## SQL2

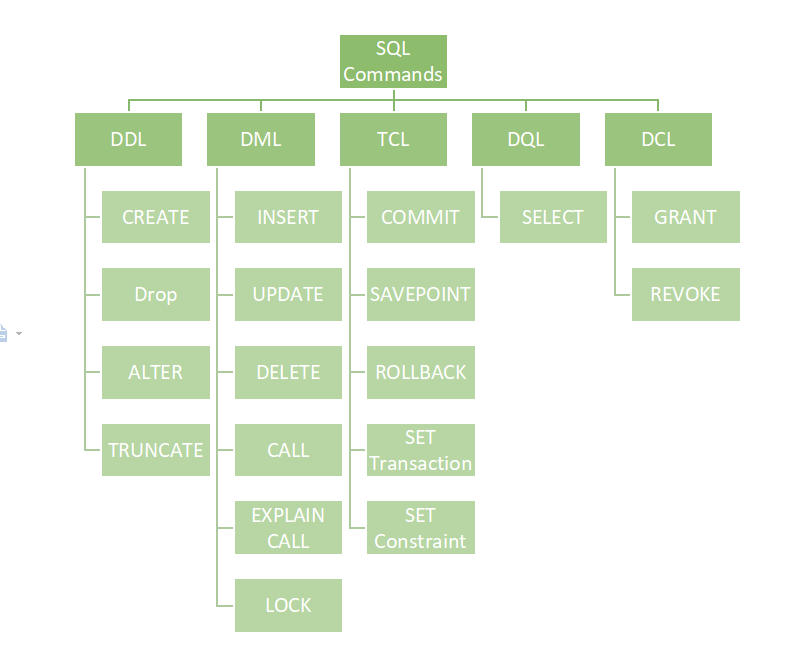
Structured Query Language (SQL), as we all know, is the database language by the use of which we can perform certain operations on the existing database, and also we can use this language to create a database. [SQL](https://www.geeksforgeeks.org/structured-query-language/) uses certain commands like CREATE, DROP, INSERT, etc. to carry out the required tasks.

[SQL commands](https://www.geeksforgeeks.org/basic-sql-commands/) are like instructions to a table. It is used to interact with the database with some operations. It is also used to perform specific tasks, functions, and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

These [SQL](https://www.geeksforgeeks.org/sql-concepts-and-queries/) commands are mainly categorized into five categories:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language

Now, we will see all of these in detail.



## **DDL (Data Definition Language)**

[DDL](https://www.geeksforgeeks.org/features-of-structured-query-language-sql/) or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

* [CREATE](https://www.geeksforgeeks.org/sql-create/): This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
* [DROP](https://www.geeksforgeeks.org/sql-drop-truncate/): This command is used to delete objects from the database.
* [ALTER](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)**:** This is used to alter the structure of the database.
* [TRUNCATE](https://www.geeksforgeeks.org/sql-drop-truncate/)**:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
* [COMMENT](https://www.geeksforgeeks.org/sql-comments/): This is used to add comments to the data dictionary.
* [RENAME](https://www.geeksforgeeks.org/sql-alter-rename/)**:** This is used to rename an object existing in the database.

## **DQL (Data Query Language)**

**DQL** statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows: it is a component of SQL that allows getting data from the database and imposing order upon it. It includes the SELECT statement. This command allows getting the data out of the database to perform operations with it. When a SELECT is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.

List of DQL:

* [SELECT](https://www.geeksforgeeks.org/sql-select-clause/)**:** It is used to retrieve data from the database.

## **DML(Data Manipulation Language)**

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

List of DML commands:

* [INSERT](https://www.geeksforgeeks.org/sql-insert-statement/): It is used to insert data into a table.
* [UPDATE](https://www.geeksforgeeks.org/sql-update-statement/)**:** It is used to update existing data within a table.
* [DELETE](https://www.geeksforgeeks.org/sql-delete-statement/): It is used to delete records from a database table.
* [LOCK:](https://www.geeksforgeeks.org/sql-lock-table/) Table control concurrency.
* **CALL:** Call a PL/SQL or JAVA subprogram.
* **EXPLAIN PLAN:** It describes the access path to data.

## **DCL (Data Control Language)**

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of  DCL commands:

[GRANT:](https://www.geeksforgeeks.org/mysql-grant-revoke-privileges/) This commandgives users access privileges to the database.

**Syntax:**

GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

[REVOKE:](https://www.geeksforgeeks.org/difference-between-grant-and-revoke/)This command withdraws the user’s access privileges given by using the GRANT command.

**Syntax:**

REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

## **TCL (Transaction Control Language)**

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions[*here*](https://www.geeksforgeeks.org/sql-transactions/). Hence, the following TCL commands are used to control the execution of a transaction:

**BEGIN:** Opens a Transaction.

[COMMIT](https://www.geeksforgeeks.org/sql-transactions/)**:** Commits a Transaction.

**Syntax:**

COMMIT;

[ROLLBACK](https://www.geeksforgeeks.org/sql-transactions/)**:** Rollbacks a transaction in case of any error occurs.

**Syntax:**

ROLLBACK;

[SAVEPOINT](https://www.geeksforgeeks.org/sql-transactions/)**:** Sets a save point within a transaction.

**Syntax:**

SAVEPOINT SAVEPOINT\_NAME;

Examples

## DATA DEFINITION LANGUAGE COMMANDS

##### COMMANDS

SQL> create table stud (sname varchar2(30), sid varchar2(10), sage number(2), sarea varchar2(20));

| Table created.  SQL> desc stud; Name | Null? | Type |
| --- | --- | --- |
| SNAME |  | VARCHAR2(30) |
| SID |  | VARCHAR2(10) |
| SAGE |  | NUMBER(2) |
| SAREA |  | VARCHAR2(20) |

SQL>alter table stud modify ( sage number(10)); Table altered.

SQL> alter table stud add ( sdept varchar2(20));

| Table altered.  SQL> desc stud; Name | Null? | Type |
| --- | --- | --- |
| SNAME |  | VARCHAR2(30) |
| SID |  | VARCHAR2(10) |
| SAGE |  | NUMBER(10) |
| SAREA |  | VARCHAR2(20) |
| SDEPT |  | VARCHAR2(20) |

SQL> alter table stud drop ( sdept varchar2(20)); Table altered.

| SQL> desc studs; Name | Null? | Type |
| --- | --- | --- |
| SNAME |  | VARCHAR2(30) |
| SID |  | VARCHAR2(10) |
| SAGE |  | NUMBER(10) |
| SAREA |  | VARCHAR2(20) |
| SQL> truncate table studs; |  |  |
| Table truncated. |  |  |
| SQL> desc studs; Name | Null? | **Type** |
| SNAME |  | VARCHAR2(30) |
| SID |  | VARCHAR2(10) |
| SAGE |  | NUMBER(10) |
| SAREA |  | VARCHAR2(20) |
| SDEPT |  | VARCHAR2(20) |
| SQL> drop table studs; |  |  |
| Table dropped. |  |  |



## DATA MANIPULATION LANGUAGE COMMANDS

##### DESCRIPTION

**THE ORACLE TABLE – DUAL**

Dual is a small oracle table which consists of only one row and one column and contains the value X in that column.

##### INSERT

This command is used to insert values into the table.

##### SELECT

This command is used to display the contents of the table or those of a particular column.

##### RENAME

This command renames the name of the table.

##### ARITHMETIC OPERATIONS

Various operations such as addition, multiplication, subtraction and division can be performed using the numbers available in the table.

##### DISTINCT

This keyword is used along with select keyword to display unique values from the specified column. It avoids duplicates during display.

##### ORDER BY CLAUSE

The order by clause arranges the contents of the table in ascending order (by default) or in descending order (if specified explicitly) according to the specified column.

##### CONCATENATION OPERATOR

This combines information from two or more columns in a sentence according to the format specified.

##### LOGICAL OPERATORS

* AND : The oracle engine will process all rows in a table and displays the result only when all of the conditions specified using the AND operator are specified.
* OR : The oracle engine will process all rows in a table and displays the result only when any of the conditions specified using the OR operators are satisfied.
* NOT : The oracle engine will process all rows in a table and displays the result only when none of the conditions specified using the NOT operator are specified.
* BETWEEN : In order to select data that is within a range of values, the between operator is used. (AND should be included)

##### PATTERN MATCH

* LIKE PREDICATE : The use of like predicate is that it allows the comparison of one string value with another string value, which is not identical. This is achieved by using wildcard characters which are % and \_. The purpose of % is that it matches any string and \_ matches any single character.
* IN AND NOT IN PREDICATE : The arithmetic operator = compares a single value to another single value. In case a value needs to be compared to a list of values then the in

predicate is used.The not in predicate is the opposite of the in predicate. This will select all the rows whose values do not match all of the values in the list.

##### NUMERIC FUNCTIONS

* ABS: It returns the absolute value of ‘n’.
* POWER: It returns m raised to nth power. n must be an integer else an error is returned.
* ROUND: It returns n rounded to m places right of the decimal point. If m is omitted, n is rounded to zero places. m must be an integer.
* SQRT: It returns square root of n. n should be greater than zero.

##### STRING FUNCTIONS

* LOWER: It returns char with letters in lower case.
* INITCAP: It returns char with the first letter in upper case.
* UPPER: It returns char with all letters forced to upper case.
* SUBSTR: It returns a portion of char beginning at character m, exceeding up to n characters. If n is omitted result is written up to the end character. The 1st position of char is one.
* LENGTH: It returns the length of char
* LTRIM: It removes characters from the left of char with initial characters removed up to the 1st character not in set.
* RTRIM: It returns char with final characters removed after the last character not in the set. Set is optional. It defaults to spaces.
* LPAD: It returns char1, left padded to length n with the sequence of characters in char2. char2 defaults to blanks.
* RPAD: It returns char1, right padded to length n with the characters in char2, replicated as many times as necessary. If char2 is omitted, it is padded with blanks.

##### AGGREGATE FUNCTIONS

* AVG (N): It returns average value of n ignoring null values.
* MIN (EXPR): It returns minimum value of the expression.
* COUNT (EXPR): It returns the number of rows where expression is not null.
* COUNT (\*): It returns the number of rows in the table including the duplicates and those with null values.
* MAX (EXPR): It returns maximum value of the expression.
* SUM(N): It returns sum of values of n.

##### CONVERSION FUCTIONS

* TO\_NUMBER(CHAR): It converts the char value containing a number to a value of number data type.
* TO\_CHAR(N,FMT): It converts a value of number data type to a value of char data type, using the optional format string. It accepts a number n and a numeric format fmt in which the number has to appear. If fmt is omitted, n is converted to a char value exactly long enough to hold significant digits.
* TO\_CHAR(DATE, FMT): It converts a value of data type to char value. It accepts a date as well as the format in which the date has to appear. Fmt must be a date format. If fmt is omitted, date is the default date format.

##### DATE FUNCTIONS

* SYSDATE : The sysdate is a pseudo column that contains the current date and time. It requires no arguments when selected from the table dual and returns the current date.
* ADD\_MONTHS(D,N): It returns date after adding the number of months specified with the function.
* LAST\_DAY(D): It returns the last date of the month specified with the function
* MONTHS\_BETWEEN(D1,D2): It returns number of months between D1 and D2.
* NEXT\_DAY(DATE, CHAR): It returns the date of the first week day named by char . char must be a day of the week.

##### GROUP BY CLAUSE

The group by clause is another section of the select statement. This optional class tells oracle to group rows based on distinct values that exists for specified columns.

##### HAVING CLAUSE

The having clause can be used in conjunction with the group by clause. Having imposes a condition on the group by clause, which further filters the groups created by the group by clause. **SET OPERATIONS**

* UNION CLAUSE: Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of two or more queries into a single set of rows and columns.
* INTERSECT CLAUSE: Multiple queries can be put together and their output can be combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected. The output in an intersect clause will include only those rows that are retrieved by both the queries.

##### JOIN OPERATIONS

* INNER JOIN/ NATURAL JOIN/ JOIN: It is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
* OUTER JOIN: It is an extension of join operation to deal with missing information.

Left Outer Join: It takes tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join.

Right Outer Join: It takes tuples in the right relation that did not match with any tuple in the left relation, pads the tuples with null values for all other attributes from the left relation and adds them to the result of the natural join.

Full Outer Join: It combines tuples from both the left and the right relation and pads the tuples with null values for the missing attributes and them to the result of the

natural join.

##### COMMANDS

**CREATION OF TABLE**

SQL>create table stud (sname varchar2(30), sid varchar2(10), sage number(10), sarea varchar2(20), sdept varchar2(20));

Table created.

##### INSERTION OF VALUES INTO THE TABLE

SQL> insert into stud values ('ashwin',101,19,'anna nagar','aeronautical');

1 row created.

SQL> insert into stud values ('bhavesh',102,18,'nungambakkam','marine'); 1 row created.

SQL> insert into stud values ('pruthvik',103,20,'anna nagar','aerospace'); 1 row created.

SQL> insert into stud values ('charith',104,20,'kilpauk','mechanical'); 1 row created.

SQL> select \* from stud;

SNAME SID SAGE SAREA SDEPT

| ashwin | 101 | 19 anna nagar | aeronautical |
| --- | --- | --- | --- |
| bhavesh | 102 | 18 nungambakkam | marine |
| pruthvik | 103 | 20 anna nagar | aerospace |
| charith | 104 | 20 kilpauk | mechanical |

##### RENAMING THE TABLE ‘STUD’

SQL> rename stud to studs; Table renamed.

##### ARITHMETIC OPERATION

SQL> select sname, sid+100 "stid" from studs; SNAME stid

| ashwin | 201 |
| --- | --- |
| bhavesh | 202 |
| pruthvik | 203 |
| charith | 204 |

##### CONCATENATION OPERATOR

SQL> select sname || ' is a ' || sdept || ' engineer. ' AS "PROFESSION" from studs; PROFESSION

ashwin is a aeronautical engineer. bhavesh is a marine engineer. pruthvik is a aerospace engineer. charith is a mechanical engineer.

##### DISPLAY ONLY DISTINCT VALUES

SQL> select distinct sarea from studs; SAREA

anna nagar kilpauk nungambakkam

##### USING THE WHERE CLAUSE

SQL> select sname,sage from studs where sage<=19;

SNAME SAGE



ashwin 19

bhavesh 18

##### BETWEEN OPERATOR

| bhavesh | nungambakkam | 102 |
| --- | --- | --- |
| pruthvik | anna nagar | 103 |
| charith  **IN PREDICATE** | kilpauk | 104 |

SQL> select sname,sarea, sid from studs where sid between 102 and 104; SNAME SAREA SID

SQL> select sname,sarea , sid from studs where sid in(102,104); SNAME SAREA SID

| bhavesh | nungambakkam | 102 |
| --- | --- | --- |
| charith | kilpauk | 104 |

##### PATTERN MATCHING

SQL> select sname, sarea from studs where sarea like '%g%';

SNAME SAREA



ashwin anna nagar

bhavesh nungambakkam

pruthvik anna nagar

##### LOGICAL AND OPERATOR

SQL> select sname ,sid from studs where sid>102 and sarea='anna nagar';

SNAME SID



pruthvik 103

##### LOGICAL OR OPERATOR

SQL> select sname ,sid from studs where sid>102 or sarea='anna nagar';

| SNAME | SID |
| --- | --- |
| ashwin | 101 |
| pruthvik | 103 |
| charith | 104 |

##### NOT IN PREDICATE

SQL> select sname, sid from studs where sid not in(102,104);

SNAME SID



ashwin 101

pruthvik 103

##### UPDATING THE TABLE

SQL> alter table studs add ( spocket varchar2(20) ); Table altered.

SQL> update studs set spocket=750 where sid=101; 1 row updated.

SQL> update studs set spocket=500 where sid=102; 1 row updated.

SQL> update studs set spocket=250 where sid=103; 1 row updated.

SQL> update studs set spocket=100 where sid=104; 1 row updated.

SQL> select \* from studs;

SNAME SID SAGE SAREA SDEPT



| SPOCKET |  | | | |
| --- | --- | --- | --- | --- |
| ashwin | 101 | 19 | anna nagar | aeronautical |
| 750 |  |  |  |  |
| bhavesh | 102 | 18 | nungambakkam | marine |
| 500 |  |  |  |  |
| pruthvik | 103 | 20 | anna nagar | aerospace |
| 250 |  |  |  |  |
| charith | 104 | 20 | kilpauk | mechanical |
| 100 |  |  |  |  |

##### AGGREGATE FUNCTIONS

SQL> select avg( spocket ) result from studs;

RESULT



400

SQL> select min(spocket) result from studs; RESULT



100

SQL> select count(spocket) result from studs; RESULT



4

SQL> select count(\*) result from studs; RESULT



4

SQL> select count(spocket) result from studs where sarea='anna nagar'; RESULT



2

SQL> select max(spocket) result from studs;

RESULT



750

SQL> select sum(spocket) result from studs; RESULT



1600

##### NUMERIC FUNCTIONS

SQL> select abs(-20) result from dual;

RESULT



20

SQL> select power (2,10) result from dual; RESULT



1024

SQL> select round(15.359,2) result from dual; RESULT



15.36

SQL> select sqrt (36) result from dual; RESULT



6

##### STRING FUNCTIONS

SQL> select lower('ORACLE') result from dual; RESULT

oracle

SQL> select upper('oracle') result from dual; RESULT



ORACLE

SQL> select initcap('Oracle') result from dual;

RESULT



Oracle

SQL> select substr('oracle' ,2 ,5) result from dual;

RESULT



racle

SQL> select lpad('oracle',10,'#') result from dual; RESULT



####oracle

SQL> select rpad ('oracle',10,'^') result from dual; RESULT



oracle^^^^

##### CONVERSION FUNCTIONS

SQL> update studs set sage=to\_number(substr(118,2,3)); 4 rows updated.

SQL> select \* from studs;

| SNAME | SID | SAGE SAREA SDEPT |
| --- | --- | --- |
| SPOCKET |  |  |
| ashwin | 101 | 18 anna nagar aeronautical |
| 750 |  |  |
| bhavesh | 102 | 18 nungambakkam marine |
| 500 |  |  |
| pruthvik | 103 | 18 anna nagar aerospace |
| 250 |  |  |
| charith | 104 | 18 kilpauk mechanical |
| 100 |  |  |

SQL> select to\_char( 17145, '099,999') result from dual;

RESULT



017,145

SQL> select to\_char(sysdate,'dd-mon-yyyy') result from dual; RESULT



16-jul-2008

##### DATE FUNCTIONS

SQL> select sysdate from dual;

SYSDATE



16-JUL-08

SQL> select sysdate,add\_months(sysdate,4) result from dual; SYSDATE RESULT



16-JUL-08 16-NOV-08

SQL> select sysdate, last\_day(sysdate) result from dual; SYSDATE RESULT



16-JUL-08 31-JUL-08

SQL> select sysdate, next\_day(sysdate,'sunday') result from dual; SYSDATE RESULT



16-JUL-08 20-JUL-08

SQL> select months\_between('09-aug-91','11-mar-90') result from dual; RESULT



16.935484

##### GROUP BY CLAUSE

SQL> select sarea, sum(spocket) result from studs group by sarea; SAREA RESULT

| anna nagar |  | 1000 |
| --- | --- | --- |
| nungambakkam |  | 500 |
| kilpauk |  | 100 |

##### HAVING CLAUSE

SQL> select sarea, sum(spocket) result from studs group by sarea having spocket<600;

SAREA RESULT



nungambakkam 500

kilpauk 100

##### DELETION

SQL> delete from studs where sid=101; 1 row deleted.

SQL> select \* from studs;

| SNAME | SID | SAGE SAREA SDEPT |
| --- | --- | --- |
| SPOCKET |  |  |
| bhavesh | 102 | 18 nungambakkam marine |
| 500 |  |  |
| pruthvik | 103 | 20 anna nagar aerospace |
| 250 |  |  |
| charith | 104 | 20 kilpauk mechanical |
| 100 |  |  |

##### CREATING TABLES FOR DOING SET OPERATIONS

TO CREATE PRODUCT TABLE

SQL> create table product(prodname varchar2(30), prodno varchar2(10)); Table created.

SQL> insert into product values('table',10001); 1 row created.

SQL> insert into product values('chair',10010); 1 row created.

SQL> insert into product values('desk',10110);

1 row created.

SQL> insert into product values('cot',11110); 1 row created.

SQL> insert into product values('sofa',10010); 1 row created.

SQL>

SQL> insert into product values('tvstand',11010); 1 row created.

SQL> select \* from product;

PRODNAME PRODNO

| table | 10001 |
| --- | --- |
| chair | 10010 |
| desk | 10110 |
| cot | 11110 |
| sofa | 10010 |
| tvstand | 11010 |

TO CREATE SALE TABLE

SQL> create table sale(prodname varchar2(30),orderno number(10),prodno varchar2(10)); Table created.

SQL> insert into sale values('table',801,10001); 1 row created.

SQL> insert into sale values('chair',805,10010); 1 row created.

SQL> insert into sale values('desk',809,10110); 1 row created.

SQL> insert into sale values('cot',813,11110);

1 row created.

SQL> insert into sale values('sofa',817,10010);

| 1 row created.  SQL> select \* from sale; |  | |
| --- | --- | --- |
| PRODNAME | ORDERNO | PRODNO |
| table | 801 | 10001 |
| chair | 805 | 10010 |
| desk | 809 | 10110 |
| cot | 813 | 11110 |
| sofa | 817 | 10010 |
| **SET OPERATIONS** |  |  |

SQL> select prodname from product where prodno=10010 union select prodname from sale where prodno=10010;

PRODNAME



chair sofa

SQL> select prodname from product where prodno=11110 intersect select prodname from sale where prodno=11110;

PRODNAME



cot

##### CREATING TABLES FOR DOING JOIN AND NESTED QUERY OPERATIONS

TO CREATE SSTUD1 TABLE

SQL> create table sstud1 ( sname varchar2(20) , place varchar2(20)); Table created.

SQL> insert into sstud1 values ( 'prajan','chennai'); 1 row created.

SQL> insert into sstud1 values ( 'anand','chennai'); 1 row created.

SQL> insert into sstud1 values ( 'kumar','chennai'); 1 row created.

SQL> insert into sstud1 values ( 'ravi','chennai'); 1 row created.

SQL> select \* from sstud1;

SNAME PLACE



prajan chennai

anand chennai

kumar chennai

ravi chennai

TO CREATE SSTUD2 TABLE

SQL> create table sstud2 ( sname varchar2(20), dept varchar2(10), marks number(10)); Table created.

SQL> insert into sstud2 values ('prajan','cse',700); 1 row created.

SQL> insert into sstud2 values ('anand','it',650); 1 row created.

SQL> insert into sstud2 values ('vasu','cse',680); 1 row created.

SQL> insert into sstud2 values ('ravi','it',600); 1 row created.

SQL> select \* from sstud2;

SNAME DEPT MARKS

| prajan | cse | 700 |
| --- | --- | --- |
| anand | it | 650 |
| vasu | cse | 680 |
| ravi | it | 600 |

## VIEWS

Views are created for the following reasons,

* Data simplicity
* To provide data security
* Structural simplicity (because view contains only limited number of rows and colmns)

##### TYPES OF VIEWS

* Updatable views – Allow data manipulation
* Read only views – Do not allow data manipulation

##### TO CREATE THE TABLE ‘FVIEWS’

SQL> create table fviews( name varchar2(20),no number(5), sal number(5), dno number(5)); Table created.

SQL> insert into fviews values('xxx',1,19000,11); 1 row created.

SQL> insert into fviews values('aaa',2,19000,12); 1 row created.

SQL> insert into fviews values('yyy',3,40000,13); 1 row created.

SQL> select \* from fviews;

NAME NO SAL DNO

| xxx | 1 | 19000 | 11 |
| --- | --- | --- | --- |
| aaa | 2 | 19000 | 12 |
| yyy | 3 | 40000 | 13 |

##### TO CREATE THE TABLE ‘DVIEWS’

SQL> create table dviews( dno number(5), dname varchar2(20)); Table created.

SQL> insert into dviews values(11,'x'); 1 row created.

SQL> insert into dviews values(12,'y'); 1 row created.

SQL> select \* from dviews;

DNO DNAME



1. x
2. y

##### CREATING THE VIEW ‘SVIEW’ ON ‘FVIEWS’ TABLE

SQL> create view sview as select name,no,sal,dno from fviews where dno=11; View created.

SQL> select \* from sview;

| NAME |  | NO |  | SAL |  | DNO |
| --- | --- | --- | --- | --- | --- | --- |
| xxx |  | 1 |  | 19000 |  | 11 |

##### UPDATES MADE ON THE VIEW ARE REFLECTED ONLY ON THE TABLE WHEN THE STRUTURE OF THE TABLE AND THE VIEW ARE NOT SIMILAR -- PROOF

SQL> insert into sview values ('zzz',4,20000,14); 1 row created.

SQL> select \* from sview;

NAME NO SAL DNO



xxx 1 19000 11

SQL> select \* from fviews;

| NAME | NO | SAL |  | DNO |
| --- | --- | --- | --- | --- |
| xxx | 1 | 19000 |  | 11 |
| aaa | 2 | 19000 |  | 12 |
| yyy | 3 | 40000 |  | 13 |
| zzz | 4 | 20000 |  | 14 |

##### UPDATES MADE ON THE VIEW ARE REFLECTED ON BOTH THE VIEW AND THE TABLE WHEN THE STRUTURE OF THE TABLE AND THE VIEW ARE SIMILAR – PROOF

**CREATING A VIEW ‘IVIEW’ FOR THE TABLE ‘FVIEWS’**

SQL> create view iview as select \* from fviews; View created.

SQL> select \* from iview;

| NAME | NO | | SAL | DNO |
| --- | --- | --- | --- | --- |
| xxx | 1 | 19000 | 11 | |
| aaa | 2 | 19000 | 12 | |
| yyy | 3 | 40000 | 13 | |
| zzz | 4 | 20000 | 14 | |

##### PERFORMING UPDATE OPERATION

SQL> insert into iview values ('bbb',5,30000,15); 1 row created.

SQL> select \* from iview;

NAME NO SAL DNO



| xxx | 1 | 19000 | 11 |
| --- | --- | --- | --- |
| bbb | 5 | 30000 | 15 |

SQL> select \* from fviews;

NAME NO SAL DNO

| xxx | 1 | 19000 | 11 |
| --- | --- | --- | --- |
| aaa | 2 | 19000 | 12 |
| yyy | 3 | 40000 | 13 |
| zzz | 4 | 20000 | 14 |
| bbb | 5 | 30000 | 15 |

##### CREATE A NEW VIEW ‘SSVIEW’ AND DROP THE VIEW

SQL> create view ssview( cusname,id) as select name, no from fviews where dno=12; View created.

SQL> select \* from ssview;

CUSNAME ID



aaa 2

SQL> drop view ssview; View dropped.

##### TO CREATE A VIEW ‘COMBO’ USING BOTH THE TABLES ‘FVIEWS’ AND ‘DVIEWS’

SQL> create view combo as select name,no,sal,dviews.dno,dname from fviews,dviews where fviews.dno=dviews.dno;

View created.

SQL> select \* from combo;

NAME NO SAL DNO DNAME

| xxx | 1 | 19000 | 11 x |
| --- | --- | --- | --- |
| aaa | 2 | 19000 | 12 y |

##### TO PERFORM MANIPULATIONS ON THIS VIEW

SQL> insert into combo values('ccc',12,1000,13,'x');

insert into combo values('ccc',12,1000,13,'x')

\*

ERROR at line 1:

ORA-01779: cannot modify a column which maps to a non key-preserved table

This shows that when a view is created from two different tables no manipulations can be performed using that view and the above error is displayed.

## DATACONTROL LANGUAGE COMMANDS

EX NO: 5 DATE:

##### AIM

To study the various data language commands (DCL) and implement them on the database.

##### DESCRIPTION

The DCL language is used for controlling the access to the table and hence securing the database. This language is used to provide certain priveleges to a particular user. Priveleges are rights to be allocated. The privilege commands are namely,

* Grant
* Revoke

The various priveleges that can be granted or revoked are,

* Select
* Insert
* Delete
* Update
* References
* Execute
* All

GRANT COMMAND**:** It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND**:** Using this command , the DBA can revoke the granted database priveleges from the user.

##### SYNTAX

**GRANT COMMAND**

Grant < database\_priv [database\_priv…..] > to <user\_name> identified by <password> [,<password…..];

Grant <object\_priv> | All on <object> to <user | public> [ With Grant Option ];

##### REVOKE COMMAND

Revoke <database\_priv> from <user [, user ] >;

Revoke <object\_priv> on <object> from < user | public >;

<database\_priv> -- Specifies the system level priveleges to be granted to the users or roles. This includes create / alter / delete any object of the system.

<object\_priv> -- Specifies the actions such as alter / delete / insert / references / execute / select / update for tables.

<all> -- Indicates all the priveleges.

[ With Grant Option ] – Allows the recipient user to give further grants on the objects.

The priveleges can be granted to different users by specifying their names or to all users by using the “Public” option.

##### EXAMPLES

Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES” Their schemas are as follows ,

Departments ( dept \_no , dept\_ name , dept\_location ); Employees ( emp\_id , emp\_name , emp\_salary );

SQL> Grant all on employees to abcde; Grant succeeded.

SQL> Grant select , update , insert on departments to abcde with grant option; Grant succeeded.

SQL> Revoke all on employees from abcde; Revoke succeeded.

SQL> Revoke select , update , insert on departments from abcde; Revoke succeeded.

**PL /SQL**

PL/SQL is a completely portable, high-performance transaction-processing language.

PL/SQL provides a built-in, interpreted and OS independent programming environment.PL/SQL can also directly be called from the command-line **SQL\*Plus interface**.Direct call can also be made from external programming language calls to database.PL/SQL's general syntax is based on that of ADA and Pascal programming language.Apart from Oracle, PL/SQL is available in **TimesTen in-memory database** and **IBM DB2**.

## Features of PL/SQL

PL/SQL has the following features −

* PL/SQL is tightly integrated with SQL.
* It offers extensive error checking.
* It offers numerous data types.
* It offers a variety of programming structures.
* It supports structured programming through functions and procedures.
* It supports object-oriented programming.
* It supports the development of web applications and server pages.

## Advantages of PL/SQL

PL/SQL has the following advantages −

SQL is the standard database language and PL/SQL is strongly integrated with SQL. PL/SQL supports both static and dynamic SQL. Static SQL supports DML operations and transaction control from PL/SQL block. In Dynamic SQL, SQL allows embedding DDL statements in PL/SQL blocks.

PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.

PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.

PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.

Applications written in PL/SQL are fully portable.

PL/SQL provides a high security level.

PL/SQL provides access to predefined SQL packages.

PL/SQL provides support for Object-Oriented Programming.

PL/SQL provides support for developing Web Applications and Server Pages.

Sample Programme:

**DECLARE**

**x NUMBER := 100;**

**BEGIN**

**FOR i IN 1..10 LOOP**

**IF MOD(i,2) = 0 THEN**

**-- i is even**

**INSERT INTO temp VALUES (i, x, 'i is even');**

**ELSE**

**INSERT INTO temp VALUES (i, x, 'i is odd');**

**END IF;**

**x := x + 100;**

**END LOOP;**

**COMMIT;**

**END;**

**Triggers**

**Trigger** is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes. Triggers are used to specify certain integrity constraints and referential constraints that cannot be specified using the constraint mechanism of SQL.

In a **database management system (DBMS)**, a **trigger** is a powerful feature that allows you to automatically execute specific actions or statements when certain events occur in the

database. Let’s explore the syntax and usage of triggers:

**Trigger Structure**:

A trigger has the following structure:

CREATE TRIGGER trigger\_name

[BEFORE | AFTER] [INSERT | UPDATE | DELETE] ON table\_name

[FOR EACH ROW | FOR EACH COLUMN]

AS

BEGIN

-- Trigger action or statements

END;

The trigger\_name must be unique within the schema.You can specify whether the trigger fires **before** or **after** the specified event (insert, update, or delete).The table\_name indicates the table associated with the trigger.

**Example –**   
Suppose, we are adding a tuple to the ‘Donors’ table that is some person has donated blood. So, we can design a trigger that will automatically add the value of donated blood to the ‘Blood\_record’ table.

**Types of Triggers –**   
We can define 6 types of triggers for each table:

1. **AFTER INSERT** activated after data is inserted into the table.
2. **AFTER UPDATE:** activated after data in the table is modified.
3. **AFTER DELETE:** activated after data is deleted/removed from the table.
4. **BEFORE INSERT:** activated before data is inserted into the table.
5. **BEFORE UPDATE:** activated before data in the table is modified.
6. **BEFORE DELETE:** activated before data is deleted/removed from the table.

**Write a trigger to ensure that no employee of age less than 25 can be inserted in the database.**

delimiter $$

CREATE TRIGGER Check\_age BEFORE INSERT ON employee

FOR EACH ROW

BEGIN

IF NEW.age < 25 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'ERROR:

AGE MUST BE ATLEAST 25 YEARS!';

END IF;

END; $$kkkj

delimiter;

**Explanation:** Whenever we want to insert any tuple to table ’employee’, then before inserting this tuple to the table, trigger named ‘Check\_age’ will be executed. This trigger will check the age attribute. If it is greater than 25 then this tuple will be inserted into the tuple otherwise an error message will be printed stating “ERROR: AGE MUST BE ATLEAST 25 YEARS!”

**Cursor**

A cursor acts as a **pointer** to a specific row within the result set of an SQL query.It enables you to sequentially access and process each row of data returned by a query.Cursors are particularly useful when you need to perform operations on individual rows or iterate through query results.

**Types of Cursors**:

**Implicit Cursors**:

Automatically created by the DBMS for **single-row queries** (queries that return only one row).

No explicit declaration is required.

Implicit cursors are associated with DML statements (INSERT, UPDATE, DELETE) and provide information about the execution of these statements.

Common cursor attributes include:

%FOUND: Indicates whether the last query returned any rows.

%NOTFOUND: Opposite of %FOUND.

%ISOPEN: Indicates whether the cursor is open.

%ROWCOUNT: Returns the number of rows affected by the last DML statement

**Explicit Cursors**:

Defined explicitly by the programmer.

Used for **multi-row queries** (queries that return multiple rows).

Explicit cursors provide more control and flexibility.

Steps involved in using explicit cursors:

**Cursor Declaration**: Define the cursor in the declaration section of your PL/SQL block.

**Cursor Opening**: Allocate memory for the cursor using the OPEN statement.

**Cursor Fetching**: Retrieve data from the cursor using the FETCH statement.

**Cursor Closing**: Release the allocated memory using the CLOSE statement.

**Example of an Explicit Cursor**:

**SQL**

DECLARE

empId employees.EMPLOYEEID%TYPE;

empName employees.EMPLOYEENAME%TYPE;

empCity employees.EMPLOYEECITY%TYPE;

CURSOR c\_employees IS

SELECT EMPLOYEEID, EMPLOYEENAME, EMPLOYEECITY

FROM employees;

BEGIN

OPEN c\_employees;

LOOP

FETCH c\_employees INTO empId, empName, empCity;

EXIT WHEN c\_employees%NOTFOUND;

dbms\_output.put\_line(empId || ' ' || empName || ' ' || empCity);

END LOOP;

CLOSE c\_employees;END;/

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

In this example, we declare an explicit cursor c\_employees to retrieve employee data from the employees table,We fetch each row and display the employee ID, name, and city.

**Queries in PL/SQL**

The SELECT statement retrieves data from one or more tables. For example:

DECLARE

emp\_name VARCHAR2(50);

BEGIN

SELECT employee\_name INTO emp\_name

FROM employees

WHERE employee\_id = 100;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_name);

END;

In this example, we retrieve the name of an employee with employee\_id 100 from the employees table.

**Queries using in Trigger**

Certainly! Let’s explore some examples of how you can use **triggers** in SQL. Triggers are procedural code blocks that automatically execute in response to specific events on a specified table. They play a crucial role in maintaining data integrity and ensuring accuracy. Here are some common scenarios and examples:

**Before Insert Trigger**:

Suppose you want to ensure that no employee younger than 25 years is inserted into the database. You can create a trigger like this:

CREATE TRIGGER CheckAgeBeforeInsert

BEFORE INSERT ON employeesFOR EACH ROW

BEGIN

IF NEW.age < 25 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'ERROR: AGE MUST BE AT LEAST 25 YEARS!';

END IF;

END;

In this example, the trigger checks the age of the employee before inserting a new record.

**After Update Trigger**:

Let’s say you want to log changes made to the orders table. You can create an after-update trigger like this:

CREATE TRIGGER LogOrderChanges

AFTER UPDATE ON ordersFOR EACH ROW

BEGIN

INSERT INTO order\_logs (order\_id, action, timestamp)

VALUES (NEW.order\_id, 'Updated', NOW());

END;

In this case, the trigger logs the update action along with the timestamp.

**Before Delete Trigger**:

Suppose you want to prevent deletion of records from the customers table if they have pending orders. You can create a trigger like this:

CREATE TRIGGER PreventCustomerDeletion

BEFORE DELETE ON customersFOR EACH ROW

BEGIN

DECLARE order\_count INT;

SELECT COUNT(\*) INTO order\_count

FROM orders

WHERE customer\_id = OLD.customer\_id;

IF order\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'ERROR: Cannot delete customer with pending orders!';

END IF;

END;

This trigger ensures that customers with pending orders cannot be deleted.

**SQL USING CURSOR**

A cursor is an object that enables traversal over the rows of a result set.

It allows you to process individual rows returned by a query.

Here’s the basic structure for using a cursor in SQL Server:

DECLARE @product\_name VARCHAR(MAX), @list\_price DECIMAL;

DECLARE cursor\_product CURSOR FOR

SELECT product\_name, list\_price

FROM production.products;

OPEN cursor\_product;

FETCH NEXT FROM cursor\_product INTO @product\_name, @list\_price;

WHILE @@FETCH\_STATUS = 0BEGIN

PRINT @product\_name + CAST(@list\_price AS VARCHAR);

FETCH NEXT FROM cursor\_product INTO @product\_name, @list\_price;END;

CLOSE cursor\_product;DEALLOCATE cursor\_product;

**Explanation**:

this example:

We declare two variables (@product\_name and @list\_price) to hold product information. We create a cursor named cursor\_product that retrieves data from the production.products table. We open the cursor using OPEN cursor\_product.

We fetch each row from the cursor into the variables. The WHILE loop processes all rows. Finally, we close and deallocate the cursor.

**Customize**:

You can replace the query inside the cursor with your own SQL statement based on your requirements.