Indexing for Performance

* Symptoms of Poor Indexing

1. **Lack of Indexes Is Causing Table Scans. showplan** output reports whether the table is being accessed via a table scan or index.
2. **Index Is Not Selective Enough.** An index is selective if it helps the optimizer find a particular row or a set of rows. An index on a unique identifier such as a social security number is highly selective, since it lets the optimizer pinpoint a single row. An index on a nonunique entry such as sex (M, F) is not very selective, and the optimizer would use such an index only in very special cases.

# Index Does Not Support Range Queries. Generally, clustered indexes and covering indexes provide good performance for range queries and for SARGs that match any rows. Range queries that reference the keys of noncovering indexes use the index for ranges that return a limited number of rows. As the number of rows the query returns increases, however, using a nonclustered index or a clustered index on a data-only-locked table can cost more than a table scan.

1. **Too Many Indexes Slow Data Modification**
2. **Index Entries Are Too Large.**

**create table demotable (c10 char(10),**

**c20 char(20),**

**c40 char(40))**

**create index t10 on demotable(c10)**

**create index t20 on demotable(c20)**

**create index t40 on demotable(c40)**

**sp\_estspace demotable, 500000**

Table 9-1 shows the results.

**Table 9-1: Effects of key size on index size and levels**

**Index, key size Leaf-level pages Index levels**

*t1*0, 10 bytes 4311 3

*t2*0, 20 bytes 6946 3

*t4*0, 40 bytes 12501 4

* Index Limits and Requirements 9-5

The following limits apply to indexes in Adaptive Server:

• You can create only one clustered index per table, since the data

for a clustered index is ordered by index key.

• You can create a maximum of 249 nonclustered indexes per table.

• A key can be made up of as many as 31 columns. The maximum

number of bytes per index key is 600.

• When you create a clustered index, Adaptive Server requires

empty free space to copy the rows in the table, allocate space for

the clustered index pages, and space to re-create any

nonclustered indexes on the table. Space required can vary,

depending on how full the table’s pages are when you begin and

what space management properties are applied to the table and

index pages. 120% of the current space usage is a general rule; if

you have very long keys, you may need even more free space; if

many pages are half-empty, the copy takes less room than the

current space. See “Determining the Space Available for

Maintenance Activities” on page 30-10 for more information.

• The referential integrity constraints **unique** and **primary key** create

unique indexes to enforce their restrictions on the keys. By

default, **unique** constraints create nonclustered indexes and **primary**

**key** constraints create clustered indexes.

* Choosing Indexes

Questions to ask when working with index selection are:

• What indexes are associated currently with a given table?

• What are the most important processes that make use of the

table?

• What is the ratio of select operations to data modifications

performed on the table?

• Has a clustered index been created for the table?

• Can the clustered index be replaced by a nonclustered index?

• Do any of the indexes cover one or more of the critical queries?

• Is a composite index required to enforce the uniqueness of a

compound primary key?

• What indexes can be defined as unique?

• What are the major sorting requirements?

• Do some queries using descending ordering of result sets?

• Do the indexes support joins and referential integrity checks?

• Does indexing affect update types (direct vs. deferred)?

• What indexes are needed for cursor positioning?

• If dirty reads are required, are there unique indexes to support

the scan?

• Should IDENTITY columns be added to tables and indexes to

generate unique indexes? (Unique indexes are required for

updatable cursors and dirty reads.)

When deciding how many indexes to use, consider:

• Space constraints

• Access paths to table

• Percentage of data modifications vs. select operations

Performance requirements of reports vs. OLTP

• Performance impacts of index changes

• How often you can use **update statistics**

**Index Keys and Logical Keys**

Index keys need to be differentiated from logical keys. Logical keys

are part of the database design, defining the relationships between

tables: primary keys, foreign keys, and common keys. When you

optimize your queries by creating indexes, the logical keys may or

may not be used as the physical keys for creating indexes. You can

create indexes on columns that are not logical keys, and you may

have logical keys that are not used as index keys.

Choose index keys for performance reasons. Create indexes on

columns that support the joins, search arguments, and ordering

requirements in queries. A common error is to create the clustered

index for a table on the primary key, even though it is never used for

range queries or ordering result sets.

**Guidelines for Clustered Indexes**

These are general guidelines for clustered indexes:

• Most allpages-locked tables should have clustered indexes or use

partitions to reduce contention on the last page of heaps. In a

high-transaction environment, the locking on the last page

severely limits throughput.

• If your environment requires a lot of inserts, do not place the

clustered index key on a monotonically increasing value such as

an IDENTITY column. Choose a key that places inserts on

“random” pages to minimize lock contention while remaining

useful in many queries. Often, the primary key does not meet this

condition. This problem is less severe on data-only-locked tables,

but is a major source of lock contention on allpages-locked tables.

• Clustered indexes provide very good performance when the key

matches the search argument in range queries, such as:

**where colvalue >= 5 and colvalue < 10**

In allpages-locked tables, rows are maintained in key order and

pages are linked in order, providing very fast performance for

queries using a clustered index. In data-only-locked tables, rows

are in key order after the index is created, but the clustering can

decline over time.

• Other good choices for clustered index keys are columns used in

**order by** clauses and in joins.

• If possible, do not include frequently updated columns as keys in

clustered indexes on allpages-locked tables. When the keys are

updated, the rows must be moved from the current location to a

new page. Also, if the index is clustered, but not unique, updates

are done in deferred mode.

**Choosing Clustered Indexes**

Choose indexes based on the kinds of **where** clauses or joins you

perform. Choices for clustered indexes are:

• The primary key, if it is used for **where** clauses and if it randomizes

inserts

• Columns that are accessed by range, such as:

**col1 between 100 and 200**

**col12 > 62 and < 70**

• Columns used by **order by**

• Columns that are not frequently changed

• Columns used in joins

If there are several possible choices, choose the most commonly

needed physical order as a first choice. As a second choice, look for

range queries. During performance testing, check for “hot spots” due

to lock contention.

• Techniques for Choosing Indexes 9-13

• Index and Statistics Maintenance 9-16

• Additional Indexing Tips 9-19