Using Indexes

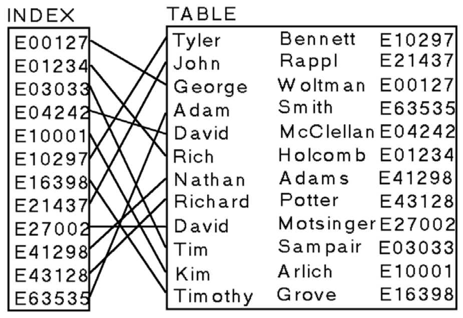
Introduction

There are a number of ways in which you can improve the performance of database activity using indexes. We provide only general guidelines that apply to most databases. Consult your database vendor's documentation for more detailed information.

For information regarding how to create and drop indexes, see your database system documentation.

An index is a database structure that you can use to improve the performance of database activity. A database table can have one or more indexes associated with it.

An index is defined by a field expression that you specify when you create the index. Typically, the field expression is a single field name, like EMP_ID. An index created on the EMP_ID field, for example, contains a sorted list of the employee ID values in the table. Each value in the list is accompanied by references to the records that contain that value.



A database driver can use indexes to find records quickly. An index on the EMP_ID field, for example, greatly reduces the time that the driver spends searching for a particular employee ID value. Consider the following Where clause:

```
WHERE emp_id = 'E10001'
```

Without an index, the driver must search the entire database table to find those records having an employee ID of E10001. By using an index on the EMP_ID field, however, the driver can quickly find those records.

Indexes may improve the performance of SQL statements. You may not notice this improvement with small tables but it can be significant for large tables; however, there can be disadvantages to having too many indexes. Indexes can slow down the performance of some inserts, updates, and deletes when the driver has to maintain the indexes as well as the database tables. Also, indexes take additional disk space.

Improving Record Selection Performance

For indexes to improve the performance of selections, the index expression must match the selection condition exactly. For example, if you have created an index whose expression is last_name, the following Select statement uses the index:

```
SELECT * FROM emp WHERE last_name = 'Smith'
```

This Select statement, however, does not use the index:

```
SELECT * FROM emp WHERE UPPER(last_name) = 'SMITH'
```

The second statement does not use the index because the Where clause contains UPPER(LAST_NAME), which does not match the index expression LAST_NAME. If you plan to use the UPPER function in all your Select statements and your database supports indexes on expressions, then you should define an index using the expression UPPER(LAST_NAME).

Indexing Multiple Fields

If you often use Where clauses that involve more than one field, you may want to build an index containing multiple fields. Consider the following Where clause:

```
WHERE last_name = 'Smith' and first_name = 'Thomas'
```

For this condition, the optimal index field expression is LAST_NAME, FIRST_NAME. This creates a concatenated index.

Concatenated indexes can also be used for Where clauses that contain only the first of two concatenated fields. The LAST_NAME, FIRST_NAME index also improves the performance of the following Where clause (even though no first name value is specified):

```
last_name = 'Smith'
```

Consider the following Where clause:

```
WHERE last_name = 'Smith' and middle_name = 'Edward' and
     first_name = 'Thomas'
```

If your index fields include all the conditions of the Where clause in that order, the driver can use the entire index. If, however, your index is on two nonconsecutive fields, say, LAST_NAME and FIRST_NAME, the driver can use only the LAST_NAME field of the index.

The driver uses only one index when processing Where clauses. If you have complex Where clauses that involve a number of conditions for different fields and have indexes on more than one field, the driver chooses an index to use. The driver attempts to use indexes on conditions that use the equal sign as the relational operator rather than conditions using other operators (such as greater than). Assume you have an index on the EMP_ID field as well as the LAST_NAME field and the following Where clause:

```
WHERE emp_id >= 'E10001' AND last_name = 'Smith'
```

In this case, the driver selects the index on the LAST_NAME field.

If no conditions have the equal sign, the driver first attempts to use an index on a condition that has a lower and upper bound, and then attempts to use an index on a condition that has a lower or upper bound. The driver always attempts to use the most restrictive index that satisfies the Where clause.

In most cases, the driver does not use an index if the Where clause contains an OR comparison operator. For example, the driver does not use an index for the following Where clause:

```
WHERE emp_id >= 'E10001' OR last_name = 'Smith'
```

Deciding Which Indexes to Create

Before you create indexes for a database table, consider how you will use the table. The two most common operations on a table are to:

- Insert, update, and delete records
- Retrieve records

If you most often insert, update, and delete records, then the fewer indexes associated with the table, the better the performance. This is because the driver must maintain the indexes as well as the database tables, thus slowing down the performance of record inserts, updates, and deletes. It may be more efficient to drop all indexes before modifying a large number of records, and re-create the indexes after the modifications.

If you most often retrieve records, you must look further to define the criteria for retrieving records and create indexes to improve the performance of these retrievals. Assume you have an employee database table and you will retrieve records based on employee name, department, or hire date. You would create three indexes—one on the DEPT field, one on the HIRE_DATE field, and one on the LAST_NAME field. Or perhaps, for the retrievals based on the name field, you would want an index that concatenates the LAST_NAME and the FIRST_NAME fields (see “Indexing Multiple Fields” on page 489 for details).

Here are a few rules to help you decide which indexes to create:

- If your record retrievals are based on one field at a time (for example, dept='D101'), create an index on these fields.
- If your record retrievals are based on a combination of fields, look at the combinations.
- If the comparison operator for the conditions is AND (for example, CITY = 'Raleigh' AND STATE = 'NC'), then build a concatenated index on the

CITY and STATE fields. This index is also useful for retrieving records based on the CITY field.

If the comparison operator is OR (for example, DEPT = 'D101' OR HIRE_DATE > {01/30/89}), an index does not help performance. Therefore, you need not create one.

If the retrieval conditions contain both AND and OR comparison operators, you can use an index if the OR conditions are grouped. For example:

```
dept = 'D101' AND (hire_date > {01/30/89} OR
exempt = 1)
```

In this case, an index on the DEPT field improves performance.

If the AND conditions are grouped, an index does not improve performance. For example:

```
(dept = 'D101' AND hire_date) > {01/30/89}) OR
exempt = 1
```

Improving Join Performance

When joining database tables, index tables can greatly improve performance. Unless the proper indexes are available, queries that use joins can take a long time.

Assume you have the following Select statement:

```
SELECT * FROM dept, emp WHERE dept.dept_id = emp.dept
```

In this example, the DEPT and EMP database tables are being joined using the department ID field. When the driver executes a query that contains a join, it processes the tables from left to right and uses an index on the second table’s join field (the DEPT field of the EMP table).

To improve join performance, you need an index on the join field of the second table in the From clause. If there is a third table in the From clause, the driver also uses an index on the field in the third table that joins it to any previous table. For example:

```
SELECT * FROM dept, emp, addr

WHERE dept.dept_id = emp.dept AND emp.loc = addr.loc
```

In this case, you should have an index on the EMP.DEPT field and the ADDR.LOC field.

We welcome your feedback! Please send any comments concerning documentation, including suggestions for other topics that you would like to see, to:

docgroup@datadirect.com

FOR MORE INFORMATION

800-876-3101

info@datadirect.com

Worldwide Sales

Belgium (French)	0800 12 045
Belgium (Dutch)	0800 12 046
France	0800 911 454
Germany	0800 181 78 76
Japan	0120.20.9613
Netherlands	0800 022 0524
United Kingdom	0800 169 19 07
United States	800 876 3101

DataDirect TECHNOLOGIES

DataDirect Technologies is focused on data access, enabling software developers at both packaged software vendors and in corporate IT departments to create better applications faster. DataDirect Technologies offers the most comprehensive, proven line of data connectivity components available anywhere. Developers worldwide depend on DataDirect Technologies to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC and ADO.NET, as well as cutting-edge XML query technologies. More than 250 leading independent software vendors and thousands of enterprises rely on DataDirect Technologies to simplify and streamline data connectivity. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS).

www.datadirect.com