This web paper describes triggers and indexes in the OpenEdge Release 10 releases. In a future OpenEdge release, the contents of this web paper will be folded into a programming manual as part of the OpenEdge product documentation set.

This web paper describes:

- ABL Triggers
- ABL Index Usage
Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior written consent, in writing, from Progress Software Corporation. The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document. The references in this manual to specific platforms supported are subject to change.

A (and design), Actional, Actional (and design), Affinities Server, Allegrix, Allegrix (and design), Apana, Business Empowerment, DataDirect (and design), DataDirect Connect, DataDirect Connect64, DataDirect Connect OLE DB, DataDirect Technologies, DirectAlert, Dynamic Index Utility, Dynamic Routing Architecture, EasyAsk, EdgeXtend, Empowerment Center, eXcelon, Futhom, Halo, IntelliStream, Iwave Integrator, LiveMine, Madrid, Neon, Neon 24x7, Neon New Era of Networks, Neon Upload, O (and design), ObjectStore, OpenEdge, PDF, PeerDirect, Persistence (and design), POSSENET, Powered by Progress, PowerTier, ProCare, Progress, Progress Dynamics, Progress Business Empowerment, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Progress Profiles, Progress Results, Progress Software Developers Network, ProtoSpeed, ProVision, SequeLink, Shadow, ShadowDirect, Shadow Interface, Shadow Web Interface, Shadow Web Server, Shadow TLS, Sonic ESB, SonicMQ, SOAPStation, Sonic Software (and design), SonicSynergy, SpeedLoad, SpeedScript, Speed Upload, Stylus Studio, Technical Empowerment, UIMX, Visual Edge, Voice of Experience, WebSpeed, and Your Software, Our Technology–Experience the Connection are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, Apana Dashboard Studio, Apana Event Manager, Apana Event Modeler, Apana Event Store, AppsAlive, AppServer, ASPnP, ASP-in-a-Box, BusinessEdge, Cache-Centric, Connect Everything, Achieve Anything, DataDirect Spy, DataDirect SupportLink, DataDirect XQuery, DataXtend, Future Proof, Ghost Agents, GVAC, ObjectCache, ObjectStore Inspector, ObjectStore Performance Expert, Panterro, POSSE, ProDataSet, Progress DataXtend, Progress ESP Event Manager, Progress ESP Event Modeler, Progress Event Engine, Progress RFID, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataAdapter, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Sonic Business Integration Suite, Sonic Process Manager, Sonic Collaboration Server, Sonic Continuous Availability Architecture, Sonic Database Service, Sonic Workbench, Sonic Orchestration Server, Sonic XML Server, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and other countries. Vermont Views is a registered trademark of Vermont Creative Software in the U.S. and other countries. IBM is a registered trademark of IBM Corporation. JMX and JMX-based marks and Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Any trademarks or service marks contained herein are the property of their respective owners.

Third Party Product Notices for OpenEdge


OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML)); and Copyright © 2000-2003 The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “Ant,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission from the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes software developed by IBM and others. Copyright © 1999, International Business Machines Corporation and others. All rights reserved.

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icsj4j.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Keni-Pong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED “AS IS”, WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes software developed by the World Wide Web Consortium. Copyright ©1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-200201231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes the JMX Technology from Sun Microsystems, Inc.

OpenEdge includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name “ModelObjects” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “ModelObjects”, nor may “ModelObjects” appear in their name, without prior written permission. For written permission, please contact djjcobs@modelobjects.com.

OpenEdge includes files that are subject to the Netscape Public License Version 1.1 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright © 1998-1999 Netscape Communications Corporation. All Rights Reserved.
ABL Triggers

A database trigger is a block of ABL (Advanced Business Language) code that executes whenever a specific database event occurs. A database event is an action performed against the database.

This chapter discusses ABL triggers. Specifically, it contains the following sections:

- Trigger definition
- ABL database events
- Schema and session database triggers
- Best practices for ABL triggers

Trigger definition

An ABL trigger is a block of ABL code that executes whenever a specific database event occurs. A database event is an action performed against the database. For example, when you write a record to a database, a WRITE event occurs.

Because database triggers execute whenever a database event occurs, they are useful for tasks such as referential integrity. For example, if you delete a customer record from a database, you might also want to delete all of the customer’s order records. The event (deletion of a customer record) initiates the task (deletion of all associated order records). A database trigger is ideal for this type of processing because the same task must always be performed when the particular event occurs. Other suitable tasks are maintaining database security or writing database audit trails.

For more information on using ABL to write trigger code, see OpenEdge Development: ABL Reference and OpenEdge Development: ABL Essentials.

ABL database events

Database triggers associate a table or field with a database event. When the event occurs, the trigger executes. ABL does not provide events for all database actions. For example, although you can dump database definitions from a database, you cannot write a trigger for a DUMP event because ABL does not provide a DUMP event.

However, ABL does provide replication-related triggers in addition to standard triggers for certain events. Replication-related triggers help you implement database replication. For more information on replication-related triggers, see OpenEdge Development: ABL Essentials.

The database events that ABL supports follow.

CREATE

When the ABL Virtual Machine (AVM) executes a CREATE or INSERT statement for a particular database table, the AVM creates the record, then fires all applicable CREATE triggers, then fires all applicable REPLICATION–CREATE triggers.
**DELETE**

When the AVM executes a DELETE statement for a particular database table, the AVM fires all applicable DELETE triggers, then fires all applicable REPLICATION-DELETE triggers, then validates the delete, then performs the delete.

**FIND**

When the AVM reads a record in a particular database table using a FIND or GET statement or a FOR EACH loop, the AVM fires all applicable FIND triggers. FIND triggers fire only for records that completely satisfy the full search condition, such as a WHERE clause specifies. FIND triggers do not fire in response to the CAN-FIND function.

Note that if a FIND trigger fails, the AVM behaves as though the record had not met the search criteria. If the FIND is within a FOR EACH block, the AVM simply proceeds to the next record. If your application uses the BREAK option of the PRESELECT phrase (which forces the AVM to retrieve two records at a time, so it can find the break), the AVM executes the FIND trigger twice during the first FIND, which is actually two FINDs in succession. Thereafter, the AVM looks one record ahead of the record currently in the record buffer, and executes the FIND trigger before it places the next record in the buffer.

**WRITE**

When the AVM changes the contents of a record and validates it for a particular database table, the AVM first fires all applicable WRITE triggers, then fires all applicable REPLICATION-WRITE triggers. The AVM automatically validates a record when releasing it. You can also use the VALIDATE statement to explicitly validate a record. In either case, WRITE triggers execute before the validation occurs (so WRITE triggers can correct values and do more sophisticated validation). the AVM might execute the WRITE triggers for a single record more than once before it writes the record to the database, if it validates the record more than once and you modify the record between validations. (A modification is considered any change, even if you return the original value.).

**ASSIGN**

When the AVM updates a particular field in a database record, the AVM fires all applicable ASSIGN triggers. Unlike the other database events, this one monitors a specific field rather than a table. ASSIGN triggers execute when the contents of the associated field are modified. The trigger procedure executes at the end of a statement that assigns a new value to the field and after any necessary re-indexing. If the statement contains several field assignments (for example, UPDATE name city st), the AVM fires each applicable ASSIGN trigger at the end of the statement. If any trigger fails, the AVM undoes the statement (unless the code specifies NO-UNDO). For more information on replication-related triggers and database replication, see *OpenEdge Data Management: Database Administration*.

**Schema and session database triggers**

ABL supports two types of database triggers: schema and session. A schema trigger is a .p procedure that you add, through the Data Dictionary, to the schema of a database. Schema trigger code defined in the database is executed by database clients.

A session trigger is a section of code that you add to a larger, enclosing procedure.
**Schema triggers**

You create schema triggers through the **Table** or **Field Properties** dialog box in the Data Dictionary. When you use the Data Dictionary to define a schema trigger for a table or field, the trigger is automatically added to the table or field’s data definitions. OpenEdge allows you to define the trigger while you are creating or modifying a table or field. This trigger definition is stored in a trigger procedure. For information on using the Data Dictionary to create and delete triggers, see *OpenEdge Development: Basic Database Tools* (Character only) and in graphical interfaces, the online help for the Data Dictionary. For more information schema triggers, see *OpenEdge Development: ABL Essentials*.

**Differences between schema and session triggers**

Although their syntax is slightly different, schema and session triggers provide similar functionality. The important difference between them is that schema triggers are independent procedures; whereas session triggers are contained within a larger procedure. Because of this difference, schema triggers always execute when a specified event occurs, regardless of what application initiates the event.

Session triggers are defined as part of a particular application and are only in effect for that particular application. Since session triggers are executed from within an enclosing procedure, they have access to the frames, widgets, and variables defined in the enclosing procedure.

Since schema triggers are compiled separately from the procedure that initiates their execution, they do not have access to the procedure’s frames, widgets, and variables. Use schema triggers for processing that you always want to perform for a specific event. For example, when an order record is deleted, you might always want to delete the corresponding order-line records.

Use session triggers to perform additional or independent processing when the event occurs. Both types of trigger scan return **ERRORS** that cause the associated event to fail. For more information on the **ERROR** option of the **RETURN** statement, see *OpenEdge Development: ABL Reference*.

**Trigger interaction**

You can define a schema and a session trigger for the same table/event or field/event pair. How the triggers interact depends on how you define them.

Ordinarily, both triggers execute, with the session trigger executing first (except for the **FIND** session trigger, which executes after the **FIND** schema trigger). In this way, the schema trigger always has precedence over the session trigger. For a **WRITE**, **DELETE**, **CREATE**, or **ASSIGN** event, the schema trigger can override the session trigger. For a **FIND** event, the schema trigger can preempt the session trigger.

**Best practices for ABL triggers**

When you write ABL triggers, there are several general considerations that you should keep in mind.

**Metaschema tables**

ABL does not allow any database triggers on events for metaschema tables and fields (tables or fields named with an initial underscore). You can only intercept database events for an application database object.
User-interaction code

ABL allows you to include any type of ABL statement within a database trigger block, including those that involve user interaction. However, it is not recommended to include any statements that call for input from the user.

For example, if the user runs your procedure in batch mode, a trigger with a prompt causes the procedure to stop, waiting for user input.

FIND NEXT and FIND PREV

You cannot delete or advance any record in a buffer passed to a database trigger procedure (as with a FIND NEXT or FIND PREV statement) within the trigger procedure.

Triggers execute other triggers

An action within one trigger procedure can execute another trigger procedure. For example, if a trigger procedure assigns a value to a field and you defined an ASSIGN trigger for that field, that ASSIGN trigger executes. You must carefully design your triggers to avoid conflicts. For example, trigger A could change data, which could cause trigger B to execute. Trigger B could change the same data, a change you did not anticipate or want.

Triggers can start transactions

By their nature, CREATE, DELETE, WRITE, and ASSIGN triggers execute from within a transaction. (The triggering event itself must occur within a transaction.)

This means that while you can start a subtransaction within a trigger procedure for these events, you cannot start a new transaction. However, a FIND might or might not be executed within a transaction. Therefore, you should not assume that you can start a subtransaction within a FIND trigger; it might turn out to be a complete transaction.

Where triggers execute

Database triggers (including replication-related triggers) execute in the application logic, which consists of OpenEdge Application Servers and local ABL procedures. For more information on OpenEdge Application Servers, see OpenEdge Application Server: Developing AppServer Applications.

Storing trigger procedures

You can store a database’s trigger procedures in an operating system directory or in an ABL r-code library.

SQL considerations

Because of inherent differences between ABL and SQL, triggers might not execute in exactly the same way. For more information on how SQL and database triggers interact, see OpenEdge Data Management: SQL Reference. The following rules apply:

- SQL triggers will not fire when ABL events occur.
- ABL triggers will not fire when SQL events occur.
- To ensure integrity, you must have both types of triggers, or use SQL to only read data, or have only a SQL table.
ABL Index Usage

This chapter explains how an OpenEdge application uses indexes. Specifically, it contains the following sections:

- Finding out which indexes are used
- Maintaining indexes through ABL
- Using the ABL ASSIGN statement
- Indexes and unknown values
- Indexes and case sensitivity
- How ABL chooses and brackets indexes to satisfy queries
- Index-related hints

Finding out which indexes are used

To find out which indexes the AVM will use at run time for a particular query, use the XREF option in the COMPILE statement. The SEARCH lines in the XREF output file show the indexes that are accessed for every record selection statement.

Table 1–1 provides a list of tags the XREF compile option generates.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEARCH</td>
<td>Indicates an index bracket or look up will be used. The logical database name, table name, and index names are listed. When multiple brackets and indexes are used for the same query, you will see one search line for each bracket.</td>
</tr>
<tr>
<td>SEARCH . . .</td>
<td>Indicates that a suitable bracket could not be constructed and an index scan over the entire table will be performed using the index noted.</td>
</tr>
<tr>
<td>WHOLE–INDEX</td>
<td>Indicates that the query result is to be ordered by a particular column value and no suitable index exists. A sort of the query result on the noted column value is required.</td>
</tr>
<tr>
<td>ACCESS</td>
<td>Indicates that the specified table and field value is used at this point in the program.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Indicates that a record is created at this location in the program.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Indicates that a record is deleted at this location in the program.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Indicates that the specified field value of a table is updated at this location in the program.</td>
</tr>
</tbody>
</table>
An alternative method to determine index usage is to use the index statistics virtual system table. The startup parameters that enable this are described in *OpenEdge Data Management: Database Administration*.

**Maintaining indexes through ABL**

As you work with your application, you will want to know when the AVM creates and updates indexes. The AVM *creates* a new index entry for a record at the first occurrence of any one of the following:

- At the end of a statement in which the AVM assigns values to all components of the index entry.
- At the end of the closest iterating subtransaction block in which the AVM creates the record.
- When the AVM processes a `VALIDATE` statement.
- When the AVM releases the record from the record buffer.
- At the end of the transaction in which the AVM creates the record.

The AVM *updates* an index at the end of any statement in which it changes the values for one or more index fields. Because the AVM updates indexes immediately (at the end of an `UPDATE` statement), the AVM immediately finds records in the order of the new index, while the data in the found record is unchanged. The AVM changes the data in the record at the end of the scope of the record or when it releases the record.

**Note:** The AVM does not update an index if the value you try to assign to the index field is the same as the current value of the index field.

You can change the name of an index at any time. You can also delete nonprimary indexes. However, before letting you delete a primary index, OpenEdge requires that you first designate another index as primary.

If there is only one index, you must create a new index before deleting the existing index. You cannot change any of the component definitions of an index. Instead, you must delete the index and re-create it using the modified component definitions.

Remember that OpenEdge assumes that the first index you create is the primary index, so create your primary index first.

**Using the ABL ASSIGN statement**

When you want to make changes to several indexed components in an ABL procedure, use the `ASSIGN` statement to define these new values. This method allows you to change several values with minimum I/O processing. Otherwise, the AVM re-indexes records at the end of each statement that changes the value of an index component.
The following code demonstrates how you can change two index values with the `ASSIGN` statement:

```abl
r-sgn2.p

DEFINE VARIABLE neword LIKE order-line.order-num
   LABEL "New Order".
DEFINE VARIABLE newordli LIKE order-line.line-num
   LABEL "New Order Line".
REPEAT:
   PROMPT-FOR order-line.order-num line-num.
   FIND order-line USING order-line.order-num AND line-num.
   SET neword newordli.
   FIND order WHERE order.order-num = neword.
   ASSIGN order-line.order-num = neword
       order-line.line-num = newordli.
END.
```

This procedure changes the order number and line number of an order-line record. (It copies an order-line from one order to another.) It sets the new values into variables and modifies the record with a single `ASSIGN` statement that contains two assignment phrases in the form `field=expression`. So both fields are changed within a single statement. Because order-num and line-num are used jointly in one index, this method avoids having the indexing done until both values change.

### Indexes and unknown values

If an index contains an Unknown value (?), the AVM sorts that value higher than any other value. When you define a unique index, ABL ensures its uniqueness. For example, if cust-num is a unique index, and there is already a cust-num with a value 10, ABL does not allow you to create a cust-num with the value 10. However, ABL does not prohibit users from entering any number of records with unknown values in index fields. You can prevent users from doing this by making the unique index fields mandatory.

### Examples

Example 1: Using the sports database, the following query will display all records where cust-num is > 10 because cust-num is an indexed field and the Unknown value (?) will sort high in an indexed field:

```abl
FOR EACH cust WHERE cust-num >10 AND cust-num <= ?
```

However, the query below will display ZERO records because cust-num is the chosen index for the query. Since zip is not the chosen index, the Unknown value (?) will not sort high and the second part of the query will be false. No records are returned when one part of an `AND` is `FALSE`:

```abl
FOR EACH cust WHERE cust-num >10 AND cust-num <= ? AND zip >0 AND zip <?
```
Example 2: The same rule can affect queries where Unknown value (?) value is not explicitly used. Using the sports database, if you create three order records where order.cust-num = 1 and order-date = ?, then the following query will return the three records:

```
FOR EACH order WHERE order-date >= 1/1/97
```

However, the following query will return no records:

```
FOR EACH order WHERE order-date >= 1/1/97 AND cust-num = 1
```

**Indexes and case sensitivity**

The values for indexed fields are normally stored as all uppercase letters. This ensures that the AVM sorts character values properly without regard to case. For example, it treats the character values “JOHN”, “John”, and “john” the same. Also, if “JOHN” is already in a unique index, then any attempt to insert “John” is rejected.

For case-sensitive fields, the AVM stores the values exactly as entered. This means that in the above example, it accepts “John” as a different value. Also, when sorted, the uppercase values appear first, then lowercase. So following the same example, “JOHN”, “John”, and “john” all appear in a different order. Note, however, that word indexes on case-sensitive fields are treated as if the field is case insensitive.

Case sensitivity is a characteristic of the field, not the index. Therefore, if an index contains some fields that are case sensitive and some that are not, then the different sorting rules apply.

Field names are not case sensitive; they can be uppercase, lowercase, or a combination of both. If you name a field “Phone” in the Data Dictionary, you can refer to it as “phone” or “PHONE” in your procedures.

Ordinarlly, OpenEdge character fields are not case sensitive (“SMITH”=“Smith”=“smith”). However, on rare occasions, you might want to define a field that is case sensitive. For example, part numbers that contain both uppercase and lowercase characters should be stored in a case-sensitive field. Case-sensitive fields are not recommended, because they depart from standard OpenEdge usage. However, if you require strict adherence to the ANSI SQL standard, you might have to define all character fields as case sensitive. Once a field is defined as case sensitive, you can change it back and forth, unless it is a component of an index. If a field is a component of an index, you must delete the index, then re-create it using the modified field.

Case-sensitive fields can be indexed, and they can be grouped with case-insensitive field components in an index. With case-sensitive indexes, “JOHN”, “John”, and “john” are all unique values. However, sort order depends on the code page of your database. Note that you can (and should) define case-sensitive variables to hold values moving to and from case-sensitive fields. For more information on case sensitivity, see the ANSI SQL (-Q) startup parameter in <i>OpenEdge Deployment: Startup Command and Parameter Reference</i>. 

---

10
How ABL chooses and brackets indexes to satisfy queries

Knowing the importance of creating indexes to support common data access patterns is a big step toward efficient design. However, to make your query code and indexes work together effectively, you must understand how ABL chooses indexes to satisfy a particular query.

Efficient query coding is outside the scope of this book, but its relevance to index design and creation cannot be overstated. Therefore, this section briefly explains how ABL chooses the most efficient indexes based on the ABL query. Then, you will learn how ABL brackets an index, when possible, to minimize the number of fetched records.

**Background and terminology**

This section provides a concise, abbreviated summary of the concepts and terminology required to discuss how ABL chooses the most efficient indexes to satisfy a query. For the purposes of index selection, there are three general types of *WHERE* clause.

**Syntax**

```
WHERE searchExpr [ BY field ]
```

For example:

```
WHERE Cust-num > 6
WHERE Name "D" BY Sales-Rep
WHERE Country = "Zimbabwe"
WHERE Comments CONTAINS "Com*
```

**Syntax**

```
WHERE searchExpr AND searchExpr [ BY field ]
```

For example:

```
WHERE Cust-num > 6 AND comments CONTAINS "ASAP"
WHERE Name = "Rogers" AND Postal-Code BEGINS "017"
```

**Syntax**

```
WHERE searchExpr OR searchExpr [ BY field ]
```

For example:

```
WHERE Cust-num > 1000 OR Sales-Rep BEGINS "S"
WHERE Postal-Code =< "01500" OR Postal-Code >= "25000"
```
The optional **BY field** clause imposes a sort order on returned records and is called a **sort match**. A **searchExpr** typically has one of the following forms:

<table>
<thead>
<tr>
<th>BY field</th>
<th>Sort match</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>field</strong> = <strong>expression</strong></td>
<td>Equality match</td>
</tr>
<tr>
<td><strong>field</strong> &lt; / =&lt; / &gt; / &gt;= <strong>expression</strong></td>
<td>Range match</td>
</tr>
<tr>
<td><strong>field</strong> BEGINS <strong>expression</strong></td>
<td>Range match</td>
</tr>
<tr>
<td><strong>wordIndexedField</strong> CONTAINS <strong>stringExpression</strong></td>
<td>Equality (simple string)</td>
</tr>
<tr>
<td></td>
<td>Range (wild card string)</td>
</tr>
<tr>
<td></td>
<td>None (&gt;1 string, joined logically)</td>
</tr>
</tbody>
</table>

For more information, see the Record Phrase and FOR statement reference entries in *OpenEdge Development: ABL Reference*.

Because these expressions effectively select the records to return—and the indexes to use—they are called **search conditions**. Commonly, but not always, **field** is an indexed field. Also, a **searchExpr** can include other **searchExpr**s joined by **ANDs** and **ORs**, forming arbitrarily complex queries.

The Compiler constructs a logical tree from a query and evaluates both sides of each **AND** or **OR**, looking for index criteria. ABL counts equality, range, and sort matches (for **OR**) and uses them to select and bracket indexes. The precise rules are numerous and complex, and it is not important to fully understand their details. The next sections outline the rules in sufficient detail to help you develop a feel for index usage. In addition, you should experiment by coding various queries, compiling them with the **XREF** option, and examining index usage as reported in the **SEARCH** lines of the **XREF** output file.

The index selection examples that follow are based on the sports database.

**Case 1: WHERE searchExpr**

If there is an index on the **field** in **searchExpr**, or if **field** is the first component in a multi-field index, ABL uses the index. Otherwise, ABL uses the primary index:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WHERE Customer.Name BEGINS &quot;B&quot;</strong></td>
<td>Name</td>
</tr>
<tr>
<td><strong>WHERE Customer.Postal-Code BEGINS &quot;01&quot;</strong></td>
<td>Cust-Num (primary)</td>
</tr>
</tbody>
</table>

If the **searchExpr** references a word-indexed field, ABL uses the word index.

If there is a **BY **field clause, and **field** is indexed, ABL uses the index to sort returned records as long as there is no index on the **WHERE** clause. If **field** is not indexed, ABL creates a temporary sort table and sorts the records at run time.
Case 2: WHERE searchExpr AND searchExpr

For a compound WHERE clause, ABL builds a logic tree and evaluates index usage on either side of the AND. When used with the FOR EACH statement, if both sides of the AND include equality matches on all components of non-unique indexes, both indexes are used. When used with the FIND statement, if both sides of the AND are equality matches on indexed fields, only a single index is used. Note that a word index expression with a simple string is an equality match; a wild card string constitutes a range match:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Name = &quot;Mary&quot; AND Customer.Sales-Rep = &quot;Higgins&quot;</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Sales-Rep</td>
</tr>
<tr>
<td>WHERE Comments CONTAINS &quot;small&quot; AND Country = &quot;USA&quot; AND Postal-Code = &quot;01730&quot;</td>
<td>Comments</td>
</tr>
<tr>
<td></td>
<td>Country-Post</td>
</tr>
</tbody>
</table>

If the selection criteria do not support multiple index usage, see the “General rules for choosing a single index” section on page 14.

If ABL uses multiple indexes to select and return records, the precise return order is not predictable. If necessary, you can use the USE-INDEX or BY options to guarantee record return order. In the following example, the BY clause guarantees records are sorted by Cust-Num:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Country = &quot;USA&quot; AND Customer.Sales-Rep = &quot;Higgins&quot; BY Cust-Num</td>
<td>Sales-Rep</td>
</tr>
</tbody>
</table>

Case 3: WHERE searchExpr OR searchExpr

For a compound WHERE clause, ABL builds a logic tree and evaluates index usage on either side of the OR. In general, if all selection criteria on both sides of the OR include matches—equality, range, or sort—on successive, leading components of two non-unique indexes, ABL uses both indexes:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Comments CONTAINS &quot;to*&quot; OR Customer.Name = &quot;Carlin&quot;</td>
<td>Comments</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td>WHERE Name &gt; &quot;Beaudette&quot; OR Country &gt; &quot;Zambia&quot;</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>Country-Post</td>
</tr>
</tbody>
</table>

In addition, if one side of the OR includes a CONTAINS clause (that is, it uses a word index), ABL uses the word index and then a second index to satisfy the other side of the OR:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Comments CONTAINS &quot;credit&quot; OR Postal-Code &gt; &quot;01000&quot;</td>
<td>Comments</td>
</tr>
<tr>
<td></td>
<td>Cust-Num</td>
</tr>
</tbody>
</table>
In this example, the right side of the OR includes a range match, but Postal-Code is the second component of the County-Post index, so the match is not active. ABL uses the primary index to satisfy this piece of the query and, as always, uses the word index to satisfy a CONTAINS clause as shown in this example:

```
WHERE Comments CONTAINS "credit"
  OR Postal-Code < "01000"
  BY Sales-Rep
```

If the selection criteria do not support multiple index usage, see the “General rules for choosing a single index” section on page 14.

**Note:** If any expression on either side of the OR does not use an index or all its components, ABL must scan all records using the primary index.

### General rules for choosing a single index

When the selection criteria do not support multiple index usage, ABL uses these general rules (in this order) to select the most efficient index:

1. If there is a CONTAINS clause (which is legal only for word indexed fields), use the word index:

   **Sample WHERE clause**

   ```
   WHERE Customer.Comments CONTAINS "big"
   AND Customer.Country = "Canada"
   ```

   **Indexes used**

<table>
<thead>
<tr>
<th>Comments</th>
</tr>
</thead>
</table>

2. If an index is unique, and all of its components are used in active equality matches, use the unique index. It invariably returns 0 or 1 records:

   **Sample WHERE clause**

   ```
   WHERE Customer.Cust-Num = 10
   AND Customer.Sales-Rep = "DR"
   ```

   **Indexes used**

<table>
<thead>
<tr>
<th>Cust-Num</th>
</tr>
</thead>
</table>
3. Use the index with the most active equality matches. Equality matches are active if both of the following conditions are met:

- They apply to successive, leading index components.
- They are joined by ANDs (not ORs or NOTs).

This disqualifies equality matches on, for example, components 2 and 3 of an index with three components, and it disqualifies matches on components 1 and 2 if they surround an OR:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Name = &quot;Harrison&quot; AND Customer.Sales-Rep BEGINS &quot;S&quot;</td>
<td>Name</td>
</tr>
<tr>
<td>WHERE Customer.Name = &quot;Harrison&quot; AND (Customer.Country = &quot;Finland&quot; OR Customer.Country = &quot;Denmark&quot;)</td>
<td>Name</td>
</tr>
</tbody>
</table>

4. Use the index with the most active range matches. For a range match to be active it must stand alone or be connected to other selection criteria by ANDs. In addition, it must apply to an index component having any one of two properties:

- The component is the first or only one in the index.
- All preceding components in the index key have active equality matches.

The following table provides some index examples:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Contact = &quot;DLC&quot; AND Customer.Sales-Rep BEGINS &quot;S&quot;</td>
<td>Sales-Rep</td>
</tr>
<tr>
<td>WHERE Customer.Contact = &quot;Ritter&quot; AND Comments CONTAINS &quot;compute*&quot;</td>
<td>Comments</td>
</tr>
</tbody>
</table>
5. Use the index with the most sort matches (all sort matches are active):

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Contact = &quot;Wilson&quot; AND Customer.Credit-Limit &gt; 2000 BY Name</td>
<td>Name</td>
</tr>
<tr>
<td>WHERE Name = &quot;Wilson&quot; OR Customer.Credit-Limit = 2000 BY Sales-Rep</td>
<td>Sales-Rep</td>
</tr>
</tbody>
</table>

6. Use the index that comes first alphabetically. That is, if there is a tie—if multiple indexes have the same number of active equality, range, and/or sort matches—use the alphabet to decide:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Name = &quot;Samali&quot; AND Customer.Sales-Rep = &quot;BCW&quot;</td>
<td>Name</td>
</tr>
</tbody>
</table>

7. Use the primary index:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Customer.Contact = &quot;MK&quot; AND (Customer.Sales-Rep BEGINS &quot;S&quot; OR Customer.Sales-Rep BEGINS &quot;B&quot;)</td>
<td>Cust-Num</td>
</tr>
<tr>
<td>WHERE Customer.Postal-Code &gt;= &quot;01000&quot; AND Customer.City = &quot;Boston&quot;</td>
<td>Cust-Num</td>
</tr>
<tr>
<td>WHERE &quot;meaningless expression&quot;</td>
<td>Cust-Num</td>
</tr>
</tbody>
</table>

**Bracketing**

Having selected one or more indexes to satisfy a query, the AVM tries immediately to isolate the smallest necessary index subset, so as to return as few records as possible. This is called *bracketing*. Careful query design can increase the opportunities for bracketing, thereby preventing ABL from scanning entire indexes and examining all records. The rules for bracketing are simple:

- Bracket on active equality matches.
- Bracket an active range match, but no further brackets are possible for that index.
The following table provides some bracketing examples:

<table>
<thead>
<tr>
<th>Sample WHERE clause</th>
<th>Indexes used</th>
<th>Brackets</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Contact = &quot;DLC&quot; AND (Sales-Rep BEGINS &quot;S&quot; OR Sales-Rep BEGINS &quot;B&quot;)</td>
<td>Cust-Num</td>
<td>None</td>
</tr>
<tr>
<td>WHERE Postal-Code &gt;= &quot;01000&quot; AND City = &quot;Boston&quot;</td>
<td>Cust-Num</td>
<td>None</td>
</tr>
<tr>
<td>WHERE Name = &quot;Harrison&quot; AND Sales-Rep BEGINS &quot;S&quot;</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>WHERE Contact = &quot;DLC&quot; AND Sales-Rep BEGINS &quot;S&quot;</td>
<td>Sales-Rep</td>
<td>Sales-Rep</td>
</tr>
<tr>
<td>WHERE Country BEGINS &quot;EC&quot; AND Sales-Rep BEGINS &quot;S&quot; BY Country</td>
<td>Country-Post</td>
<td>Country-Post</td>
</tr>
<tr>
<td>WHERE Comments CONTAINS &quot;big&quot; AND Country = &quot;USA&quot; AND Postal-Code = &quot;01730&quot;</td>
<td>Comments Country-Post</td>
<td>Country Postal-Code</td>
</tr>
</tbody>
</table>

The following recommendations are intended to help you maximize query performance. They are only recommendations, and you can choose to ignore one or more of them in specific circumstances:

- Avoid joining range matches with AND.
- Avoid ORs if any expression on either side of the OR does not use an index (or all its components). Be aware that the AVM must scan all records using the primary index.
- With word indexes, avoid using AND with two wild card strings, either in the same word index (WHERE comments CONTAINS “fast* & grow*”) or in separate word indexes (WHERE comments CONTAINS “fast*” AND report CONTAINS “ris*”).
- Avoid WHERE clauses that OR a word index reference and a non-indexed criterion (WHERE comments CONTAINS “computer” OR address2 = “Bedford”).

**Index-related hints**

The following list contains index-related hints:

- As the ratio of records that satisfy the query to total records decreases, the desirability of indexed access increases.
- As a rule of thumb, use fewer indexes per table with heavy update activity and limited access; use more indexes per table with heavy access but limited update activity.
- Most index processing—including, for example, all word index evaluations—takes place on the OpenEdge server side, and the minimum required records are returned to the client.