ILDER

@保存到为知笔记

@网路冷眼

【Top 10 performance tuning tips for relational databases】 http://t.cn/RqTWCXh 关系型数据库10大性能调节技巧。

Top 10 performance tuning tips for relational databases

Although newer relational databases and faster hardware run most SQL queries with a significantly small response time, there is always room for improvement. This paper lists 10 tips that every developer/DBA should consider when designing their database or writing SQL scripts. It also talks about some common mistakes developers typically make and how to avoid them.
Since most relational databases share same design concepts under their hood, this paper is not specific to any particular vendor. Although in our example we talk about five databases, these tips apply to a wider range of RDBMS.

Scenario

To help explain SQL queries mentioned in this document, we will assume a database with following tables. The remainder of this paper uses these tables as reference in explaining pros and cons of the design.

Table schema for Customer

Field Name	Data Type	Primary Key
CustomerID	Integer	Yes
FirstName	Varchar(30)	
LastName	Varchar(30)	
City	Varchar(30)	
State	Varchar(10)	
Phone	Varchar(15)	
AccountCreatedOn	Datetime	

Indexes for Customer

Index Name	Fields	Clustered
CustPK	CustomerID	Yes
IdxState	State	
IdxPhone	Phone	
IdxName	FirstName, LastName	
IdxCity	City, State	

Table schema for Orders

Field Name	Data Type	Primary Key
OrderID	Integer	Yes
CustomerID	Integer	
OrderDate	Datetime	
OrderCost	Decimal	
Shipped	Char(1)	

Indexes for Orders

Index Name	Fields	Clustered
OrderPK	OrderID	Yes
IdxCustDate	CustomerID, OrderDate	

Tip 1 - Database statistics

The most important resource to any SQL optimizer is the statistics collected for different tables within the catalog. Statistics is the information about indexes and their distribution with respect to each other. Optimizer uses this information to decide the least expensive path that satisfies a query. Outdated or missing statistics information will cause the optimizer to take a less optimized path hence increasing the overall response time.
Following table lists SQL commands for different database that is used to update statistics. For further details, refer to the reference manual for your particular RBDMS

Oracle:	ANALYZE command or DBMS_UTILITY package
DB2:	RUNSTATS command
MS SQL Server:	UPDATE STATISTICS
Informix:	UPDATE STATISTICS
Sybase ASE:	UPDATE STATISTICS

Optimizers always tend to select the least expensive path \u2013 one that returns least number of rows in fastest time. Why do optimizers rely on statistics? Consider the following query that is run against our sample database to answer this question.

```
select *
from customer
where city = 'New York City'
    and phone = '212-555-1212'
```

Notice that the above query contain two fields in the "WHERE" clause and there are two indexes defined, each containing one field. One very important notion to remember is that the optimizer can only use ONE index per table. Therefore, it has to make a decision as to which index to use. Since phone number should return least amount of rows, our query will run much faster if the optimizer always uses IdxPhone. However, if statistics are not updated, the optimizer does not know which index is better and may decide to choose IdxCity since 'city' field appears first in our WHERE clause. Once you update statistics the database will know more about the data distribution and will correctly choose the better index to run your query.

Tip 2 - Create optimized indexes

SQL optimizer heavily depends on indexes defined for a particular table. Indexes are double-edged sword: no index will degrade performance of your SELECT statements and too many indexes will slow down your DML (INSERT, UPDATE, and DELETE) queries. Therefore, it is important to have a right balance of index on tables. Besides the number of indexes, fields that are involved and their order is also very important.

When creating indexes, estimate the number of unique values the column(s) will have for a particular field. For example, the idxCity index in our sample database is not a good candidate for an index. When you wish to search for customers in New York City, it can potentially return thousands of rows, which are then searched sequentially. Such

candidate for an index. When you wish to search for customers in New York City, it can potentially return thousands of rows, which are then searched sequentially. Such indexes seldom help in speeding up SELECT queries and reduce the response time for DML queries.

Composite index - Indexes containing more than one field are called composite index. Such indexes should be created if you expect to run queries that will have multiple fields in the WHERE clause and all fields combined will give significantly less rows than the first field alone.

For example, in our sample database the index "IdxCustDate" is a composite index. This index is only useful if the ratio between customers and the number of their orders is high \u2013 meaning an average customer places more than 1000 orders. Creating a composite index on the Orders table when most customers have placed only a handful of order wastes not only spaces on the hard disk but have negative impact on DML queries.

Clustered index - A clustered index determines the physical order of data in a table - meaning the actual data is sorted according to the fields in the index. This is similar to a telephone directory, which arranges data by last name. There can be only one clustered index per table. These indexes are particularly efficient on columns that are often searched for range of values.

Although most databases support such index, they use a different terminology. For example Oracle calls it Index-Organized Table (IOT) whereas DB2, Informix, MS SQL Server and Sybase all call it clustered index.

Tip 3 - Avoid functions on RHS of the operator

Often developers use functions or method with their SQL queries. Consider the following example.

```
select *
from Customer
where YEAR(AccountCreatedOn) == 2005
and  MONTH(AccountCreatedOn) = 6
```

Note that even though AccountCreatedOn has an index, the above query changes the where clause such a way that this index cannot be used anymore.

Rewriting the query in the following way will increase the performance tremendously.

```
Select *
From Customer
Where AccountCreatedOn between '6/1/2005'
    and '6/30/2005'
```

Tip 4 - Predetermine expected growth

As mentioned earlier indexes have a negative impact on DML queries. One way to minimize this negative affect is to specify an appropriate value for fill factor when creating indexes.
When an index is created, the data for indexed columns is stored on the disk. When new rows of data are inserted into the table or the values in the indexed columns are changed, the database may have to reorganize the storage of the data to make room for the new rows. This reorganization can take additional toll on DML queries. However, if you expect new rows on a regular basis in any table, you can specify the expected growth for an index. The terminology used for this expected growth is different in every database. The following table lists the terms used by different RDBMS for expected growth.

Oracle:	PCTFREE - Percent Free
DB2:	PCTFREE - Percent Free
MS SQL Server:	FILL FACTOR
Informix:	FILL FACTOR
Sybase ASE:	FILL FACTOR

Tip 5 - Specify optimizer hints in SELECT

Although in most cases the query optimizer will pick the appropriate index for a particular table based on statistics, sometimes it is better to specify the index name in your SELECT query. For example, consider the following

```
SELECT *
FROM customer
WITH ( Index(IdxPhone))
WHERE city = 'New York City'
    and phone = '212-555-1212'
```

Notice the additional "WITH" clause after FROM. This example is specific to MS SQL Server. Every database use different syntax for specifying this value and they are quite different from each other. Refer to your RDBMS documentation for details.

Tip 6 - Use EXPLAIN

Most databases return the execution plan for any SELECT statement that is created by the optimizer. This plan is very useful in fine tuning SQL queries. The following table lists SQL syntax for different databases.

Oracle:	EXPLAIN PLAN FOR >Your query<
DB2:	EXPLAIN PLAN SET queryno = xxx for >Your query<
MS SQL Server:	Set SHOWPLAN_ALL ON >Your query<
Informix:	SET EXPLAIN
Sybase ASE:	Set SHOWPLAN_ALL ON >Your query<

You can also use third party tools, such as WinSQL Professional from Synametrics Technologies to run EXPLAIN commands against databases.

Tip 7 - Avoid foreign key constraints

Foreign keys constraints ensure data integrity at the cost of performance. Therefore, if performance is your primary goal you can push the data integrity rules to your application layer. A good example of a database design that avoids foreign key constraints is the System tables in most databases. Every major RDBMS has a set of tables known as system tables. These tables contain meta data information about user databases. Although there are relationships among these tables, there is no foreign key relationship. This is because the client, in this case the database itself, enforces these rules.

Tip 8 - Two heads are better than one

Hard disk I/O is among the slowest resource on a computer, which becomes apparent as the size of your database increase. Many databases allow users to split there database onto multiple physical hard drives. In fact, some even go a step further and allow splitting the contents of a table on multiple disks. When you use multiple physical disks, I/O operations speed up significantly since more heads fetch data in parallel.

Tip 9 - Select limited data

The less data retrieved, the faster the query will run. Rather than filtering on the client, push as much filtering as possible on the server-end. This will result in less data being sent on the wire and you will see results much faster. Eliminate any obvious or computed columns. Consider the following example.

```
Select  FirstName, LastName, City
Where City  = 'New York City'
```

In the above example, you can easily eliminate the "City" column, which will always be "New York City". Although this may not seem to have a large effect, it can add up to a significant value for large result sets.

Tip 10 - Drop indexes before loading data

Consider dropping the indexes on a table before loading a large batch of data. This makes the insert statement run faster. Once the inserts are completed, you can recreate the index again.

If you are inserting thousands of rows in an online system, use a temporary table to load data. Ensure that this temporary table does not have any index. Since moving data from one table to another is much faster than loading from an external source, you can now drop indexes on your primary table, move data from temporary to final table, and finally recreate the indexes.

Summary

This document provides some helpful tips related to performance improvements against any relational database. We hope that after reading this document, you can modify their design and/or queries within their application that returns optimum result.