Database Management System Lab Practical File

SUBMITTED TO JASMINE MAM

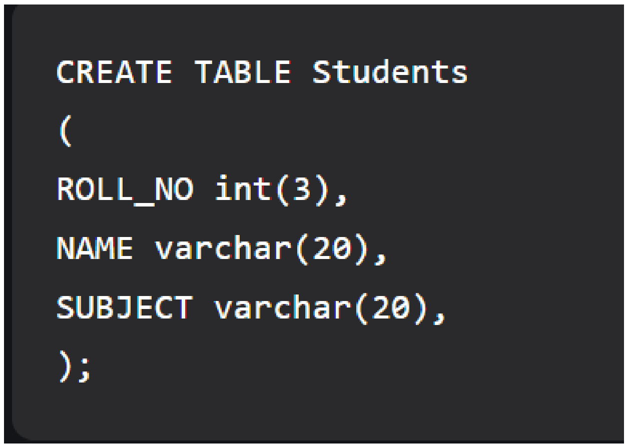
Gautam Jain | B Tech CSE | 1915312/1905785

Practical – 1

Aim: Write the queries for Data Definition (create, drop, alter and rename) and Data Manipulation Language (select, insert, update and delete).

Create:

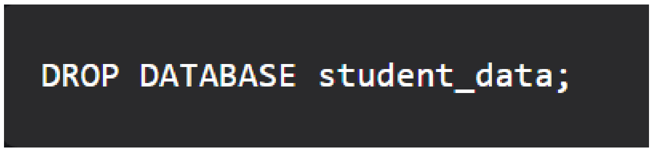
A command to Create a table in MySql.



```
CREATE TABLE Students
(
ROLL_NO int(3),
NAME varchar(20),
SUBJECT varchar(20),
);
```

Drop:

A command to Drop a table or database in MySql.



```
DROP DATABASE student_data;
```

Alter:

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

```
ALTER TABLE Customers
ADD Email varchar(255);
```

Rename:

The RENAME TABLE and ALTER TABLE syntax help in changing the name of the table.

```
RENAME old_table _name To new_table_name ;
```

Select:

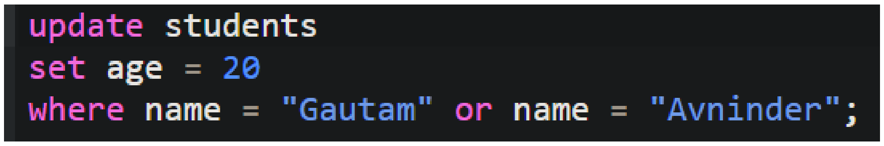
The SELECT statement is used to select data from a database.

```
SELECT column1, column2, ...
FROM table_name;
```

```
101|Gautam|9/10/2000|Ludhiana|20
121|Avninder|9/03/2001|Ludhiana|20
```

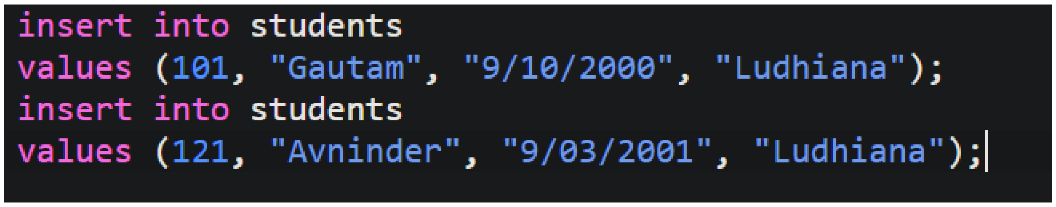
Update:

The UPDATE statement is used to modify the existing records in a table.



```sql
update students
set age = 20
where name = "Gautam" or name = "Avninder";
```

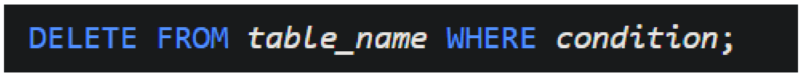
Insert:



```sql
insert into students
values (101, "Gautam", "9/10/2000", "Ludhiana");
insert into students
values (121, "Avninder", "9/03/2001", "Ludhiana");
```

Delete:

DELETE FROM attendance; It will DELETE all the rows from attendance table.



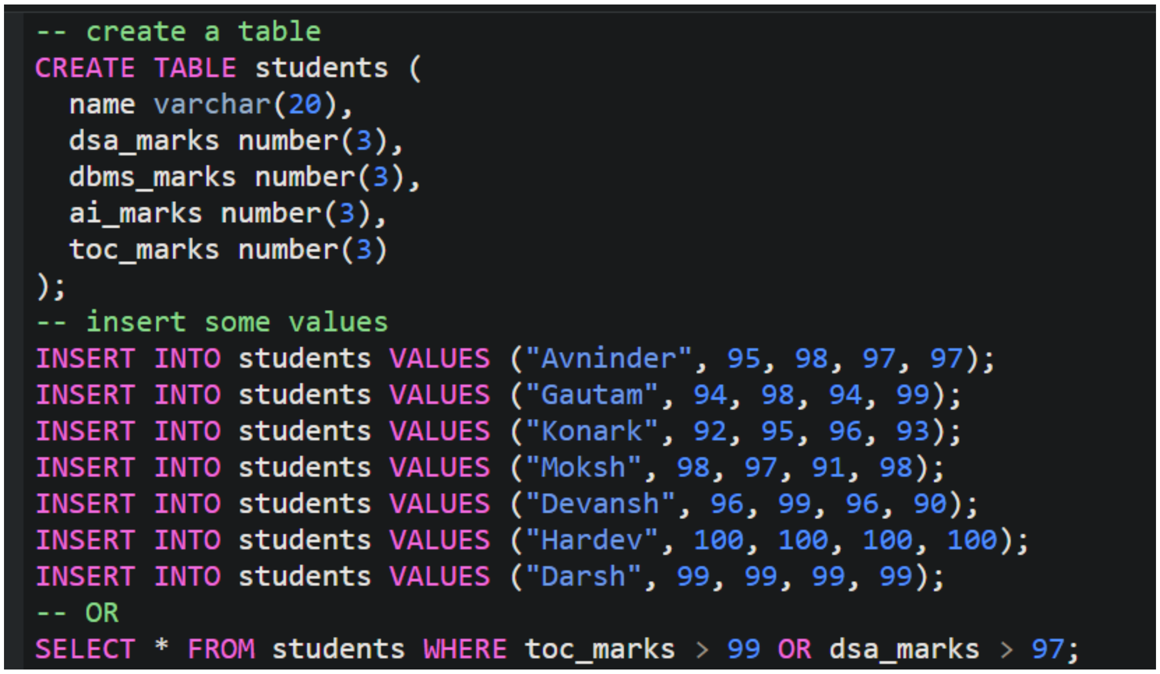
```sql
DELETE FROM table_name WHERE condition;
```

Practical - 2

Aim: Write SQL queries using logical operators (,=etc).

Logical Operators: Logical operators that can be used in SQL sentences are:

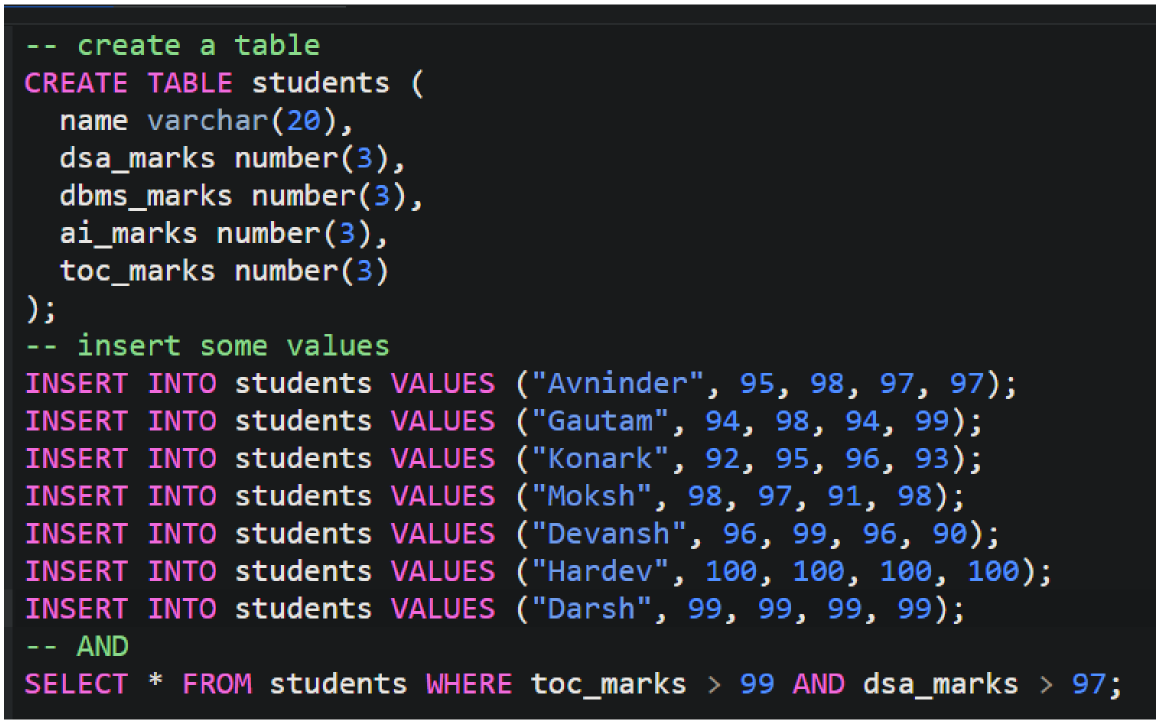
1. The AND operator: The AND operator displays the result only when all of the conditions specified using the AND operator are specified and satisfied.
Example:



```sql
-- create a table
CREATE TABLE students (
  name varchar(20),
  dsa_marks number(3),
  dbms_marks number(3),
  ai_marks number(3),
  toc_marks number(3)
);
-- insert some values
INSERT INTO students VALUES ("Avninder", 95, 98, 97, 97);
INSERT INTO students VALUES ("Gautam", 94, 98, 94, 99);
INSERT INTO students VALUES ("Konark", 92, 95, 96, 93);
INSERT INTO students VALUES ("Moksh", 98, 97, 91, 98);
INSERT INTO students VALUES ("Devansh", 96, 99, 96, 90);
INSERT INTO students VALUES ("Hardev", 100, 100, 100, 100);
INSERT INTO students VALUES ("Darsh", 99, 99, 99, 99);
-- OR
SELECT * FROM students WHERE toc_marks > 99 OR dsa_marks > 97;
```

2. The OR operator: The OR operator displays the result when any one condition is true.

Example:



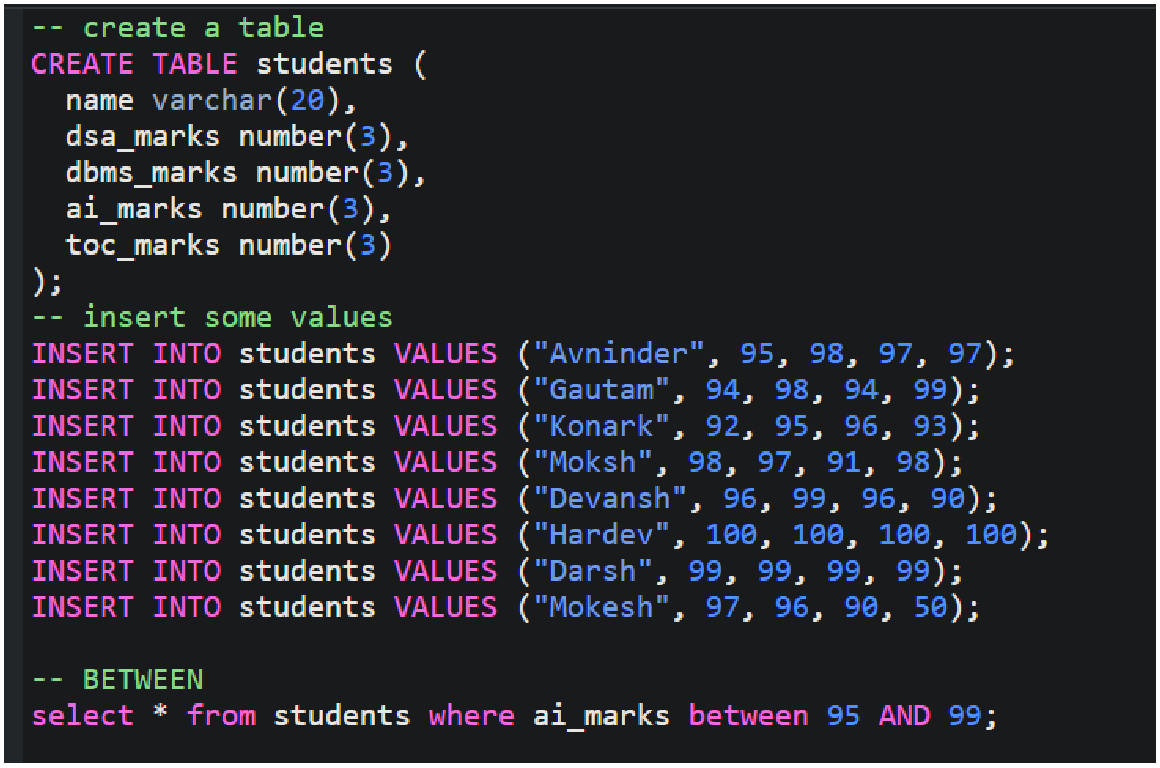
```sql
-- create a table
CREATE TABLE students (
  name varchar(20),
  dsa_marks number(3),
  dbms_marks number(3),
  ai_marks number(3),
  toc_marks number(3)
);
-- insert some values
INSERT INTO students VALUES ("Avninder", 95, 98, 97, 97);
INSERT INTO students VALUES ("Gautam", 94, 98, 94, 99);
INSERT INTO students VALUES ("Konark", 92, 95, 96, 93);
INSERT INTO students VALUES ("Moksh", 98, 97, 91, 98);
INSERT INTO students VALUES ("Devansh", 96, 99, 96, 90);
INSERT INTO students VALUES ("Hardev", 100, 100, 100, 100);
INSERT INTO students VALUES ("Darsh", 99, 99, 99, 99);
-- AND
SELECT * FROM students WHERE toc_marks > 99 AND dsa_marks > 97;
```

Practical – 3

Aim: Write SQL queries using SQL operators (between, and, or, in, like, null).

The BETWEEN operator allows the selection of rows that contain values within a specified lower and upper limit. The range coded after the BETWEEN is inclusive.

Example:

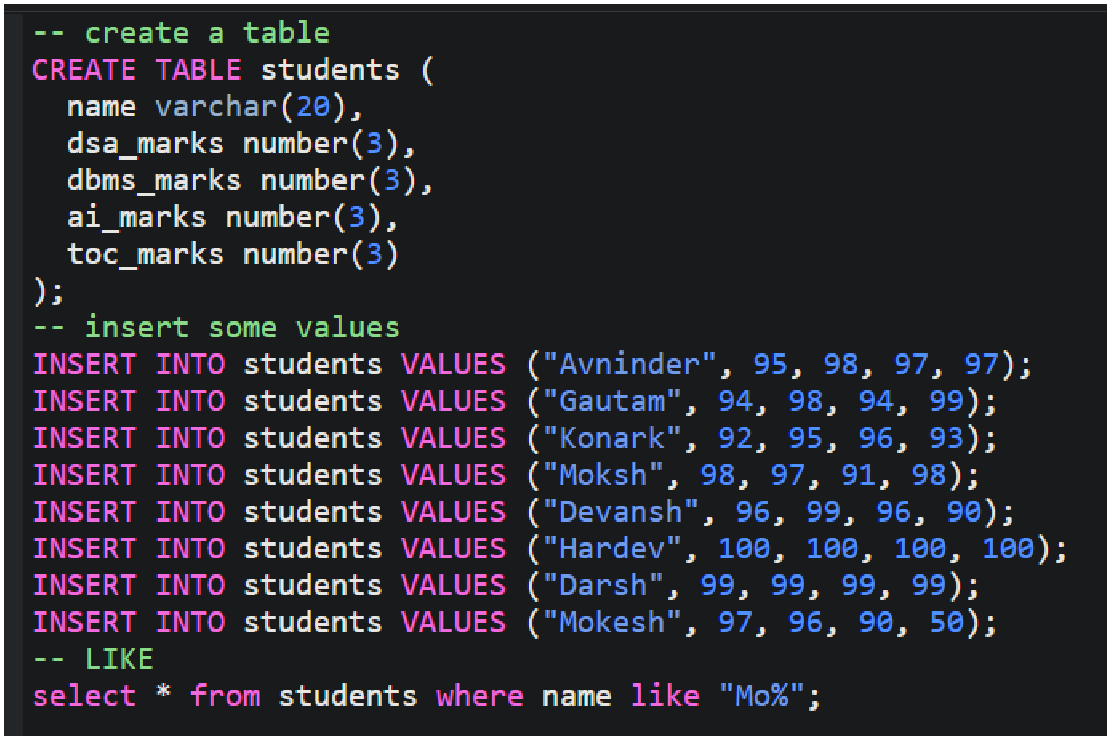


```
-- create a table
CREATE TABLE students (
  name varchar(20),
  dsa_marks number(3),
  dbms_marks number(3),
  ai_marks number(3),
  toc_marks number(3)
);
-- insert some values
INSERT INTO students VALUES ("Avninder", 95, 98, 97, 97);
INSERT INTO students VALUES ("Gautam", 94, 98, 94, 99);
INSERT INTO students VALUES ("Konark", 92, 95, 96, 93);
INSERT INTO students VALUES ("Moksh", 98, 97, 91, 98);
INSERT INTO students VALUES ("Devansh", 96, 99, 96, 90);
INSERT INTO students VALUES ("Hardev", 100, 100, 100, 100);
INSERT INTO students VALUES ("Darsh", 99, 99, 99, 99);
INSERT INTO students VALUES ("Mokesh", 97, 96, 90, 50);

-- BETWEEN
select * from students where ai_marks between 95 AND 99;
```

The LIKE predicate: The LIKE predicate allows comparison of one string value, which is not identical. This is achieved by using wildcard characters. Two wildcard characters that are used are % and -.

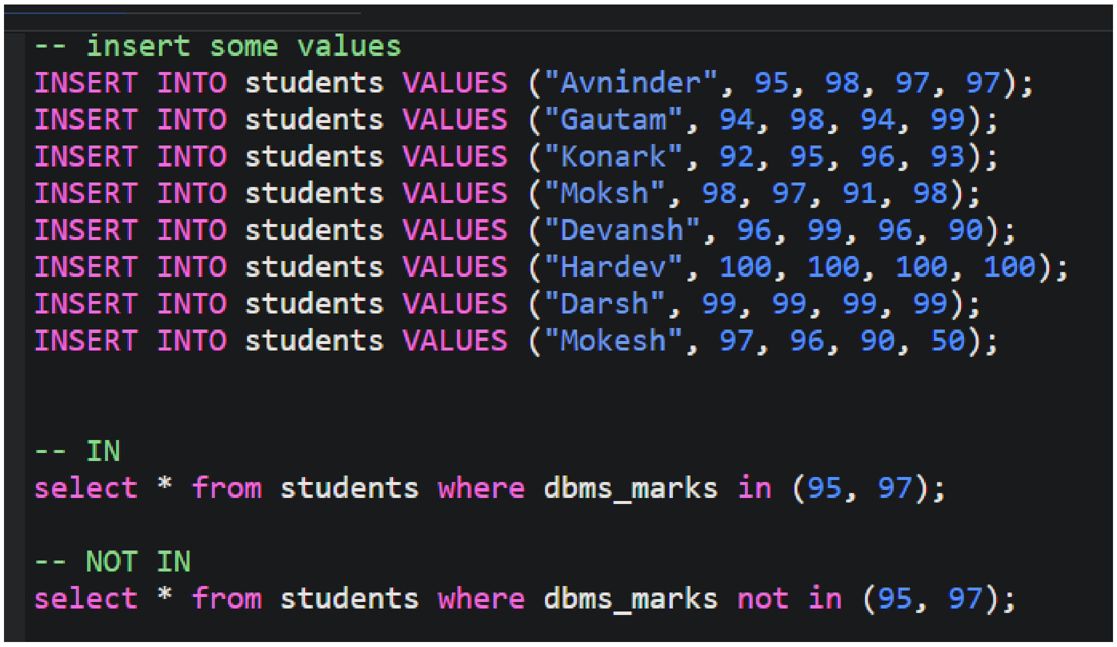
Example:



```sql
-- create a table
CREATE TABLE students (
  name varchar(20),
  dsa_marks number(3),
  dbms_marks number(3),
  ai_marks number(3),
  toc_marks number(3)
);
-- insert some values
INSERT INTO students VALUES ("Avninder", 95, 98, 97, 97);
INSERT INTO students VALUES ("Gautam", 94, 98, 94, 99);
INSERT INTO students VALUES ("Konark", 92, 95, 96, 93);
INSERT INTO students VALUES ("Moksh", 98, 97, 91, 98);
INSERT INTO students VALUES ("Devansh", 96, 99, 96, 90);
INSERT INTO students VALUES ("Hardev", 100, 100, 100, 100);
INSERT INTO students VALUES ("Darsh", 99, 99, 99, 99);
INSERT INTO students VALUES ("Mokesh", 97, 96, 90, 50);
-- LIKE
select * from students where name like "Mo%";
```

c) The IN and NOT IN predicate: The arithmetic operator (=) compare a single value to another single value. In case a value needs to be compared to a list of 9 values then the IN predicate is used. The NOT IN predicate is the opposite of IN predicate. This will select all the rows where values do not match the values in the list.

Example:



```sql
-- insert some values
INSERT INTO students VALUES ("Avninder", 95, 98, 97, 97);
INSERT INTO students VALUES ("Gautam", 94, 98, 94, 99);
INSERT INTO students VALUES ("Konark", 92, 95, 96, 93);
INSERT INTO students VALUES ("Moksh", 98, 97, 91, 98);
INSERT INTO students VALUES ("Devansh", 96, 99, 96, 90);
INSERT INTO students VALUES ("Hardev", 100, 100, 100, 100);
INSERT INTO students VALUES ("Darsh", 99, 99, 99, 99);
INSERT INTO students VALUES ("Mokesh", 97, 96, 90, 50);


-- IN
select * from students where dbms_marks in (95, 97);

-- NOT IN
select * from students where dbms_marks not in (95, 97);
```

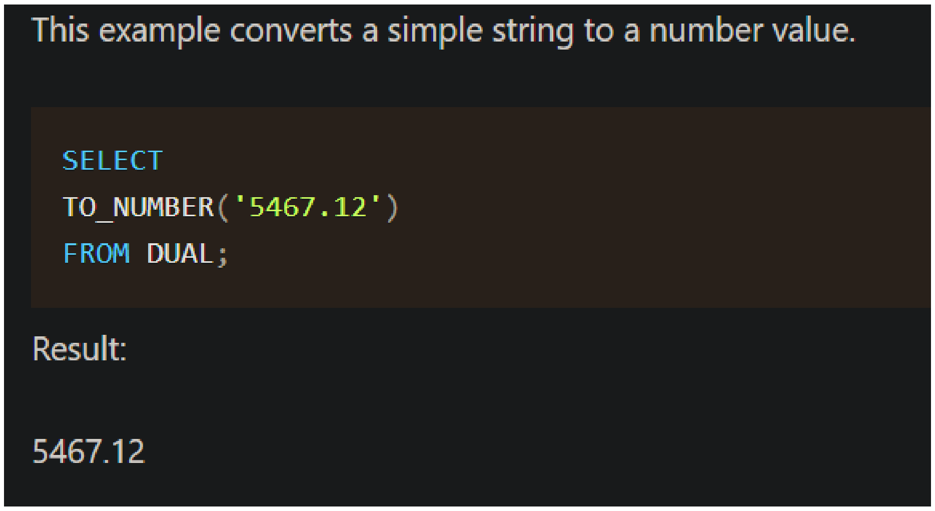
Practical – 4

Aim: Write SQL query using character, number, date and group functions.

a) TO_NUMBER: The TO_NUMBER function converts a character value to a numeric data type. If the string being converted contains nonnumeric characters, the function returns an error.

Syntax: TO_NUMBER (string1, [format], [nls_parameter])

Example:



This example converts a simple string to a number value.

```
SELECT
TO_NUMBER('5467.12')
FROM DUAL;
```

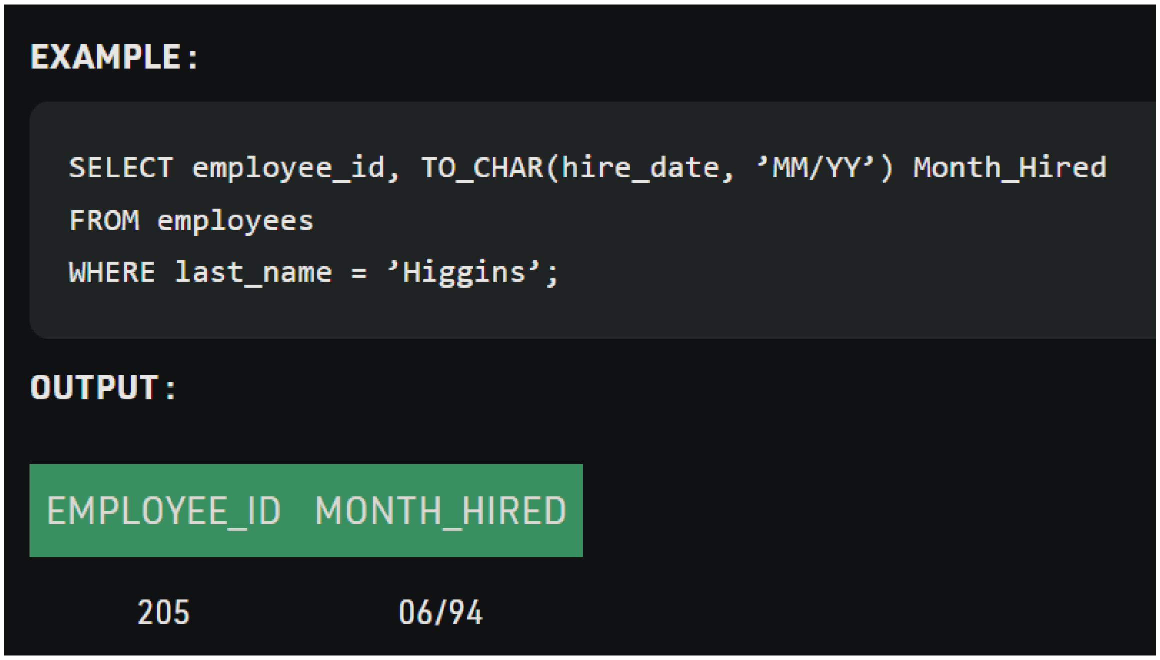
Result:

```
5467.12
```

b) TO_CHAR: TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

Syntax: TO_CHAR(number1, [format], [nls_parameter])

Example:



EXAMPLE:

```
SELECT employee_id, TO_CHAR(hire_date, ’MM/YY’) Month_Hired
FROM employees
WHERE last_name = ’Higgins’;
```

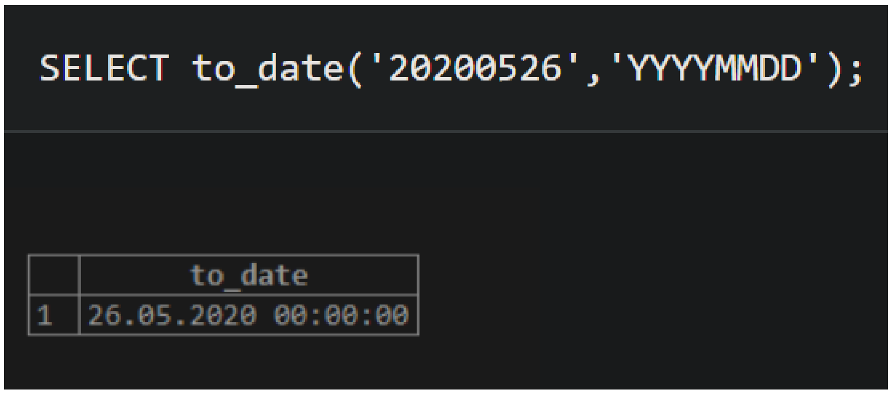
OUTPUT:

```
EMPLOYEE_ID  MONTH_HIRED
     205         06/94
```

c) TO_DATE: The function takes character values as input and returns formatted date equivalent of the same. The TO_DATE function allows users to enter a date in any format, and then it converts the entry into the default format used by Oracle 11g.

Syntax: TO_DATE(string1, [format_mask], [nls_language])

Example:



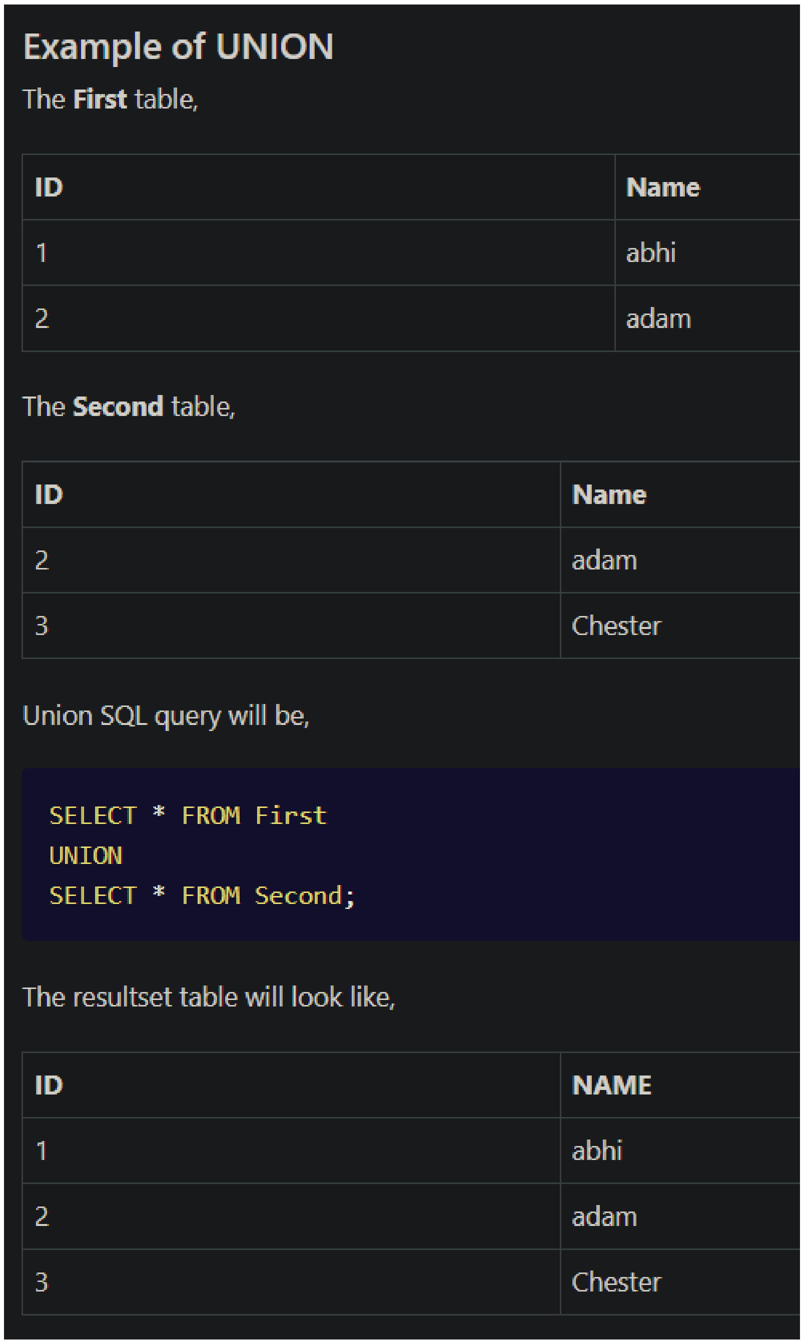
```
SELECT to_date('20200526','YYYYMMDD');
```

```
    to_date
1   26.05.2020 00:00:00
```

Practical – 5

Aim: Write SQL queries for Relational Algebra (union, intersect, and minus, etc.).

Union:



Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

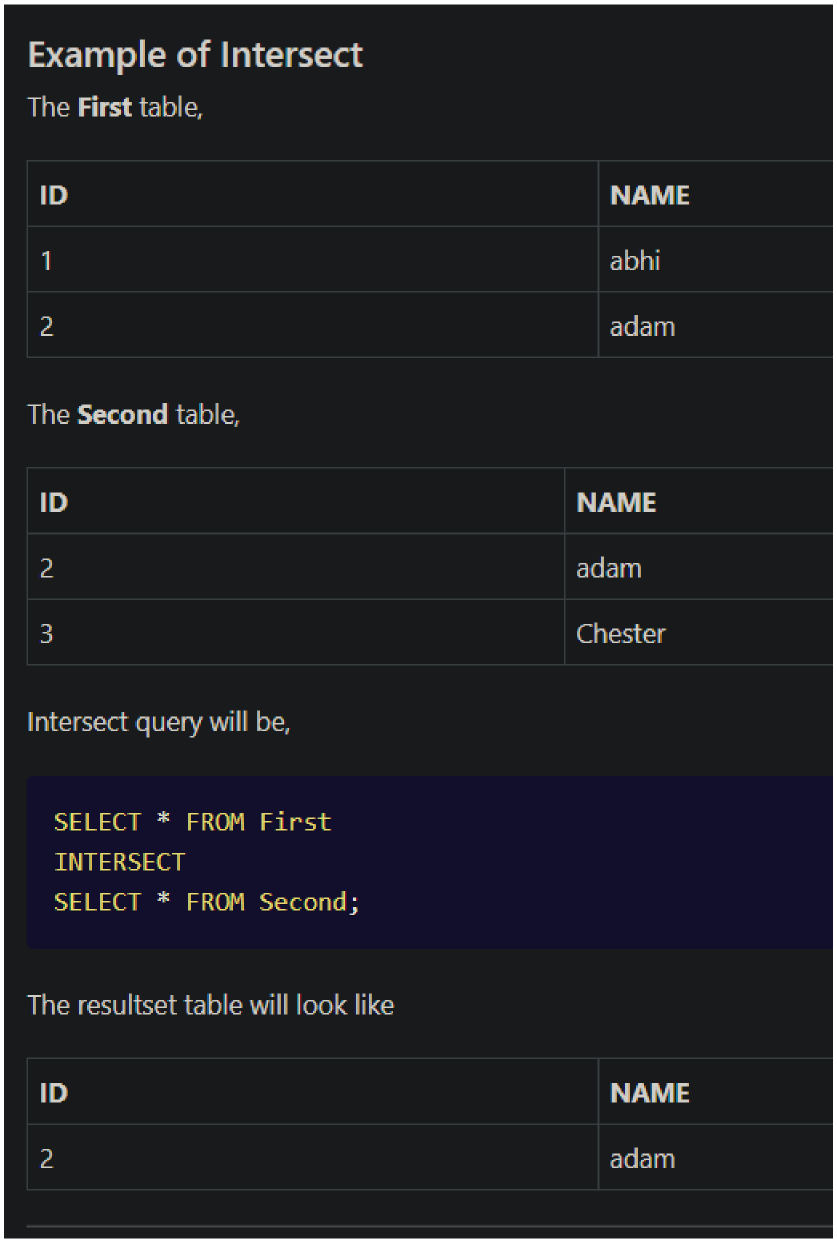
Union SQL query will be,

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

Intersect:



Example of Intersect

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

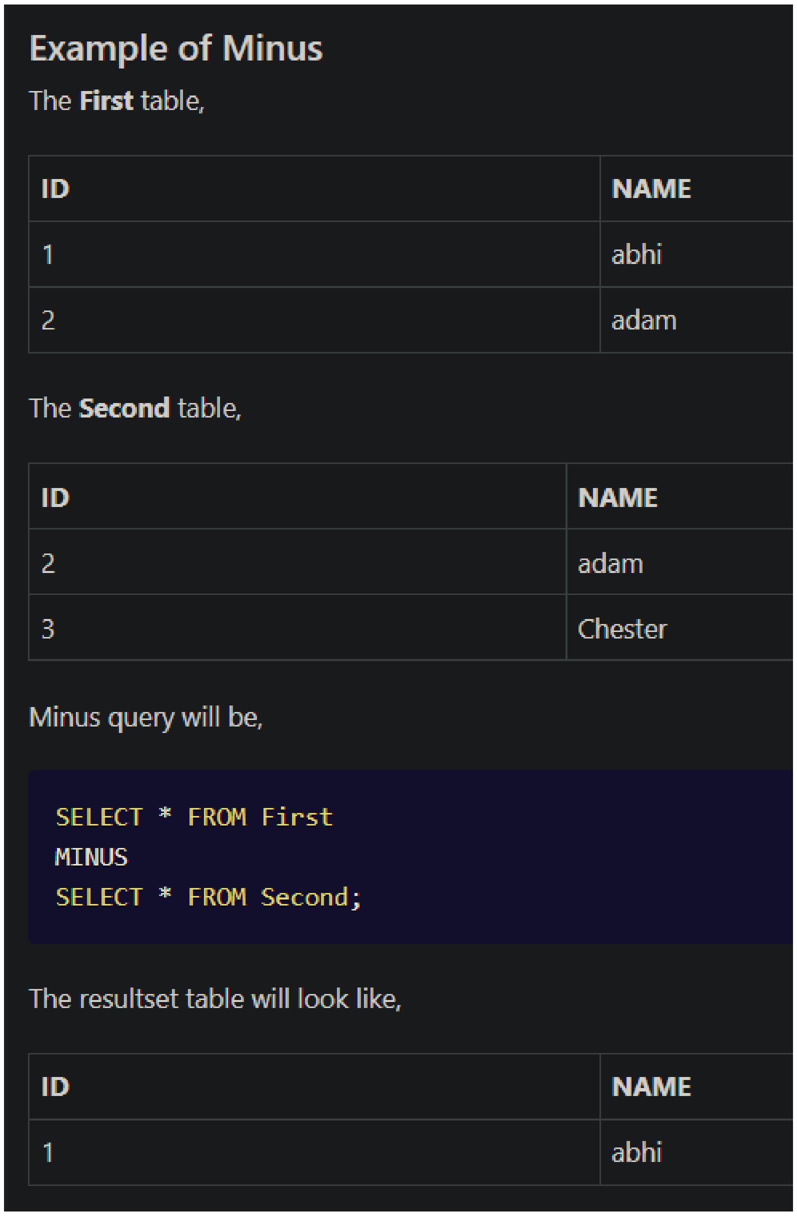
Intersect query will be,

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like

ID	NAME
2	adam

Minus:



Example of Minus

The **First** table,

ID	NAME
1	abhi
2	adam

The **Second** table,

ID	NAME
2	adam
3	Chester

Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

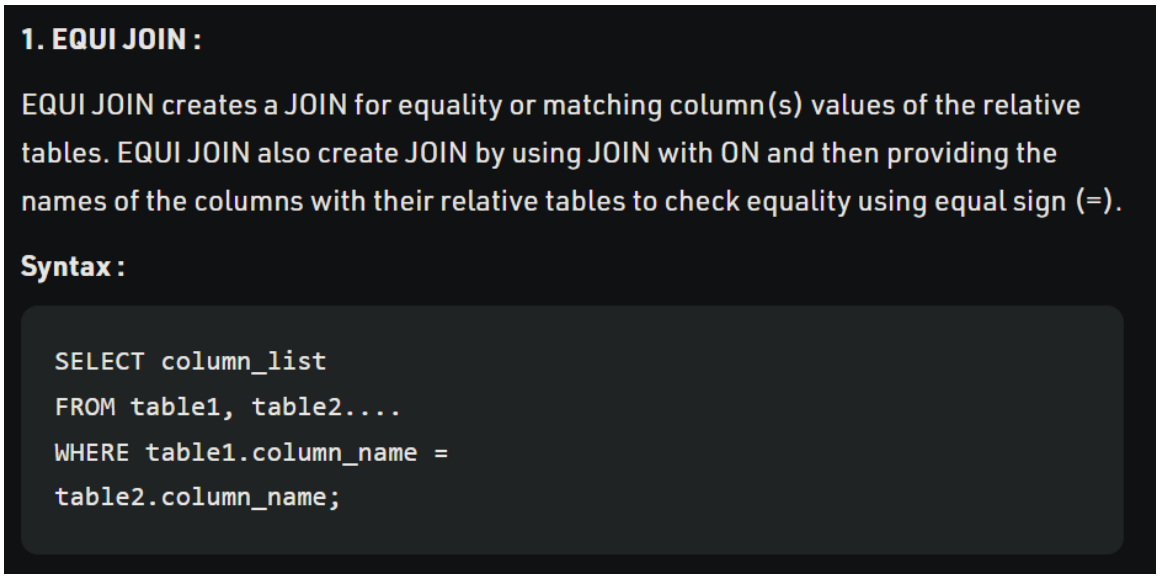
The resultset table will look like,

ID	NAME
1	abhi

Practical – 6

Aim: Write SQL queries for extracting data from more than one table (equi-join, non equi-join, outer join).

Equi-Join:

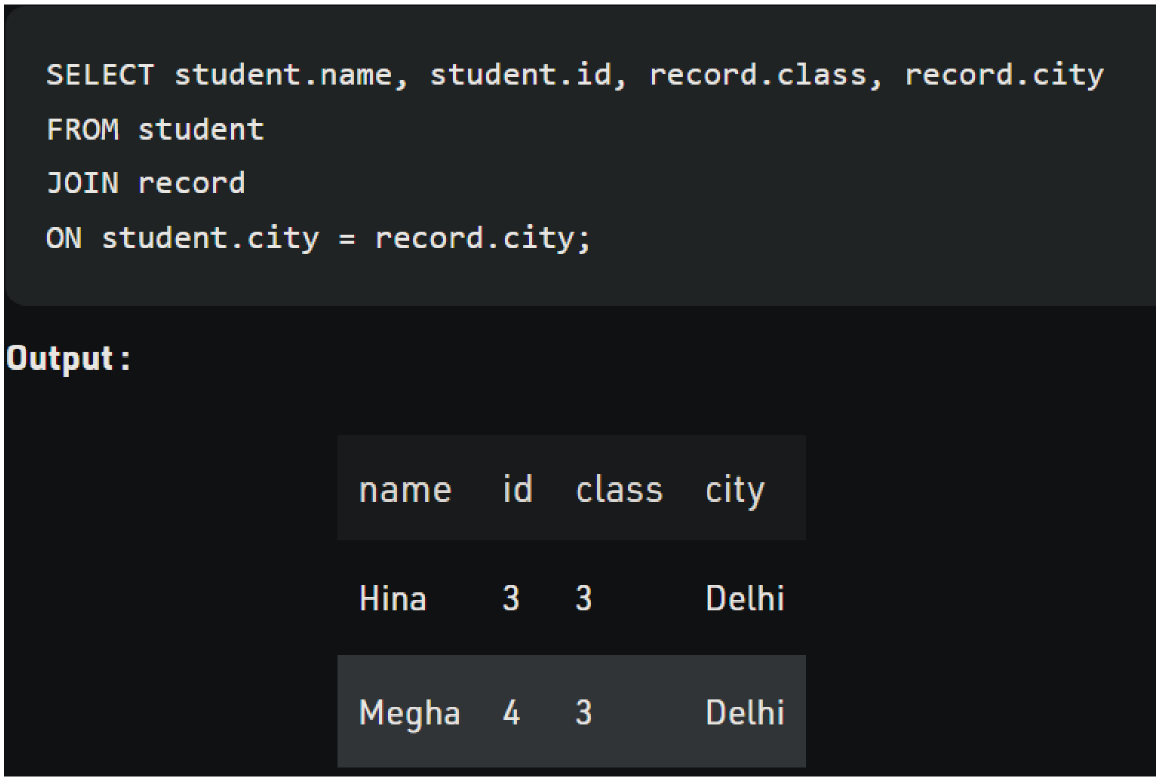


1. EQUI JOIN :

EQUI JOIN creates a JOIN for equality or matching column(s) values of the relative tables. EQUI JOIN also create JOIN by using JOIN with ON and then providing the names of the columns with their relative tables to check equality using equal sign (=).

Syntax :

```
SELECT column_list
FROM table1, table2....
WHERE table1.column_name =
table2.column_name;
```

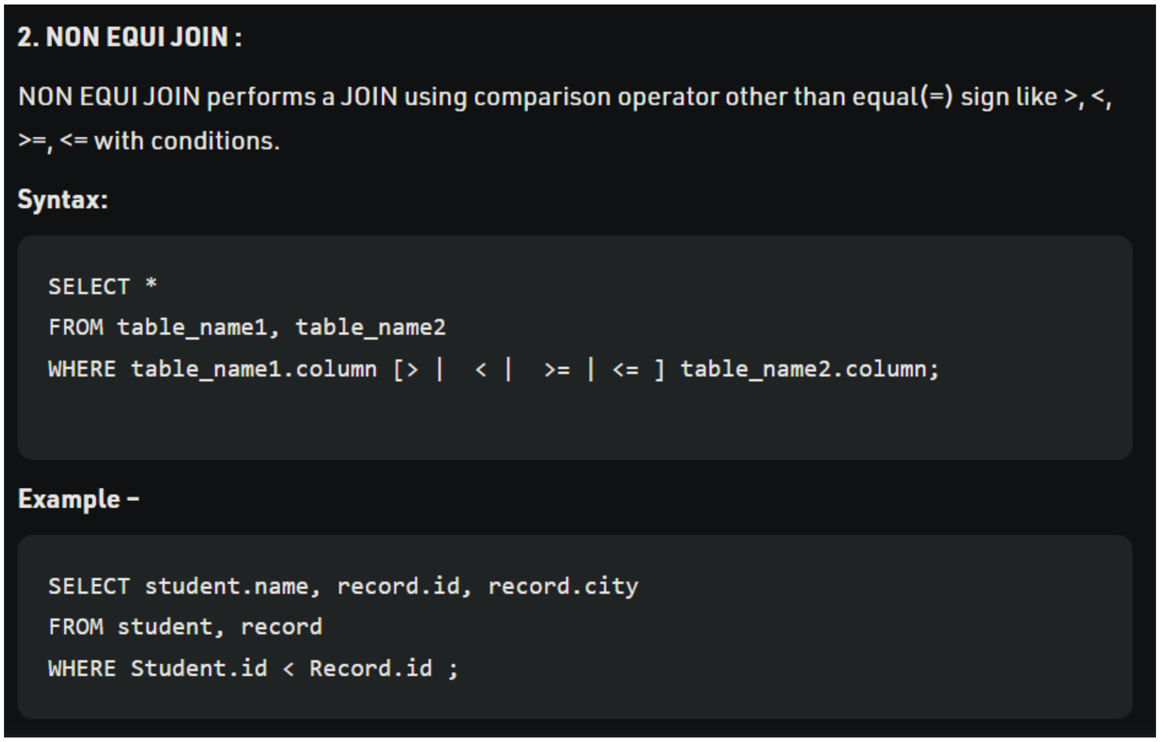


```
SELECT student.name, student.id, record.class, record.city
FROM student
JOIN record
ON student.city = record.city;
```

Output :

name	id	class	city
Hina	3	3	Delhi
Megha	4	3	Delhi

Non-Equi-join:



2. NON EQUI JOIN :

NON EQUI JOIN performs a JOIN using comparison operator other than equal(=) sign like >, <, >=, <= with conditions.

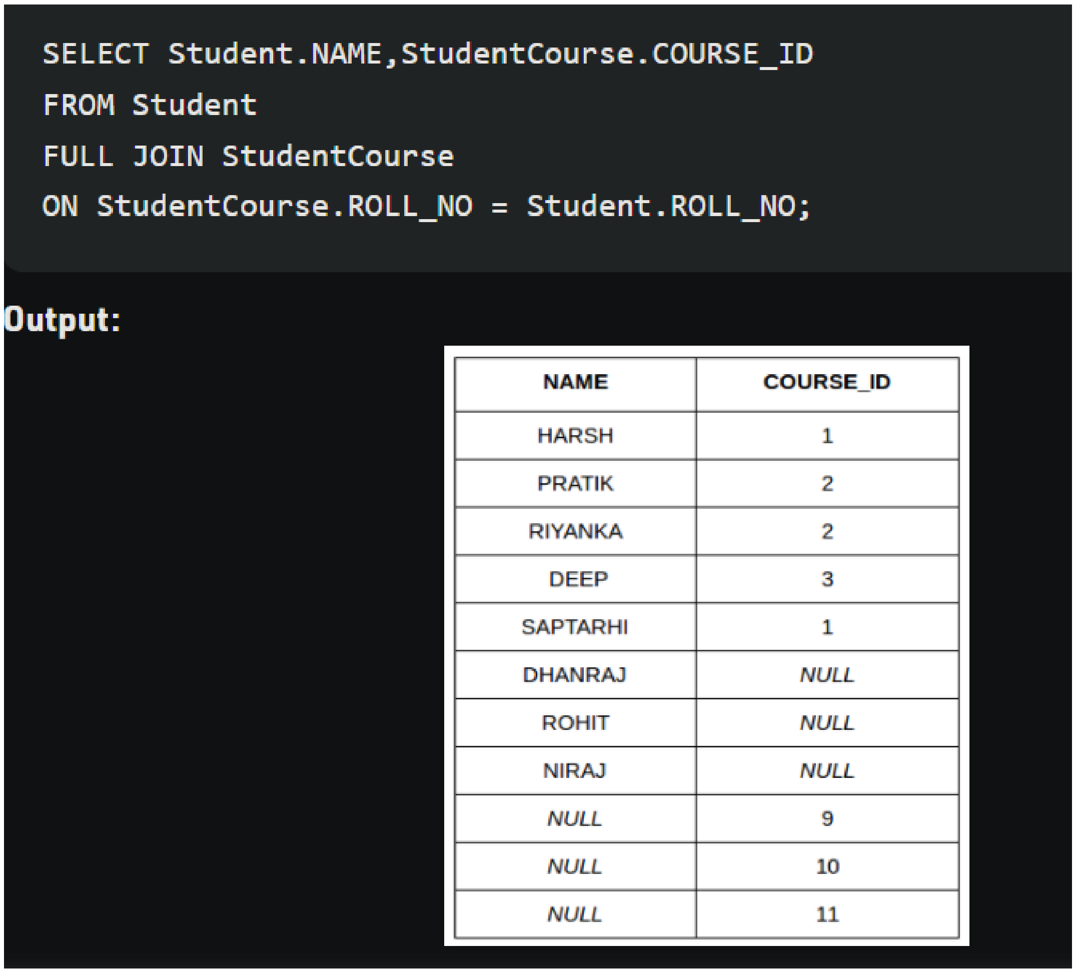
Syntax:

```
SELECT *
FROM table_name1, table_name2
WHERE table_name1.column [> |  < |  >= | <= ] table_name2.column;
```

Example –

```
SELECT student.name, record.id, record.city
FROM student, record
WHERE Student.id < Record.id ;
```

Outerjoin:



```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

INNER JOIN

In SQL, INNER JOIN selects records that have matching
values in both tables as long as the condition is satisfied. It returns the
combination of all rows from both the tables where the condition satisfies.

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example –

```
SELECT students_99.name, students_99.roll_no, students_99.age, students_99.branch
FROM students_99
INNER JOIN students_table_3
ON students_99.name = students_table_3.name;
```

Output –

NAME	ROLL_NO	AGE	BRANCH
Avninder	1	20	CSE
Stephney	2	21	CSE
David	4	20	CSE
Dave	6	21	CSE
Jane	7	20	CSE

LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it
will returns NULL.

```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example –

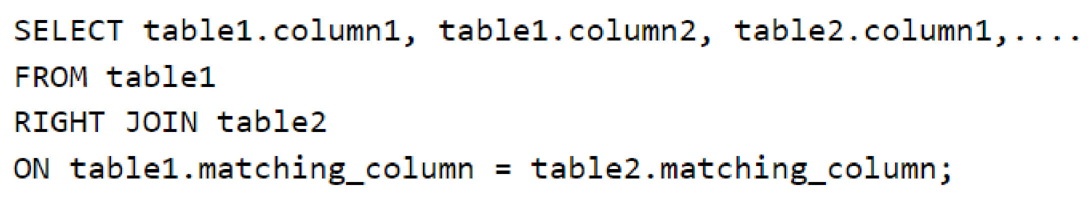
```
SELECT students_99.name, students_99.roll_no, students_99.age, students_99.branch
FROM students_99
LEFT JOIN students_table_3
ON students_99.name = students_table_3.name;
```

Output –

NAME	ROLL_NO	AGE	BRANCH
Avninder	1	20	CSE
Stephney	2	21	CSE
David	4	20	CSE
Dave	6	21	CSE
Jane	7	20	CSE
Mike	3	19	CSE
Laurel	5	19	CSE

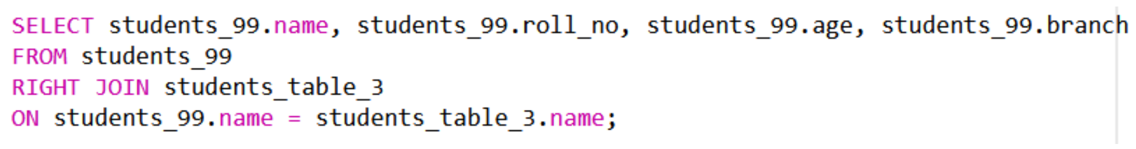
RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values
from the rows of right table and the matched values from the left table. If
there is no matching in both tables, it will return NULL.



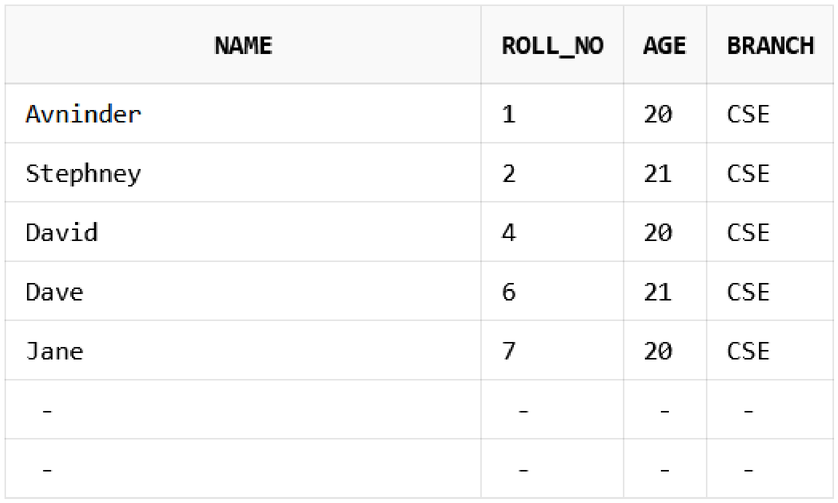
```
SELECT table1.column1, table1.column2, table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example –



```
SELECT students_99.name, students_99.roll_no, students_99.age, students_99.branch
FROM students_99
RIGHT JOIN students_table_3
ON students_99.name = students_table_3.name;
```

Output –

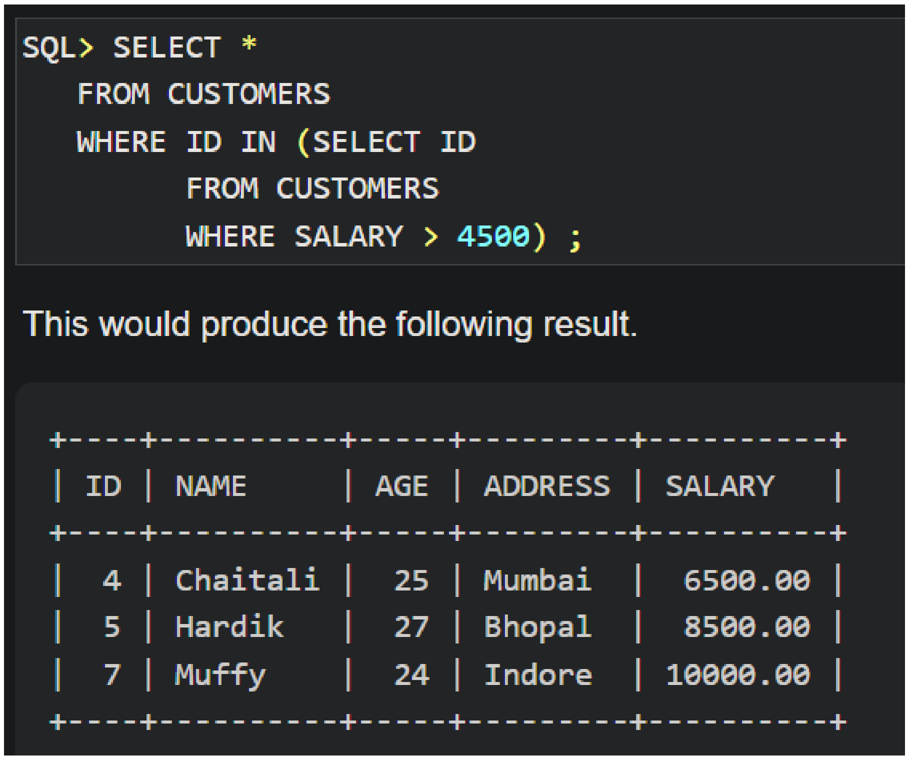


NAME	ROLL_NO	AGE	BRANCH
Avninder	1	20	CSE
Stephney	2	21	CSE
David	4	20	CSE
Dave	6	21	CSE
Jane	7	20	CSE
-	-	-	-
-	-	-	-

Practical – 7

Aim: write sql queries for sub-queries, nested queries

Sub-queries:



```
SQL> SELECT *
   FROM CUSTOMERS
   WHERE ID IN (SELECT ID
         FROM CUSTOMERS
         WHERE SALARY > 4500) ;
```

This would produce the following result.

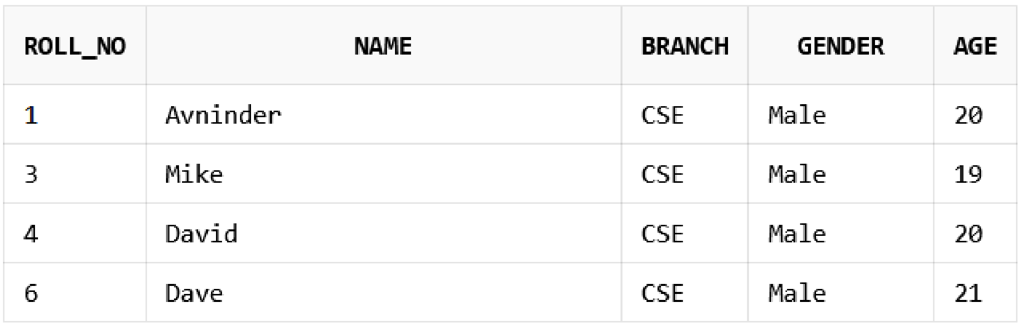
```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Nested Query –

```
Select * from Table_name
where Table_name.column_name in (Select column_name from Table_name where Condition);
```

Example –

```
Select * from students_99
where roll_no in (Select roll_no from students_99 where gender = 'Male');
```



ROLL_NO	NAME	BRANCH	GENDER	AGE
1	Avninder	CSE	Male	20
3	Mike	CSE	Male	19
4	David	CSE	Male	20
6	Dave	CSE	Male	21

Practical – 8

Aim: Creation, altering and dropping of tables and inserting rows into a table (use constraints while creating tables) examples using Select command.

1. **I/O constraints –**

   This data constraints determines the speed at which data can be inserted or extracted an oracle table.

2. **Business Rule constraints –**

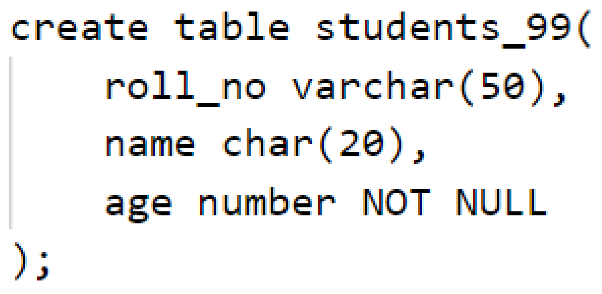
   This data constraints can be implemented in oracle by using check constraints. Business rule validation check are perform when the user perform a right operation on table i.e. insert, update and delete data. Any insert, update and delete statement causes the relevant constraints to be evaluated. Constraints are stored as a part of the global table definition by the oracle engine in its system table.

- **Null value concept -** When a column is defined as not null, it becomes mandatory column.

  Syntax: NOT NULL-

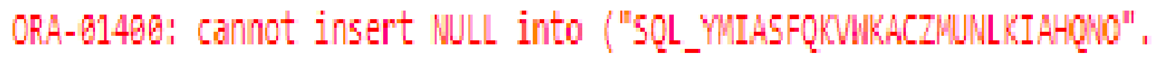
  Create table name (column1 data type, column2 data type NOTNULL ...);

  Example –



```
create table students_99(
    roll_no varchar(50),
    name char(20),
    age number NOT NULL
);
```

Error code -

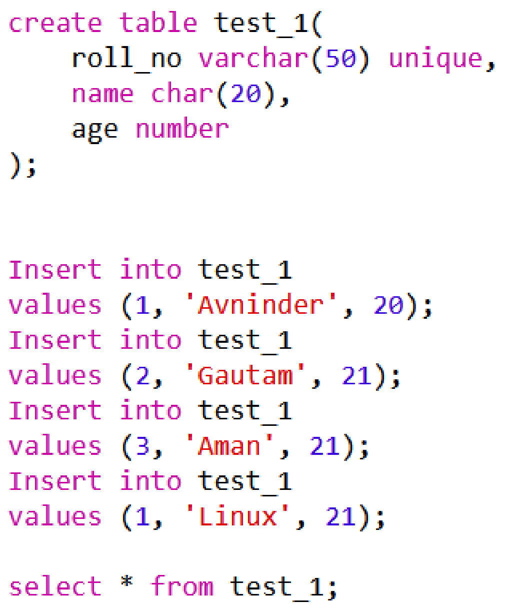


```
ORA-01400: cannot insert NULL into ("SQL_YMIASFQKVWKACZMUNLKIAHQNO".
```

- **Unique constraints**: The unique column constraint permit multiple entries of
  NULL into the column.

At Column Level-

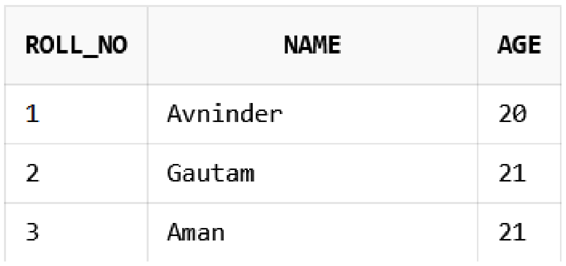
Syntax-
create table table_name (column1 data type, column2 data type unique, column3 data type unique ...);



```
create table test_1(
    roll_no varchar(50) unique,
    name char(20),
    age number
);


Insert into test_1
values (1, 'Avninder', 20);
Insert into test_1
values (2, 'Gautam', 21);
Insert into test_1
values (3, 'Aman', 21);
Insert into test_1
values (1, 'Linux', 21);

select * from test_1;
```

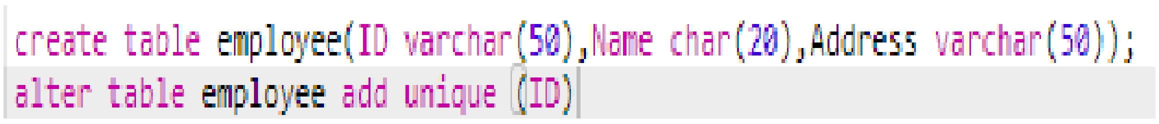


ROLL_NO	NAME	AGE
1	Avninder	20
2	Gautam	21
3	Aman	21

- **Primary key constraints -** In a table we implement the primary key constraint only one column that make it unique from other values. Primary key constraint
  cannot contain Null value.

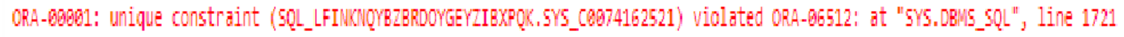
  Syntax - At column level-

  Create table table_name (column2 data type, column2 data type, column2 data type ... primary (column1, column2….));



```
create table employee(ID varchar(50),Name char(20),Address varchar(50));
alter table employee add unique (ID)
```

(If we put same primary key we get an error)



```
ORA-00001: unique constraint (SQL_LFINKNQYBZBRDOYGEYZIBXPQK.SYS_C0074162521) violated ORA-06512: at "SYS.DBMS_SQL", line 1721
```

- **Simple key –**

  In a database table, a simple key is just a single attribute that can uniquely identify a row.

- **Composite key –**

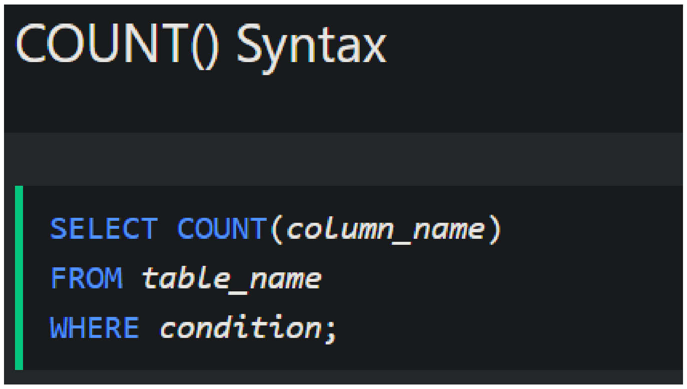
  A composite key in SQL can be defined as a combination of multiple columns, and these columns are used to identify all the rows that are involved uniquely.

Practical – 9

Aim: Queries using aggregate functions (count, sum, avg, max and min), group by, having and creation and dropping of views.

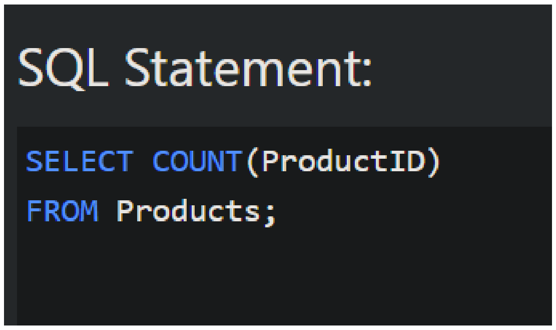
Count:

The COUNT() function returns the number of rows that matches a specified criterion.



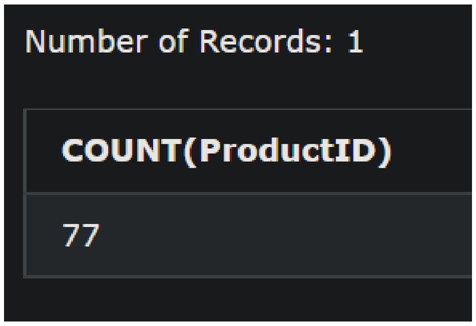
COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```



SQL Statement:

```
SELECT COUNT(ProductID)
FROM Products;
```

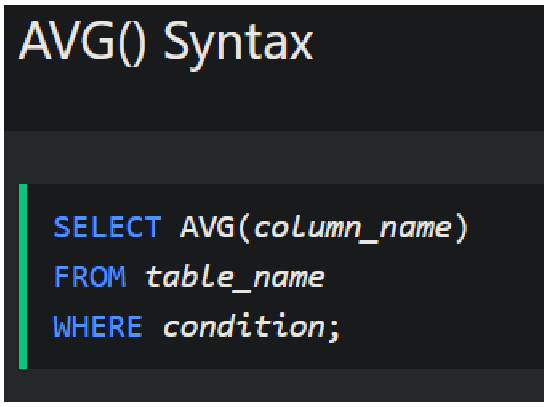


```
Number of Records: 1

COUNT(ProductID)
77
```

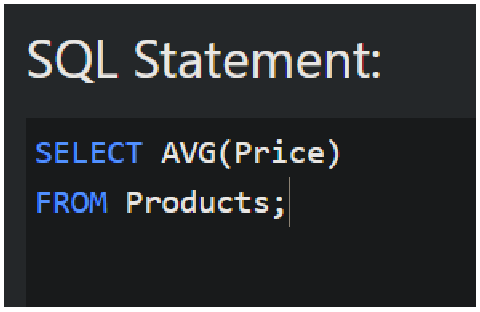
AVG():

The AVG() function returns the average value of a numeric column.

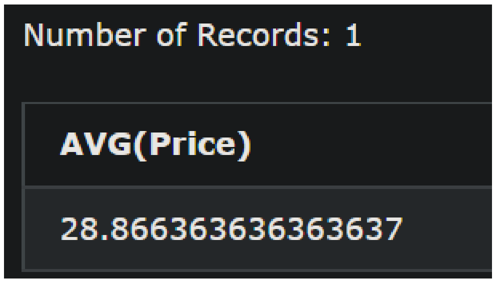


AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```


SQL Statement:

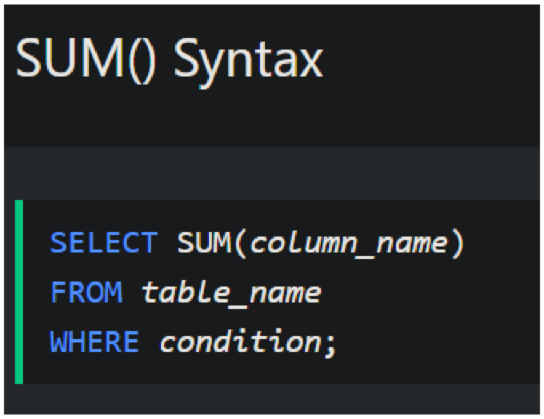
```
SELECT AVG(Price)
FROM Products;
```


Number of Records: 1

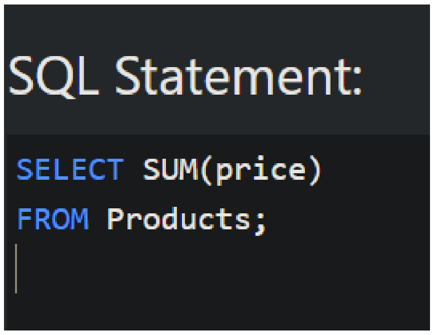
```
AVG(Price)
28.866363636363637
```

Sum():

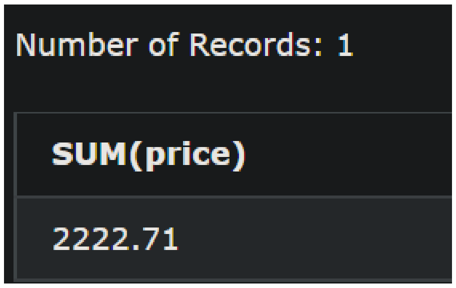
The SUM() function returns the total sum of a numeric column.


SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```


SQL Statement:

```
SELECT SUM(price)
FROM Products;
```

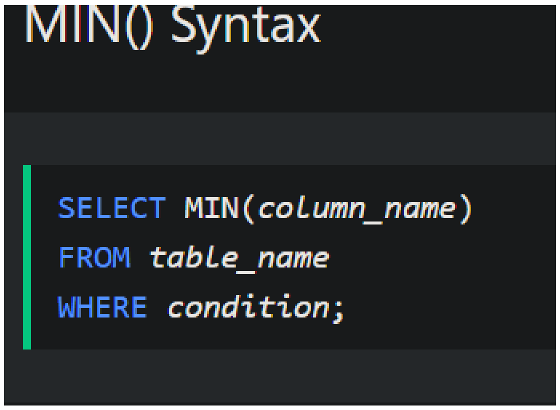


```
Number of Records: 1

SUM(price)
2222.71
```

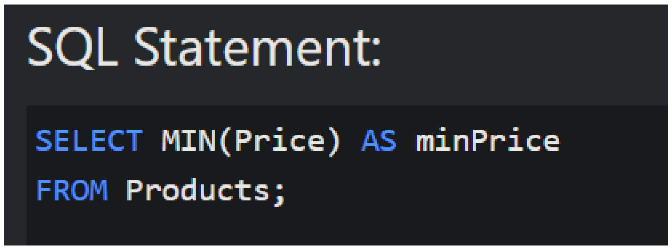
Min():

The MIN() function returns the smallest value of the selected column.



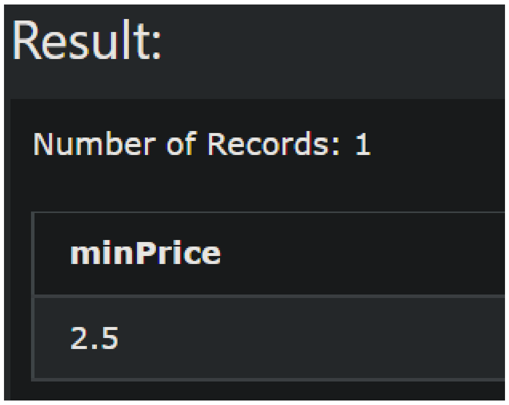
MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```



SQL Statement:

```
SELECT MIN(Price) AS minPrice
FROM Products;
```

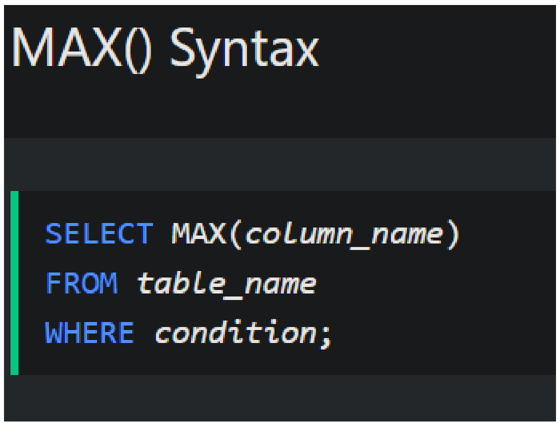

Result:

```
Number of Records: 1
```

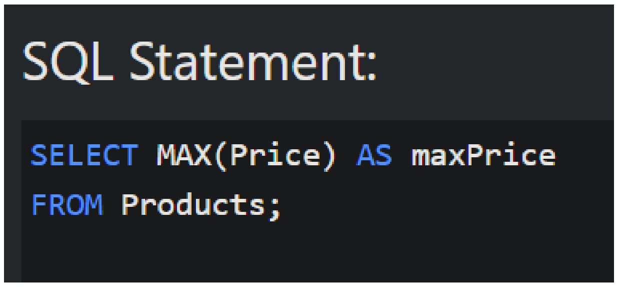
minPrice
2.5

Max():

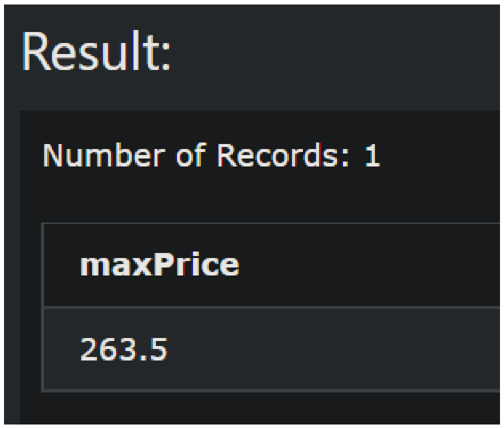
The MAX() function returns the largest value of the selected column.


MAX() Syntax

```sql
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```


SQL Statement:

```sql
SELECT MAX(Price) AS maxPrice
FROM Products;
```

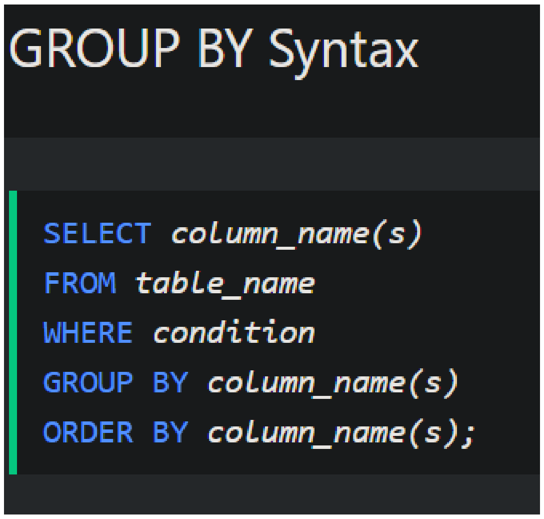

Result:

Number of Records: 1

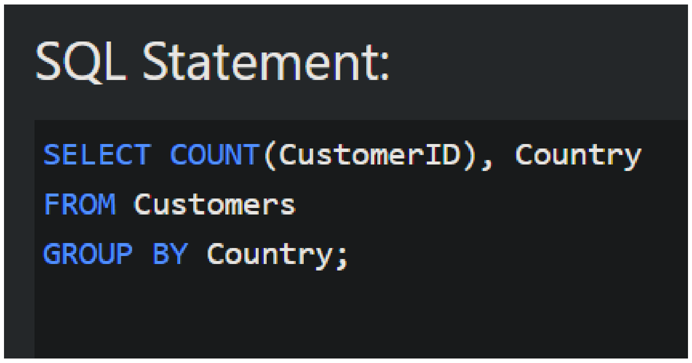
maxPrice
263.5

GroupBy:

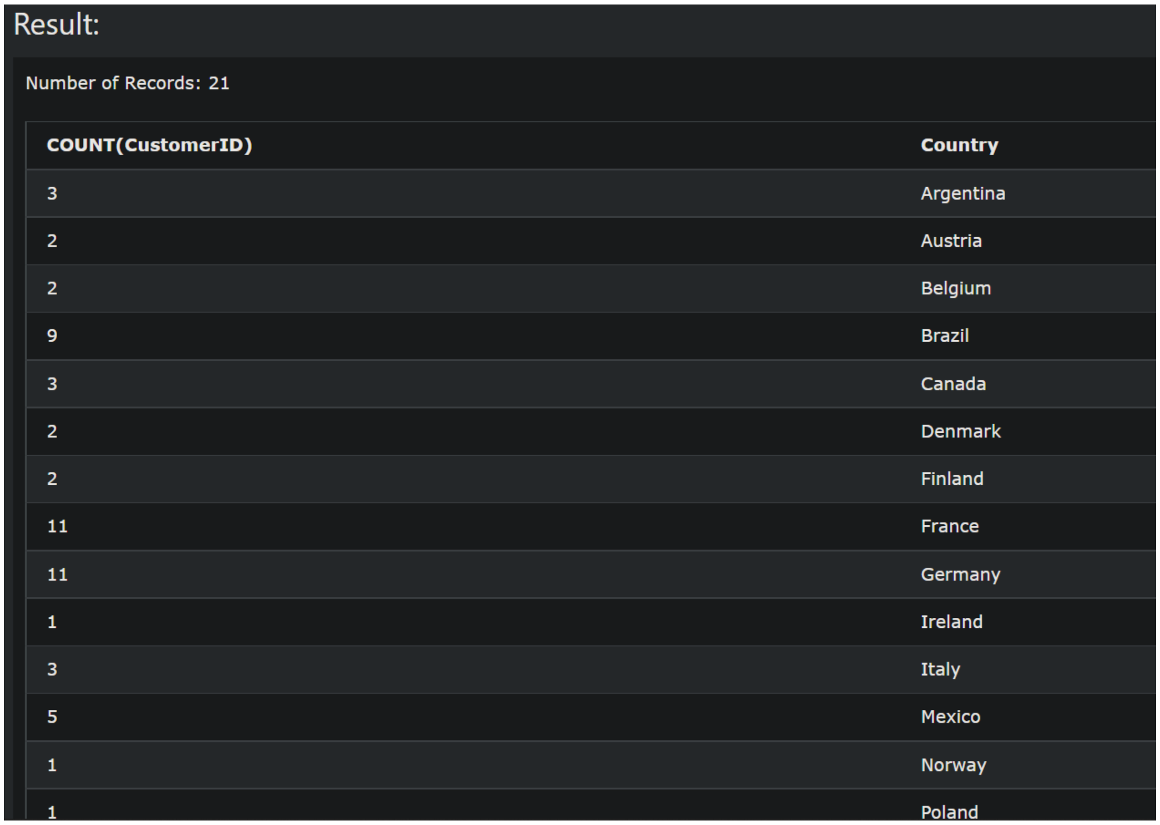
The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".


GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```


SQL Statement:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

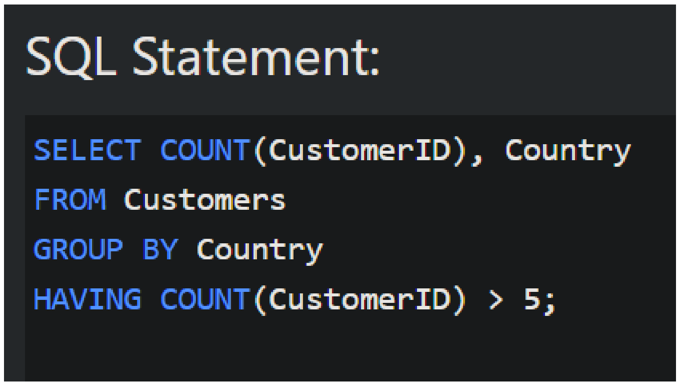

Result:

Number of Records: 21

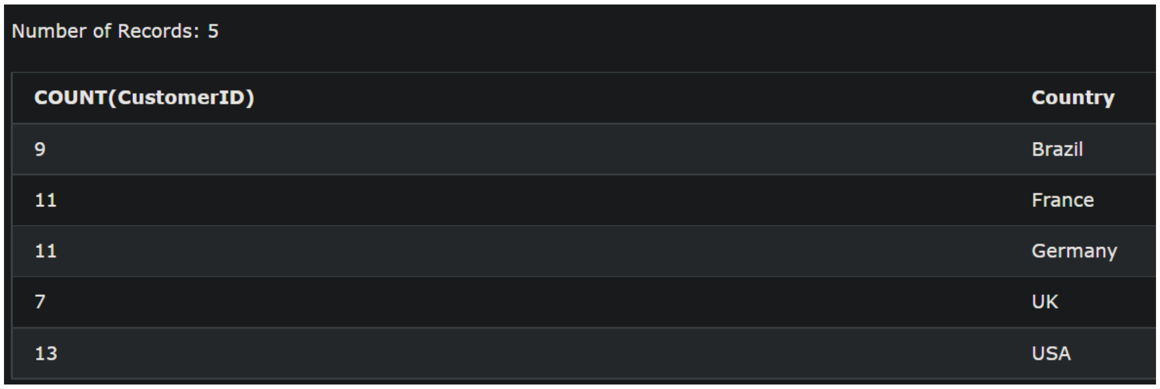
COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland
11	France
11	Germany
1	Ireland
3	Italy
5	Mexico
1	Norway
1	Poland

Having():

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.


SQL Statement:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```


Number of Records: 5

COUNT(CustomerID)	Country
9	Brazil
11	France
11	Germany
7	UK
13	USA

CREATE VIEW EXAMPLE:

we create view aman1 from table hard by using the following syntax:

CREATE VIEW Syntax

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

```
create view test_5 as select id ,name, dbms_marks from test_4 where 1;
```

DROP VIEW:

A view is deleted with the DROP VIEW statement.

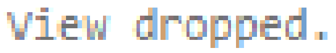
Syntax

```
DROP VIEW view_name;
```

EXAMPLE:

```
drop view test_5;
```

Output –



```
View dropped.
```

Practical – 10

Aim: Queries using conversion functions (to_char, to_number and to_date), string functions (Concatenation, lpad, rpad, ltrim, rtrim, lower, upper, initcap, length, substr and instr), date functions (Sysdate, next_day, add_months, last_day, months_between, least, greatest, trunc, round, to_char, to_date).

Conversion Functions

:- These functions are used to convert the value from one

type to another type.

1) TO_CHAR (N [,FMT]) :- Converts 'N' of numeric data type to Varchar2 datatype

using optional number format FMT.

TO_CHAR (N [,FMT])

Example 1) Convert 12345 to string.

SELECT TO_CHAR (12345) FROM DUAL;

31

2) Display system date after converting to varchar2 data type.

SELECT TO_CHAR (SYSDATE) FROM DUAL;

3) Display system date in 'MON-DD-YYYY' format after converting to varchar2 data

type.

SELECT TO_CHAR (SYSDATE, 'MON-DD-YYYY') FROM DUAL;

2) TO_NUMBER (CHAR) :- This conversion function is used to convert string to

number data type.

Ex :- Convert string '123.45' to number data type.

SELECT TO_NUMBER ('123.45') FROM DUAL;

3) TO_DATE :- Converts character data type data to date type data.

Ex:- Display '09-02-2010' converted to DDD-MM-YY format using to_char &

to_date functions.

SELECT TO_CHAR (TO_DATE ('09-02-2010', 'DD-MM-YYYY'),('DDD-MM-YY')

FROM DUAL;

String Functions:

String Functions :-

1) Concatenation :- Concatenates two stings from a given list.

CONCAT (CHAR1, CHAR2)

EX 1) Concate the string 'Rajesh' with 'Raghu'

SELECT CONCAT ('Rajesh', 'Raghu') FROM DUAL;

2) Concat bid & bname of Boats & display along with color.

SELECT CONCAT (BID, BNAME), COLOR FROM BOATS;

2) LPAD (CHAR1, N, CHAR2) :- Returns CHAR1 left padded to length ‗N' with

sequence of characters in CHAR2. The default value of CHAR2 is a single blank

space.

Ex 1) Lpad the string 'Rajesh' to length 30 with the set of characters in string '-*-'

SELECT LPAD ('Rajesh', 30, '-*-') FROM DUAL;

2) Lpad the string bname to length 20 with ‘-*’ set of characters and string color by

‘/’.

3) RPAD (CHAR1, N, CHAR2) :- Returns CHAR1 right padded to length ‘N’ with

sequence of characters in CHAR2. The default value of CHAR2 is a single blank

space.

Ex 1) Rpad the string ‘Rajesh’ to length 30 with the set of characters in string ‘*#’

SELECT RPAD (‘Rajesh’, 30, ‘*#’) FROM DUAL;

33

2) Rpad the string sname to length 25 with ‘-*’ set of characters and remaining

attributes in normal way.

3) Rpad the string bname to length 20 with ‘-*’ set of characters and lpad the same

to make it length 30 with ‘#’ and remaining attributes in normal way.

4) LTRIM (CHAR, SET) :- Returns characters from the left of CHAR by deleting all

leftmost characters that appear in set.

Ex:- 1) Display all sailors information by removing characters of sname if starts with ‘R’. SELECT SID, LTRIM (SNAME,’R’) FROM SAILORS

2) Display all sailors information by removing characters of boat name if starts with

‘T’

5) RTRIM (CHAR, SET) :- Returns characters from the right of CHAR by deleting all

rightmost characters that appear in set.

Ex:- 1) Display all sailors information by removing characters of sname if ends with

‘i’.

SELECT SID, RTRIM (SNAME,’i’) FROM SAILORS;

6) LOWER(CHAR) :- Converts all characters to lowercase characters in a sting

CHAR. Ex:- 1) Display all Boats information by showing their names in lower case.

SELECT BID, LOWER (BNAME), COLOR FROM BOATS;

34

7) UPPER(CHAR) :- Converts all characters to uppercase characters in a sting CHAR.

Ex:- 1) Display all Sailors information by showing their names in Upper case.

SELECT SID, UPPER (SNAME), AGE, RATING FROM SAILORS;

8) INITCAP(CHAR) :- Converts first character of each word in a sting CHAR to

uppercase. Ex:-1) Display all Sailors information by showing their names in

Capitalizing first char. SELECT SID, INITCAP (SNAME), AGE, RATING FROM

SAILORS;

2) Capatilize first letter of each word in ‘rajesh raghu’

9) LENGTH (CHAR) :- Returns the length of the string CHAR i.e. number of

characters present in the given string.

Ex:-1) Find the number of characters in the string ‘Information Technology’

SELECT LENGTH (‘Information Technology’) FROM DUAL;

2) Display length of string SID, SNAME from Sailors along with their values. SELECT

SID, LENGTH (SID), SNAME, LENGTH (SNAME) FROM SAILORS;

35

10) SUBSTR (CHAR, M, N) :- It returns substring from CHAR string starting with

index M & gives N characters.

Ex : Display boats information by starting their names with 3rd character & show only 4 characters.

SELECT BID, SUBSTR (BNAME, 3, 4), COLOR FROM BOATS;

11) INSTR (CHAR1, CHAR2, M, N) :- It searches CHAR1 beginning with Mth

character for Nth occurrence of CHAR2 and returns the position after character in

CHAR1.

If N is negative value, then it searches backwards from the end of CHAR1. The

default value of M & N is 1.

Ex : Display the index of string 'AB' after 2nd character & 3rd occurrence in the

given string 'ABCDABCDABABAB'. SELECT INSTR ('ABCDABCDABABAB','AB', 2, 3)

FROM DUAL;

12) TRANSLATE (CHAR, FROM, TO) :- It returns characters with all occurrences of

each character in FROM replaced by its corresponding character in TO.

Ex :1) Replace ‗A' with ‗D' in the given string ‗ABCDABCDABABAB'. SELECT

TRANSLATE ('ABCDABCDABABAB','A','B') FROM DUAL;

13) REPLACE (CHAR, S, R) :- It returns characters with every occurrences of S

replaced with R. If R is not given or NULL, all occurrences of S are removed or

deleted.

36

Ex :1) Display BNAME by replacing ‘DA’ with ‘MA’.

SELECT REPLACE (BNAME, ‘DA’, ‘MA’) FROM BOATS;

Date Functions :-

1) SYSDATE :- Displays the system date for a system.

SELECT SYSDATE FROM SELECT TO_CHAR (SYSDATE, 'DD-MON-YYYY HH:MI:SS')

FROM dual;

2) NEXT_DAY (D, DAY) :- Displays next date on DAY after date D. Ex: Display date

on Thu after 20th Feb, 2018.

SELECT NEXT_DAY (’20-FEB-2018’, ‘THU’) FROM DUAL

3) ADD_MONTHS (D, N) :- Returns a date after adding a specified day D with

specified number of months N.

Ex: Display SID, Day of Reservation by adding 20 months to given day.

SELECT SID, DAY, ADD_MONTHS (DAY, 20) FROM RESERVES;

4) LAST_DAY(D) :- Returns the date corresponding to last day of the month.

Ex: Display Sname, Day of Reservation and date corresponding to last date of the

month.

SELECT S.SNAME, DAY, LAST_DAY (DAY) FROM SAILORS S, RESERVES R WHERE

S.SID = R.SID;

37

5) MONTHS_BETWEEN (D1, D2) :- Returns number of months between given two

dates D1 & D2.

Ex: Display SID, Day of Reservation and months between System Date & day of

reservation.

SELECT SID, DAY, MONTHS_BETWEEN (SYSDATE, DAY) FROM RESERVES;

6)LEAST (EXPR1 [, EXPR2, ... EXPR_N]) :- Returns the smallest value in a list of

expressions. Ex : 1) Find least value in 12, 53, 17, 2.

SELECT LEAST (12, 53, 17, 2) FROM DUAL;

7) GREATEST (EXPR1 [, EXPR2, ... EXPR_N]) :- Returns the largest value in a list of

expressions

Ex : 1) Find least value in 12, 13, 17, 2.

SELECT GREATEST (12, 13, 17, 2) FROM DUAL;

8) TRUNC (NUMBER [, DECIMAL_PLACES]) :- returns a number truncated to a

certain number of decimal places.

Ex :- Truncate the 5632.98345 number to 0, 1, 2 decimal places.

SELECT TRUNC (5632.98345) FROM DUAL;

SELECT TRUNC (5632.98345, 1) FROM DUAL;

9) ROUND (NUMBER [, DECIMAL_PLACES]) :- Returns a number rounded to a

certain number of decimal places. SELECT ROUND (5632.98345) FROM DUAL;

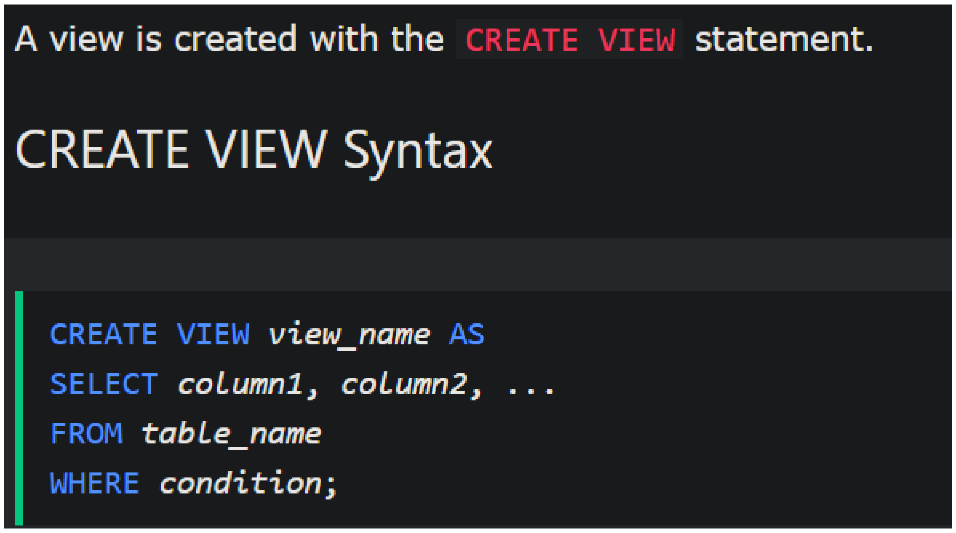
Practical – 11

Aim: Write SQL queries to create views and also apply different operations on views.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

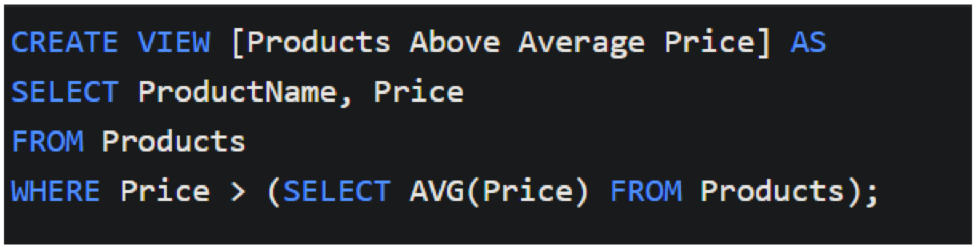
A view is created with the CREATE VIEW statement.



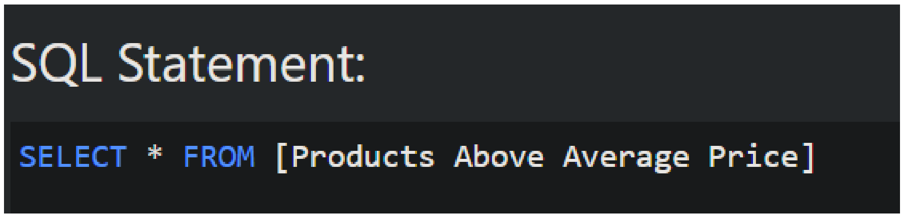
A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

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

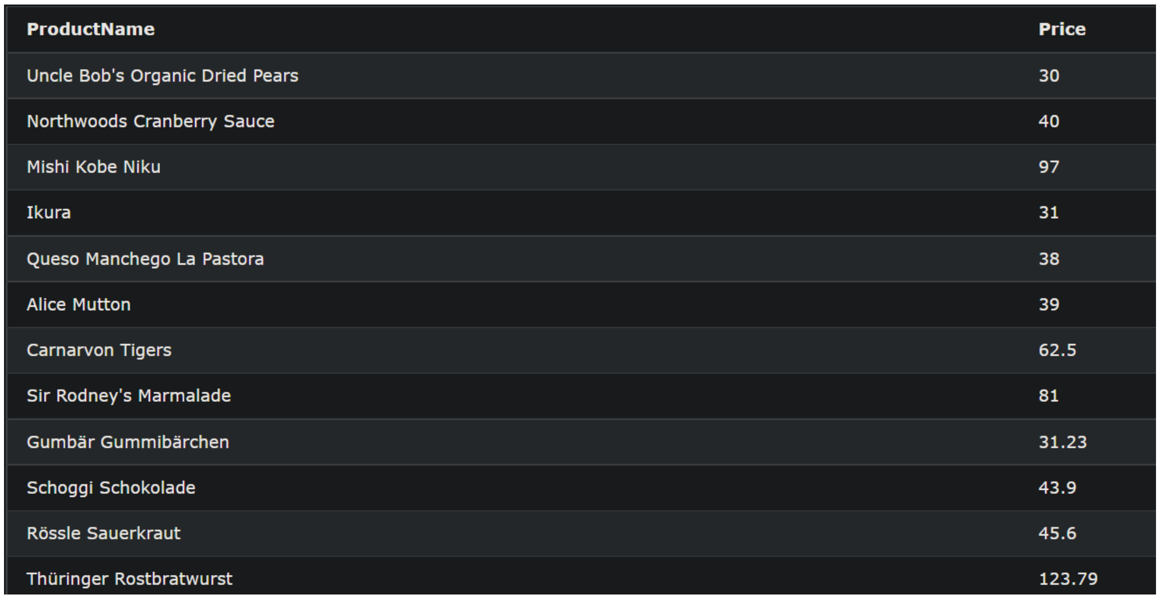


```sql
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```



SQL Statement:

```sql
SELECT * FROM [Products Above Average Price]
```



ProductName	Price
Uncle Bob's Organic Dried Pears	30
Northwoods Cranberry Sauce	40
Mishi Kobe Niku	97
Ikura	31
Queso Manchego La Pastora	38
Alice Mutton	39
Carnarvon Tigers	62.5
Sir Rodney's Marmalade	81
Gumbär Gummibärchen	31.23
Schoggi Schokolade	43.9
Rössle Sauerkraut	45.6
Thüringer Rostbratwurst	123.79