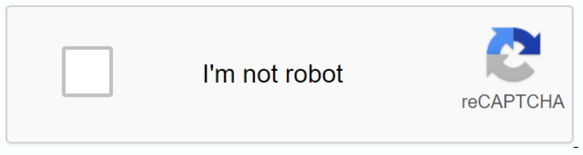

Continue

Query processing in dbms pdf

Query processing refers to the entire process or activity that translates query translation into low-level guidance, optimizing queries for storing resources, estimating or evaluating query costs, and extracting data from the database. Goal: To find an efficient query execution plan for a given SQL query, especially if you want to minimize the time. Cost factors: Disk access [typically time consuming], read/write operations [typically requiring resources such as memory/RAM]. The main steps involved in handling queries are described in the figure below. Figure 1 - Let's look at the entire process with an example of the database query processing steps. Let's consider the following two relationships as an exemplary table of discussion. Employees (Inno, Enami, Phone) Proj_Assigned (Inno, Proj_No, Role, DOP) Where, Ino is the employee number, Enami is the employee name, the Proj_No employee is the project number, the role is the role of the employee in the project, and the period of the project for several months. Using this information, let's write a query to find a list of all the employees working on a project that is more than 10 months old. Choose an Emmy name from the staff, Proj_Assigned staff. Eno = Proj_Assigned.Eno and DOP > 10; Input: Queries written in SQL are provided as input to the query processor. In our case, let us consider the SQL queries written above. Step 1: In this step of parsing, the parsing machine in the query processor module checks the syntax of the query, the permissions of the user executing the query, the name of the table, and the property name. The correct table name, property name, and user's permissions can be obtained from the system catalog (data dictionary). Step 2: If you have written a valid query, the translation is converted from advanced language SQL to a low-level command of relational algeum. For example, sql queries can be converted to relational algetypes, such as: φDOP>10 Δ Employee.Eno=Proj_Assigned.Eno (Employee X Prof_Assigned)3: The optimizer uses statistical data stored as part of the data dictionary. Statistical data is information about table size, record length, indexes generated in the table, and so on. The optimizer also checks for conditions and conditional attributes that are part of the query. Step 4: Action Plan A queries can be represented in several ways. In this step, the query processor module uses the information collected in Step 3 to find the same different relational algebraexpressions and return the results of the algebra that has already been created. For example, queries created with relational algerate can be written as queries below. γEname (employee ⋈Eno (Prof_Assigned) so far, we have two execution plans. The amount of time consumed or the space required to execute the query is different to execute the same result (obvious). Therefore, it is essential that one plan spends less money. In this step, choose one of the many execution plans that you have developed. This execution plan accesses the data in the database to provide the final result. In our example, the second plan can be good. The first plan joins two relationships (costly tasks) and then applies conditions (conditions are considered filters) to the combined relationship. This allows you to spend more time as well as space. In the second plan, one of the tables (Proj_Assigned) is filtered, and the results are combined with the Employee table. This join may need to reduce the number of records. Therefore, the second plan is the best (not always known information). Output: The end result is visible to the user. The full information described above is described in Figure 2 below. Figure 2 - Query Processing [Note: NJ in Step 4 means natural join] Skipping key content indicates the editing and execution of query specifications, typically expressed in declarative database query language (SQL), such as structured query language (SQL) when skipping to a table of query processing content. Query processing consists of compilation time and runtime steps. At compile time, the query compiler converts the query specifications into executable programs. This translation process (often referred to as query compilation) consists of query optimization and code generation steps, as well as vocabulary, syntax, and tax analysis of query specifications. The generated code typically consists of the actual operators on the database computer. These operators implement data access, joins, selection, projection, grouping, and aggregation. At runtime, the database engine interprets and executes programs that implement query specifications to produce query results. In the 1960s and 1970s, navigation database... This is a preview of the subscription content, check the access login. Code E.F. A relational data model of a large shared data bank. Komun. ACM., 13 (6):377-387, 1970.zbMATHGoogle ScholarFreytag J.C., Meyer D., Bosen G. 1994. Processing queries to advanced database systems. Morgan Kaufman, Google ScholarGraefe G. Volcano – scalable and parallel query evaluation system. IEEE Transnoll. Data Eng., 6 (1):120-135, 1994.Scholar GoogleGraefe G. Query evaluation technology for large databases. ACM computing. Surv., 25(2):73-170, 1993.Google Azarari R.A. and Fisher N.J. Access specification language for relational database systems. IBM J. Res. Dev., 23 (3):286-298, 1979.zbMATHGoogle ScholarMarkl V., Haas P.J., Kutsch M., Meadow N., Srivastava U., Tran T.M. Consistent Selectivity Estimates through Maximum Entropy. VLDB J., 16 (1):55-76, 2007.Google Scholar P.G., Astrahan M., Chamberlain D.D., Laurie R.A., and Price T.G. Access Path Selection from relational databases System. 1979, 1979, pp. 23-34.Google ScholarYu C.T. and ACM SIGMOD Int. on database query processing principles for advanced applications. Conf. Morgan Kaufman, 1997.Google Scholar© Springer Science + Business Media, LLC 20091.IBM Almaden Research Center San JoseUSA Database the main goal is to store, access, and manipulate relevant data in one place when users need it. Accessing and manipulating data must be done efficiently. It's easy and fast to access. However, the database is a system and the user is a different system or application or person. Users can request data in a language they understand. However, DBMS has its own language (SQL). As a result, users are prompted to query the database in that language. This SQL is a high-level language designed to build a bridge between the user and DBMS for communication. However, the underlying system of DBMS does not understand SQL. There must be a low-level language that these systems can understand. Typically, all queries written in SQL are converted to a low-level language with a relational algerate that the system can understand. However, it will be difficult for all users to write their own relational algetypes of queries. It requires thorough knowledge of it. So what DBMS does is ask the user to write a query in SQL. Check your code and then convert it to a lower-level language. It then selects the best execution path, executes the query, and gets the data from internal memory. All of these processes are called query processing. Query processing it is a step-by-step step of breaking a high-level language into a low-level language that the machine can understand and perform the requested work for the user. The query processor in DBMS does this. The diagram above shows how to handle queries in the database to display results. When a query is submitted to the database, the query compiler receives the query. The query is then examined and divided into individual tokens. When a token is created, it is verified by the parser. Tokenized queries are then converted to different relational expressions, relational trees, and relational graphs (query plans). The query optimizor then selects it to identify the best query plan to handle. Examine constraints and indexes in the system catalog and determine the best query plan. Create different execution plans for the query plan. The query execution plan then determines the best optimized execution plan for execution. The command processor then uses this execution plan to retrieve data from the database and return the results. An overview of how query processing works. Let's take a closer look below. There are four phases to handle a typical query. Parseand and translation query optimization trial or query code On the runtime processor of the DB. Parsing and translation is the first step in processing all queries. Users typically write requests in SQL language. To process and execute this request, dbms must be converted to a lower level, which is a machine-understandable language. All queries published to the database are selected first by the query processor. Scan and parse queries with individual tokens and check the accuracy of queries. Check the validity of the table/view used and the syntax of the query. When passed, each token is converted into relations, trees, and graphs. These are easily handled by other façades of DBMS. I'll use the examples to understand these steps. Suppose you want to see the student details you are studying in class DESIGN_01. When a user says , search for student details DESIGN_01 class, DBMS doesn't understand. Therefore, DBMS provides a language called SQL that allows users and DBMS to understand and communicate with each other. This SQL is written in simple English, such as a form that both understand. Therefore, the user writes his request to SQL as follows: STD_ID selection, STD_NAME, address, dob of student s, class c where s.CLASS_ID = c.CLASS_ID and c.CLASS_NAME = 'DESIGN_01'; When this query is issued, DBMS reads and converts it into a form that can be used to further process and synthesize DBMS. This query processing step is known as the parsing and translation phase. The query processor examines the submitted SQL queries and is divided into individual meaningful tokens. In our examples, 'choose*', 'student', 'class c', 'where', 's.CLASS_ID = c.CLASS_ID', 'AND' and 'c.CLASS_NAME = 'DESIGN_01' are other tokens. This tokenized query form is readily available on the processor for further processing. A query is generated from the data dictionary table to determine whether there are tables and columns for these tokens. If the data is not in the dictionary, the submitted query will fail at this stage. It also proceeds to find out if the syntax used in the query is correct. it does not check whether DESIGN_01 exists in the table, and checks whether there are SQL definition syntax, such as 'select* FROM', 'WHERE', 's.CLASS_ID = c.CLASS_ID', and 'AND'. Validating the syntax translates into relational algetypes, relational trees, and graph representations. This is easily understood and handled by the optimizer for further processing. The above query can be converted to two forms of relationship algesturotrial, as shown below. The first query first identifies the students in the DESIGN_01 class, and then selects only the requested columns. The other queries first select the requested columns from the student table, and then filter for DESIGN_01. Both results.∏ STD_ID, STD_NAME, address, DOB (σ CLASS_NAME = DESIGN_01 student ∞class) CLASS_NAME or DESIGN_01 (∏ STD_ID, STD_NAME, address, DOB (student ∞)) It can be represented in relational structures such as trees and graphs: Query processors apply rules and algorithms to these relational structures to represent a more efficient and robust structure that is only used by DBMS. These structures are based on mapping between tables, joining siesused, and the cost of the execution algorithm of these queries. When applying filters, etc., determine the selection and projection and projection and selection structure, which are efficient processing methods. In the third phase of query processing, the best structure and plan selected by the optimizer is selected and executed. Dig the database memory and retrieve the records according to your plan. In some cases, queries can be processed, compiled, and stored in the DB for use by the runtime DB processor. The results are then returned to the user. If a complex query is simple, it is the entire step that DBMS handles. All of these processes take a few seconds. However, the ideal optimization and execution path selection will create a faster value of query cost query costQuery costquery is the time it took to press the database by query and return the results. Query processing time,for example The time it takes to parse, translate, optimize, evaluate, run, and return results to the user is called query cost. It is used for a few seconds, but includes multiple subtasks and time taken from each task. To execute an optimized query involves pressing primary and secondary memory based on file organization methods. Depending on the file organization and the index esthen used, the time it took to retrieve the data may vary. Most of the time the query is used to access data in memory. There are several factors that determine the cost of access time, such as disk I/O time, CPU time, and network access time. Disk access time is the time it took for the processor to retrieve and locate records in secondary memory and return the results. The query takes most of the time to process. You can ignore other cases compared to disk I/O time. While calculating disk I/O time, only two factors are generally considered – and the time and transfer time are pursued. The search time is the time the processor took to find a single record in disk memory and is represented by a tS. For example, to find the student ID of student 'John', the processor is taken from memory based on index and file organization methods. It is called the time it took for the processor to hit a block of disks and retrieve the identity. The time spent by the disk returning the imported results to the processor/user is called transfer time and is represented by tT. Suppose the query needs to find the S time to get the record and return block B to the user. Then the disk I/O cost is calculated as below (S*tS)+ (B*tT), that is, the sum of the total time spent on s time and total search Shoot to transfer blocks b. Other costs, such as CPU costs, RAM costs, etc. are ignored because they are relatively small. Only disk I/O is considered query cost. However, because memory space/buffers depend on the number of queries running in parallel, you must calculate the maximum time spent on the query if the buffer is full or there is no buffer. Not all queries use buffers and cannot determine how many buffers/blocks are available for the query. The processor may have to wait for all the blocks of memory to be obtained. The impact of the index on CostWe has already seen the index and index-based file organization system. So you can imagine how it affects query processing and how you pin the search time. This significantly reduces the time to run queries. Let's look at the various indexes and their roles in reducing processing time. Density Index This type of index is indexed for each search key value and sorted based on the search key. For example, if the STD_ID is a search key, there is an entry for each STD_ID in the index table. In such cases we do not need to pass through the file from the beginning until we get the value we want. This index helps you get records directly into a single search. If you want to search for a range of records, it is sufficient to retrieve the initial record. Therefore, only one pursuit should be performed. Therefore the cost can be calculated as :tS+ (B*tT) and the total time it takes to transfer the b block of query and a single seeking time. In this type of sparse index index, the search key values are grouped into multiple blocks and only the start key values of the block are indexed. That is, ; If the STD_ID is a search key, its value is 10 records each (i.e., 100, 110, 120, 130...) grouped into and stored the index in the index table only 100, 110, 120, and 130. Therefore, when you need to retrieve a record, you can get a block where the data is stored in a single search, but it must pass linearly in the block until you get the requested record. That is, ; =119, in a single pursuit we will get into a block of 110 if we need to search STD_ID, but inside that block we must seek 9 times to get the crossing / 119. Therefore we need 10 pursuits in this case. Assuming you need to search for a range of records from 119 to 125, you should record index block 110 to get 119, and record 120 to get records from 120 to 125. Therefore, the number of blocks accessed and transmitted is 2.READ file processing system (S*tS) + (B*tT), so it is less efficient than the density index, but it is better than linear search at the beginning of the file (for example, finding numbers is less than linear search). However, there is less space compared to the dense index. Also, you don't have to worry about storing an index for a newly added or updated record. The default Index method is the primary key itself. The index is stored for these primary keys. It can be a dense or sparse index. However, because it is the primary key, the records are stored in an ordered format. So when we need to find the key, we will get it into one quest. Even if you need to search for values in a range, a single search can obtain the start record of the range. Then the rest of the records are imported sequentially. Therefore, the number of blocks to search is small. Secondary indexes where the index is created from a key other than the search key. Therefore we need to store each value of the secondary index in the index table, making it dense. In this method, it is difficult to retrieve the key value, and each record is traversed to get the record. In multi-level indexing of this method, the search key values are grouped into more than one level. That is, ; The initial index level includes a search key value of 100,200,300, each of which refers to a secondary level that has a range of search key values, such as 100 and 110,120. 200,210,.. These secondary-level indexes can point directly to the records in the file at the third level or. This reduces the number of block reads that can be seen in the sparse index. This is typically useful for binary search. The clustering index is clustered here with two or more search key values, pointing to the actual data in the file. It can also be multi-level. It reduces the search time as well as the number of blocks to read. B+ Tree Index This method keeps all search key values equal away from the root, and all search key values are on child nodes. The intermediary node only has pointers to the actual search key node. The height of the B+ tree is the same for all nodes. The maximum search required to retrieve key values in this method is logarithmic and log (h). However, this is compared to other indexes that can retrieve records in a single search. The advantages of the B+ tree are aligned and balanced. You don't have to rearrange records when there's an automatic insertion or deletion. This provides better performance for high-performance essential systems. System.