Database Indexing

Reference : <https://www.youtube.com/watch?v=0X9bbtwTnuE&index=1&list=PLUWdEQb_2yqVXnx-P1881VVtec8moW1sk>

**What are indexes ?**

Indexes are database objects which can be created on table or table clusters , speedup data retrieval in certain cases.

When we store data in table it will be stored in memory as file and it can have its own flie structure.

The entire table data is stored and distributed in different memory blocks. There are two different data do indexing

B-tree (Balanced tree)

B-Map (Bit map)

**How to check in sql ?**

Explain plan for select \* from [\**tablename*\*] where id =10

Select \* from table(dbms\_xplan.display)

**How indexes are stored ?**

ID RowId

1. 1AA67549BBCC34
2. 1AA67549BBCC35
3. 2AB67549BBCC34

The indexes are stored like mentioned figure . Rowids are nothing but address of memory blocks and id are the values of indexes. So when we search with an id it will not scan the entire table it will just pick the memory block address and show us the data as per the rowed.

If we create a unique constrain in backend dbms by default create a unique index .

Select \* from user\_segments

Segment\_name partition\_name segment\_type bytes Blocks

SYS\_C007364 null INDEX 65536 8

So the inex is one kind of segment in dbms which contains a physical memory in system and structured with 8 blocks.

**Why we can’t create indexes for every field of the table as increases retrieval data performance ?**

Indexes will hamper insert, update , delete performance. Because after creating index we need to insert or update not only table row but also the indexes. It can reduce the crud operation performance. Try to keep index number per table as low as possible.

To get rid of this problem we can make the indexes invisible at crud operation time and for retrieval time again make it visible. DBMS won’t remove drop the index but do not use the index while storing data.

Alter index SYS\_C007364 invisible

Alter index SYS\_C007364 visible

**Index scan methods**

* Index unique scan
* Index range scan
* Full index scan
* Fast full index scan
* Index skip scan

**In B-tree data structure how index methods work?**

Root 🡪 branch block 🡪 Traverse leaf block 🡪 table access by index row id

**Index plan for foreign keys ?**

A referential integrity constraint exists on the indexed column or columns. The index is a means to avoid a full table lock that would otherwise be required if you update the parent table primary key. If a parent table primary key is updated or deleted it will lock the entire child table which has foreign key relationship.

As a remedy create indexes for foreign key columns. If a table contains one or more foreign key columns that reference other tables, you need to index those columns. Foreign key columns are usually non-unique because they're on the "many" side of the one-to-many (1:M) relationship. Therefore, you expect to see duplicate values in the foreign key column every time one row in the master table (on the "one" side) is related to two or more rows in the detail table (on the "many" side). You need to create a unique index on a foreign key column only if you're enforcing a mandatory one-to-one (1:1) relationship between two tables. In a 1:1 relationship, the values in the foreign key column are a unique set that corresponds to the primary key values in the master table. When the foreign key column in a table is indexed, SQL Server can scan the indexes for the rows it needs instead of scanning both tables.

**Composite index**

Index can be made on more than one columns in a table. Suppose if we create a composite key on a table DBMS will create a composite index for me. For composite index the order is very importatnt.

Example if we create a primary key constrain(field1, field2, field3)

And query select \* from tablename where field2 = ‘ nmnmn’

In this case DBMS is not going to use the index, as field2 is mentioned as a second in order.

If we modify the query select \* from tablename where field2 = ‘ nmnmn’ and field1 = ‘kjjkjk’

This time the index will be used.

**Can we create multiple indexes on same column or set of columns ?**Starting with [Oracle 12c](http://www.doblerconsulting.com/tag/oracle-12c/), multiple [indexes](http://www.doblerconsulting.com/tag/indexes/) on same column or same set of columns can be created, as long as only one index is visible, and the [indexes](http://www.doblerconsulting.com/tag/indexes/) are physically different. It is not possible to create two bitmap [indexes](http://www.doblerconsulting.com/tag/indexes/) on the same column or set of columns.

Create the first index on start\_dt column. By default, [Oracle](http://www.doblerconsulting.com/tag/oracle/) creates B-tree index.

SQL>  create index idx1\_ind\_test1 on ind\_test1(start\_dt);

Index created.

Now, create a second invisible index on the same column

SQL> create index idx2\_ind\_test1 on ind\_test1(start\_dt) invisible;

create index idx2\_ind\_test1 on ind\_test1(start\_dt) invisible

\*

ERROR at line 1:

ORA-01408: such column list already indexed

Index failed because these 2 are the same type of indexes.

You can create a bitmap index on the same column.

SQL>  create bitmap index  idx2\_ind\_test1 on ind\_test1(start\_dt) invisible;

Index created.

This feature is useful when you want to test the performance of different indexes without dropping the existing one. If you already have a b-tree index on a set of columns and you want to test whether creating bitmap index on the same set of columns will improve performance, you can make the existing one invisible and create a new visible one and perform the test.

But we can create combination of columns in B-tree index, like

Index1 = combination of field1, field2

Index2 = combination of field3, field2

Index3 = combination of field2, field1

**When to use which index scan methods ?**

<https://gerardnico.com/db/oracle/index_scans>

**How the clustering factor can affect cost.**

Example Effects of Clustering Factor on Cost

Assume the following situation:

* There is a table with 9 rows.
* There is a non-unique index on col1 for table.
* The c1 column currently stores the values A, B, and C.
* The table only has three Oracle blocks.

Case 1: The index clustering factor is low for the rows as they are arranged in the following diagram.

Block 1 Block 2 Block 3

------- ------- --------

A A A B B B C C C

This is because the rows that have the same indexed column values for c1 are located within the same physical blocks in the table. The cost of using a range scan to return all of the rows that have the value A is low, because only one block in the table needs to be read.

Case 2: If the same rows in the table are rearranged so that the index values are scattered across the table blocks (rather than collocated), then the index clustering factor is higher.

Block 1 Block 2 Block 3

------- ------- --------

A B C A B C A B C

This is because all three blocks in the table must be read in order to retrieve all rows with the value A in col1