Oracle:

Oracle is one of the powerful RDBMS products that provide efficient solutions for database applications. Oracle is the product of Oracle Corporation which was founded by LAWRENCE ELLISION in 1977. The first commercial product of oracle was delivered in 1970. The first version of oracle 2.0 was written in assembly language. Nowadays commonly used versions of oracle are ORACLE 8, 8i and 9i, Oracle 8 and onwards provide tremendous increase in performance, features and functionality.

FEATURES OF ORACLE

- Client/Server Architecture
- Large database and Space Management
- Concurrent Processing
- High transaction processing performance
- High Availability
- Many concurrent database users
- Controlled availability
- Openness industry standards
- Manageable security
- Database enforced integrity
- Distributed systems
- Portability
- Compatibility

ORACLE SERVER TOOL

Oracle is a company that produces most widely used server based multi-user RDBMS. Oracle server is a program installed on server hard-disk drive. This program must be loaded in RAM to that it can process the user requests. Oracle server takes care of following functions. Oracle server tools are also called as back end. Functions of server tool:

- Updates the data
- Retrieves the data sharing
- Manages the data sharing
- Accepts the query statements PL/SQL and SQL
- Enforce the transaction consistency

ORACLE CLIENT TOOL

Once Oracle engine is loaded into sever memory user would have to log in to engine in order to work done. Client tools are more useful in commercial application development. It provides facilities to work on database objects. These are more commonly used in commercial applications. Oracle client tools are also called front end.

DDL – DATA DEFINATION LANGUAGE

SQL sentences that are used to create these objects are called DDL's or Data Definition Language. The SQL provides various commands for defining relation schemas, deleting relations, creating indexes and modify relation schemas. DDL is part of SQL which helps a user in defining the data structures into the database. Following are the various DDL commands are:

- Alter table, create table & drop table
- Create index & drop index
- Create view & drop view

DML – DATA MANIPULATION LANGUAGE

The SQL sentences used to manipulate data within these objects are called DML's or Data Manipulation Language. It is language that enables users to access or manipulate data as organized by appropriate data model. By data manipulation we have

- Retrieval of information stored in database.
- Insertion of new information into database.
- Deletion of information from database.
- Modification of data stored in database.

Two types of DML are

- Procedural DML
- Non-procedural DML

Following are DML commands are

- Select
- Update
- Delete
- Insert

DCL – DATA CONTROL LANGUAGE

The SQL sentences, which are used to control the behavior of these objects, are called DCL's or Data Control Language. It is language used to control data and access to the database. Following are some DCL commands are :

- Commit
- Rollback
- Save point
- Set transaction

DATA TYPES OF SQL

- **CHAR :** This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of characters (i.e. the size) this data type can hold is 255 characters. Syntax is CHAR(SIZE)

  Example is CHAR (20)

- **VARCHAR** : This data type is used to store variable length alphanumeric data. The maximum this data type can hold is 4000 characters. One difference between this data type and the CHAR data type is ORACLE compares VARCHAR values using non-padded comparison semantics i.e. the inserted values will not be padded with spaces. Syntax is VARCHAR(SIZE)

  Example is VARCHAR (20) OR VARCHAR2 (20)

- **NUMBER** : The NUMBER data type is used to store numbers (fixed or floating point). Numbers of virtually any magnitude maybe stored up to 38 digits of precision. Numbers as large as 9.99 * 10 to the power of 124, i.e. followed by 125 zeros can be stored. The precision, (P), determines the maximum length of the data, whereas the scale, (S), determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted values are stored with their original precision up to the maximum of 38 digits.

  Syntax : NUMBER (P, S) Example is NUMBER (10, 2)

- **DATE** : This data type is used to represent data and time. The standard format id DD-MM-YY as in 13-JUL-85. To enter dates other than the standard format, use the appropriate functions. Date Time stores date in the 24-hour format. By default, the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day of the current month.

  Syntax : DATE

1. Create table

Relations can be described in the form of tables, and in SQL :

create table table_name(column_name datatype, …);

Home > SQL > SQL Commands

Autocommit Display 10

```
create table emp(id number(6), name varchar(10),salary number(6),address varchar(10))
```

Results Explain Describe Saved SQL History

```
Table created.
```

0.05 seconds

2. Insert

To enter the data, we can do manually by entering each data input with proper input values. Null can also be written if the value is unknown or if do not exist.

insert into table_name values(data…) ;

Autocommit Display 10

```
create table emp(id number(6), name varchar(10),salary number(6),address varchar(10))
insert into emp values(11,'ankit',56000,'rajasthan')
insert into emp values(35,'vijay',6000,'punjab')
```

Results Explain Describe Saved SQL History

```
1 row(s) inserted.
```

0.00 seconds

3. Insert with form means

We can add the tuples by entering each and every input data according to the form means. One of the method is :

insert into student values(:column_name, …) ;

Autocommit Display 10

```
create table emp(id number(6), name varchar(10),salary number(6),address varchar(10))
insert into emp values(11,'ankit',56000,'rajasthan')
insert into emp values(35,'vijay',6000,'punjab')
insert into emp values(:id,:name,:salary,:address)
```

Results Explain Describe Saved SQL History

```
1 row(s) inserted.
```

0.00 seconds

4. Describe the table

We can describe the basic elements of the table i.e. column-properties, primary-key,etc.

desc table_name;

Autocommit Display 10

```
create table emp(id number(6), name varchar(10),salary number(6),address varchar(10))
insert into emp values(11,'ankit',56000,'rajasthan')
insert into emp values(35,'vijay',6000,'punjab')
insert into emp values(:id,:name,:salary,:address)
desc emp
```

Results Explain Describe Saved SQL History

Object Type **TABLE** Object **EMP**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMP	ID	Number	-	6	0	-	✓	-	-
	NAME	Varchar2	10	-	-	-	✓	-	-
	SALARY	Number	-	6	0	-	✓	-	-
	ADDRESS	Varchar2	10	-	-	-	✓	-	-

1 - 4

5. Describe the table content/tuples.

We can describe the tuples or, the data stored in these tables by:

Select * from table_name;

Autocommit Display 10

```
insert into emp values(:id,:name,:salary,:address)
desc emp
select * from emp
```

Results Explain Describe Saved SQL History

ID	NAME	SALARY	ADDRESS
11	ankit	56000	rajasthan
35	vijay	6000	punjab

2 rows returned in 0.01 seconds CSV Export

6. Add column

Due to several modification, a table may need to add any column. We can do this by this command :

Alter table table_name add column_name datatype;

Autocommit Display 10

```
desc emp
select * from emp
alter table emp add phone_no number(10);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

0.03 seconds

7. Modify table column datatype

The datatype of any column may need to change like from integer to floating numbers. So, this can also be done by altering the table as:

alter table table_name modify column_name new_datatype ;

Autocommit Display 10

```
desc emp
select * from emp
alter table emp add phone_no number(10);
alter table emp modify phone_no varchar(10);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

0.01 seconds

8. Drop column

Any column, that is no longer needed as the database can be dropped by altering the table as :

alter table table_name drop column column_name;

Autocommit Display 10

```
select * from emp
alter table emp add phone_no number(10);
alter table emp modify phone_no varchar(10);
alter table emp drop column phone_no;
```

Results Explain Describe Saved SQL History

```
Table dropped.
```

0.01 seconds

9. Change of column name in table

Column name can also be altered by using alter table command as :

alter table table_name rename column column_name to new_column_name ;

Autocommit Display 10

```
alter table emp add phone_no number(10);
alter table emp modify phone_no varchar(10);
alter table emp drop column phone_no;
alter table emp rename column id to emp_id;
```

Results Explain Describe Saved SQL History

```
Table altered.
```

0.01 seconds

10. Change of table name

Change of table_name is also possible by altering the table as :

alter table table_name rename to new_table_name ;

Home > SQL > SQL Commands

Autocommit Display 10

```
alter table emp modify phone_no varchar(10);
alter table emp drop column phone_no;
alter table emp rename column id to emp_id;
alter table emp rename to employee;
desc employee;
```

Results Explain Describe Saved SQL History

Object Type **TABLE** Object **EMPLOYEE**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEE	EMP_ID	Number	-	6	0	-	✓	-	-
	NAME	Varchar2	10	-	-	-	✓	-	-
	SALARY	Number	-	6	0	-	✓	-	-
	ADDRESS	Varchar2	10	-	-	-	✓	-	-

1 - 4

11. Add a primary key

A primary key is needed to uniquely identify the tuples. And hence, in the oracle, it can be done by altering the table as:

alter table table_name add primary key(column_name);

Home > SQL > SQL Commands

Autocommit Display 10

```
desc employee;

alter table employee add primary key(emp_id);
```

Results Explain Describe Saved SQL History

```
Table altered.
```

0.03 seconds

On describing the table, it can be seen as the primary key has turned 1 for the column the key has been enabled.

Home > SQL > SQL Commands

Autocommit Display 10

```
alter table employee add primary key(emp_id);
desc employee;
```

Results Explain Describe Saved SQL History

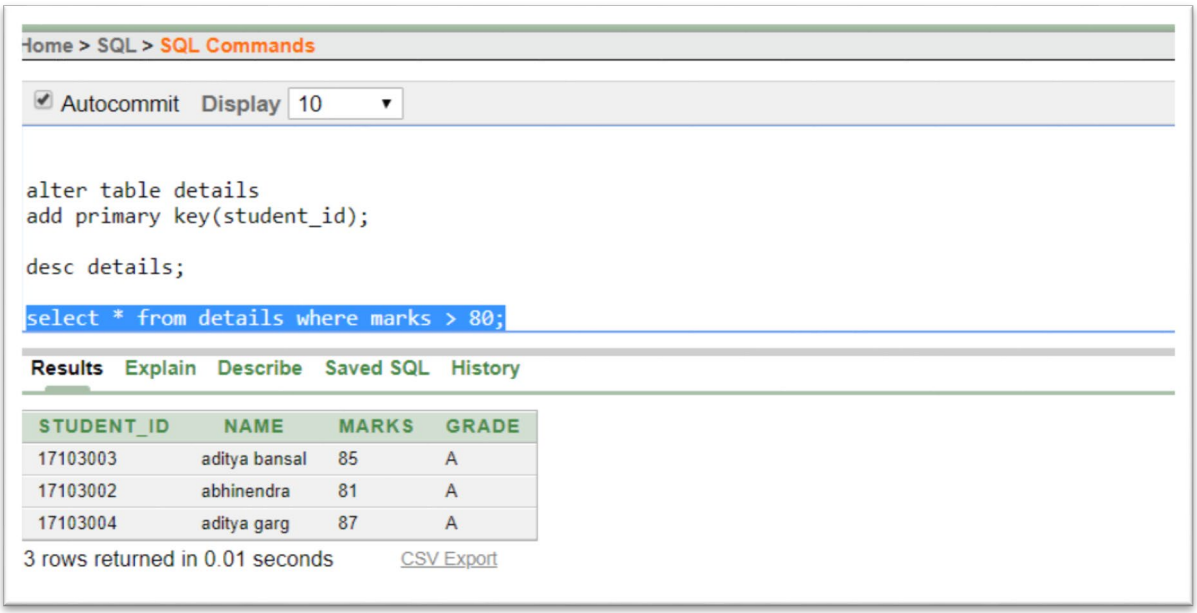
Object Type **TABLE** Object **EMPLOYEE**

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEE	EMP_ID	Number	-	6	0	1	-	-	-
	NAME	Varchar2	10	-	-	-	✓	-	-
	SALARY	Number	-	6	0	-	✓	-	-
	ADDRESS	Varchar2	10	-	-	-	✓	-	-

1 - 4

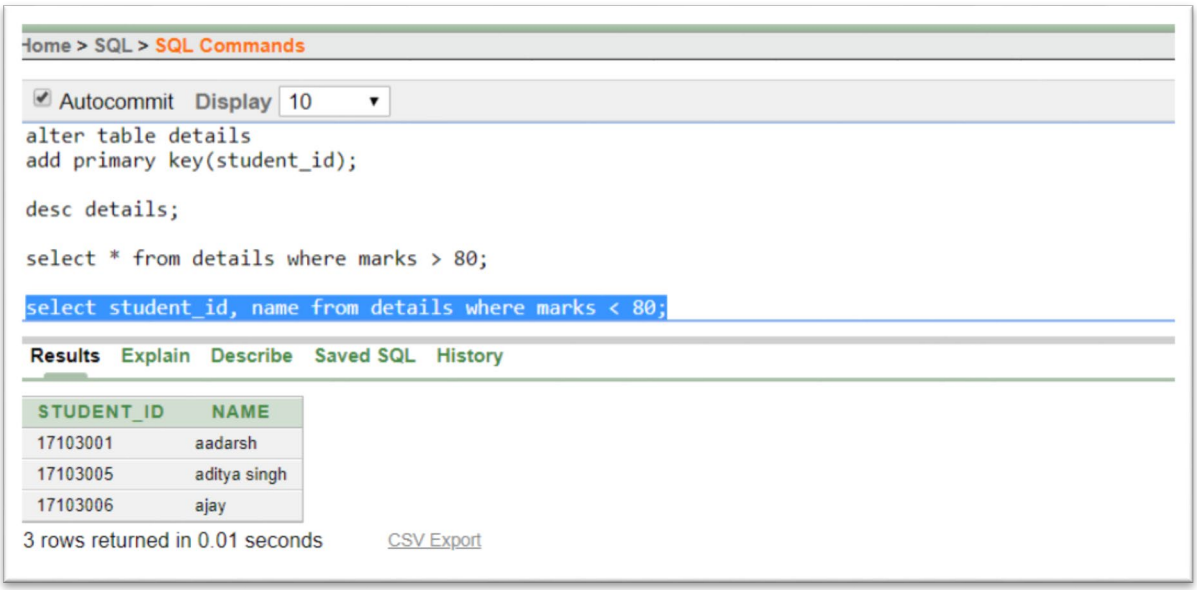
12. Select a particular tuple (Use of where clause)

Selecting all the tuples that manages a condition can be evaluated/expressed by where clause as :



13. Select a particular column (use of where clause)

We can also select some unique column instead of the whole length tuple and can be shown as a list for a view perspective hiding all the irrelevant information, using select as :



14. Arithmetic calculation on selection.

Any arithmetic changes in the values can be just shown without actually performing it in the original database. It can be achieved by :

Home > SQL > SQL Commands

Autocommit Display 10

```
desc details;

select * from details where marks > 80;

select student_id, name from details where marks < 80;

select student_id, name, marks*1.1 from details where marks>=85;
```

Results Explain Describe Saved SQL History

STUDENT_ID	NAME	MARKS*1.1
17103003	aditya bansal	93.5
17103004	aditya garg	95.7

2 rows returned in 0.00 seconds CSV Export

15. Update the values (using update)

Any value can be updated by using update and set keyword that is used as :

Update table_name
Set column_name = " … "
From table_name where " condition " ;

Home > SQL > SQL Commands

Autocommit Display 10

```
select * from details;

update details
set state='UP', marks=marks-2
where student_id < 17103004;

select * from details;
```

Results Explain Describe Saved SQL History

STUDENT_ID	NAME	MARKS	GRADE	STATE
17103001	aadarsh	77	B	UP
17103003	aditya bansal	83	A	UP
17103002	abhinendra	79	A	UP
17103004	aditya garg	87	A	-
17103005	aditya singh	78	B	-
17103006	ajay	75	B	-

6 rows returned in 0.00 seconds CSV Export

16. Select tuples having a particular column as NULL

We can select any tuples having a particular column with a NULL value. It can be done by using is NULL clause.

Select * from table_name where column_name is NULL;

Home > SQL > SQL Commands

Autocommit Display 10

```
update details
set state='UP', marks=marks-2
where student_id < 17103004;

select * from details;

select * from details where state is NULL;
```

Results Explain Describe Saved SQL History

STUDENT_ID	NAME	MARKS	GRADE	STATE
17103004	aditya garg	87	A	-
17103005	aditya singh	78	B	-
17103006	ajay	75	B	-

3 rows returned in 0.03 seconds CSV Export

17. Select tuples having a particular column as NOT NULL

We can select any tuple having a particular column with the 'is not NULL' clause.

Select * from table_name where column_name is NOT NULL;

Home > SQL > SQL Commands

Autocommit Display 10

```
set state='UP', marks=marks-2
where student_id < 17103004;

select * from details;

select * from details where state is NULL;

select * from details where state is NOT NULL;
```

Results Explain Describe Saved SQL History

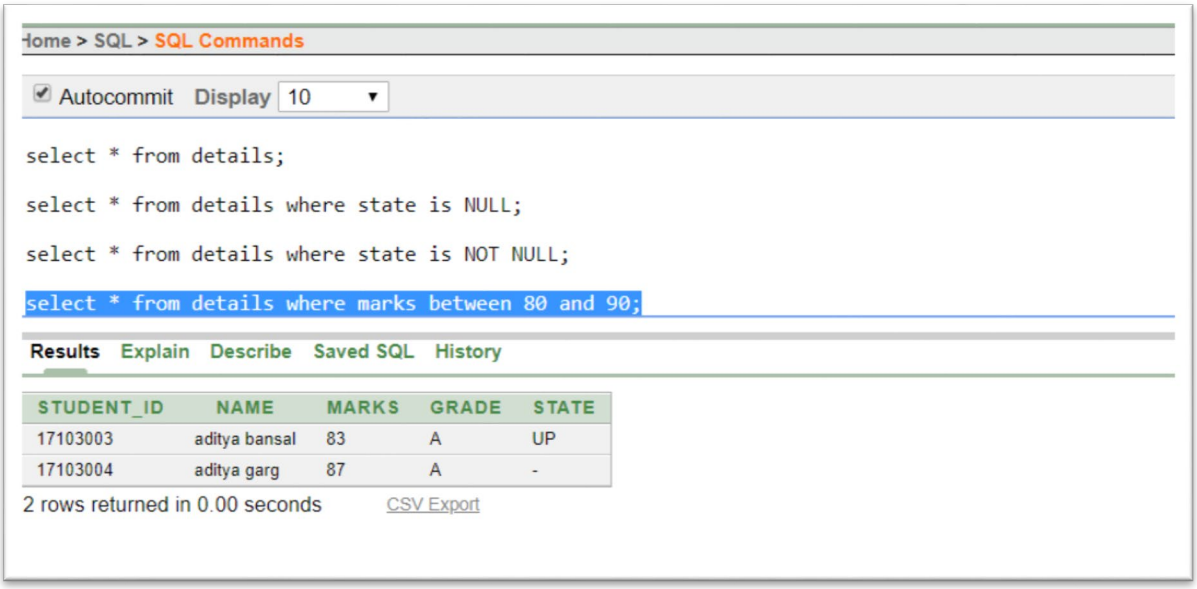
STUDENT_ID	NAME	MARKS	GRADE	STATE
17103001	aadarsh	77	B	UP
17103003	aditya bansal	83	A	UP
17103002	abhinendra	79	A	UP

3 rows returned in 0.00 seconds CSV Export

18. Use of Between Clause

To get the tuples having a particular value between the range, a between clause can be used which will show the values between the extremities.

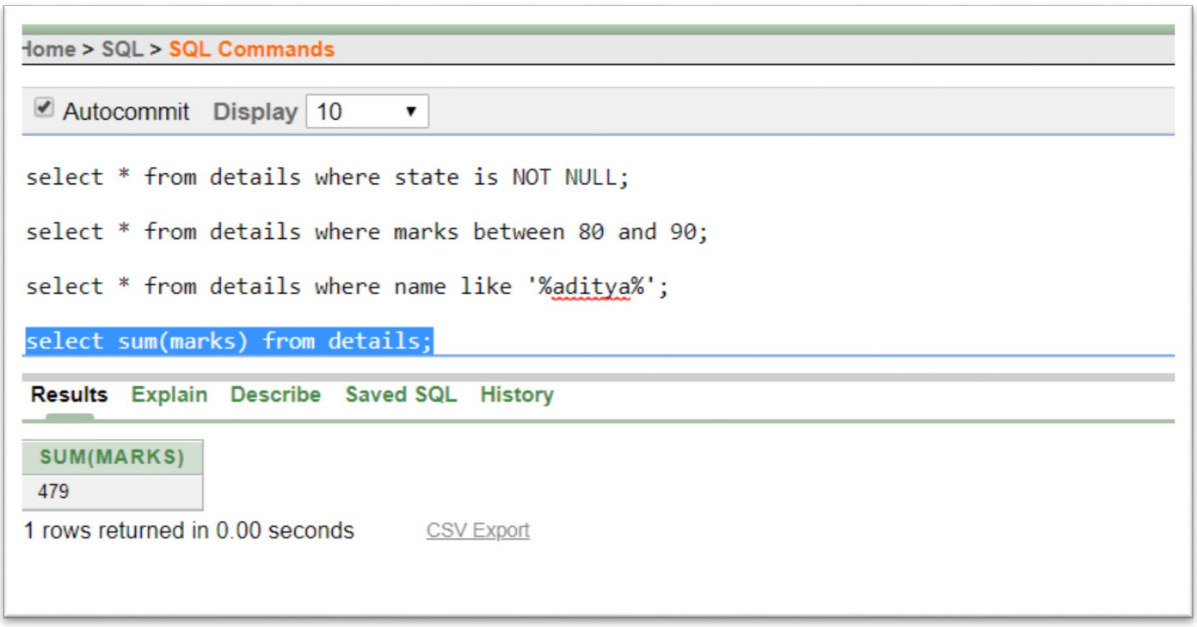
Select * from table_name where column_marks between min and max ;



19. Use of sum clause

Sum function is used to find the sum of all the numbers in a pre-defined column.

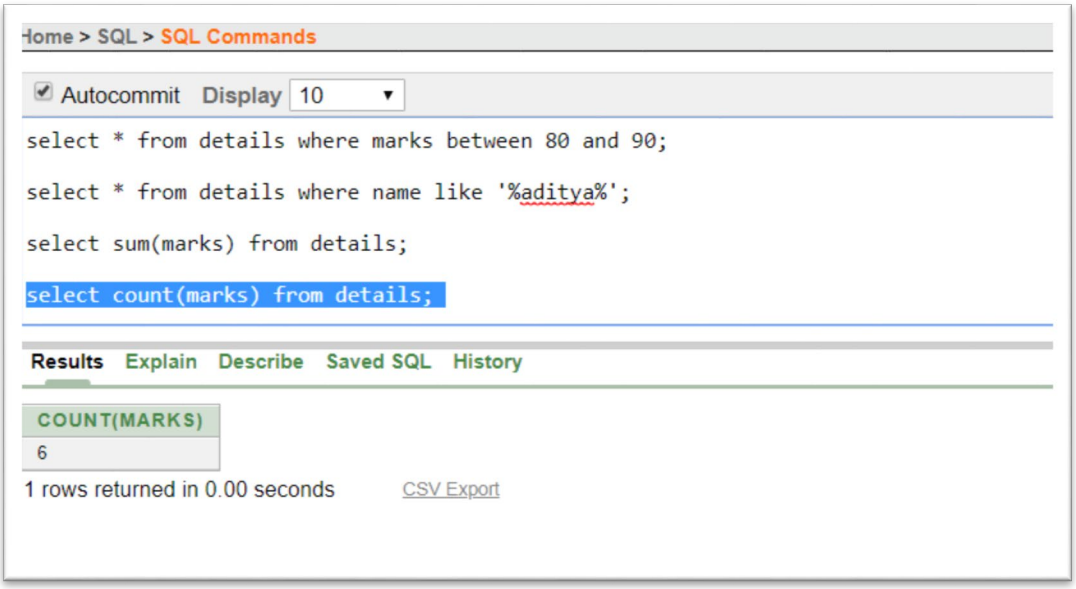
Select sum(column_name) from table_name;



20. Use of count clause

To count the number of entries/tuples, we can use count function.

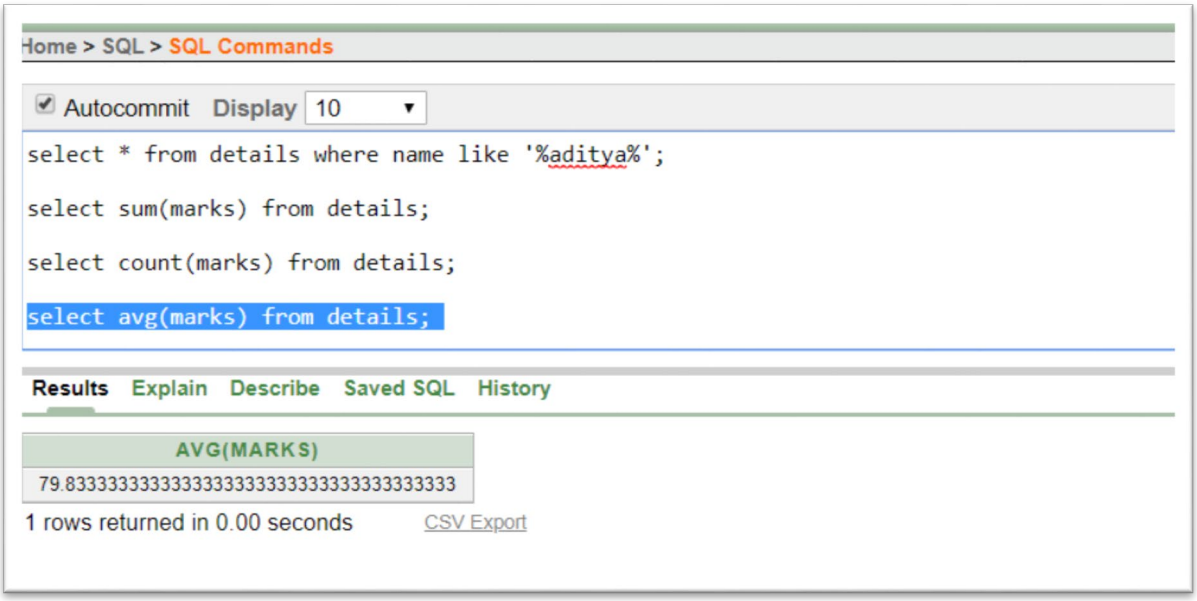
Select count(column_name) from table_name;



21. Use of average clause

The average of a particular column can be calculated as :

Select avg(column_name) from table_name;



22. Use of Maximum clause

The maximum of any particular number column can be given by this function as:

Select max(column_name) from table_name;

Home > SQL > SQL Commands

Autocommit Display 10

```
select sum(marks) from details;

select count(marks) from details;

select avg(marks) from details;

select name,marks from details where marks=( select max(marks) from details);
```

Results Explain Describe Saved SQL History

NAME	MARKS
aditya garg	87

1 rows returned in 0.00 seconds CSV Export

23. Use of Minimum clause

Similarly, minimum can be generated by using min clause.

Select min(column_name) from table_name;

Home > SQL > SQL Commands

Autocommit Display 10

```
select sum(marks) from details;

select count(marks) from details;

select avg(marks) from details;

select name,marks from details where marks > ( select min(marks) from details);
```

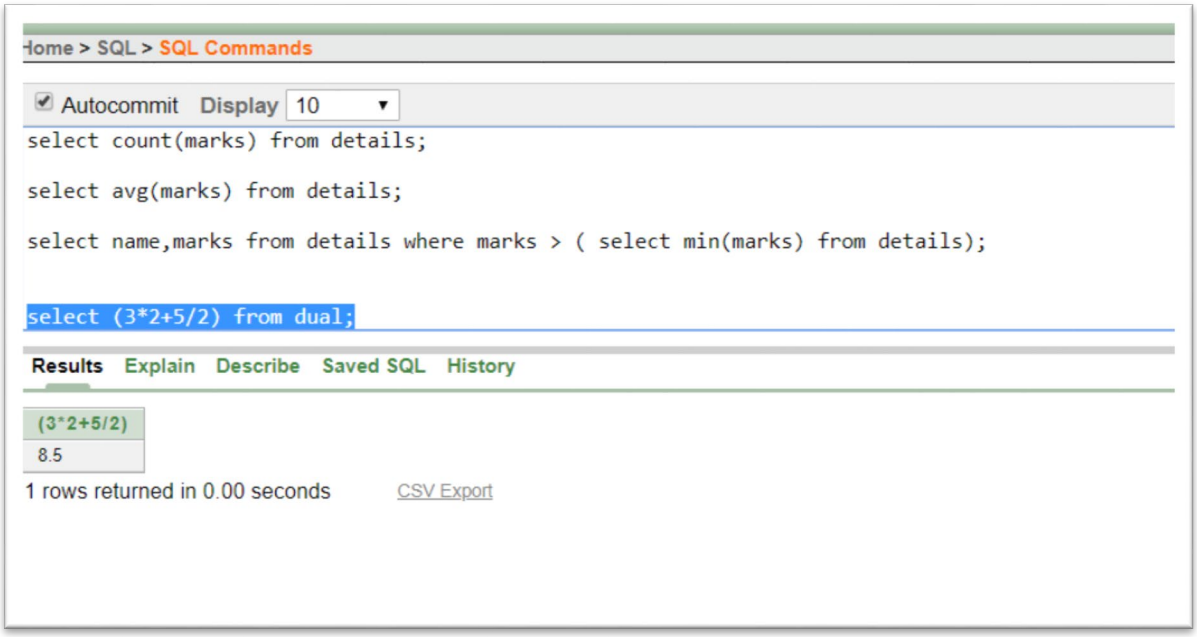
Results Explain Describe Saved SQL History

NAME	MARKS
aadarsh	77
aditya bansal	83
abhinendra	79
aditya garg	87
aditya singh	78

5 rows returned in 0.00 seconds CSV Export

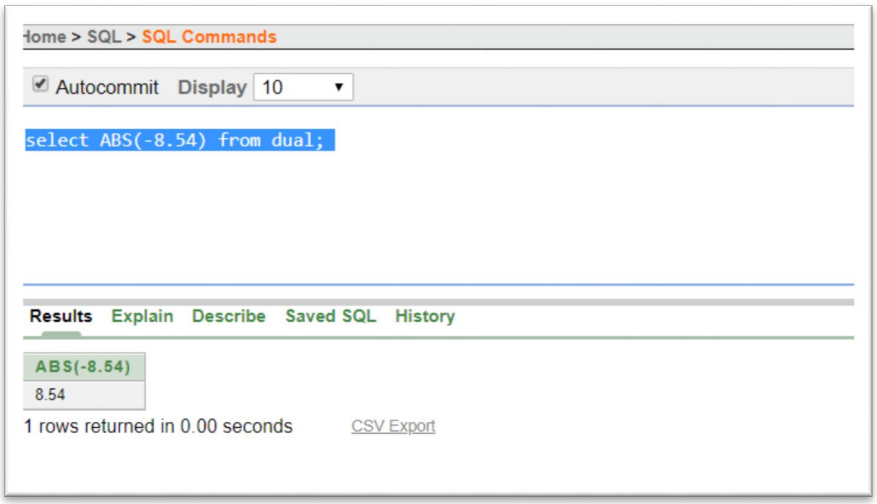
24. Use of dual

Dual is like a 1-row, 1-column table with a single record used for selecting when you're not actually interested in the data, but instead want the result of some system function in a select statement.

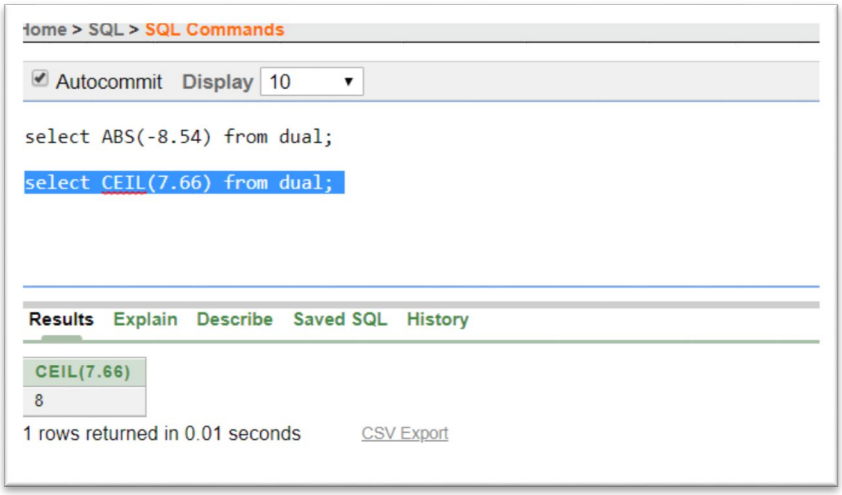


25. Use of number functions

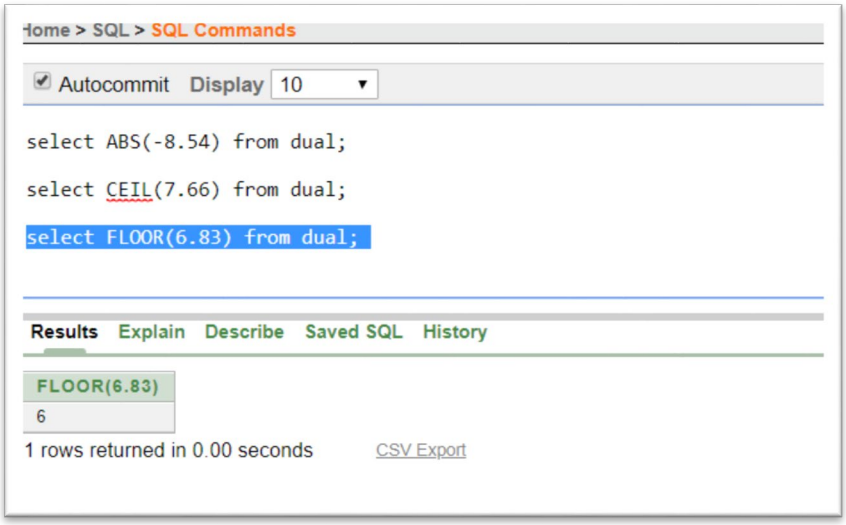
- **ABS** – Abstract function



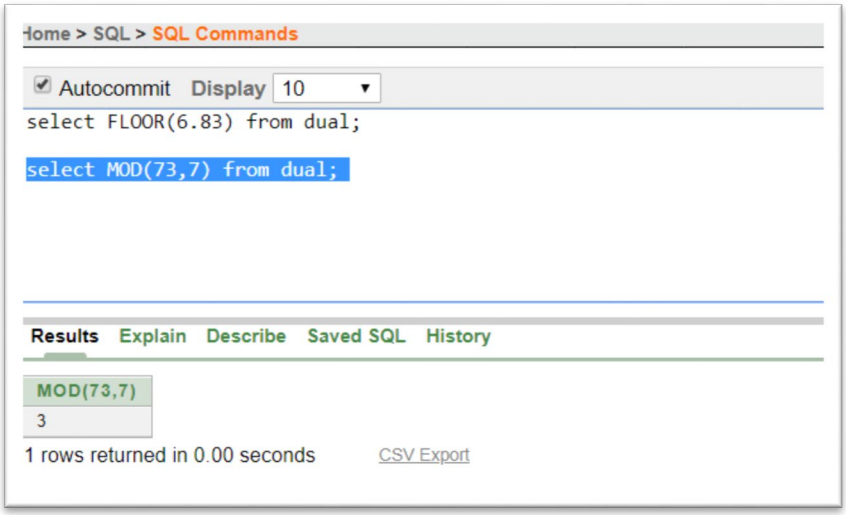
- **CEIL** – Ceiling function



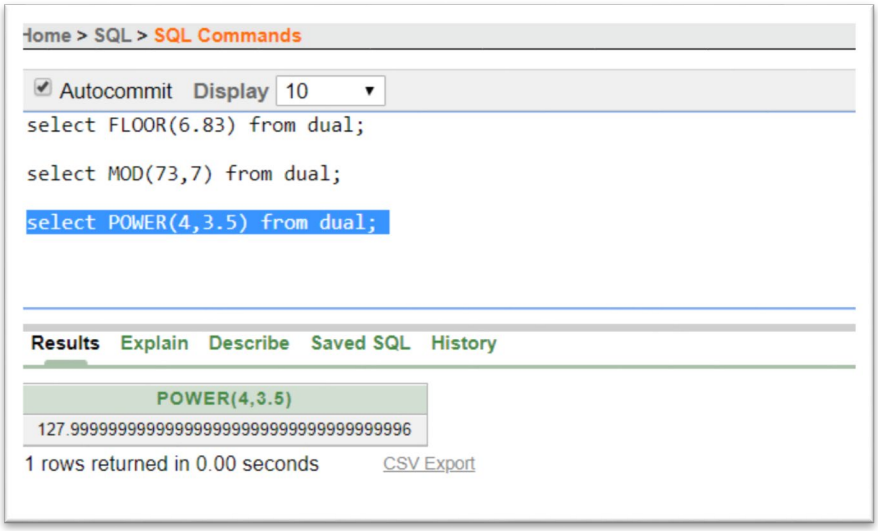
- **FLOOR** – Floor function



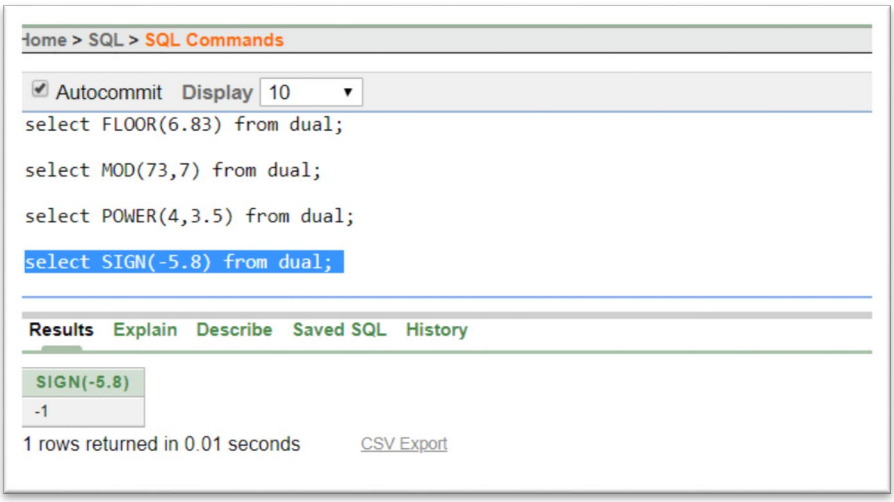
- **MOD** – Modulus function



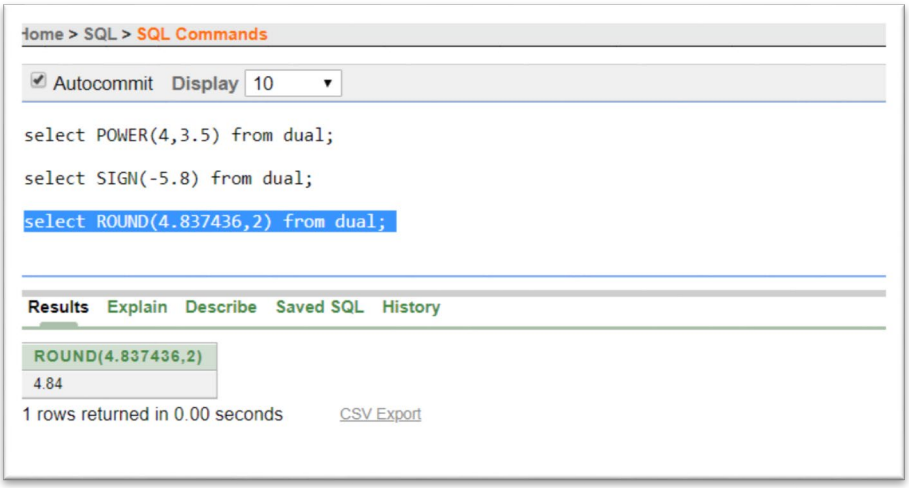
- **POWER** – power function



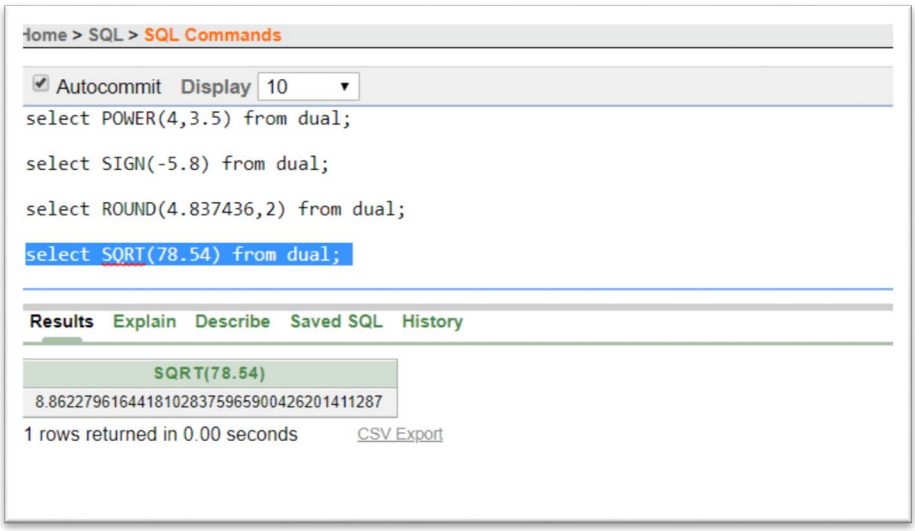
- **SIGN** – Sign function



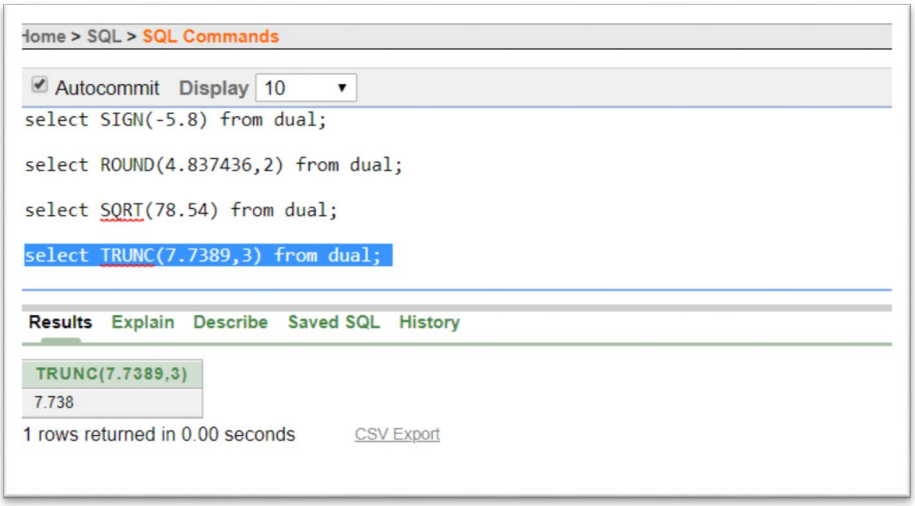
- **ROUND** – Round function



- **SQRT** – Square Root function

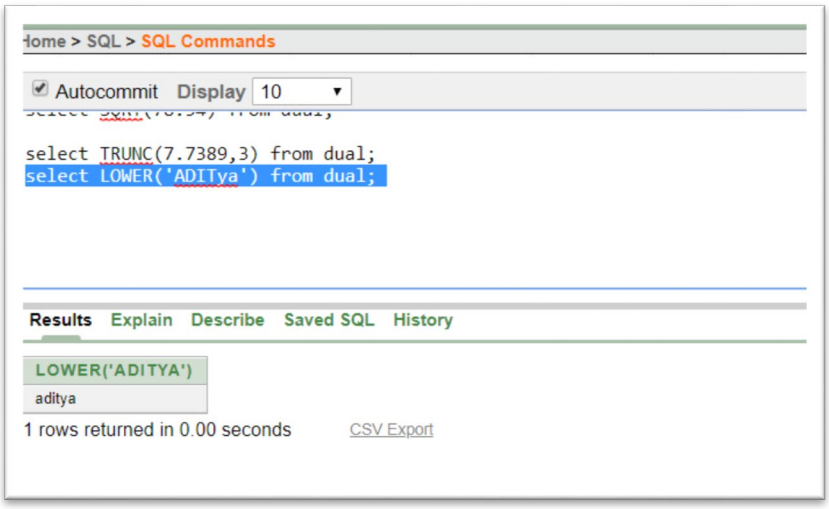


- **TRUNC** – Truncating function

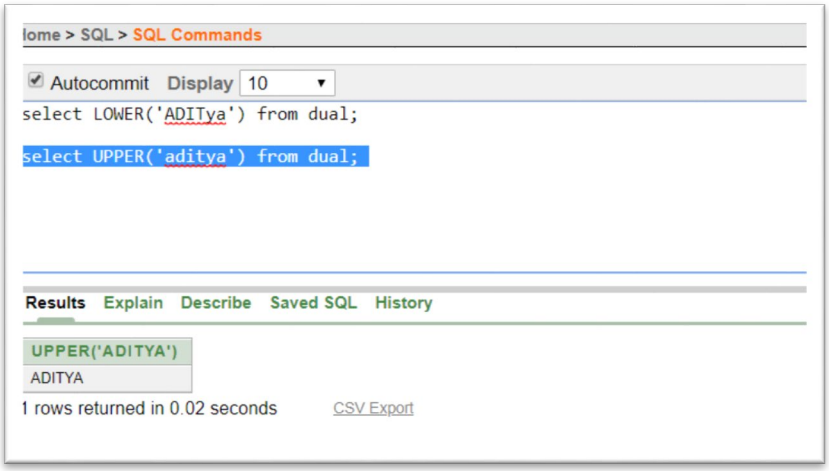


26. Use of Character function

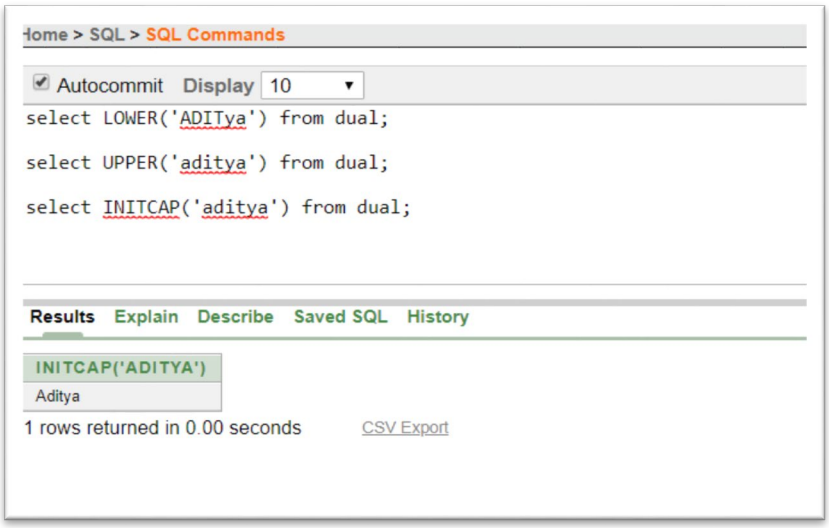
- **LOWER**



- **UPPER**



- **INITCAP**



- **CONCAT**

```
select CONCAT('aditya ','garg') from dual;
```

CONCAT('ADITYA','GARG')
aditya garg

1 rows returned in 0.00 seconds CSV Export

- **LENGTH**

```
select CONCAT('aditya ','garg') from dual;
select LENGTH('aditya garg') from dual;
```

LENGTH('ADITYAGARG')
11

1 rows returned in 0.00 seconds CSV Export

- **TRIM**

```
select CONCAT('aditya ','garg') from dual;
select LENGTH('aditya garg') from dual;
select TRIM('a' from 'aditya') from dual;
```

TRIM('A'FROM'ADITYA')
dity

1 rows returned in 0.00 seconds CSV Export

- **SUBSTR**

```
Autocommit Display 10
select CONCAT('aditya ','garg') from dual;
select LENGTH('aditya garg') from dual;
select TRIM('a' from 'aditya') from dual;
select SUBSTR('adityA GaRg', 2,8) from dual;
```

Results Explain Describe Saved SQL History

SUBSTR('ADITYAGARG',2,8)
dityA Ga

1 rows returned in 0.00 seconds CSV Export

- **INSTR**

```
Autocommit Display 10
select LENGTH('aditya garg') from dual;
select TRIM('a' from 'aditya') from dual;
select SUBSTR('adityA GaRg', 2,8) from dual;
select INSTR('aditya garg', 'a', 1,3) from dual;
```

Results Explain Describe Saved SQL History

INSTR('ADITYAGARG','A',1,3)
9

1 rows returned in 0.00 seconds CSV Export

- **LPAD**

```
Autocommit Display 10
select TRIM('a' from 'aditya') from dual;
select SUBSTR('adityA GaRg', 2,8) from dual;
select INSTR('aditya garg', 'a', 1,3) from dual;
select LPAD('adi',7,'*') from dual;
```

Results Explain Describe Saved SQL History

LPAD('ADI',7,'*')
****adi

1 rows returned in 0.00 seconds CSV Export

JOINS

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

There are different types of JOINS

- Inner join/natural join
- Left outer join
- Right outer join
- Outer join

And for the JOIN, there will be need of 2 relations as table or query format

So, here is the tables

Autocommit Display 10

```
create table salary(id number(6), salary number(6))
insert into salary values(:id,:salary)
select * from salary
```

Results Explain Describe Saved SQL History

ID	SALARY
11	2300
2	4000
4	3500
20	3200

4 rows returned in 0.00 seconds CSV Export

2nd table is

Home > SQL > SQL Commands

Autocommit Display 10

```
create table details(id number(6) primary key, name varchar(10))
insert into details values(:id,:name)
select * from details
```

Results Explain Describe Saved SQL History

ID	NAME
11	ankit
4	aditya
2	abhinendra
8	aman

4 rows returned in 0.00 seconds CSV Export

First table as SALARY describes relation between id and salary.

Second table as DETAILS describes relation between id and name.

1. Natural join/ inner join:

This join returns the records having common matching values.

Autocommit Display 10

```
select * from salary natural join details
```

Results Explain Describe Saved SQL History

ID	SALARY	NAME
11	2300	ankit
4	3500	aditya
2	4000	abhinendra

3 rows returned in 0.00 seconds CSV Export

2. Left outer join:

This join results in the all record of left table, with all/partial matched records of right table.

This may result some results as empty/NULL.

Autocommit Display 10

```
select * from salary natural left outer join details
```

Results Explain Describe Saved SQL History

ID	SALARY	NAME
11	2300	ankit
4	3500	aditya
2	4000	abhinendra
20	3200	-

4 rows returned in 0.02 seconds CSV Export

3. Right outer join:

This join results in the all record of right table, with all/partial matched records of left table.

This may result some results as empty/NULL.

Autocommit Display 10

```
select * from salary natural right outer join details
```

Results Explain Describe Saved SQL History

ID	SALARY	NAME
11	2300	ankit
2	4000	abhinendra
4	3500	aditya
8	-	aman

4 rows returned in 0.00 seconds CSV Export

4. Outer join:

This results in each and every query to be displayed, and does not depend whether it matches with its together table.

This will be the largest of all the table resulted.

Home > SQL > SQL Commands

Autocommit Display 10

```
select * from salary natural full outer join details
```

Results Explain Describe Saved SQL History

ID	SALARY	NAME
11	2300	ankit
4	3500	aditya
2	4000	abhinendra
20	3200	-
8	-	aman

5 rows returned in 0.00 seconds CSV Export

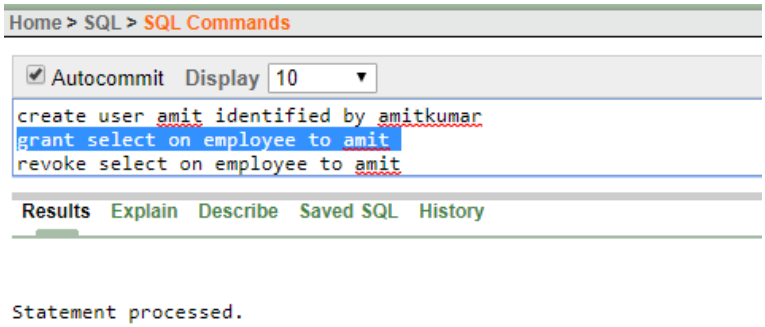
And here concludes the JOIN which can result another relation from given relations.

- ## DCL (Data Control Language) commands:

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

Examples of DCL commands:

1. **GRANT**-gives user's access privileges to database.



2. **REVOKE**-withdraw user's access privileges given by using the GRANT command.

- ## TCL (transaction Control Language) commands:

TCL commands deals with the transaction within the database

Examples of TCL commands:

1. **COMMIT**– commits a Transaction.
2. **ROLLBACK**– rollbacks a transaction in case of any error occurs.
3. **SAVEPOINT**–sets a savepoint within a transaction.

Before rollback:

Home > SQL > SQL Commands

Autocommit Display 10

```
create table depar1(dept_name char(10) primary key)
select*from depar1
insert into depar1 values('A')
insert into depar1 values('B')
commit
insert into depar1 values('C')
rollback
```

Results Explain Describe Saved SQL History

DEPT_NAME
A
B
C

after rollback

Home > SQL > SQL Commands

Autocommit Display 10

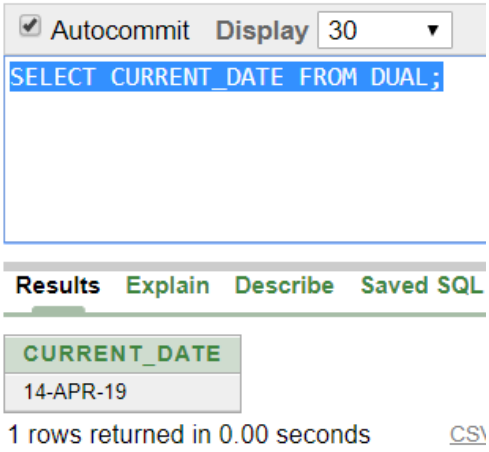
```
create table depar1(dept_name char(10) primary key)
select*from depar1
insert into depar1 values('A')
insert into depar1 values('B')
commit
insert into depar1 values('C')
rollback
```

Results Explain Describe Saved SQL History

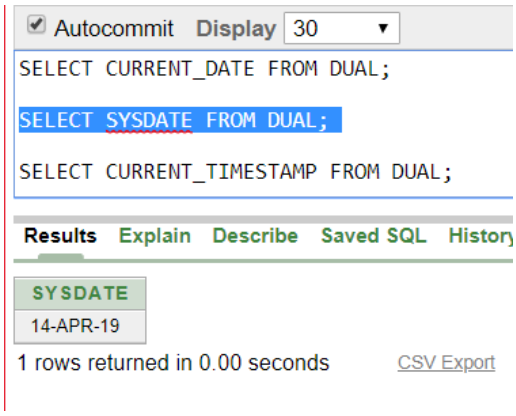
DEPT_NAME
A
B

TIME-DATE FUNCTIONS

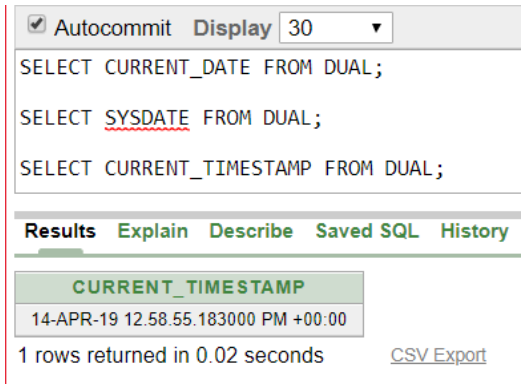
1. **CURRENT_DATE** : Returns current date.



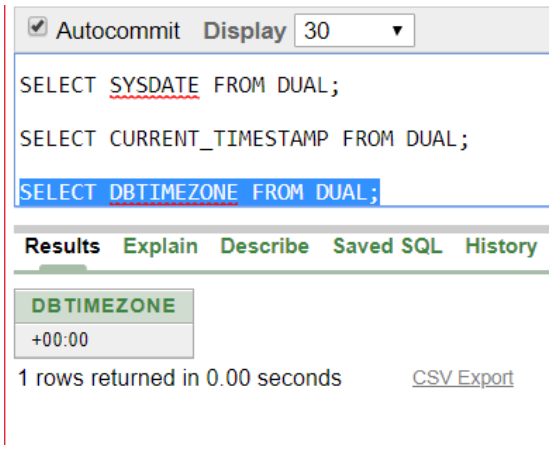
2. **SYSDATE** : Returns current date of the system.



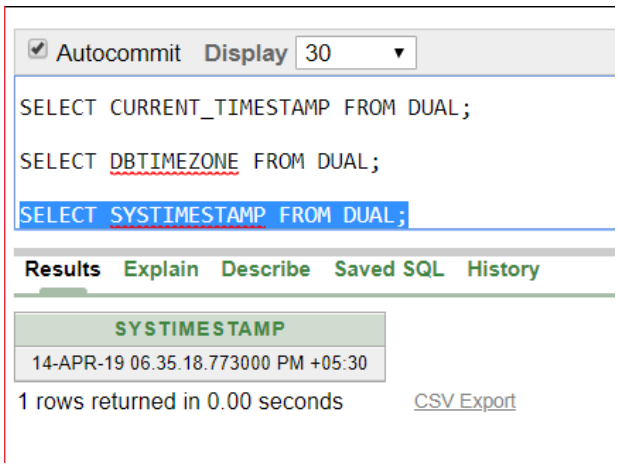
3. **CURRENT_TIMESTAMP :** Returns current time-stamp



4. **DBTIMEZONE** : returns the current database time zone



5. **SYSTIMESTAMP** : Returns the time-stamp of the system.



PL/SQL

PROGRAMMING IN SQL

PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL. PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

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.

SYNTAX:

It is a block-structured language. Hence,

1. Declarations :
   Starts with DECLARE. It is an optional section which defines all variables, cursors, subprograms, and all other elements to be used in the program.

2. Executable commands:
   This section is enclosed between BEGIN and END and it is a mandatory section. Consists of all the executable PL/SQL statements of the program. It should have atleast one executable line of code, which may be just a NULL command to indicate that nothing has been executed.

3. Exception handling:
   Starts with EXCEPTION. This optional section contains exception(s) that handle errors in the program.

Note: Identifiers are not case-sensitive**.**

Data-types:

1. Numeric
2. Character
3. Boolean
4. Date-time

Variable declaration → variable_name datatype := DEFAULT initial-value ;

- **CONDITIONS:**

1. **IF-THEN** :

```
IF condition THEN
        S;
END IF;
```

2. **IF-THEN-ELSE** :

```
IF condition THEN
        S1;
ELSE
        S2;
END IF;
```

- **CASE statement:**

```
CASE selector
    WHEN 'value1' THEN S1;
    WHEN 'value2' THEN S2;
    ...
    ...
    ELSE Sn;
END CASE;
```

- **LOOPS:**

1. **BASIC LOOP :**

```
LOOP
  Sequence of statements;
END LOOP;
```

2. **WHILE LOOP:**

```
WHILE condition LOOP
  Sequence of statements;
END LOOP;
```

3. **FOR LOOP:**

```
FOR counter IN initial_value . . final_value LOOP
    Sequence of statements;
END LOOP;
```

PROGRAMS:

1. SUM OF FIRST N NUMBERS:

Home > SQL > SQL Commands

Autocommit Display 10

```
declare
sum1 int:=0;
var1 int;
begin
for var1 in 1..10
loop
sum1:=sum1+var1;
end loop;
dbms_output.put_line(sum1);
end
```

Results Explain Describe Saved SQL History

```
55

Statement processed.
```

2. FACTORIAL OF A NUMBER:

Autocommit Display 10

```
declare
   var1 integer;
    var2 integer;
  begin
var1:=:var1;
var2:=1;
while var1>0
loop
var2:=var2*var1;
var1:=var1-1;
end loop;
dbms_output.put_line('factriol is  '||var2);
end;
```

Results Explain Describe Saved SQL History

```
factriol is  5040

Statement processed.
```

0.00 seconds

3. PRINT FIBONACCI SERIES:

Autocommit Display 10

```
    var2 integer;
   var3 integer;
    var integer;
 begin
var1:=0;
var2:=0;
var3:=1;
for var in 1..10
loop
dbms_output.put_line(' '||var2);
var1:=var3;
var3:=var3+var2;
var2:=var1;
end loop;
```

Results Explain Describe Saved SQL History

```
0
1
1
2
3
5
8
13
21
34

Statement processed.
```

0.00 seconds

- **PROCEDURE and FUNCTION:**

A subprogram is a program unit/module that performs a specific task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the calling program.

PL/SQL provides two kind of subprograms-

1. FUNCTIONS: These returns a single value; used to compute and answer.
2. PROCEDURE: These do not return a value; used to perform a subtask.

PARTS OF SUBPROGRAM-

1. Declaration
   Optional part. Contains declaration part of a subprogram. Does not start with DECLARE keyword.
2. Executable
   Mandatory part and contains statements that perform the designated part.
3. Exception handling
   It contains code that handle run-time errors.

SYNTAX FOR PROCEDURE:

CREATE OR REPLACE PROCEDURE procedure_name

[(parameter_name[IN | OUT | IN OUT] type[, , …])]

{ IS | AS }

BEGIN

<procedure_body>

END procedure_name;

SYNTAX FOR FUNCTION:

CREATE [OR REPLACE] FUNCTION function_name

[(parameter_name [IN | OUT | IN OUT] type[, . . .])]

RETURN return_datatype

{IS | AS}

BEGIN

< function_body>

END [function_name];

Explicit Procedure :

```
CREATE or REPLACE PROCEDURE greetings
AS
BEGIN
dbms_output.put_line('HELLO WORLD');
END greetings;
```

Results Explain Describe Saved SQL History

```
Procedure created.
```

```
BEGIN
greetings;
END
```

Results Explain Describe Saved SQL History

```
HELLO WORLD

Statement processed.
```

Implicit Procedure:

Home > SQL > SQL Commands

Autocommit Display 10

```
declare
a number;
b number;
c number;
procedure find_min(x in number,y in number,z out number) is
begin
if x<y then
z:=x;
else
z:=y;
end if;
end;
begin
a:=1;
b:=2;
find_min(a,b,c);
dbms_output.put_line(c);
end
```

Results Explain Describe Saved SQL History

```
1

Statement processed.
```

Explicit Functions:

Home > SQL > SQL Commands

Autocommit Display 10

```
create or replace function find_max(x in number,y in number)return number is z number;
begin
if x>y then
z:=x;
else
z:=y;
end if;
return z;
end;

declare
a number;
b number;
c number;
begin
a:=1;
b:=2;
c:=find_max(a,b);
dbms_output.put_line(c);
end
```

Results Explain Describe Saved SQL History

```
2

Statement processed.
```

Implicit Functions:

Home > SQL > SQL Commands

Autocommit Display 10

```
declare
a number;
b number;
c number;
function find_max(x in number,y in number)return number is z number;
begin
if x>y then
z:=x;
else
z:=y;
end if;
return z;
end;
begin
a:=1;
b:=2;
c:=find_max(a,b);
dbms_output.put_line(c);
end
```

Results Explain Describe Saved SQL History

```
2
```

- **PACKAGES:**

The specification is the interface to the package. It just DECLARES the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

Creating package :

```
create or replace package pcg as
procedure pro1(no in number, name out varchar);
function fun1(no in number) return varchar2;
end;
```

Results Explain Describe Saved SQL History

```
Package created.
```

Creating package body and executing :

```
create or replace package body pcg as
procedure pro1(no in number, name out varchar)
as
begin
select name into name from details
where no=student_id;
end;
function fun1(no in number) return varchar2 is
name varchar2(20);
begin
select name into name from details
where no = student_id;
return name;
end;
end;
```

Results Explain Describe Saved SQL History

```
Package Body created.
```

```
declare
a varchar(20):='aditya';
b varchar(20);
n number;
begin
pcg.pro1(17103004,a);
dbms_output.put_line(a);
b:=pcg.fun1(17103002);
dbms_output.put_line(b);
end;
```

Results Explain Describe Saved SQL History

```
aditya garg
abhinendra

Statement processed.
```

CURSORS

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT.

- **%FOUND** : Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
- **%NOTFOUND** : The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
- **%ISOPEN** : Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
- **%ROWCOUNT** : Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Example of a cursor :

```
declare
name emp1.e_name%type;
cursor abc is select e_name from emp1;
begin
open abc;
loop
fetch abc into name;
if abc%notfound then
exit;
else
dbms_output.put_line(name);
end if;
end loop;
dbms_output.put_line(abc%rowcount);
close abc;
end;
```

Results Explain Describe Saved SQL History

```
adarsh
aditya
chacha
aman
ankit
5

Statement processed.
```

0.12 seconds

TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers can be defined on the table, view, schema, or database with which the event is associated.

SYNTAX:

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER| INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE }

[OF col_name]

ON table_name

[REFERENCING OLD AS o NEW AS n]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declarative statements

BEGIN

Executable-statements

END;

```
CREATE OR REPLACE TRIGGER sal_change
BEFORE DELETE OR INSERT OR UPDATE ON employee
FOR EACH ROW
WHEN (new.E_ID > 0)
DECLARE
   sal_diff number(3);
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;

select * from employee;
create table employee(e_id number(3), name varchar(20), salary number(6));
insert into employee values(1,'aadash', 2500);
insert into employee values(2, 'abhinendra',1500);
insert into employee values(3, 'aditya', 2700);

insert into employee values(11, 'ankit', 1800);
update employee set salary=3000 where e_id = 3;
```

Results Explain Describe Saved SQL History

```
Old salary: 2700
New salary: 3000
Salary difference: 300

1 row(s) updated.
```