CHAPTER – 3

Other ORACLE Database Objects, Concurrency control using lock

Index

An index is an ordered list of the contents of a column of a table. Indexing a table is an access strategy that is a way to sort and search records in the table. Indexes are essential to improve the seed with which record can be located and retrieved from a table. Indexing involves forming a two dimensional matrix completely independent of the table on which the index is being created.

This two dimensional matrix will have a single column, which will hold sorted data, extracted from the table column on which the index is created. Another column called the address field identifies the location of the record in the oracle database. When data is inserted in the table, the oracle engine automatically inserts the data value in the index.

For every data value held in the index the oracle engine inserted a unique ROWID value. This ROWID indicates exactly where the record is stored in the table. Since the data is sorted on the indexed column the sequential search ends as soon as the oracle engine reads an index value that does not meet the search criteria.

The format of rowid is BBBBBBBBBBBBB.RRRR.FFFF

Here FFFF indicates unique data file number again this data file is further divided into data blocks. Each block again given unique id that is data block number BBBBBBBBBBB.

Also each block further stores one or more record, which has given unique record no RRRR.

Instances when the oracle engine uses an index for data extraction:

- A select statement with where clause specified on the column on which index exists.
- A select statement with order by clause specified on the column on which index exists.

Create Simple Index:

An index created on a single column of a table is called a simple index. The syntax for creating simple index that allows duplicate values is as described.

Syntax: CREATE INDEX IndexName ON TableNmae(ColumnName);
Ex. CREATE INDEX idx_emp ON EMP (name);

Create Composite Index:

An index created on more than one column is called a composite index. The syntax for creating a composite index that allows duplicate values is:
Syntax: CREATE INDEX<Index Name> ON <Table Name>
(<ColNm1>, <ColNm2>);
Ex. CREATE INDEX idx_emp ON EMP (Name, City);

Create of Unique Index:

A unique index can also be created on one or more columns. If an index is created on a single column, it is called a simple unique index.
Syntax: CREATE UNIQUE INDEX <Index Name>ON<Table Name>(<Col Name>);
Ex. Create unique index idx_enp on emp (name,city);

Composite unique index:

If an index is created on more than one column, it is called a composite unique index.
Syntax: CREATE UNIQUE INDEX<Index Name> ON <Table Name>
(<ColNm1>, <ColNm2>);

Ex. Create unique index idx_unq_emp on emp (name, city);

Dropping Indexes:

Index associated with the tables can be removed by using the drop index command.
Syntax: Drop Index <IndexName>;
Ex. Drop index idx_emp;

Using rowid to delete duplicate rows from a table:

If the user enters duplicate records in a table, a delete statement with a where clause will delete all the records that satisfy the where condition specified in the delete statement.

Now if we required that the oracle engine must retain one record and delete all other duplicate records. To retain one record, the where clause must be defined on the column that uniquely identifies a record.

So for that we can use inner select that means sub query to delete records.

Ex. Delete from client_master where rowid not in (select min (rowid) from client_master group by client_no, name, salary;

Using rownum in SQL statements:

ROWNUM column returns a number indication the order in which oracle engine selects the row from a table or set of joined rows. The first row selected has

a rownum of 1; the second has 2, and so on. So rownum can be used to limit the number of rows retrieved.

Ex. Select rownum, client_no, name from client_master
Where rownum <5;
Will give first 4 records in output.

Ex.

Select rownum, client_no ,name from client_master
Where rownum<5 order by name;
(Use above query with and without index created and see the difference in output)

View:

After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table, as required. This will answer data security requirements very well but will give rise to a great deal of redundant data being resident in tables in the database.

To reduce redundant data to the minimum possible oracle allows the creation of an object called a View. A view is mapped to a SELECT sentence. The table on which the view is based is described in the FROM clauses of the SELECT statement. The SELECT clause consists of a sub set of the columns of the tables. Thus a view, which is mapped to a table, will in effect have a sub set of the actual columns of the table from which it is built. This technique offers a simple, effective way of hiding columns of a table.

Some views are used only for looking at table data. Other views can be used to insert, update and delete table data as well as view data. If a view is used to only look at table data and nothing else the view is called a read only view. A view that is use to like at table data as well as insert, update and deletes table data is called an updateable view.

The reasons why views are created are:

- When data security is required.
- When data redundancy is to be kept to the minimum while maintaining data security

Syntax : Create view <ViewName> AS
SELECT <Colname1>,<Colname2> FROM <TableName>
WHERE <ColName>=<Expression List>;

GROUP BY <Grouping Criteria> HAVING <Predicate>

For all columns from main table:

Ex:

Create view vw_sales as select * from sales_order;

For selected columns:

Create view vw_sales as select order_no, order_date, product_no from sales_order;

Renaming the columns of view:

Create view vw_sales as select order_no o_no, order_date o_date, product_no p_no

From sales_order;

Selecting a data set from a view:

Select * from vw_sales;

Select order_no, order_date from vw_sales;

Select order_no, order_date from vw_sales where order_no = ‘o001’;

View define from single table:

- If the user wants to insert records with the help of a view, then the primary key column/s and all the not null columns must be included in the view.
- The user can update, delete records with the help of a view even if the primary key column and not null column/s are excluded from the view definition.

Ex.

Create view vw_client as

Select clinet_no, name, address, salary from client_master;

- When an insert operation is performed using the view

  Insert into vw_client values (‘c001’,’abc’,’desai vadi’,20000);

  1 row created.
- When an modify operation is performed using the view

  Update vw_client set name = ‘xyz’ where client_no =’c001’;
- When an delete operation is performed using the view

  Delete from vw_client where client_no = ‘c001’;

View Define from Multiple tables:

If a view is created from multiple tables, which were created **using a Referencing clause**, then though the Primary key columns as well as the Not Null columns are included in the view definition, the view’s behavior will be as follows:

- An INSERT operation is not allowed.
- The DELETE or MODIFY operation do not affect the master table.

- The view can used to MODIFY the columns of the detail table included in the view
- If a DELETE operation is executed on the view, the corresponding records from the detail table will be deleted.

Ex.

Create view vw_sal as select order_no, order_date, product_no, product_date
From sales_order, product_master
Where sales_order.product_no = product_master.product_no

Delete

Delete from vw_sal where order_no = 'o001' and product_no = 'p001';

Update

Update vw_sal set product_name = 'abc' where product_no = 'p002';

(Here reference between sales order and product master exists)

If a view is created from multiple tables, which not created using a Referencing clause

Then though the primary key columns as well as the not null columns are included in the view definition the view's behavior will be as follows:

- The insert, update or delete operation is not allowed. If attempted oracle display error.

Ex.

Create view vw_data as select product_no, product_name,client_no, client_name
From product_master, client_master

Sequence:

The quickest way to retrieve data from a table is to have a column in the table whose data uniquely identifies a row. By using this column and a specific value, in the where condition of select sentence the oracle engine will be able to identify and retrieve the row the fastest.

A constant is attached to a specific column in the table that ensure that the column is never left empty and that the data values in the column are unique since human beings do data entry, it is quite likely that a duplicate values could be entered, which violates this constraint and the entire row is rejected.

If the value entered into this column is computer generated it will always fulfill the unique constraint and the row will always be accepted for storage.

Oracle provides an object called Sequence that can generate numeric values. The generated can have a maximum of 38 digits. A sequence can be defined to generate numbers in ascending or descending order.

Syntax:

```
Create Sequence <Sequence Name>
        Increment by <Integer value>
        Start with <Integer value>
        Maxvalue <Integer value>
        Minvalue <Integer value>
        Cycle/Nocycle
        Cache <Integer value> /Nocache
        Order/ Noorder
```

Ex.

```
Create sequence seq_no
Increment by 1
Start with 1
Max value 9999
Cycle;
```

Alter Sequence:

```
Alter Sequence <Sequence name>
Increment by <Integer value>
Start with <Integer value>


Maxvalue <Integer value>
Minvalue <Integer value>
Cycle/Nocycle
Cache <Integer value> /Nocache
Order/ Noorder
```

Ex.

```
Alter sequence seq_no
Increment by 2
Cache 30;
```

Referencing a sequence:

Once a sequence is created SQL can be used to view the value held in its cache. To simply view sequence value one can use select statement as described below.

```
Select sequence_name.nextval from dual; (To view Next value held)
Select sequence_name.currval from dual; (To view Current value held)
```

Ex. SELECT my_seq_5.NEXTVAL FROM dual;

Dropping a Sequence:

```
Drop Sequence <Sequence Name>;
```

Chapter-3

Clause used:

Increment by:

Specifies the interval between sequence numbers, it can be any positive or negative value but not zero. If this clause is omitted, the default value is 1.

Minvalue:

Specifies the sequence minimum value.

Nominvalue:

Specifies a minimum value of 1 for an ascending sequence and -10^26 for a descending sequence.

Maxvalue:

Specifies the maximum value that a sequence can generate.

Nonmaxvale:

Specifies a maximum of 10^27 for an ascending sequence or -1 for a descending sequence. This is the default clause.

Start with:

Specifies the first sequence number to be generated. The default for an ascending sequence is the sequence minimum value(1) and for a descending sequence, it is the maximum value(-1).

Cycle:

Specifies that the sequence continues to generate repeat values after reaching either its maximum value.

NoCycle:

Specifies that a sequence cannot generate more values after reaching the maximum value.

Cache:

Specifies how many values of a sequence pre-allocates and keeps in memory for faster access. The minimum value for this parameter is two.

Nocache:

Specifies that values of a sequence are not pre-allocated.

Order:

This guarantees that sequence numbers are generated in the order of request. This is only necessary if using parallel server I parallel mode option. In exclusive mode option, a sequence always generates numbers in order.

NOorder:

This does not guarantee sequence numbers are generated in order of request. This is only necessary if you are using parallel server in parallel mode option. If the order/noorder clause is omitted, a sequence takes the noorder clause by default.

CREATING SYNONYMS

A synonym is an alternative name for objects such as tables, views, sequences, stored procedures and other database object.

Syntax: CREATE [PUBLIC] SYNONYM
SYNONYM_NAME FOR [SCHEMA..]
OBJECT_NAME [@DBLINK];

- The PUBLIC phrase means that the synonym is a public synonym and is accessible to all users. Remember though that the user must first have the appropriate privileges to the object to use the synonym.
- The SCHEMA phrase is the appropriate schema. If this phrase is omitted, oracle assumes that a reference is made to the user's own schema.
- The OBJECT_NAME phrase is the name of the object for which you are creating the synonyms. It can be one of the following: Table, Package, View, Materialized View, Sequence, Java Class Schema Object, Stored Procedure, User-Defined Object, Function, Synonym.

  Syntax: CREATE SYNONYM EMPLOYEE FOR EMP;
  Massage: Synonym Created.

DROP SYNONYMS

Syntax: drop synonym synonym_name;
Ex:

Drop the synonym named Emp
SQL> Drop Synonym Emp;

Database Link:

Database links tell oracle how to get from one database to another. You may also specify the access path in an ad hoc fashion. If you will frequently use the same connection to a remote database the database link is appropriate. Database links specify the following connection information:

- The communications protocol to use during the connection (TCP/IP)
- The host on which the remote database resides
- The name of the database on the remote host
- The name of a valid account in the remote database
- The password for that account

If you are a user in the "Local" database, you can access objects in the "Remote" database via a database link. To do this simply appends the database link name to the name of any table or view that is accessible to the remote account. When appending the database link name to a table or view name, you must precede the database line name with an "@" sign.

Creating a database link:

Ex.1 create database link myremotedb using 'system'
select sysdate from dual myremotedb

OR

Syntax:

Create database link link_name
Connect to user_name identified by password using sqlnet_string;

Ex:

Crate database link backup
Connect to abc identified by abc using 'backup_data';

Ex:

SQL> SELECT * from student@backup;

- Link_name: Used to give name to the database link.
- User name & password: Specify valid user name and password to the remote database.
- Sqlnet string: This is a valid connect string for the remote database.

****