There are four methods that you can use to filter records in a form or a datasheet: Filter by Selection, Filter by Form, Filter for Input, and Advanced Filter/Sort.

Filter by Selection

1. Start Microsoft Access, and then open the database that you are working with.
2. In a field on a form, a subform, a datasheet, or a subdatasheet, select one instance of the value that you want to filter by (for example, a name or a number).
3. On the **Records** menu, point to **Filter**, and then click **Filter by Selection**.
4. Repeat steps 2 and 3 until you have the set of records that you are looking for.**NOTE:** You can also filter for records that do not have a certain value. After you select a value, right-click, and then click **Filter Excluding Selection**.

Filter by Form

1. Open a form in Form view, or a table, a query, or a form in Datasheet view. To filter records in a subdatasheet, display the subdatasheet by clicking its expand indicator.
2. On the **Records** menu, point to **Filter**, and then click **Filter By Form** to switch to the Filter By Form window.
3. You can specify criteria for the form, the subform, the main datasheet, or any subdatasheet that is displayed. Each subform or subdatasheet has its own **Look For** and **Or** tabs.
4. Click the field in which you want to specify the criteria.
5. Enter your criteria by selecting the value that you are searching for from the list in the field (if the list includes field values), or by typing the value into the field.

   To find records in which a particular field is empty or not empty, type Is Null or Is Not Null into the field.
6. To specify additional values that records can have in the filter, click the **Or** tab for the form, the subform, the datasheet, or the subdatasheet that you are filtering, and then enter more criteria.
7. On the **Filter** menu, click **Apply Filter** to view the filter results.

Filter for Input

1. Open a form in Form view, or a table, a query, or a form in Datasheet view. To filter records in a subdatasheet, display the subdatasheet by clicking its expand indicator.
2. Right-click in the field in the form, the subform, the datasheet, or the subdatasheet that you are filtering, and then type the value that you are looking for in the **Filter For** box on the shortcut menu.
3. Press ENTER to apply the filter, and then close the shortcut menu.

Advanced Filter/Sort

1. Open a form in Form view, or a table, a query, or a form in Datasheet view.
2. Click in the form, the subform, the datasheet, or the subdatasheet that you want to filter.

3. On the **Records** menu, point to **Filter**, and then click **Advanced Filter/Sort**.
4. Add the fields that you need to specify the values or the other criteria that the filter will use to find records to the design grid.
5. To specify a sort order, click in the **Sort** cell for a field, click the arrow, and then select a sort order. Microsoft Access first sorts the leftmost field in the design grid, and then it sorts the next field to the right, and so on.
6. In the **Criteria** cell for the fields that you have included, enter the value that you are looking for or enter an expression.
7. On the **Filter** menu, click **Apply Filter** to view the filter's results.

Introduction to Database Indexes

Posted on February 15, 2006 by chris.smith

Put simply, database indexes help speed up retrieval of data. The other great benefit of indexes is that your server doesn't have to work as hard to get the data. They are much the same as book indexes, providing the database with quick jump points on where to find the full reference (or to find the database row).

There are both advantages and disadvantages to using indexes,however.

One disadvantage is they can take up quite a bit of space – check a textbook or reference guide and you'll see it takes quite a few pages to include those page references.

Another disadvantage is using too many indexes can actually slow your database down. Thinking of a book again, imagine if every "the", "and" or "at" was included in the index. That would stop the index being useful – the index becomes as big as the text! On top of that, each time a page or database row is updated or removed, the reference or index also has to be updated.

So indexes speed up finding data, but slow down inserting, updating or deleting data.

Some fields are automatically indexed. A primary key or a field marked as 'unique' – for example an email address, a userid or a social security number – are automatically indexed so the database can quickly check to make sure that you're not going to introduce bad data.

So when should a database field be indexed?

The general rule is anything that is used to limit the number of results you're trying to find.

It's hard to generalise so we'll look at some specific but common examples.

Note – the database tables shown below are used as an example only and will not necessarily be the best setup for your particular needs.

In a database table that looks like this:

Note: The SQL code shown below works with both MySQL and PostgreSQL databases.

```
CREATE TABLE subscribers (
  subscriberid INT PRIMARY KEY,
  emailaddress VARCHAR(255),
  firstname VARCHAR(255),
  lastname VARCHAR(255)
);
```

if we want to quickly find an email address, we create an index on the emailaddress field:

```
CREATE INDEX subscriber_email ON subscribers(emailaddress);
```

… and any time we want to find an email address:

```
SELECT firstname, lastname FROM subscribers WHERE emailaddress='email@domain.com';
```

… it will be quite quick to find!

Another reason for creating indexes is for tables that reference other tables. For example, in a CMS you might have a news table that looks something like this:

```
CREATE TABLE newsitem (
  newsid INT PRIMARY KEY,
  newstitle VARCHAR(255),
  newscontent TEXT,
  authorid INT,
  newsdate TIMESTAMP
);
```

and another table for authors:

```
CREATE TABLE authors (
  authorid INT PRIMARY KEY,
  username VARCHAR(255),
  firstname VARCHAR(255),
  lastname VARCHAR(255)
);
```

A query like this:

```
SELECT newstitle, firstname, lastname FROM newsitem n, authors a WHERE
n.authorid=a.authorid;
```

… will be take advantage of an index on the newsitem authorid:

```
CREATE INDEX newsitem_authorid ON newsitem(authorid);
```

This allows the database to very quickly match the records from the ‘newsitem’ table to the ‘authors’ table. In database terminology this is called a table join – you should index any fields involved in a table join like this.

Since the ‘authorid’ in the authors table is a primary key, it is already indexed. The same goes for the ‘newsid’ in the news table, so we don’t need to look at those cases.

On a side note, table aliases make things a lot easier to see what’s happening. Using ‘newsitem n’ and ‘authors a’ means we don’t have to write:

```
SELECT newstitle, firstname, lastname FROM newsitem, authors WHERE
newsitem.authorid=authors.authorid;
```

for more complicated queries where more tables are referenced this can be extremely helpful and make things really easy to follow.

In a more complicated example, a news item could exist in multiple categories, so in a design like this:

```
CREATE TABLE newsitem (
  newsid INT PRIMARY KEY,
  newstitle VARCHAR(255),
  newscontent TEXT,
  authorid INT,
  newsdate TIMESTAMP
);

CREATE TABLE newsitem_categories (
  newsid INT,
  categoryid INT
);

CREATE TABLE categories (
  categoryid INT PRIMARY KEY,
  categoryname VARCHAR(255)
);
```

This query:

```
SELECT n.newstitle, c.categoryname FROM categories c, newsitem_categories nc, newsitem n
WHERE c.categoryid=nc.categoryid AND nc.newsid=n.newsid;
```

… will show all category names and newstitles for each category.

To make this particular query fast we need to check we have an index on:

newsitem newsid
newsitem_categories newsid
newsitem_categories categoryid
categories categoryid

Note: Because the newsitem newsid and the categories categoryid fields are primary keys, they already have indexes.

We need to check there are indexes on the "join" table – newsitem_categories

This will do it:

```
CREATE INDEX newscat_news ON newsitem_categories(newsid);
CREATE INDEX newscat_cats ON newsitem_categories(categoryid);
```

We could create an index like this:

```
CREATE INDEX news_cats ON newsitem_categories(newsid, categoryid);
```

However, doing this limits some ways the index can be used. A query against the table that uses both 'newsid' and 'categoryid' will be able to use this index. A query against the table that only gets the 'newsid' will be able to use the index.

A query against that table that only gets the 'categoryid' will not be able to use the index.

For a table like this:

```
CREATE TABLE example (
  a int,
  b int,
  c int
);
```

With this index:

```
CREATE INDEX example_index ON example(a,b,c);
```

- It will be used when you check against ‘a’.
- It will be used when you check against ‘a’ and ‘b’.
- It will be used when you check against ‘a’, ‘b’ and ‘c’.
- It will not be used if you check against ‘b’ and ‘c’, or if you only check ‘b’ or you only check ‘c’.
- It will be used when you check against ‘a’ and ‘c’ but only for the ‘a’ column – it won’t be used to check the ‘c’ column as well.

A query against ‘a’ OR ‘b’ like this:

```
SELECT a,b,c FROM example where a=1 OR b=2;
```

- Will only be able to use the index to check the ‘a’ column as well – it won’t be able to use it to check the ‘b’ column.

Multi-column indexes have quite specific uses, so check their use carefully.

Now that we’ve seen when we should use indexes, let’s look at when we shouldn’t use them. They can actually slow down your database (some databases may actually choose to ignore the index if there’s no reason to use it).

A table like this:

```
CREATE TABLE news (
  newsid INT PRIMARY KEY,
  newstitle VARCHAR(255),
  newscontent TEXT,
  active CHAR(1),
  featured CHAR(1),
  newsdate TIMESTAMP
);
```

… looks pretty standard. The ‘active’ field tells us whether the news item is active and ready to be viewed on the site.

So… should we should create an index on this field for a query like this?

SELECT newsid, newstitle FROM news WHERE active='1′;

No, we shouldn't.

If most of your content is live, this index will take up extra space and slow the query down because almost all of the fields match this criteria. Imagine 500 news items in the database with 495 being active. It's quicker to eliminate the ones that aren't active than it is to list all of the active ones (if you do have an index on the 'active' field, some databases will choose to ignore it anyway because it will slow the query down).

The featured field tells us whether the news item should feature on the front page. S
hould we index this field? Yes. Most of our content is not featured, so an index on the 'featured' column will be quite useful.

Other examples of when to index a field include if you're going to order by it in a query. To get the most recent news items, we do a query like this:

SELECT newtitle, newscontent FROM news ORDER BY newsdate DESC;

Creating an index on 'newsdate' will allow the database to quickly sort the results so it can fetch the items in the right order. Indexing can be a bit tricky to get right, however there are tools available for each database to help you work out if it's working as it should.
Well there you have it — my introduction to database indexes. Hopefully you've learned something from this article and can apply what you've learned to your own databases.

Sorting Records in Form View or in Datasheet View

To sort records in Form view or in Datasheet view, follow these steps:

1. Start Microsoft Access, and then open the database that you are working with.
2. Open the table or the form whose data you want to view.
3. Click the field that you want to use for sorting records. To sort records in a subform, click the field that you want to sort. To sort records in a subdatasheet, display the subdatasheet by clicking its expand indicator, and then click the field that you want to sort.
4. On the **Records** menu, point to **Sort**, and then click **Sort Ascending** or **Sort Descending**.**NOTE:** In a form, you can sort on only one field at a time.

Sorting with Subdatasheets

In Datasheet view, when you sort the subdatasheet for one record, Microsoft Access sorts all the subdatasheets at that level. In a datasheet or subdatasheet, you can select two or more adjacent columns at the same time, and then sort them. Access sorts records starting with the leftmost selected column. When you save the form or datasheet, Access saves the sort order.

Sorting Records on a Report

1. Start Microsoft Access, and then open the database that you are working with.
2. Open the report in Design view.
3. On the **View** menu, click **Sorting and Grouping** to display the **Sorting and Grouping** dialog box.
4. In the first row of the **Field/Expression** column, select a field name or type an expression.

   NOTE: When you fill in the **Field/Expression** column, Microsoft Access sets the sort order to **Ascending**.
5. You can sort on up to 10 fields or expressions in a report. To sort your report on more than one field, add another field or expression to the **Field/Expression** column. The field or expression in the first row is the first sorting level. The second row is the second sorting level, and so on.

1.

- 1

  Open your database file in Microsoft Access.

- 2

  Use the F11 key to open the Database window.

- 3

  Click on either the Form tab or the Table tab, depending on which view you want to work in. A list of forms or tables appears.

- 4

  Select the table or form you want to work with and click Open.

- 5

  Click the field that you want to use for sorting records.

- 6

  From the Records menu, click Sort Ascending if you want to sort from 0 to 9 or from A to Z.

- 7

  Click Sort Descending if you want to sort the field from 9 to 0 or from Z to A.

SELECT Statement (Microsoft Access SQL)

Instructs the Microsoft Access database engine to return information from the database as a set of records. Syntax SELECT [predicate] { * | table .* |...

SELECT...INTO Statement (Microsoft Access SQL)

Creates a make-table query. Syntax SELECT field1 [, field2 [, …]] INTO newtable [IN externaldatabase]   FROM source The SELECT…INTO statement has these...

INSERT INTO Statement (Microsoft Access SQL)

Adds a record or multiple records to a table. This is referred to as an append query. Syntax Multiple-record append query: INSERT INTO target [(field1...

UPDATE Statement (Microsoft Access SQL)

Creates an update query that changes values in fields in a specified table based on specified criteria. Syntax UPDATE table SET newvalue WHERE criteria...

DELETE Statement (Microsoft Access SQL)

Creates a delete query that removes records from one or more of the tables listed in the FROM clause that satisfy the WHERE clause. Syntax DELETE [table...

EXECUTE Statement (Microsoft Access SQL)

Used to invoke the execution of a procedure. Syntax EXECUTE procedure [param1 [, param2 [, …]] The EXECUTE statement has these parts: Part Description...

TRANSACTION Statement (Microsoft Access SQL)

Used to initiate and conclude explicit transactions. Syntax Initiate a new transaction. BEGIN TRANSACTION Conclude a transaction by committing all work...

TRANSFORM Statement (Microsoft Access SQL)

Creates a crosstab query. Syntax TRANSFORM aggfunction selectstatement PIVOT pivotfield [IN (value1 [, value2 [, …]])] The TRANSFORM statement has these...

INNER JOIN Operation (Microsoft Access SQL)

Combines records from two tables whenever there are matching values in a common field. Syntax FROM table1 INNER JOIN table2 ON table1 . field1 compopr table2...

LEFT JOIN, RIGHT JOIN Operations (Microsoft Access SQL)

Combines source-table records when used in any FROM clause. Syntax FROM table1 [LEFT | RIGHT] JOIN table2 ON table1.field1 compopr table2.field2 The LEFT...

UNION Operation (Microsoft Access SQL)

Creates a union query, which combines the results of two or more independent queries or tables. Syntax [TABLE] query1 UNION [ALL] [TABLE] query2 [UNION...

PARAMETERS Declaration (Microsoft Access SQL)

Declares the name and data type of each parameter in a parameter query. Syntax PARAMETERS name datatype [, name datatype [, …]] The PARAMETERS declaration...

WITH OWNERACCESS OPTION Declaration (Microsoft Access SQL)

In a multiuser environment with a secure workgroup, use this declaration with a query to give the user who runs the query the same permissions as the query...

PROCEDURE Clause (Microsoft Access SQL)

Defines a name and optional parameters for a query. Note The PROCEDURE clause has been superseded by the PROCEDURE statement. Although the PROCEDURE clause...

SQL Subqueries (Microsoft Access SQL)

A subquery is a SELECT statement nested inside a SELECT, SELECT…INTO , INSERT…INTO , DELETE , or UPDATE statement or inside another subquery. Syntax You