Abstract

This is the second of a 2-part series on how database indexes work and how they are used. In this second part will cover how indexes are used by the OpenEdge 4GL, by OpenEdge SQL and by some of the various database utilities.

We will provide many coding examples to illustrate index usage as well as some tips for maintenance.

Indexes Part 2: 4GL Index Usage

Gus Björklund (gus642@gmail.com)
Head Groundskeeper
Parmington Foundation

please ask questions
if we do not explain something clearly

the only stupid question is
the one you do not ask

Subjects

- Index Use In Queries
- Compiler XREF
- Index Usage in Utilities
- OpenEdge SQL Server

Section 1:
Index Use in 4GL Queries

4GL Query Resolution Strategy

- 4GL compiler uses a rules-based algorithm to choose access paths and indexes for both static and dynamic queries
- Dynamic queries are compiled at query prepare time
- SQL is completely different

6

Index brackets

- A set of consecutive index entries, ordered by key values
- Two types:
  - Range brackets
    - starting key value
    - ending key value
  - Equality brackets
    - start and end key values the same

Index bracket scan

- Fetch all the records to which the index entries in the bracket refer
- In either order, ascending or descending

Query Resolution Strategy

- At least one index bracket is needed for every query EXCEPT
  - FIND by rowid or recid
  - FIND CURRENT
  - TABLE SCAN in Type ii data areas
  - TABLE SCAN in Type i data areas uses "whole index" range bracket scan on primary key
- Sometimes more than one bracket can be used
- What can’t be eliminated via index is done by reading and examining field values of records in the bracket

9

Two bracket query

city = “Boston” AND zip = 17777

CITY,	RECID
BOLTON	8976
BOSTON	0007
BOSTON	0022
BOSTON	0428
BOSTON	0929
BOSTON	1021
BOSTON	1456
BOSTON	6000
BOSTON	6221
BOSTON	6341
BOSTON	6666
BOSTON	7778
BOXBORO	1002

ZIP,	RECID
17776	0002
17776	8101
17777	1021
17777	6000
17777	6182
17777	6400
17777	6650
17777	6666
17777	9933
17778	3001
17778	

Result list of REC-IDs:

1021

6000

6666

City bracket cursor

Zip bracket cursor

Two bracket query

city = “Boston” AND zip = 17777 USE-INDEX zip

CITY,	RECID
BOLTON	8976
BOSTON	0007
BOSTON	0022
BOSTON	0423
BOSTON	0629
BOSTON	1021
BOSTON	1456
BOSTON	6000
BOSTON	6221
BOSTON	6311
BOSTON	6666
BOSTON	7778
BOXBORO	1002

ZIP,	RECID
17776	0002
17776	8101
17777	1021
17777	6000
17777	6182
17777	6400
17777	6650
17777	6666
17777	9933
17778	3001
17778	-------

Result list of REC-IDs:

1021
6000
6182
6400
6650
6666
9933

Zip bracket cursor

City bracket cursor NOT USED

Multi Bracket Query:
city = “Boston” or zip > 17777 or zip = 16666

CITY,	RECID
BOLTON	8976
BOSTON	0007
BOSTON	0022
BOSTON	0428
BOSTON	0929
BOSTON	1021
BOSTON	1456
--additional entries	
BOSTON	6221
BOSTON	6341
BOSTON	6666
BOSTON	7778
BOXBORO	1002

ZIP,	RECID
15555	0002
16666	8101
16666	6400
17700	6421
17722	6582
-- additional entries	
17777	6650
17778	6666
17778	9933
17779	3001
17780	1020

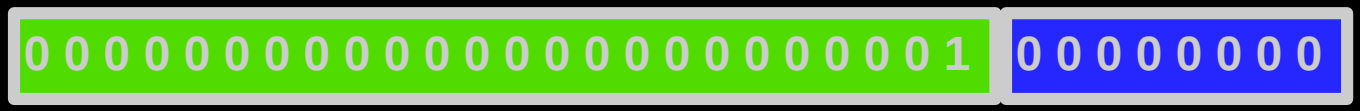
Index Selection

Index Usage Examples

Examples use CUSTOMER table with the following index definitions

Index Name	Columns	Unique
cust-num (Primary)	cust-num	yes
name	name	no
sales-rep	sales-rep	no
country-post	country, postal-code	no
comments	comments	word index

Review: Rowid



- Leaf level of index contains key value and rowids
- Unique 64-bit identifier for a record in a table
  - Partition id
  - block number in area
  - row number in block
- Encodes the “physiological” storage address
  - Used to locate record fastly
- Block header tells record byte offset in block

15

Index selection using rowid

Rowid allows direct access to record without needing any index

Statement	Indexes Chosen
find customer where rowid (customer) = myrowid.	**None**

Index selection using use-index

USE-INDEX forces specific index to be used, regardless of compiler's opinion

Statement	Indexes Chosen
find customer where cust-num = 45 use-index cust-num.	cust-num

Index selection using use-index

USE-INDEX forces specific index to be used
In this case bracket is on entire name index

Statement	Indexes Chosen
find customer where cust-num = 45 use-index name.	name

TABLE-SCAN

does not use index
range scan on entire table

Statement	Indexes Chosen
for each customer table-scan: end.	none
for each customer where cust-num = 45 table-scan: end.	none

Single Index Selection Rules

Single Index Selection Rules 1

- When a query does not use
  - ROWID
  - RECID
  - CURRENT
  - TABLE SCAN
  - USE-INDEX

  one or multiple indexes may be selected based on the WHERE or BY clause, the following rules are used for the selection of one index:

Single Index Selection Rules 2

- 1) Index which is unique, and all its components are involved in equality matches
- 2) Index with more active equality matches
- 3) Index with more active range matches
- 4) Index with more active sort matches

Single Index Selection Rules 3

- tie-breakers when multiple indexes have same score:
  - index that is primary index
  - first index alphabetically by index name (for real tables), or first index defined (for temp-tables)

Single index selection 1

Statement	Indexes Chosen
for each customer:	cust-num
for each customer where state = “MD”:	cust-num
for each customer where cust-num = 12	cust-num
find customer where cust-num = 12	cust-num

Single index selection 2

Brackets on one index

Statement	Indexes Chosen
for each customer where sales-rep = “John”:	sales-rep
for each customer where sales-rep begins “J”:	sales-rep
for each customer where cust-num > 20 and cust-num < 40	cust-num

Single index selection 3

BY clause does not always cause index to be used

Statement	Indexes Chosen
for each customer where cust-num > 56 by name:	cust-num
for each customer by name:	name

Single index selection 4

Bracket is NOT formed if you use a function or expression for a component of an index

Statement	Indexes Chosen
for each customer where substring (name, 1, 1) = “A”	cust-num

for each customer where
if myrowid <> ? then
  rowid (customer) = myrowid
else true:
	cust-num

Multiple Index Selection Rules

Multiple Index Selection

- When the where clause uses AND or OR and indexes are available for both sides of AND or OR, multiple indexes may be selected based on the following rules: (except for FIND, which always uses one index)

More multiple index selection 1

- Using AND, when all index components of each side of the AND are involved in equality matches and indexes are NOT unique.

QUIZ: why does uniqeness matter?

More multiple index selection 2

- Using OR, when at least the lead index components of each side of the OR are involved in equality or range matches.

More multiple index selection 3

- When a word index is used in the selection criteria, the word index as well as other indexes will be used.

32

Multiple index selection 1

Need equality matches on both sides of AND to use multiple indexes

Statement	Indexes Chosen
for each customer where name = “Gus” and sales-rep = “Jim”	name sales-rep
for each customer where name > “Gus” and sales-rep > “Jim”	name

Multiple index selection 2

Need all components of index for equality matches

Statement	Indexes Chosen
for each customer where country = “USA” and postal-code = “21000” and sales-rep = “Jim”	country-post sales-rep
for each customer where country = “USA” and sales-rep = “Jim”	sales-rep

Multiple index selection 3

Need at least lead component of index on each side of OR

Statement	Indexes Chosen
for each customer where country = “USA” and postal-code = “21000” OR sales-rep = “Jim”	country-post sales-rep
for each customer where country = “USA” OR sales-rep > “Jim”	country-post sales-rep

Multiple index selection 4

CONTAINS on word-indexed column, AND with equality match, OR with lead components

Statement	Indexes Chosen
for each customer where comments CONTAINS “amount” and sales-rep = “Jim”	comments sales-rep
for each customer where (comments CONTAINS “amount” and name = “John”) OR (country = “USA” and postal-code = “21000”)	comments name country-post

Section 2:
XREF

Using Compiler XREF

- Use compiler XREF option to see the indexes chosen by Progress compiler.

```
COMPILE bar.p LISTING bar.lst XREF bar.xref.
```

- SEARCH keywords indicate the index chosen.
- Multiple SEARCH keywords for the same statement indicates multiple indexes or brackets are being used.

Source text

```
for each customer where name = "Gus" and sales-rep = "Jim":
end.
for each customer where name > "Gus" and sales-rep > "Jim":
end.
for each customer where country = "USA" and
            postal-code = "21000" and sales-rep = "Jim":
end.
for each customer where country = "USA" and sales-rep = "Jim":
end.
for each customer where country = "USA" and
            postal-code = "21000" or sales-rep = "Jim":
end.
for each customer where country = "USA" or sales-rep = "Jim":
end.
for each customer where comments contains "amount" and sales-rep = "Jim":
end.
for each customer where (comments contains "amount" and name = "John") or
                        (country = "USA" and postal-code = "21000"):
end.
```

cross reference

query.p query.p 29 REFERENCE sports.Customer
query.p query.p 33 ACCESS sports.Customer Name
query.p query.p 33 ACCESS sports.Customer Sales-Rep
query.p query.p 33 STRING "Gus" 3 NONE TRANSLATABLE
query.p query.p 33 STRING "Jim" 3 NONE TRANSLATABLE
query.p query.p 33 SEARCH sports.Customer Name
query.p query.p 33 SEARCH sports.Customer Sales-Rep
query.p query.p 36 ACCESS sports.Customer Name
query.p query.p 36 ACCESS sports.Customer Sales-Rep
query.p query.p 36 STRING "Gus" 3 NONE TRANSLATABLE
query.p query.p 36 STRING "Jim" 3 NONE TRANSLATABLE
query.p query.p 36 SEARCH sports.Customer Name
query.p query.p 39 ACCESS sports.Customer Country
query.p query.p 39 ACCESS sports.Customer Postal-Code
query.p query.p 39 ACCESS sports.Customer Sales-Rep
query.p query.p 39 STRING "USA" 3 NONE TRANSLATABLE
query.p query.p 39 STRING "21000" 5 NONE TRANSLATABLE
query.p query.p 39 STRING "Jim" 3 NONE TRANSLATABLE
query.p query.p 39 SEARCH sports.Customer Country-Post
query.p query.p 39 SEARCH sports.Customer Sales-Rep

XREF output samples

SEARCH foo.customer RECID
SEARCH foo.customer cust-num
SEARCH foo.customer name WHOLE-INDEX
SEARCH foo.customer name SEARCH foo.customer sales-rep
SEARCH foo.customer cust-num SEARCH foo.customer name SEARCH foo.customer name

What About Dynamic Queries ?

- They follow the same rules as static queries
- Indexes are chosen when query is prepared
- INDEX-INFORMATION attribute of query handle returns list of indexes used
  - Check for “WHOLE-INDEX”
- INDEX-INFORMATION method of buffer returns list of indexes defined
- -zquil

Section 3:
Index Use by Utilities

proutil foo –C dump customer

- Uses primary key by default
- With –index <n>, uses specific index
- With –index 0, uses table-scan

proutil foo –C idxfix

- Has options to compare one index with another

Section 4:
Index use by OpenEdge SQL Servers

Totally unlike 4GL

- OpenEdge SQL Query Engine uses statistics to decide what indexes to use
- For every index, statistics for:
  - number of entries
  - number of unique values
  - data distribution range histogram for each key component
  - data distribution range histogram for entire key

Also, can have separate stats for

- Table and index partitions
- Tenants and their indexes in multitenant tables
- CDC system tables

SQL query engine uses stats to decide

- join order
- index usage

OE SQL Server Query Engine

- Does not use rules like 4GL query engine
- Does statistical optimization based on
  - Estimated bracket size
  - Table cardinality info
  - Join criteria
  - etc
- Can create temporary indexes

Thus endeth indexes part 2

questions