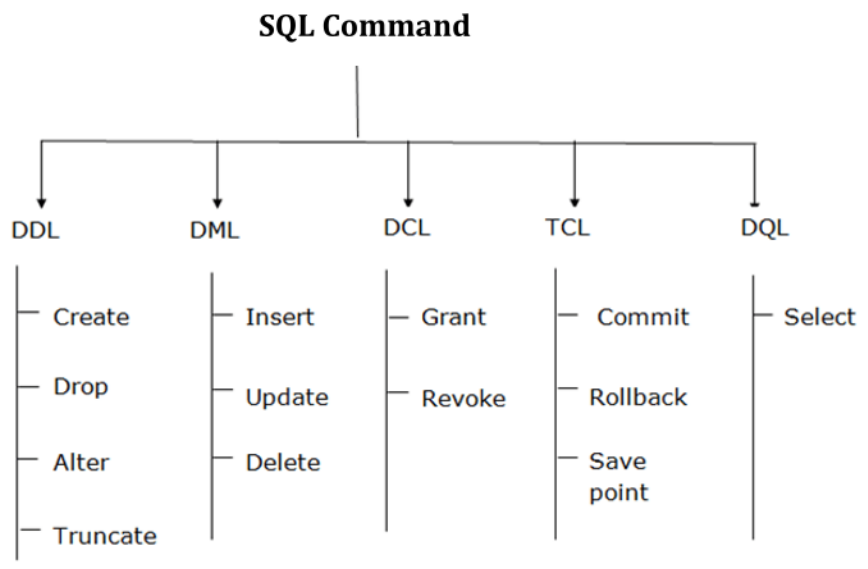
UNIT-1 Advance SQL



Que: Transactional Control: Commit, Save point, Rollback

- When one delete some data by mistake then to recover/undo those data transaction control is used.
- Transactional control is the ability to manage various transactions that may occur within a relational database management system.
- When you speak of transactions, you are referring to the INSERT, UPDATE, and DELETE commands.
- When table regarding result of student is created and student want to see/view their result then student are not allowed to do changes like insert , update , delete in result table . so to restrict the access of student table Data control is required

commit := make changes permanent in transaction.
rollback := make the changes undo
savepoint:= rollback to particuler transaction point .

COMMIT statement:
- The COMMIT statement is used to permanently save the changes made during a transaction.
- For example, if we want to insert some data into a table and commit the changes.

syntax:

```
insert into employee (id,name,salary) values(1,'Amit',50000);
COMMIT;
```

- In this example, the INSERT statement adds a new record to the "employee" table. The COMMIT statement then saves the changes made by the INSERT statement.

ROLLBACK statement:

- The ROLLBACK statement is used to undo the changes made during a transaction.
- For example, if we want to update some data in a table but then decide to undo the changes,

  syntax:

  ```
  UPDATE employee set salary = 55000 where id=1;

  Rollback;
  ```

- In this example, the UPDATE statement changes the salary of the employee with id 1.
- The ROLLBACK statement then undoes the changes made by the UPDATE statement.

SAVEPOINT statement:

- The SAVEPOINT statement is used to create a point within a transaction that can be used to roll back to a specific point in the transaction.
- For example, if we want to update some data in a table and create a savepoint before doing so, we would use the following syntax:

```
Declare
BEGIN
SAVEPOINT sp1;
UPDATE employee SET salary = 70000 where id =83;
ROLLBACK TO sp1;
END;
```

- In this example, the SAVEPOINT statement creates a savepoint called "sp1".
- The UPDATE statement changes the salary of the employee with id 1.
- The ROLLBACK TO statement then rolls back to the savepoint created by the SAVEPOINT statement, undoing the changes made by the UPDATE statement.

Que: DCL Commands : Grant and Revoke

1. **grant**
   - SQL GRANT is a command used to provide access or privileges on the database objects to the users.

Syntax:

```
grant privilageName (Update,Delete)
on objectName
 to {userName | public | roleName }
```

Here:

- **privilegeName:** It represents the permission that is to be granted.(update,delete etc)
- **objectName:** It represents the name of the database object, i.e., view, table, index, etc.
- **userName:** It represents the user to which permission is to be provided.
- **public:** It represents that all database users are given permissions.
- **role_name:** It represents that users with a particular role are given these permissions.

Example:

GRANT SELECT On person_details TO user1

Here
This query will grant the SELECT permission to person_ details table to user1.

Same more than use
GRANT SELECT(SELECT,INSERT,DELETE,UPDATE) ON Person_Details TO user1

2. revoke

- This command is used to **revoke or withdraw** permissions that were **previously granted** to an account on a database object
- The REVOKE command is the opposite of the command GRANT

Syntax:

revoke privilege_name
on object_name
from {user_name |public |role_name}

Example:

revoke select on person_details FROM user1;

Que:Types of locks

In simple words , a locking means to restrict other users from accessing data temporarily.
In oracle locks are enforced on tables in two ways:

Implicit Locking
- Lock which is manage by **system itself**.

Explicit Locking
- Lock which is manage by **User itself.**

1. **Shared lock**

- Shared locks are applied while performing read operations.
- Read operation involve only viewing of data , mostly using SELECT statement.
- Multiplication shared locks can be placed simultaneously on table on table or other object.
- As read operation does not modify table data,multiple read operation can be performed

2. **Exclusive lock**

- Exclusive locks are applied while performing write operation. Write operation involve modifying data using INSERT,DELETE or UPDATE statements
- The exclusive locks are **useful in DML operations** like INSERT, UPDATE, or DELETE statements.
- Only one exclusive lock can be placed on a table or other object.
- Multiple users cannot modify (insert update or delete) the same data simultaneously.
- Multiple users may need to access different parts of the same table.

3. **Row level locks**

- This lock is used when a condition given in WHEREclause evaluates to single row.

For example ,….. Where name=‘ AMIT ’;

In this case , only single row is locked . Other records of the Table can be accessed by other users.

4. **Page level lock**

- This lock is used when a condition given in WHERE clause evaluates **to set of rows**.

For example,… where=‘AMIT’;

In case , only a set of rows are locked . Other records of the table can be accessed by other users.

5. **Table level locks**

- This lock is used when a SQL statement does not contain WHERE clause
- In this case , a query accesses entire table , and so , entire table is locked. Due to this reason , no any other user can access other part of the table

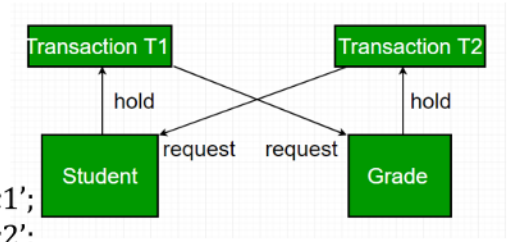
Syntax : LOCK TABLE Tablename
IN lockmode MODE[NOWAIT];

Mode	Specific
EXLUSIVE	Allow query on a table ,but prohibits any other operation. other users can only view data of a table
SHARDE	Allow concurrent queries ,but no update operations is allowed.
ROW SHARDE	Specifics row level lock. User cannot lock whole table Allowing concurrent access for all users of the table.
SHARE UPDATE	Same as above. Exists for compatibility with older versions.
ROW EXLUSIVE	Similar to ROWSHARE,but prohibits shared locking .So Only one user can access the table at a time.

- ❖ if NOWAIT specified and table is not free , this command will return with error message indicating "resource is a busy".
- ❖ various MODE of lock are EXLUSIVE , SHARED , ROW SHARE , SHARE UPDATE ,and ROW EXLUSIVE MODE.

5. Deadlock

- ➢ In a deadlock, two database operations wait for each other to release a lock.
- ➢ A deadlock occurs when two users have a lock, each on a separate object, and, they want to acquire a lock on each other's object.
- ➢ When this happens, the first user has to wait for the second user to release the lock, but the second user will not release it until the lock on the first user's object is freed. At this point, both the users are at an impasse and cannot proceed with their business.
- ➢ In such a case, Oracle detects the deadlock automatically and solves the problem by aborting one of the two transactions.



Example 1:
Transaction 1

```
BEGIN
UPDATE client_master SET salary = 500 WHERE client_no='c1';
UPDATE client_master SET salary = 500 WHERE client_no='c2';
END
```

Transaction 2

```
BEGIN
UPDATE client_master SET salary = 5000 WHERE client_no='c2';
 UPDATE client_master SET salary = 500 WHERE client_no='c1';
END
```

- ➢ Assume that Transaction 1 and Transaction2 begin exactly at the same time.
- ➢ By default Oracle automatically places exclusive lock on data that is being updated.
- ➢ This causes Transaction 1 to wait for Transaction2 to complete but in turn Transaction2 has to wait for Transaction 1 to complete.

Que: Synonym

- A **synonym** is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.
- A synonym provides you with many benefits if you use it properly.
- It can be used to hide the actual identify of the object being referenced.
  Ex:
  If there is a need to name of some particular table,then a synonym can be created to refer that table,hiding the original name.

Syntax:

```
CREATE SYNONYM  synonym-name for object-name;
```

Example:

```
CREATE SYNONYM C1 FOR CITY;
```

Output:

```
Synonym Created.
```

```
Insert into c1 values('203','CANADA');
```

- **Drop table**

Syntax:

```
Drop synnonym synonym_name
```

Example:

```
Drop synnonym c1;
```

Que: Sequences

- Oracle provides an object called a **Sequence** that can generate numeric values.
- A Sequence is simply an automatic counter, which generates sequential numbers whenever required.”
- The values generated by sequence can be used in insert and update operations.
- The value generated can have a maximum of 38 digits.
- Generate numbers in ascending or descending order.
- Provide intervals between numbers.
- Caching of sequence numbers in memory to speed up their availability
- A sequence is an independent object and can be used with any table that requires its output.

Creating Sequences

The minimum information required for generating numbers using a sequence is:

- The starting number
- The maximum number that can be generated by a sequence
- The increment value for generating the next number.

Syntax:

```
CREATE SEQUENCE <Sequence Name>
  [ START WITH value ]
  [ INCREMENT BY value ]
  [ MINVALUE value | NO MINVALUE ]
  [ MAXVALUE value | NO MAXVALUE ]
  [ CYCLE | NO CYCLE ]
  [ CACHE value | NO CACHE ];
```

Option	**Specifies…**
START WITH	Specifies the **first** sequence number. **Default** for **ascending** sequence the minimum value:1 **Default** for **descending** sequence the maximum value: **-**
INCREMENT BY	Specifies the **interval** between sequence numbers It can be any positive or negative number but not zero. **Default value is 1**
MINVALUE	Specifies the sequence minimum value
MAXVALUE value	The maximum value allowed for the sequence.
CYCLE	Specifies to **repeat cycle** of generating values after reaching maximum value
NOCYCLE	Specifies that **no more numbers can be generated** after reaching maximum value.
CACHE value	It caches the sequence numbers to minimize disk IO.
NO CACHE	It does not cache the sequence numbers.

Example:

```
CREATE SEQUENCE S1
START WITH 1
INCREAMNT BY 1
MINVALUE 1
MAXVALUE 99
CYCLE;
```

NEXTVAL AND CURRVAL

NEXTVAL and CURRVAL to access the values generated by the Sequence.
These pseudo columns are used with a Sequence name as described below:

Syntax:

SequenceName.CURRVAL

Returns the current value of the sequence.

SequenceName.NEXTVAL

Increase the value of the sequence and Returns the next value of the sequence.

Example:

-- Creating a sequence

```
CREATE SEQUENCE s1
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

-- Creating a table with a column that uses the sequence
CREATE TABLE employees (
  employee_id NUMBER PRIMARY KEY,
  first_name VARCHAR2(50),
  last_name VARCHAR2(50)
);
```

-- Inserting records into the table with automatically generated sequence values

```
INSERT INTO employees VALUES (s1.NEXTVAL, 'Amit', 'Patel');

INSERT INTO employeesVALUES (s1.NEXTVAL, 'Nish', 'Patel');
```

-- Querying the table

```
SELECT * FROM employees;
```

❖ ALTER SEQUENCE

Changes the sequence. Using this parameter you can change all sequence options, except for the sequence type.

Syntax

```
ALTER SEQUENCE <sequence>
[START <start-point>]
[INCREMENT <increment>] [
CACHE <cache>]
```

- <sequence> Defines the sequence you want to change.
- START Defines the initial sequence value.
- INCREMENT Defines the value to increment when it calls .next().
- CACHE Defines the number of values to cache, in the event that the sequence is of the type CACHED.

Examples

Alter a sequence, resetting the start value to 100:

```
ALTER SEQUENCE s1 START 100
```

Que: SQL INDEX

- The index is similar to index of a book, where a page containing required topic can be found quickly by locating an index, finding the topic and turning to the desired page directly.
- The Index in SQL is a special table used to speed up the searching of the data in the database tables.
- It also retrieves a vast amount of data from the tables frequently.
- The INDEX requires its own space in the hard disk.

❖ Simple Index

An index created on a single column of a table is called a **Simple Index .**

Syntax:
CREATE INDEX <IndexName> ON <TableName> (<ColumnName>);

Example 1:
CREATE INDEX indCustName ON Customer (name);

Output: Index created.

❖ **Composite Index**
An index created on more than one column is called a **Composite Index.**

Syntax:
CREATE INDEX <Index Name> ON <Table Name> (<ColumnNamel> <ColumnName2>);

Example 1:
CREATE INDEX indCustName ON Customer (**name, Acc_No**);

Output: Index created

❖ **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.**

➢ When the user defines a primary key or a unique key constraint at table or column level, the Oracle engine automatically creates a unique index on the primary key or unique key column

Syntax:
CREATE UNIQUE INDEX <Index Name> ON <Table Name> (<Column Name>);

Example:
Create Unique Index Idx_Custno On Customer (Cust_No);

Que: SQL views

- A view is a virtual or logical table that allows viewing or manipulating parts of the tables.
- A view is derived from one or more tables, known as base tables.
- A view is looks like and works similarly to normal tables. But unlike tables a view does not have storage space to store data.
- A view is created by a query, i.e. a select statement, which uses base tables. Data for views are extracted from these base tables based on specified query.
- A view is dynamic and always reflects the current data of the base tables.
  1. Creating view
  2. Updating view
  3. Deleting view

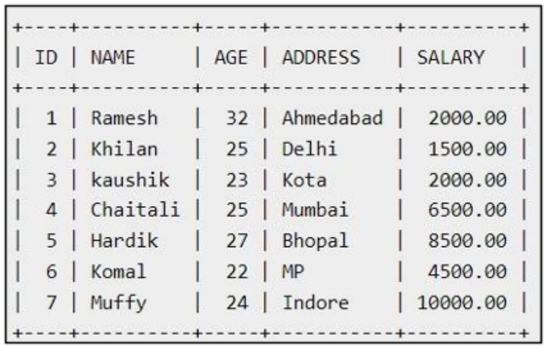
❖ Creating Views

➢ Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

➢ To create a view, a user must have the appropriate system privilege according to the specific implementation.

Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```



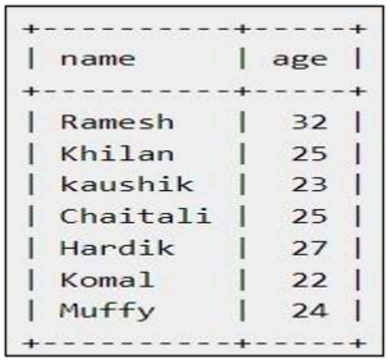
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Example

```
SQL > CREATE VIEW
CUSTOMERS_VIEW AS
        SELECT name, age
        FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```



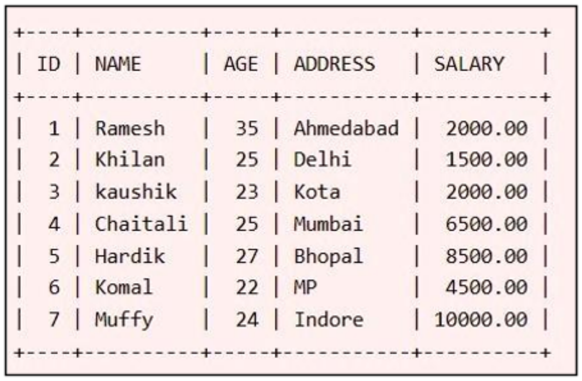
```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

❖ Update view

➢ Views can also be used for data manipulation

➢ Views on which data manipulation can be done are called Updateable Views. When an updateable view name is given in an Insert Update, or Delete SQL statement, modifications to data in the view will be immediately passed to the underlying table

```
SQL > UPDATE CUSTOMERS_VIEW
  SET AGE = 35
  WHERE name = 'Ramesh';
```

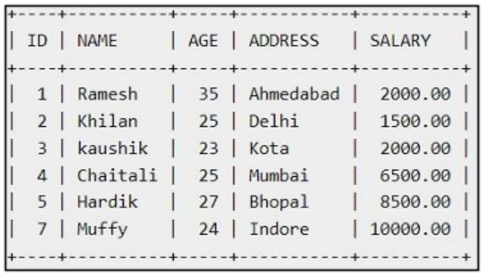


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

❖ Deleting Rows into a View

- Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.
- Following is an example to delete a record having AGE = 22.

SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```