Primary key and foreign key syntax

```
CREATE TABLE Department
 ( DeptID int PRIMARY KEY,
DName varchar (50) NOT NULL
 );

CREATE TABLE Employee  (
EmpID int PRIMARY KEY,
Name varchar (50) NOT NULL,
Salary int NULL, DeptID int,
 FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
```

Group By and Having

```sql
select * from Employee;

select * from Department;
```

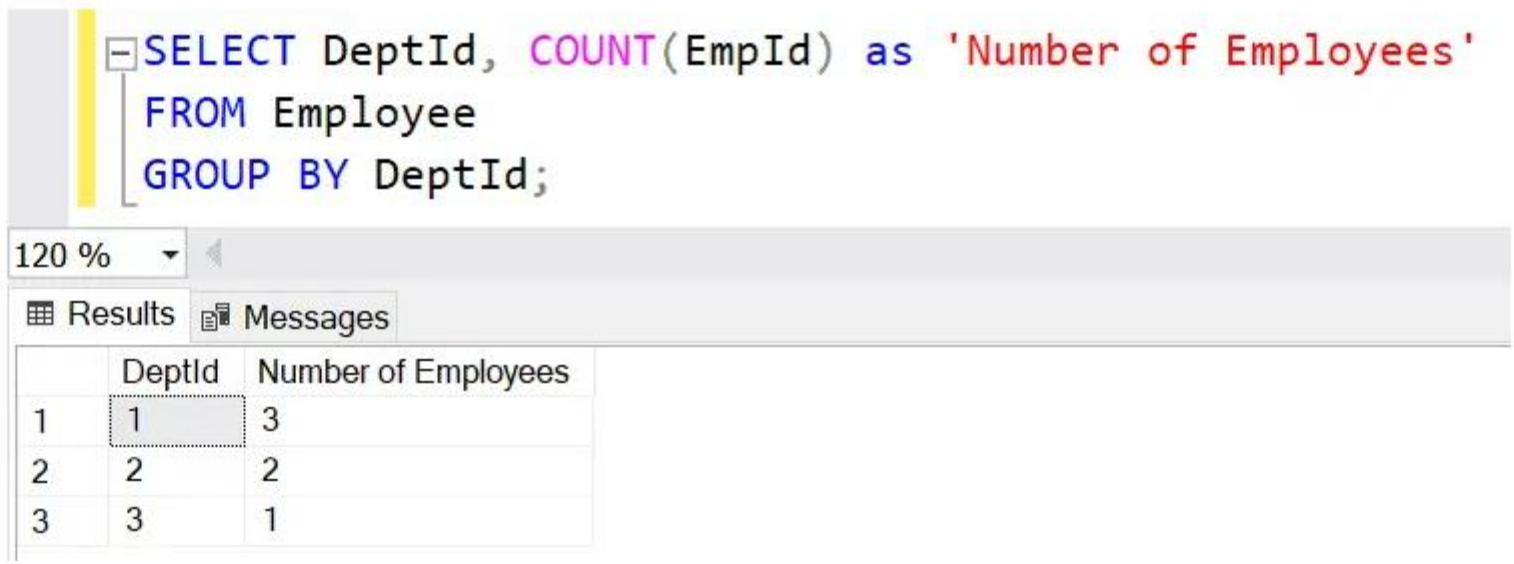
120 %

Results | Messages

	EmpId	FirstName	LastName	Email	PhoneNo	Salary	DeptId
1	1	John	King	john.king@abc.com	123.123.1834	33000	1
2	2	James	Bond	NULL	NULL	NULL	3
3	3	Neena	Kochhar	neena@test.com	123.456.4568	17000	2
4	4	Lex	De Haan	lex@test.com	123.456.4569	55000	1
5	5	Amit	Patel	NULL	NULL	18000	1
6	6	Abdul	Kalam	abdul@test.com	123.123.000	25000	2

	DeptId	DeptName
1	1	Finance
2	2	HR
3	3	Sales

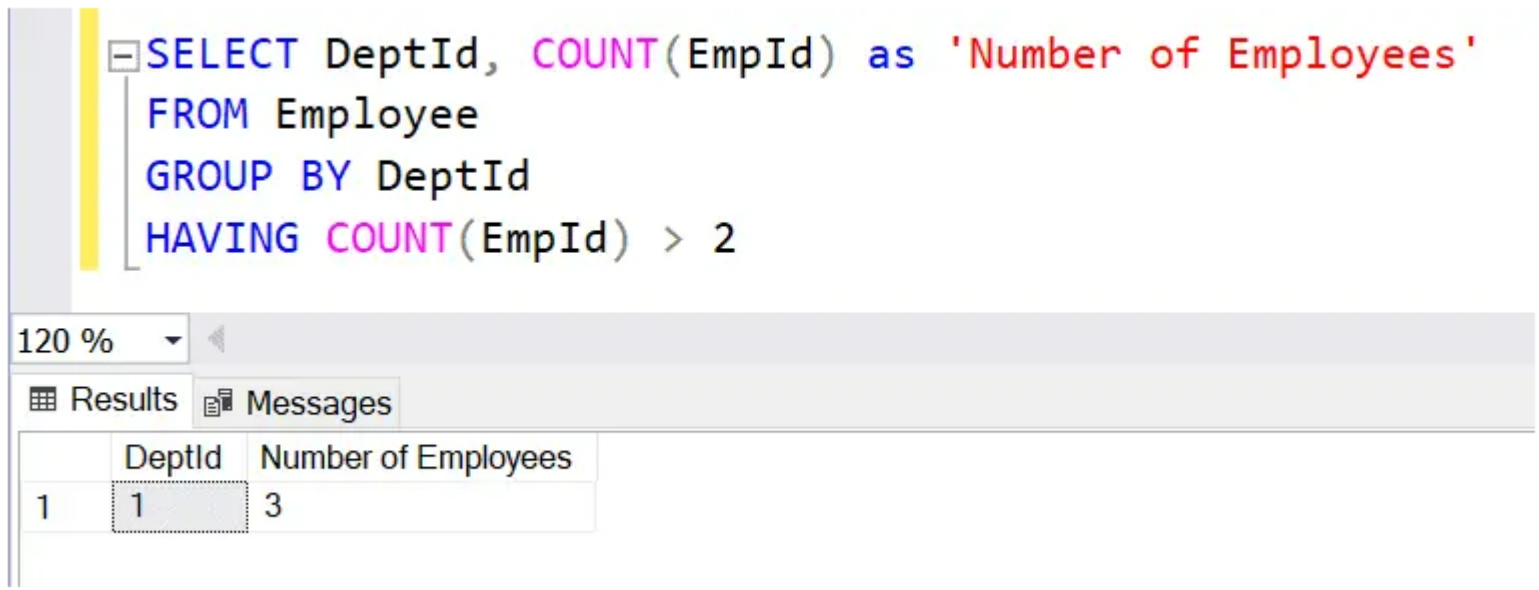
```sql
SELECT DeptId, COUNT(EmpId) as 'Number of Employees'
FROM Employee
GROUP BY DeptId;
```



	DeptId	Number of Employees
1	1	3
2	2	2
3	3	1

SQL Server: GROUP BY

```
SELECT DeptId, COUNT(EmpId) as 'Number of Employees'
FROM Employee
GROUP BY DeptId
HAVING COUNT(EmpId) > 2
```



```
SELECT DeptId, COUNT(EmpId) as 'Number of Employees'
FROM Employee
GROUP BY DeptId;
HAVING Salary > 15000
```

```
SELECT DeptId, COUNT(EmpId) as 'Number of Employees'
FROM Employee
GROUP BY DeptId
HAVING Salary > 15000
```

120 %

Messages

```
Msg 8121, Level 16, State 1, Line 8
Column 'Employee.Salary' is invalid in the HAVING clause because it is not contained in either an aggregate function
or the GROUP BY clause.
```

Example 1

```
SELECT * FROM Employee;
```

EmployeeId	Name	Gender	Salary	Department	Experience
5	Priya Sharma	Female	45000	IT	2 years
6	Rahul Patel	Male	65000	Sales	5 years
7	Nisha Gupta	Female	55000	Marketing	4 years
8	Vikram Singh	Male	75000	Finance	7 years
9	Aarti Desai	Female	50000	IT	3 years

```
SELECT Department, sum(Salary) as Salary
FROM employee
GROUP BY department;
```

Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

```
SELECT Department, sum(Salary) as Salary
FROM employee
GROUP BY department
HAVING SUM(Salary) >= 50000;
```

Department	Salary
Finance	75000
IT	95000
Marketing	55000
Sales	65000

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.
5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. We can implements this SQL clause in column operations.	6. We can implements this SQL clause in row operations.
7. It is a post-filter.	7. It is a pre-filter.
8. It is used to filter groups.	8. It is used to filter the single record of the table.

Examples

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	2000	Goa
202	Ankit	4000	Delhi
203	Bheem	8000	Jaipur
204 Ram	2000	Goa	
205	Sumit	5000	Delhi

```
SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City;
```

SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

```
SELECT SUM(Emp_Salary), Emp_City FROM Employee GROUP BY Emp_City HAVING SUM(Emp_Salary)>5000;
```

SUM(Emp_Salary)	Emp_City
9000	Delhi
8000	Jaipur

Examples


Table: Customers
customer_id | first_name | last_name | age | country
1 | John | Doe | 31 | USA
2 | Robert | Luna | 22 | USA
3 | David | Robinson | 22 | UK
4 | John | Reinhardt | 25 | UK
5 | Betty | Doe | 28 | UAE
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
COUNT(customer_id) | country
2 | UK
2 | USA

TRUNCATE

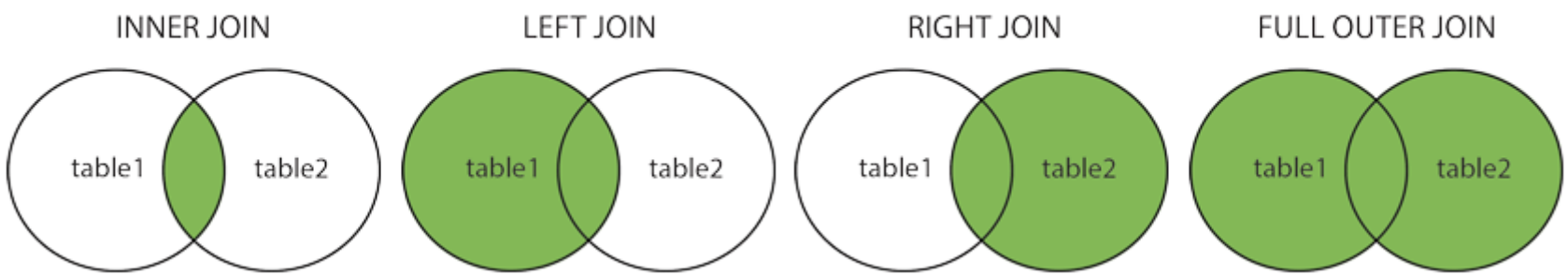
- **TRUNCATE** TABLE statement allows you to delete all data in a table.

- TRUNCATE TABLE student;
- Better performance than delete statement

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables
- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table



Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

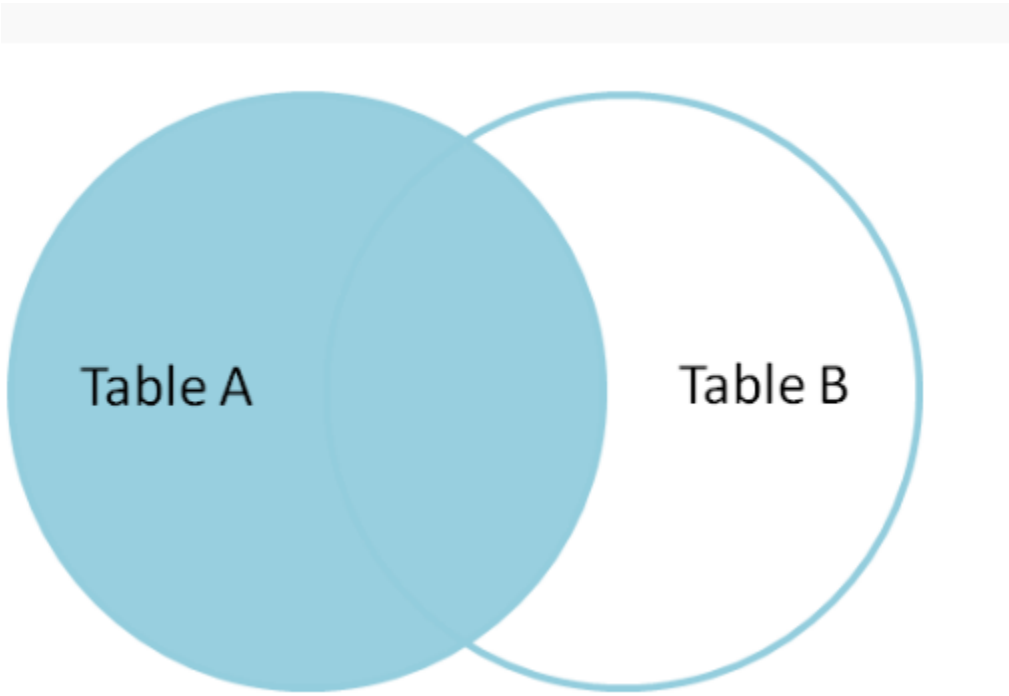
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

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

Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

Left Join

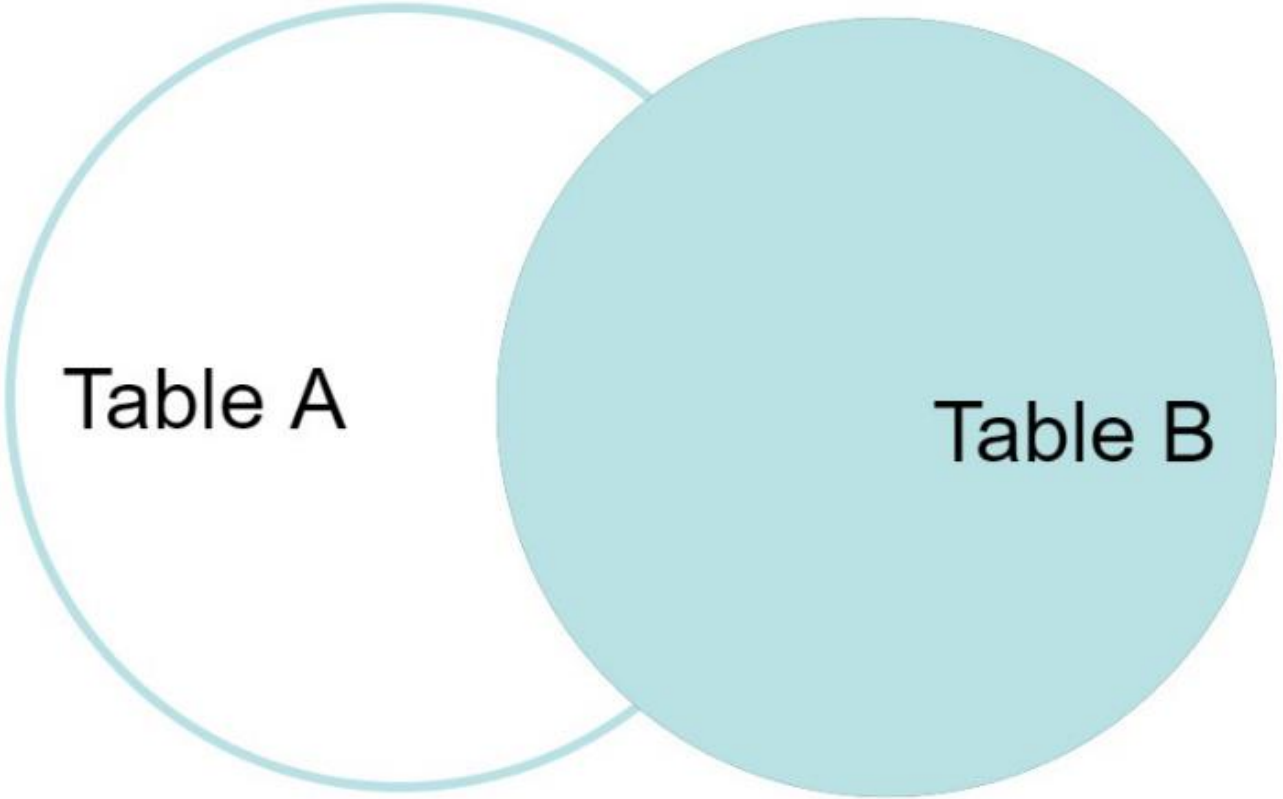


```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

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

Right Join

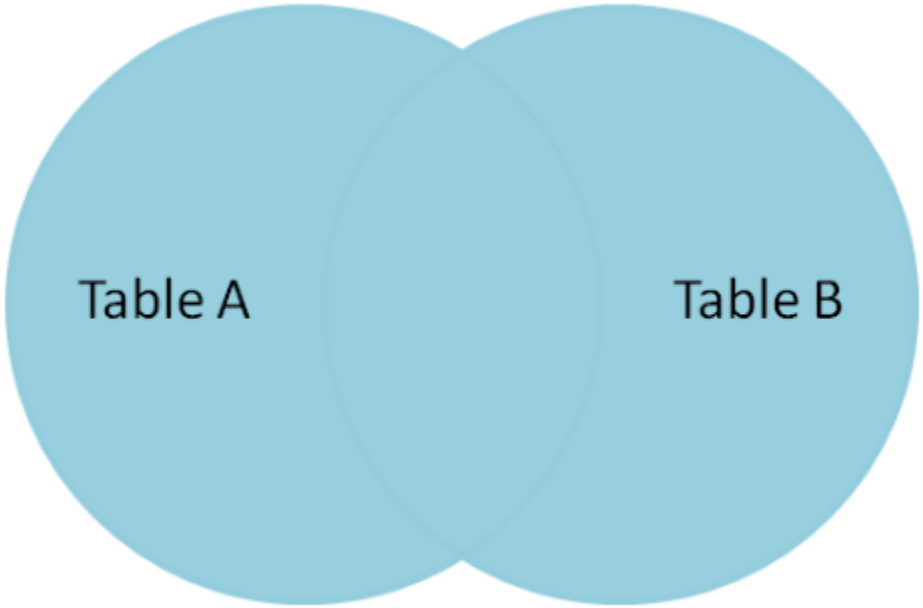


```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

Full Join



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

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

Union / Union All

```
SELECT column_1, column_2
FROM table_1
[WHERE condition]

UNION

SELECT column_1, column_2
FROM table_2
[WHERE condition]
```

```
SELECT column_1, column_2
FROM table_1
[WHERE condition]

UNION ALL

SELECT column_1, column_2
FROM table_2
[WHERE condition]
```

Union

name	location
John	Fair Street
Mary	Fair Street
Paul	West Street

name	location
Mary	Fair Street
Samantha	Fair Street
Paul	West Street

name	location
John	Fair Street
Mary	Fair Street
Paul	West Street
Samantha	Fair Street

Union ALL

name	location
John	Fair Street
Mary	Fair Street
Paul	West Street

name	location
Mary	Fair Street
Samantha	Fair Street
Paul	West Street

name	location
John	Fair Street
Mary	Fair Street
Paul	West Street
Mary	Fair Street
Samantha	Fair Street
Paul	West Street

- UNION – SELECT DISTINCT
- UNION ALL – if Rows return unique values

Copy data

create table destination_table like source_table;

CREATE TABLE Studentdetails LIKE Student;

insert into Studentdetails select * from Student

copy a part of data

insert into Studentdetails select * from Student where city=‘Mumbai'

Copy based on selecting columns

```
insert into destination_table_new
(address,city,pincode)
select address,city,pincode from source_table;
```

Tables can be of different structures

Still, Data’s are copied

Autoincrement

```
CREATE TABLE table_name
 ( column1 datatype NOT NULL AUTO_INCREMENT,
 column2 datatype [ NULL | NOT NULL ],
 ...
);
```

```
CREATE TABLE products (
product_id int(10) NOT NULL AUTO_INCREMENT,
 product_name varchar(150) NOT NULL,
PRIMARY KEY (`product_id`)
);

INSERT INTO products (product_name) VALUES(“Pens");

INSERT INTO products (product_name) VALUES(“Bags");
INSERT INTO products (product_name) VALUES(“Pencils");
```

```
SELECT * FROM products;
```

Product_id	product_name
1	Pens
2	Bags
3	Pencils

```
DELETE FROM products WHERE product_id = 2
INSERT INTO products (product_name) VALUES(“Boxes");
SELECT * FROM products;
```

Product_id	product_name
1	Pens
3	Pencils
4	Boxes

Exists and not exists

Get name of the customers who have placed minimum one order

```
SELECT fname, lname
FROM Customers
WHERE EXISTS (SELECT *
        FROM Orders
        WHERE Customers.customer_id = Orders.c_id);
```

Get name of the customers who have not placed any order

```
SELECT lname, fname
FROM Customer
WHERE NOT EXISTS (SELECT *
        FROM Orders
        WHERE Customers.customer_id = Orders.c_id);
```