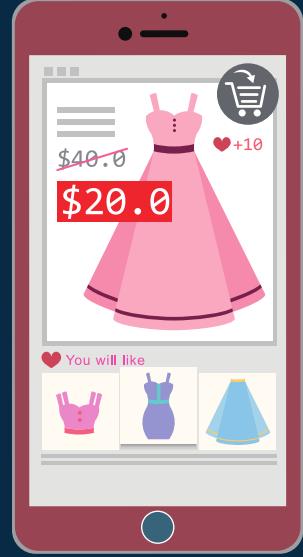


DATABASE MANAGEMENT SYSTEM PROJECT

ONLINE SHOP MANAGEMENT SYSTEM



Created by:

MAYANK
SRIVASTAVA (69)



NAMIT
GOYAL(76)



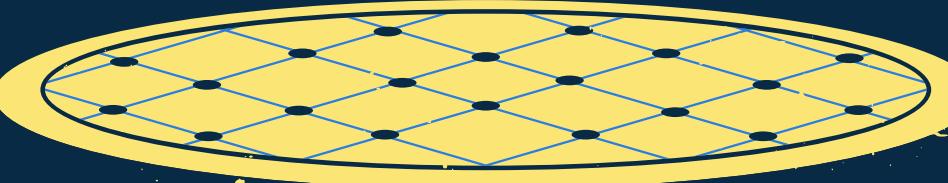
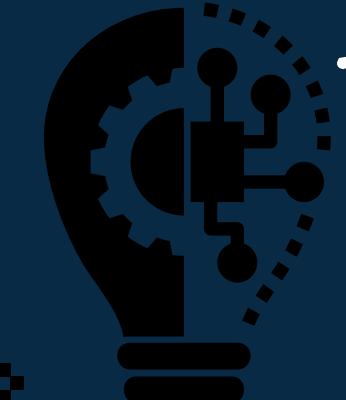
SHREYAS MISHRA
(104)



INTRODUCTION



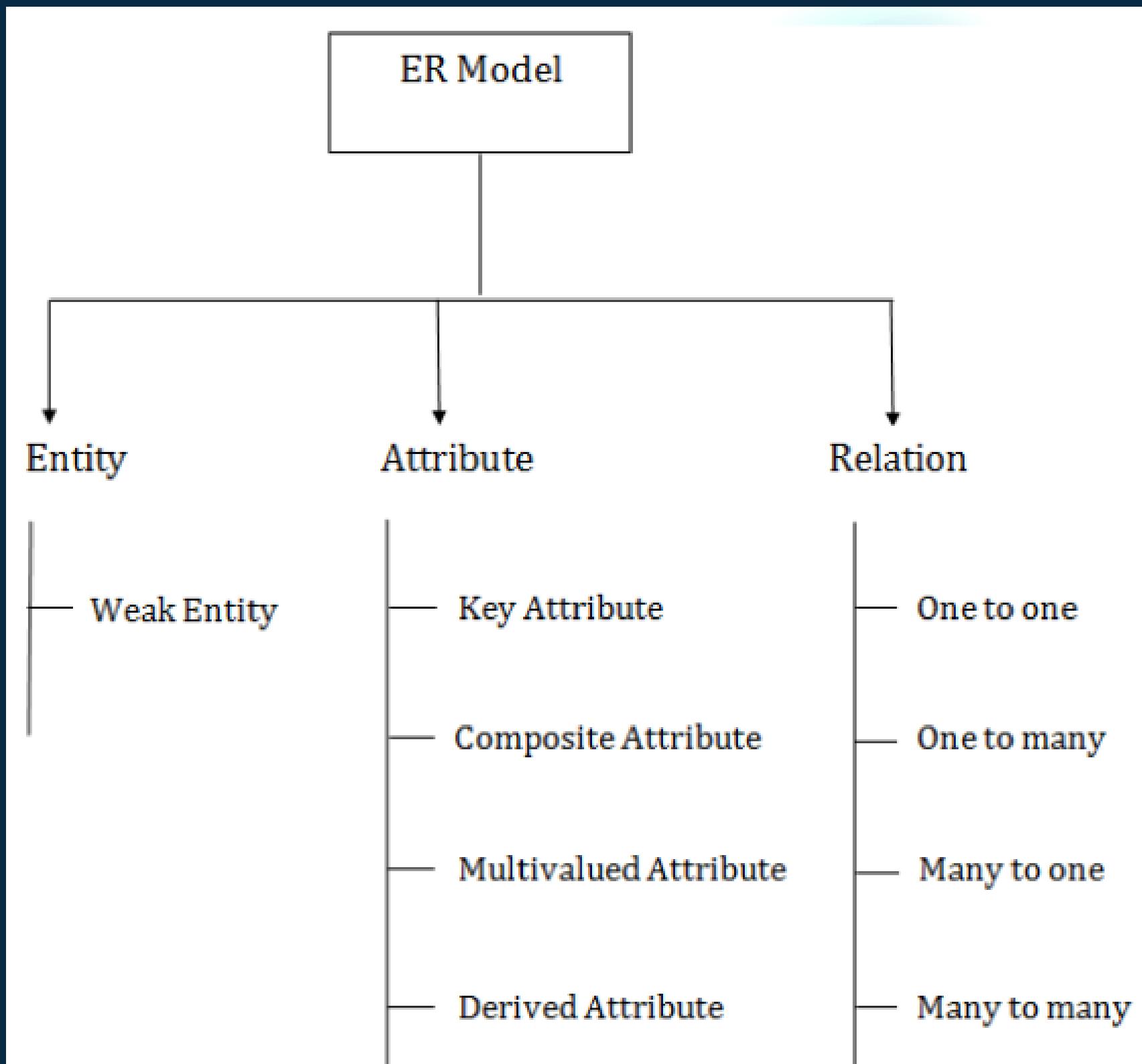
THIS PROJECT IS A WEB-BASED SHOPPING SYSTEM FOR AN EXISTING SHOP. ONLINE SHOPPING IS THE PROCESS WHEREBY CONSUMERS DIRECTLY BUY GOODS OR SERVICES FROM A SELLER IN REAL-TIME, SEE THE PRODUCTS AVAILABLE AND MAKE THE PAYMENT WITHOUT AN INTERMEDIARY SERVICE, OVER THE INTERNET. IT IS A FORM OF ELECTRONIC COMMERCE. THIS PROJECT IS AN ATTEMPT TO PROVIDE THE ADVANTAGES OF ONLINE SHOPPING TO CUSTOMERS OF A REAL SHOP.



Objective of Our Project

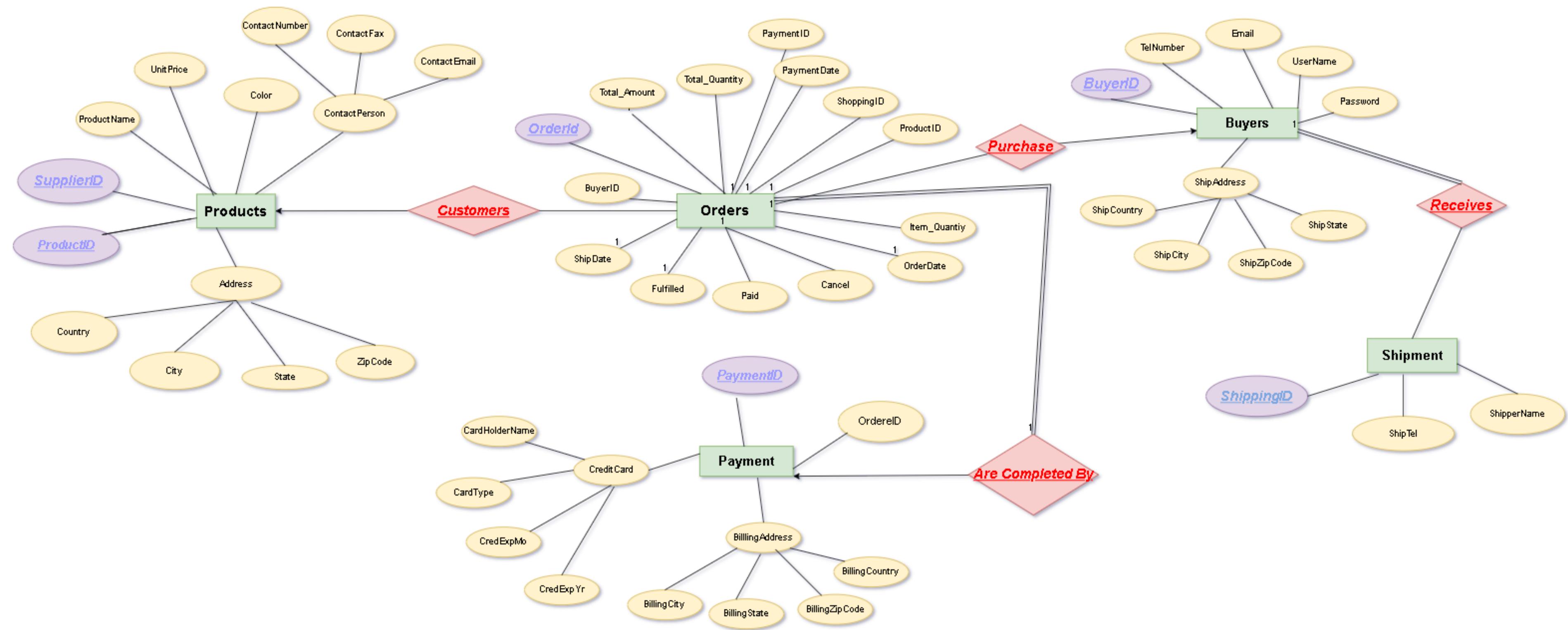
The purpose of the Shopping Management System is to act as an interface to manipulate the database records of various customers and suppliers related information such as product details, product availability, order details, shipment details and various other tasks performed by them and store them as any new record that has relation to the shop in any way. Our goal is to create a database and query system which can handle manipulation and retrieval of data as per the requirement of the user..

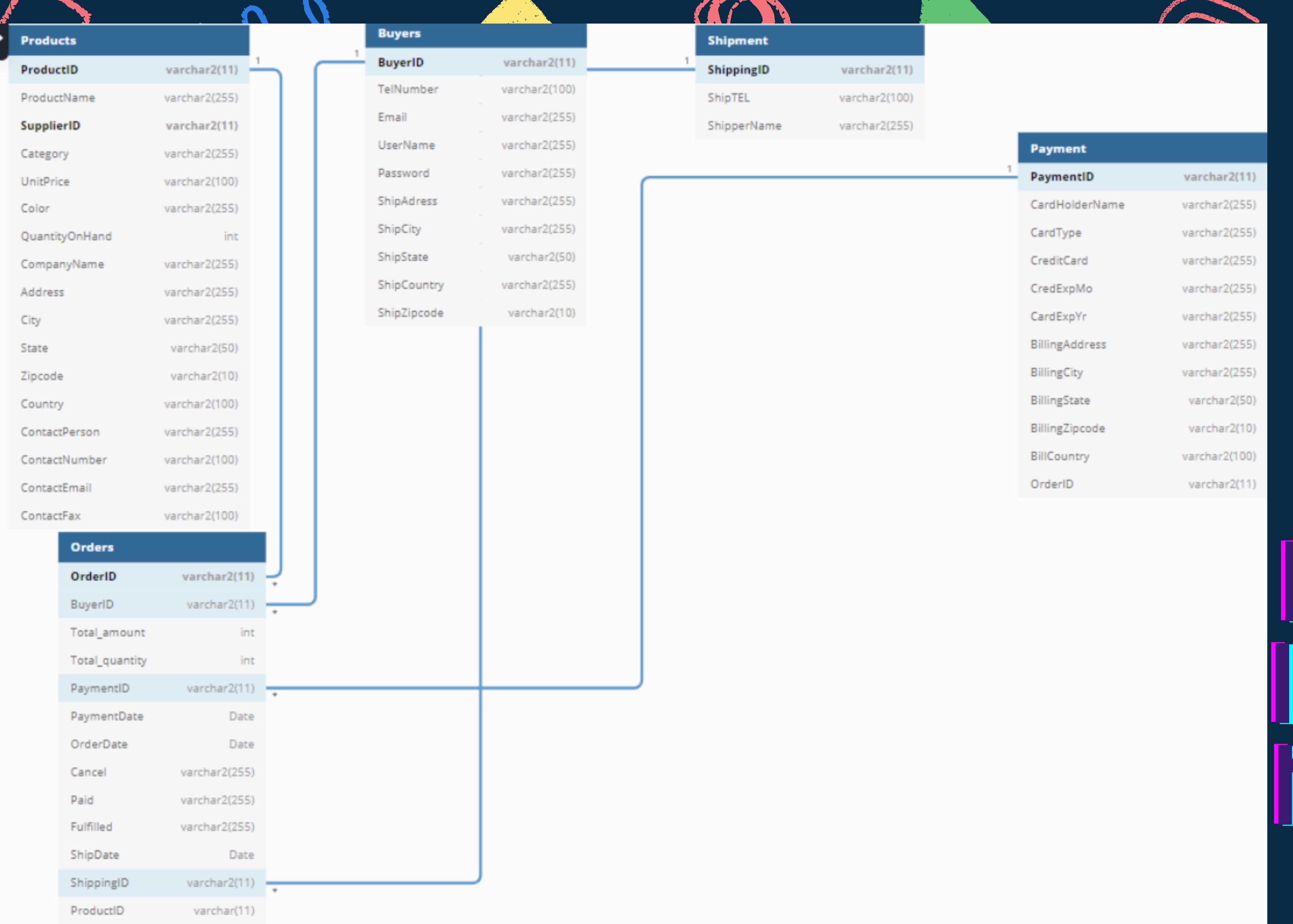
COMPONENTS OF ER DIAGRAM



ER DIAGRAM

AN ER DIAGRAM SHOWS THE RELATIONSHIP AMONG ENTITY SETS. AN ENTITY SET IS A GROUP OF SIMILAR ENTITIES AND THESE ENTITIES CAN HAVE ATTRIBUTES. IN TERMS OF DBMS, AN ENTITY IS A TABLE OR ATTRIBUTE OF A TABLE IN DATABASE, SO BY SHOWING RELATIONSHIP AMONG TABLES AND THEIR ATTRIBUTES, ER DIAGRAM SHOWS THE COMPLETE LOGICAL STRUCTURE OF A DATABASE.





RELATIONAL MODEL



AFTER
NORMALISATION



Shipment

ShippingID	varchar2(11)	PK
ShipTEL	varchar2(100)	NULL
ShipperName	varchar2(255)	NULL

Supplier

SupplierID	varchar2(11)	PK
CompanyName	varchar2(255)	NULL
Address	varchar2(255)	NULL
City	varchar2(255)	NULL
State	varchar2(50)	NULL
Zipcode	varchar2(10)	NULL
Country	varchar2(100)	NULL
ContactPerson	varchar2(255)	NULL
ContactNumber	varchar2(100)	NULL
ContactEmail	varchar2(255)	NULL
ContactFax	varchar2(100)	NULL

Products

ProductID	varchar2(11)	PK
ProductName	varchar2(255)	NULL
SupplierID	varchar2(11)	NULL FK
Category	varchar2(255)	NULL
UnitPrice	varchar2(100)	NULL
Color	varchar2(255)	NULL
QuantityOnHand	int	NULL

Orders

OrderID	varchar2(11)	PK FK
BuyerID	varchar2(11)	FK
Total_amount	int	NULL
Total_quantity	int	NULL
PaymentID	varchar2(11)	FK
PaymentDate	date	NULL
OrderDate	date	NULL
Cancel	varchar2(255)	NULL
Paid	varchar2(255)	NULL
Fulfilled	varchar2(255)	NULL
ShipDate	date	NULL
ShippingID	varchar2(11)	NULL FK

Order_Details

OrderID	varchar(11)	PK
ProductID	varchar(11)	NULL FK
ItemQuantity	int	NULL

Payment

PaymentID	varchar2(11)	PK
CardHolderName	varchar2(255)	NULL
CardType	varchar2(255)	NULL
CreditCard	varchar2(255)	NULL
CredExpMo	varchar2(255)	NULL
CardExpYr	varchar2(255)	NULL
BillingAddress	varchar2(255)	NULL
BillingCity	varchar2(255)	NULL
BillingState	varchar2(50)	NULL
BillingZipcode	varchar2(10)	NULL
BillCountry	varchar2(100)	NULL
OrderID	varchar2(11)	NULL

Buyers

BuyerID	varchar2(11)	PK
TelNumber	varchar2(100)	NULL
Email	varchar2(255)	NULL
UserName	varchar2(255)	NULL
Password	varchar2(255)	NULL
ShipAdress	varchar2(255)	NULL
ShipCity	varchar2(255)	NULL
ShipState	varchar2(50)	NULL
ShipCountry	varchar2(255)	NULL
ShipZipcode	varchar2(10)	NULL

Manufacture

Supplier	
SupplierID	INT
CompanyName	VARCHAR(45)
Address	VARCHAR(45)
City	VARCHAR(45)
State	VARCHAR(45)
Zipcode	INT
Country	VARCHAR(45)
ContactPerson	VARCHAR(45)
ContactNumber	INT
ContactEmail	VARCHAR(45)
ContactFax	VARCHAR(45)
Indexes	

Products	
ProductID	INT
ProductName	VARCHAR(45)
SupplierID	INT
{Category}	VARCHAR(45)
UnitPrice	FLOAT
Color	VARCHAR(45)
QuantityOnHand	INT
Supplier_SupplierID	INT
Indexes	
PRIMARY	

Sales

Order_Details	
Products_ProductID	INT
Orders_OrderID	INT
ItemQuantity	INT
Indexes	
PRIMARY	

Orders	
OrderID	INT
BuyerID	INT
Total_amount	FLOAT
Total_quantity	INT
PaymentID	INT
PaymentDate	DATETIME
OrderDate	DATETIME
Cancel	VARCHAR(45)
Paid	VARCHAR(45)
Fulfilled	VARCHAR(45)
ShipDate	DATETIME
ShippingID	INT
Buyer_BuyerID	INT
Indexes	
PRIMARY	

Buyer	
BuyerID	INT
TelNumber	INT
Email	VARCHAR(45)
UserName	VARCHAR(45)
Password	VARCHAR(45)
ShipAddress	VARCHAR(45)
ShipCity	VARCHAR(45)
ShipState	VARCHAR(45)
ShipCountry	VARCHAR(45)
ShipZipcode	INT
Indexes	
PRIMARY	

Logistics

Shipment	
ShippingID	INT
ShipTEL	INT
ShipperName	VARCHAR(45)
Orders_OrderID	INT
Orders_Buyer_BuyerID	INT
Indexes	
PRIMARY	

Paymant	
PaymantID	INT
CardHolderName	VARCHAR(50)
CardType	VARCHAR(45)
CreditCard	VARCHAR(45)
CredExpMo	VARCHAR(45)
CardExpYr	VARCHAR(45)
BillingAddress	VARCHAR(225)
BillingCity	VARCHAR(45)
BillingState	VARCHAR(45)
BillingZipcode	INT
BillingCountry	VARCHAR(45)
Orders_OrderID	INT
Indexes	
PRIMARY	

MMM TYPES OF SQL COMMANDS



DDL Command

DDL 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.

CREATE

used to create the database or its objects



ALTER

used to alter the structure of the database.

RENAME

is used to rename an object existing in the database.



DDL COMMANDS

I. CREATE

```
CREATE TABLE Order_Details (
    OrderID varchar(11) NOT NULL,
    ProductID varchar(11) default NULL,
    ItemQuantity int default NULL,
    PRIMARY KEY (OrderID);
```

HERE IS THE OUTPUT :

Add Column	Modify Column	Rename Column	Drop Column	Rename
Copy	Drop	Truncate	Create Lookup Table	Create App
Column Name	Data Type	Nullable	Default	Primary Key
ORDERID	VARCHAR2(11)	No	-	1
PRODUCTID	VARCHAR2(11)	Yes	NULL	-
ITEMQUANTITY	NUMBER	Yes	NULL	-

THIS IS USED TO CREATE THE DATABASE OR ITS OBJECTS (LIKE TABLE, INDEX, FUNCTION, VIEWS, STORE PROCEDURE AND TRIGGERS).

2.RENAME

```
ALTER TABLE Supplier  
RENAME COLUMN Zipcode to Zipc;
```

HERE IS THE OUTPUT :

Column Name	Data Type	Nullable	Default	Primary Key
SUPPLIERID	VARCHAR2(11)	No	-	1
COMPANYNAME	VARCHAR2(255)	Yes	NULL	-
ADDRESS	VARCHAR2(255)	Yes	NULL	-
CITY	VARCHAR2(255)	Yes	-	-
STATE	VARCHAR2(50)	Yes	NULL	-
ZIPC	VARCHAR2(10)	Yes	NULL	-
COUNTRY	VARCHAR2(100)	Yes	NULL	-
CONTACTPERSON	VARCHAR2(255)	Yes	NULL	-
CONTACTNUMBER	VARCHAR2(100)	Yes	NULL	-
CONTACTEMAIL	VARCHAR2(255)	Yes	NULL	-

THIS IS USED TO RENAME
AN OBJECT
(TABLE,COLUMN) EXISTING
IN THE DATABASE.

3.ALTER

```
ALTER TABLE Products  
ADD FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID);
```

4.TRUNCATE:
IS USED TO REMOVE ALL RECORDS FROM A TABLE,
INCLUDING ALL SPACES ALLOCATED FOR THE RECORDS ARE REMOVED.

USED TO MODIFY THE DATABASE BASICALLY

5.RENAME:
IS USED TO RENAME AN OBJECT (TABLE,COLUMN) EXISTING IN THE DATABASE.

MM

DML Command

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. THE COMMAND OF DML IS NOT AUTO-COMMITTED THAT MEANS IT CAN'T PERMANENTLY SAVE ALL THE CHANGES IN THE DATABASE. THEY CAN BE ROLLED BACK.

INSERT

used to insert data into a table.

UPDATE

used to update existing data within a table.

DELETE

used to delete records from a database table.

DML COMMAND

I. INSERT

EDIT	SUPPLIERID	COMPANYNAME	ADDRESS	CITY	STATE	ZIPC	COUNTRY	CONTACTPER
	39656653836	Shreyas Mishra	Sector 7 B , Chandigarh	Little Rock	Arkansas	72723	United States	Galvin Keller
	39599259968	P&G	4166 Id Avenue	Athens	GA	93824	United States	Xanthus Phillip
	30918186609	Nascetur	8496 Sed St.	Annapolis	Maryland	36447	United States	Montana Griffin
	31704489846	Luctus sit	747-1877 Fusce Street	Montpelier	Vermont	45116	United States	Breanna Tran
	12716365607	Sed	Ap #924-2093 Nisi Road	South Burlington	Vermont	61177	United States	Venus Tyson
	70268860023	Vivamusdapibus	P.O. Box 185, 7096 Fusce St.	Minneapolis	MN	12392	United States	Jamal Ward

```
INSERT INTO Supplier VALUES ('30918186609','Nascetur','8496 Sed  
INSERT INTO Supplier VALUES ('70268860023','Vivamusdapibus','P.O.  
INSERT INTO Supplier VALUES ('31704489846','Luctus sit','747-1877  
INSERT INTO Supplier VALUES ('68659281779','Mauris','6077 Id St.',  
INSERT INTO Supplier VALUES ('12716365607','Sed','Ap #924-2093 Nis  
INSERT INTO Supplier VALUES ('30751277561','Aliquam','Ap #559-963  
INSERT INTO Supplier VALUES ('39656653836','Phasellus','396-5140 S  
INSERT INTO Supplier VALUES ('39599259968','P&G','4166 Id Avenue',  
INSERT INTO Supplier VALUES ('12241497023','Praesent eu dui','P.O.  
INSERT INTO Supplier VALUES ('43636501698','tortor Integer','824-4
```

TO INSERT TABLES IN
DATABASE

2. UPDATE

```
UPDATE Supplier  
Set SupplierID=39599259968  
WHERE State='GA';
```

THIS COMMAND IS USED
TO UPDATE OR MODIFY
THE VALUE OF A COLUMN
IN THE TABLE.

3. DELETE
IT IS USED TO REMOVE
ONE OR MORE ROWS
FROM A TABLE.

OUTPUT

SUPPLIERID	COMPANYNAME	ADDRESS	CITY	STATE
39656653836	Phasellus	396-5140 Sodales Rd.	Little Rock	Arkansas
39599259968	P&G	4166 Id Avenue	Athens	GA
30918186609	Nascetur	8496 Sed St.	Annapolis	Maryland
31704489846	Luctus sit	747-1877 Fusce Street	Montpelier	Vermont

MM

DCL Command

DCL INCLUDES COMMANDS SUCH AS GRANT AND REVOKE WHICH MAINLY DEALS WITH THE RIGHTS, PERMISSIONS AND OTHER CONTROLS OF THE DATABASE SYSTEM.

GRANT

gives user's
access privileges to
database.

SYNTAX:- GRANT
SELECT,UPDATE ON
TABLE_NAME TO USER1,
USER2,...USERN;

REVOKE

withdraw user's
access privileges
given by using the
GRANT command

SYNTAX:- REVOKE SELECT,UPDATE
ON TABLE_NAME FROM USER1,
USER2,...USERN;

MM

TCL Command

TCL INCLUDES STATEMENTS THAT ARE USED TO MANAGE THE CHANGES THAT ARE MADE FROM DML STATEMENTS. IT ENHANCES THE TRANSACTIONAL NATURE OF ORACLE SQL.

COMMIT

aves any changes
made to the
database.

ROLLBACK

undoes any changes
made to the
database.

SAVEPOINT

creates a point in
your transaction to
which you can roll
back to.

COMMIT– Commit command is used to save all the transactions to the database.

Syntax:-

COMMIT; (After a transaction,if keyword COMMIT is written then all the transaction is saved permanently).

Example:- COMMIT;

SAVEPOINT– It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:-

SAVEPOINT savepoint_name; (Whenever any savepoint is executed then transaction is saved till that point).

Example:- SAVEPOINT savepoint_name;

· **ROLLBACK**– Rollback command is used to undo transactions that have not already been saved to the database.

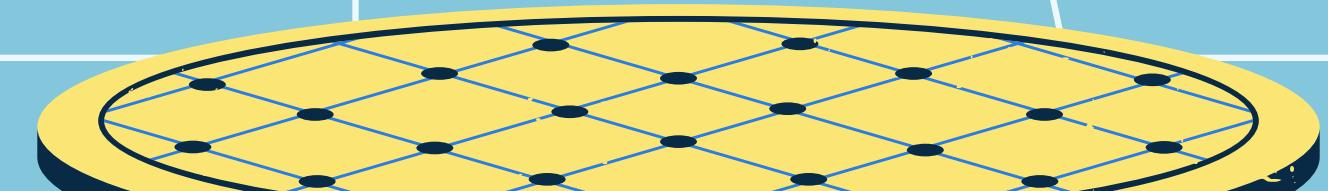
Syntax:- ROLLBACK; (If after a transaction, keyword rollback is executed, the database will be restored to its previous state).

Example:- ROLLBACK TO savepoint_name;



DATA CONSTRAINTS AND ITS TYPES

NAMING OF CONSTRAINT THE SQL CONSTRAINTS ARE AN INTEGRITY WHICH DEFINES SOME CONDITIONS THAT RESTRICT THE COLUMN TO REMAIN TRUE WHILE INSERTING OR UPDATING OR DELETING DATA IN THE COLUMN. CONSTRAINTS CAN BE SPECIFIED WHEN THE TABLE IS CREATED FIRST WITH A CREATE TABLE STATEMENT OR AT THE TIME OF MODIFICATION OF THE STRUCTURE OF AN EXISTING TABLE WITH ALTER TABLE STATEMENT. WE CAN NAME THE CONSTRAINT BY OUR OWN WHILE CREATING OR ALTERING THE TABLE TO GIVE THE CONSTRAINT A UNIQUE NAME.



TYPES OF CONSTRAINTS

Column Level Constraints

Column level constraints refer to a single column in the table and do not specify a column name (except check constraints). They refer to the column that they follow.

Table Level Constraints

Table level constraints refer to one or more columns in the table. Table level constraints specify the names of the columns to which they apply. Table level CHECK constraints can refer to 0 or more columns in the table.

Examples on Column and Table Level

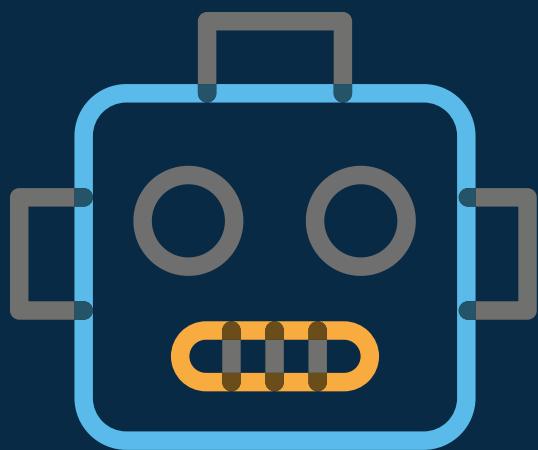
COLUMN LEVEL

```
create table orders(name char(10) not null,  
                    id integer references idtable(id),  
                    price integer check (price > 0));
```

Constraints

TABLE LEVEL

```
create table supplier(firstname char(20) not null,  
                      lastname char(20) not null,  
                      unique(firstname, lastname));
```



DIFFERENT CONSTRAINTS IN SQL



I. PRIMARY KEY

PRIMARY KEY IS A FIELD WHICH UNIQUELY IDENTIFIES EACH ROW IN THE TABLE. IF A FIELD IN A TABLE AS PRIMARY KEY, THEN THE FIELD WILL NOT BE ABLE TO CONTAIN NULL VALUES AS WELL AS ALL THE ROWS SHOULD HAVE UNIQUE VALUES FOR THIS FIELD.



EXAMPLE

```
CREATE TABLE Shipment (
    ShippingID varchar2(11) NOT NULL,
    ShipTEL varchar2(100) default NULL,
    ShipperName varchar2(255),
    PRIMARY KEY (ShippingID));
```



2. FOREIGN KEY

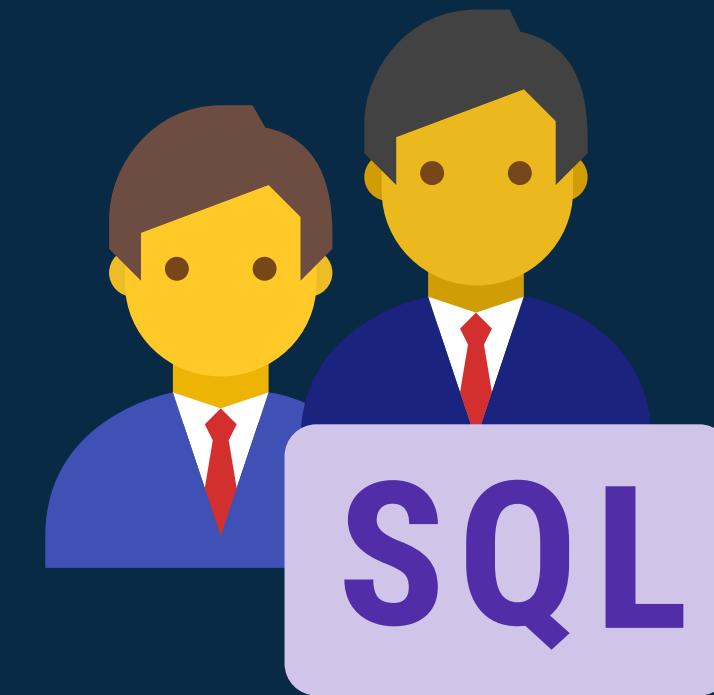
FOREIGN KEY IS A FIELD IN A TABLE WHICH UNIQUELY IDENTIFIES EACH ROW OF A ANOTHER TABLE. THAT IS, THIS FIELD POINTS TO PRIMARY KEY OF ANOTHER TABLE. THIS USUALLY CREATES A KIND OF LINK BETWEEN THE TABLES.

```
CREATE TABLE Orders (
    OrderID varchar2(11) NOT NULL,
    BuyerID varchar2(11) NOT NULL,
    Total_amount int default NULL,
    Total_quantity int default NULL,
    PaymentID varchar2(11) NOT NULL,
    PaymentDate Date,
    OrderDate Date,
    Cancel varchar2(255) default NULL,
    Paid varchar2(255) default NULL,
    Fulfilled varchar2(255) default NULL,
    ShipDate Date,
    ShippingID varchar2(11) default NULL,
    PRIMARY KEY (OrderID),
    Constraint Orders_FK1 FOREIGN KEY (BuyerID) REFERENCES Buyers(BuyerID),
    Constraint Orders_FK2 FOREIGN KEY (PaymentID) REFERENCES Payment(PaymentID),
    Constraint Orders_FK3 FOREIGN KEY (ShippingID) REFERENCES Shipment(ShippingID));
```

3. UNIQUE

THIS CONSTRAINT HELPS TO UNIQUELY IDENTIFY EACH ROW IN THE TABLE.

I.E. FOR A PARTICULAR COLUMN, ALL THE ROWS SHOULD HAVE UNIQUE VALUES. WE CAN HAVE MORE THAN ONE UNIQUE COLUMNS IN A TABLE.



```
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

4. NOT NULL

IF WE SPECIFY A FIELD IN A TABLE TO BE NOT NULL. THEN THE FIELD WILL NEVER ACCEPT NULL VALUE. THAT IS, YOU WILL BE NOT ALLOWED TO INSERT A NEW ROW IN THE TABLE WITHOUT SPECIFYING ANY VALUE TO THIS FIELD.

```
CREATE TABLE Payment (
    PaymentID varchar2(11) NOT NULL,
    CardHolderName varchar2(255) default NULL,
    CardType varchar2(255) default NULL,
    CreditCard varchar2(255),
    CredExpMo varchar2(255) default NULL,
    CardExpyr varchar2(255) default NULL,
    BillingAddress varchar2(255) default NULL,
    BillingCity varchar2(255),
    BillingState varchar2(50) default NULL,
    BillingZipcode varchar2(10) default NULL,
    BillCountry varchar2(100) default NULL,
    OrderID varchar2(11) default NULL,
    PRIMARY KEY (PaymentID));
```

5. CHECK

USING THE CHECK CONSTRAINT WE CAN SPECIFY A CONDITION FOR A FIELD, WHICH SHOULD BE SATISFIED

AT THE TIME OF ENTERING VALUES FOR THIS FIELD.

```
CREATE TABLE Order_Details (
    OrderID varchar(11) NOT NULL CHECK(OrderID>0),
    ProductID varchar(11) default NULL,
    ItemQuantity int default NULL,
    PRIMARY KEY (OrderID));
```

MySQL

6. DEFAULT

THIS CONSTRAINT IS USED TO PROVIDE A DEFAULT VALUE FOR THE FIELDS. THAT IS, IF AT THE TIME OF ENTERING NEW RECORDS IN THE TABLE IF THE USER DOES NOT SPECIFY ANY VALUE FOR THESE FIELDS THEN THE DEFAULT VALUE WILL BE ASSIGNED TO THEM.

```
CREATE TABLE Products (
    ProductID varchar2(11) NOT NULL,
    ProductName varchar2(255) default NULL,
    SupplierID varchar2(11) default NULL,
    Category varchar2(255) default NULL,
    UnitPrice varchar2(100) default NULL,
    Color varchar2(255) default NULL,
    QuantityOnHand int default NULL,
    PRIMARY KEY (ProductID));
```



BEHAVIORS OF FOREIGN KEY TABLE:-

MM

ON DELETE / UPDATE RESTRICT -

RESTRICT MEANS THAT ANY ATTEMPT TO DELETE AND/OR UPDATE THE PARENT WILL FAIL THROWING AN ERROR.

THIS IS

THE DEFAULT BEHAVIOUR IN THE EVENT THAT A REFERENTIAL

ACTION IS NOT EXPLICITLY SPECIFIED. FOLLOWING QUERY TRIES

TO DELETE A ROW FROM THE TABLE BILL, BUT THE ORDER STABLE REFERENCES THE BILL TABLE'S PRIMARY KEY, SO IT THROWS AN ERROR.



ON DELETE / UPDATE SET NULL

DELETE OR UPDATE THE ROW FROM THE PARENT TABLE, AND SET THE FOREIGN KEY COLUMN OR COLUMNS IN THE CHILD TABLE TO NULL. ORACLE DOESN' T SUPPORT ON UPDATE SET NULL.

ON UPDATE SET NULL

```
CREATE TABLE TABLE_NAME(COL1 DATATYPE,...,COLN FOREIGN KEY REFERENCES  
TABLE2_NAME(COLX) ON UPDATE SET NULL);
```

ON DELETE SET NULL

```
CREATE TABLE TABLE_NAME(COL1 DATATYPE,...,COLN FOREIGN KEY REFERENCES  
TABLE2_NAME(COLX) ON DELETE SET NULL);
```

~~MM~~

ON DELETE / UPDATE CASCADE -

ON DELETE CASCADE MEANS THAT IF THE PARENT RECORD IS DELETED, ANY CHILD RECORDS ARE ALSO DELETED. ONUPDATE CASCADE MEANS THAT IF THE PARENT PRIMARY KEY IS CHANGED, THE CHILD VALUE WILL ALSO CHANGE TO REFLECT THAT. ORACLE DOESN'T SUPPORT ON UP-DATE CASCADE.

```
CREATE TABLE TABLE_NAME(COL1 DATATYPE,...,  
COLN DATATYPE FOREIGN KEY REFERENCES  
TABLE2_NAME(COLX) ON UPDATE CASCADE);
```

BB

BEHAVIORS OF FOREIGN KEY TABLE:-

MM

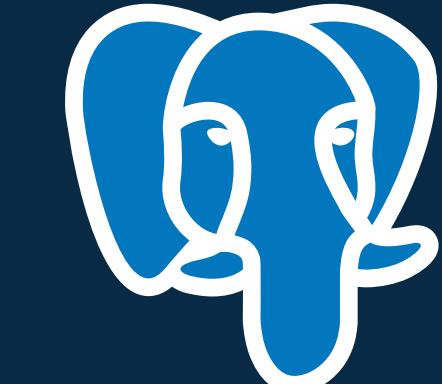
INTEGRITY CONSTRAINTS VIA ALTER TABLE COMMAND

```
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```



BB

SQL OPERATIONS AND FUNCTIONS

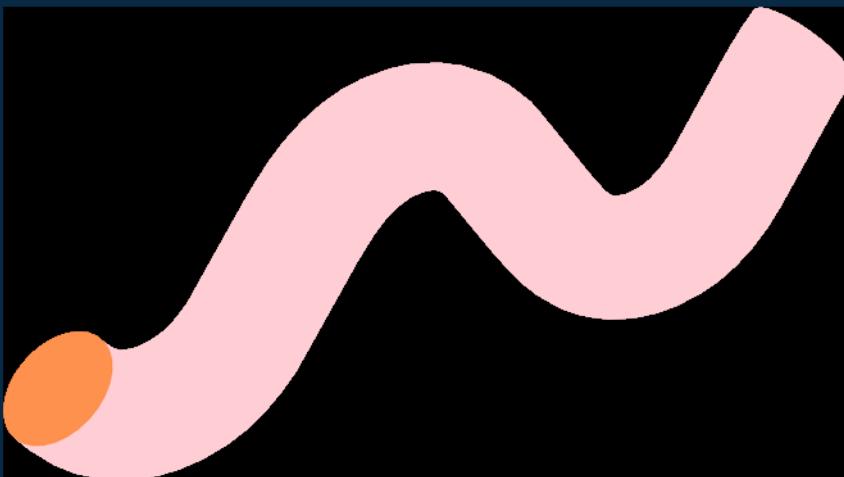


OPERATIONS

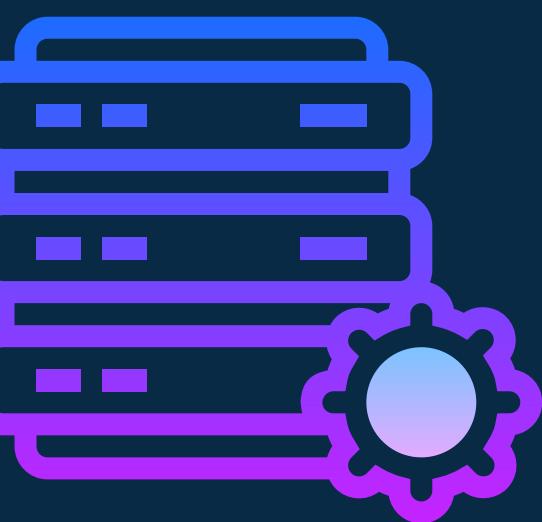
ARITHMETIC
OPERATORS

LOGICAL
OPERATORS

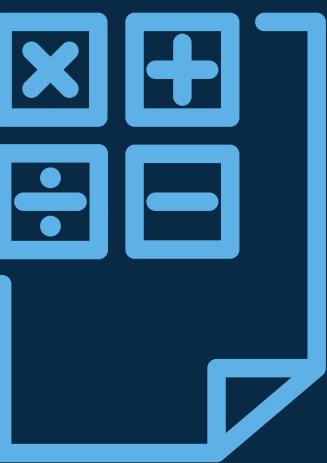
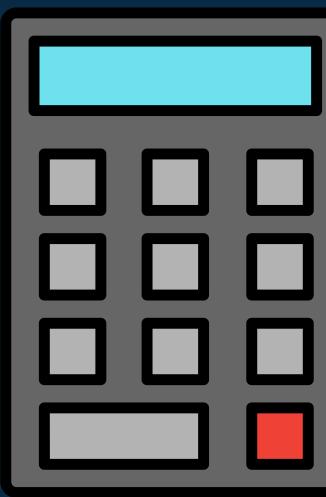
RELATIONAL
OPERATORS



ARITHMETIC OPERATORS



S.No	Operator	Description	Sample Query	Output
1.	+	Addition	Select 6 + 2;	8
2.	-	Subtraction	Select 6 - 2;	4
3.	*	Multiply	Select 6 * 2;	12
4.	/	Division	Select 6 / 2;	3
5.	%	Modulo	Select 6 % 2;	0



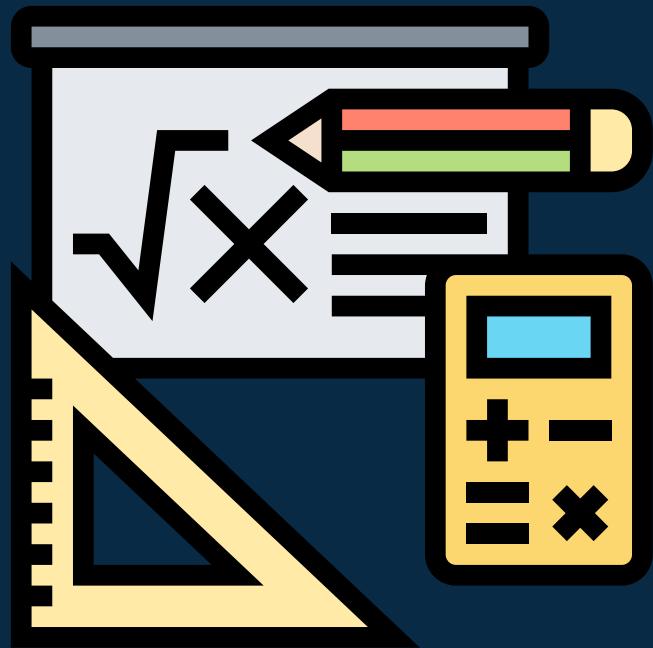
 

LOGICAL OPERATORS

Logical Operators:

Logical operators in SQL will return either true or false value.

Operators	Function	Example
AND	Check two conditions are true	Select * from emp where basic >= 10000 AND basic <= 20000;
OR	Check any of the two conditions are true	Select * from emp where basic >= 10000 OR dept = 'Sales';
NOT	Reversed the result of logical expression	Select * from emp where NOT(basic >= 10000 OR dept = 'Sales');

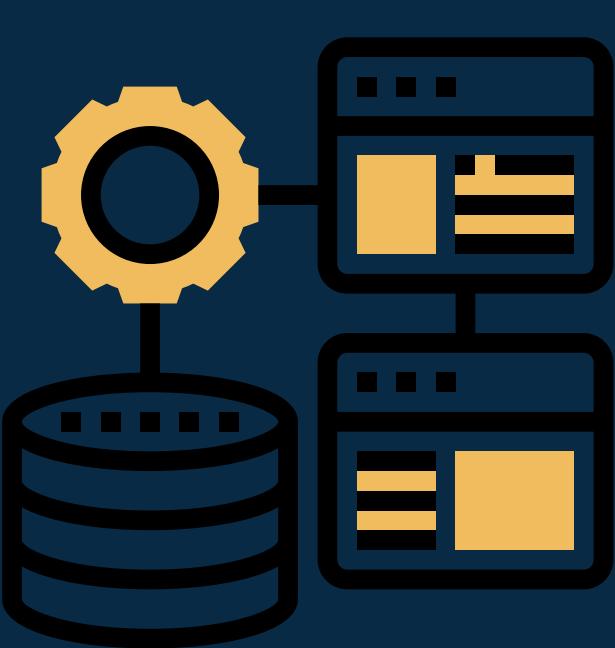
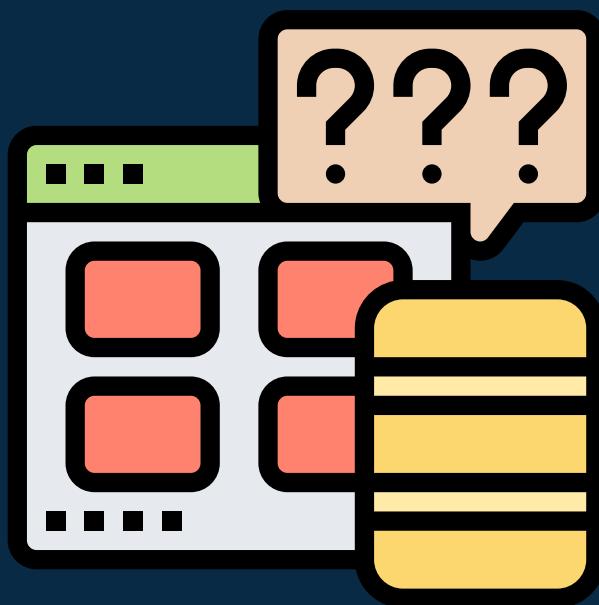




RELATIONAL OPERATORS



Operator	Description	Example
=	Equal to	$(x=y)$ is not true
!=	Equal or not	$(x \neq y)$ is true
< >	Not equal to	$(x < > y)$ is true
>	Greater than	$(x > y)$ is not true
<	Less than	$(x <= " " \text{ td} = " " >)$
>=	Greater than or equal to	$(x >= y)$ is not true
<=	Less than or equal to	$(x <= y)$ is true
!<	Not less than	$(x \not<= " " \text{ td} = " " >)$
!>	Not greater than	$(x \not> y)$ is true



EXAMPLES OF OPERATORS



| >

```
SELECT * FROM orders WHERE TOTAL_AMOUNT > ANY (SELECT TOTAL_AMOUNT FROM orders WHERE TOTAL_AMOUNT>=7);
```

HERE IS THE OUTPUT :

ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY
35764031654	94908631679	928	2
47217024154	26350935579	662	5
70701609058	89264121725	238	13
07261862587	12714972662	177	6
39878325779	72330070747	134	11
99251300121	81921795835	68	8
96945305380	42711312958	56	4
90817183713	6771139299	54	9
21942566549	50345881058	48	1

2>

```
SELECT * FROM orders WHERE TOTAL_QUANTITY BETWEEN 5 AND 20;
```

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History
ORDERID	BUYERID	TOTAL_AMOUNT	TOTAL_QUANTITY	
07261862587	12714972662	177	6	
89878325779	72330070747	134	11	
70701609058	89264121725	238	13	
05553373345	75521050479	22	7	
99251300121	81921795835	68	8	
90817183713	67711139299	54	9	
47217024154	26350935579	662	5	
7 rows returned in 0.02 seconds		Download		

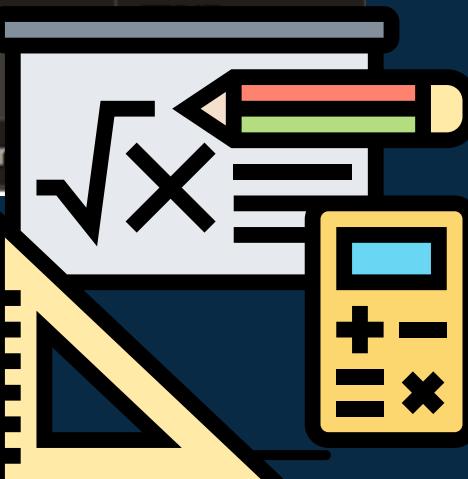


3>

```
select * from buyers
```

HERE IS THE OUTPUT :

BUYERID	TELNUMBER	EMAIL	USERNAME	PASSWORD	SHIPADDRESS	SHIPCITY	SHIPSTATE	SHIPCOUNTRY	SHIPZIPCODE
94908631679	1-772-737-2701	molestie@idantedictum.ca	Clark	JFA79CQJ8XU	P.O. Box 504, 7630 Lacus Rd.	Viransehir	Sanliurfa	Liberia	65242
89264121725	1-905-188-8108	mi.pede@non.edu	Noble	AOF35ISM0QN	2510 Tellus Avenue	Joliet	IL	Christmas Island	86876
50345881058	1-850-899-9543	euismod.Etiam@etultrices.edu	Tashya	PDE87LWL3PY	8200 Quis, Street	Vezirköprü	Samsun	Guinea-Bissau	M5Y 1Z0
81921795835	1-583-615-6760	velit.dui@sapien.org	Aretha	ANP81YWC9CU	Ap #516-795 Nisi Road	Mildura	VIC	Singapore	4505
26350935579	1-865-217-7844	ac.mattis@felispurus.org	Abbot	VSR52BPA7OY	951-727 A Street	Laramie	WY	Morocco	007708
12714972662	1-102-293-5153	Quisque@elitdictumeu.net	Vance	RST79SIK9CB	1341 In St.	Jerez de la Frontera	AN	Singapore	53506
75521050479	1-959-804-3019	id.libero.Donec@gmail.com	Matthew	PWN64YUG3AJ	253-4847 Rutrum Ave	Callander	PE	Mongolia	ZN53 8TG
72330070747	1-184-195-5645	gravida.sagittis.Duis@magna.ca	Dominic	OFX66VPY2TS	Ap #976-802 Dignissim Street	Islahiye	Gaz	Finland	
42711312958	1-317-594-7754	Nunc.ac@molestieSedid.ca	Lynn	JEE58NSU1DO	7377 Cras Rd.	Gotzis	Vbg	Seychelles	
6771139299	1-632-629-5645	gravida.non@urnasuscipitnonummy.net	Dustin	BSW81VNC4BY	Ap #972-7019 Nulla Street	Orangeville	ON	Marshall Islan	



4>

LIKE OPERATION

```
Select Distinct(ProductName) from Products  
where Products.ProductName Like 'S%';
```

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History
Shampoo				
Sets	.			

2 rows returned in 0.01 seconds [Download](#)

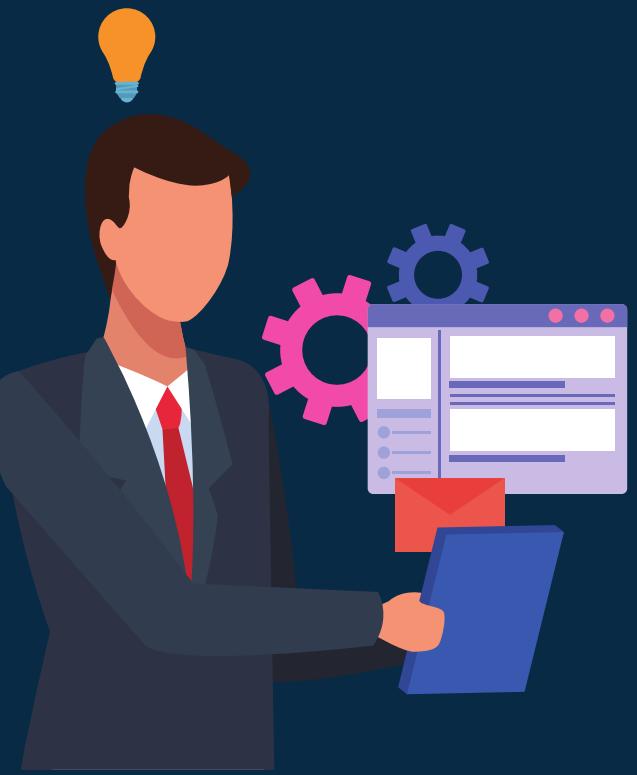
STATEMENTS



5>

IN OPERATION

```
SELECT BuyerID,Email,UserName,ShipCountry  
FROM Buyers  
WHERE ShipCountry IN ('Singapore', 'Morocco');
```



HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History
BUYERID	EMAIL	USERNAME	SHIPCOUNTRY	
81921795835	velit.dui@sapien.org	Aretha	Singapore	
12714972662	Quisque@elitdictumeu.net	Vance	Singapore	
26350935579	ac.mattis@felispurus.org	Abbot	Morocco	

3 rows returned in 0.02 seconds [Download](#)

JOIN

OPERATIONS

A SQL JOIN STATEMENT IS USED TO COMBINE DATA OR ROWS FROM TWO OR MORE TABLES BASED ON A COMMON FIELD BETWEEN THEM.

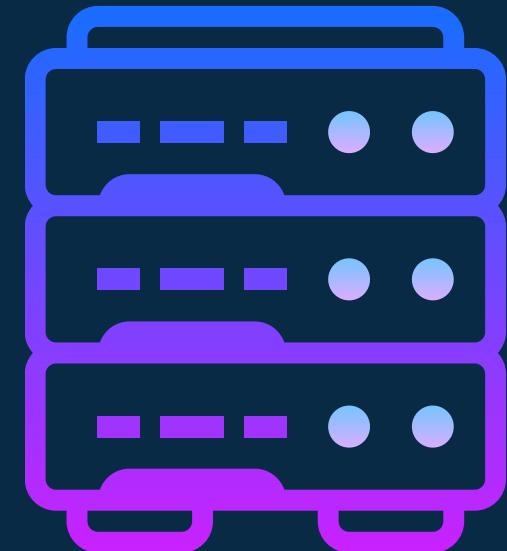
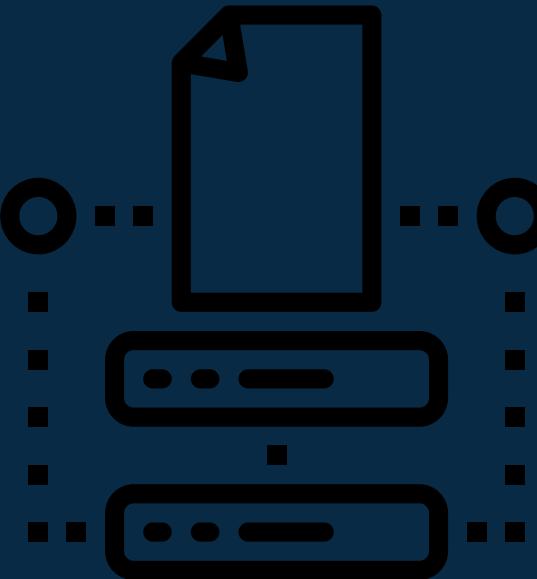
TYPES OF JOINS

CROSS JOIN

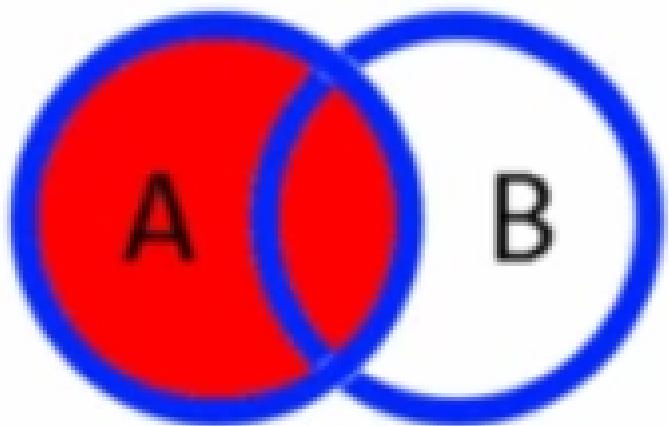
NATURAL JOIN

INNER JOIN

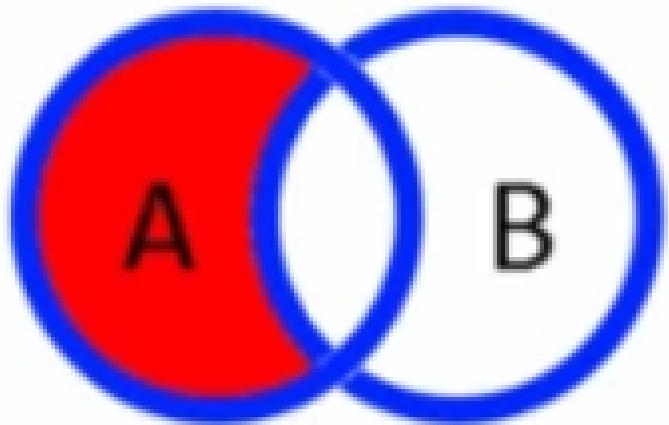
OUTER JOIN



LEFT OUTER JOIN

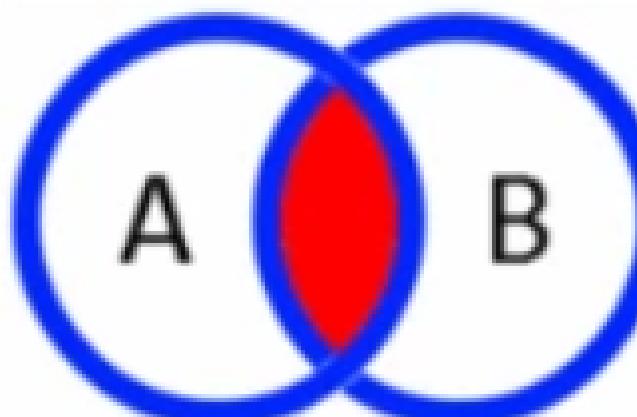


```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.KEY = b.KEY
```



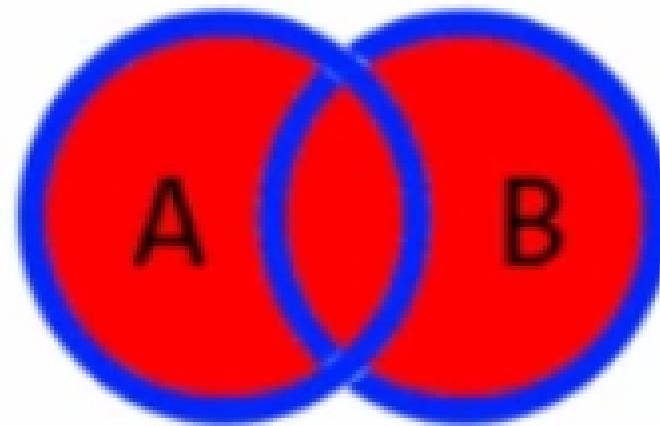
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.KEY = b.KEY  
WHERE b.KEY IS NULL
```

INNER JOIN

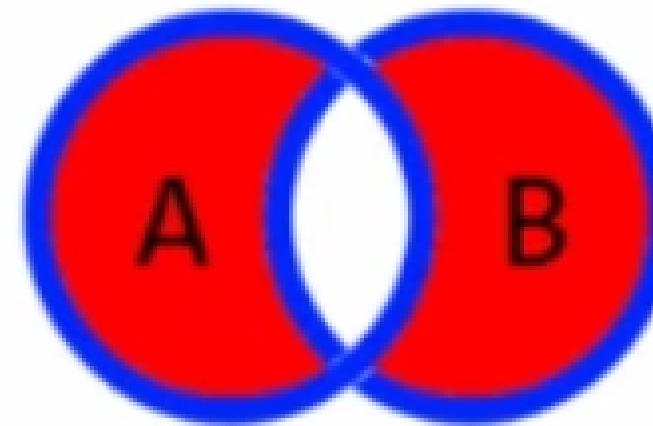


```
SELECT *  
FROM TableA a  
INNER JOIN TableB b  
ON a.KEY = b.KEY
```

FULL OUTER JOIN

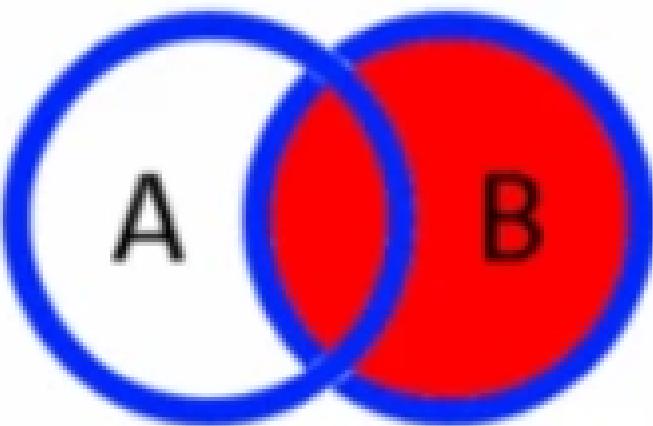


```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.KEY = b.KEY
```

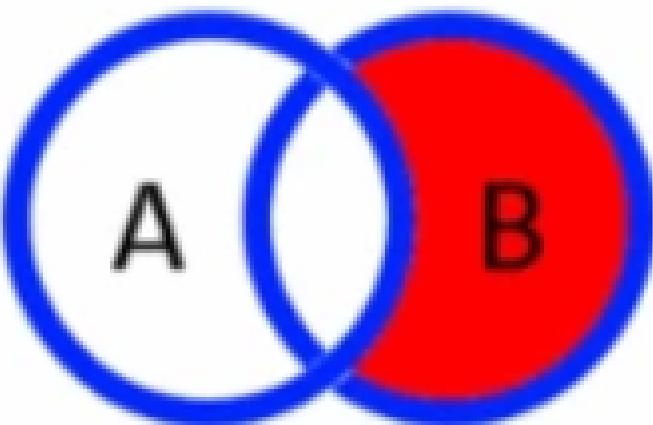


```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.KEY = b.KEY  
WHERE a.KEY IS NULL  
OR b.KEY IS NULL
```

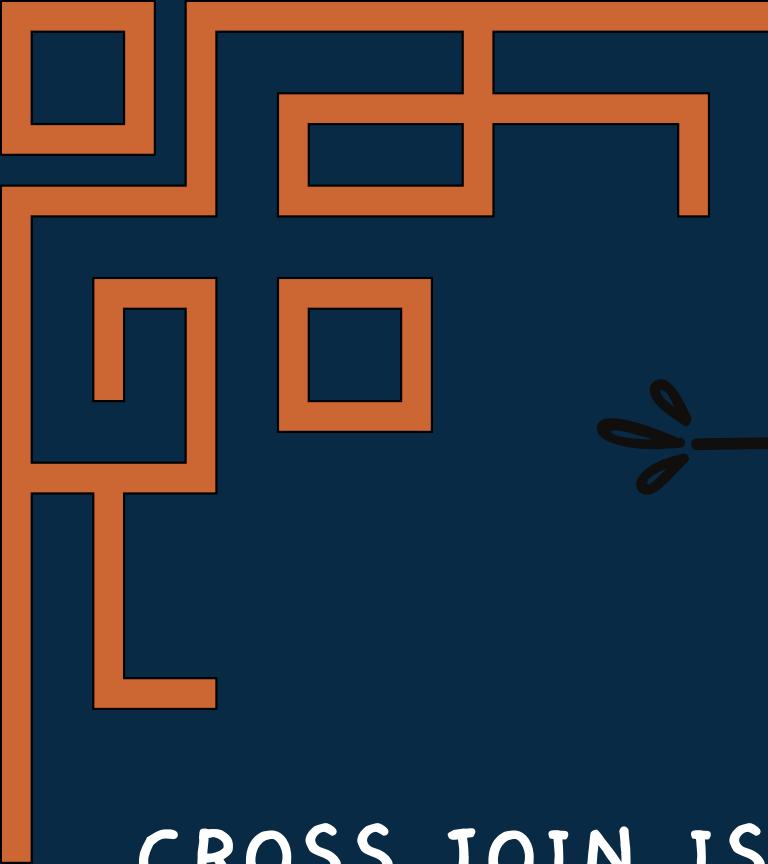
RIGHT OUTER JOIN



```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.KEY = b.KEY
```



```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.KEY = b.KEY  
WHERE a.KEY IS NULL
```



CROSS JOIN



CROSS JOIN IS USED TO COMBINE EACH ROW OF THE FIRST TABLE WITH EACH ROW OF THE SECOND TABLE. IT IS ALSO KNOWN AS THE CARTESIAN JOIN SINCE IT RETURNS THE CARTESIAN PRODUCT OF THE SETS OF ROWS FROM THE JOINED TABLES.

OUTPUT ON NEXT PAGE

```
SELECT *
FROM Products
CROSS JOIN Orders
WHERE Products.ProductName = 'Lotion' OR Products.ProductName = 'Primer';
```

HERE IS THE OUTPUT :

NATURAL JOIN

NATURAL JOIN JOINS TWO TABLES BASED ON SAME ATTRIBUTE NAME AND DATA TYPES. THE RESULTING TABLE WILL CONTAIN ALL THE ATTRIBUTES OF BOTH THE TABLE BUT KEEP ONLY ONE COPY OF EACH COMMON COLUMN.



EXAMPLE

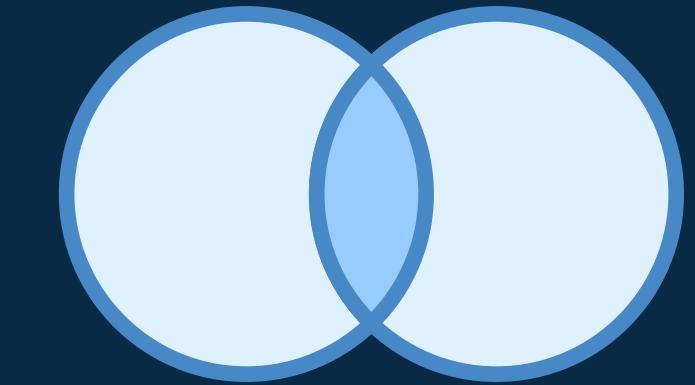
```
SELECT *  
FROM Orders  
NATURAL JOIN Buyers  
Where Total_amount>=6;
```

OUTPUT ON NEXT PAGE

HERE IS THE OUTPUT :

BUYERID	ORDERID	TOTAL_AMOUNT	TOTAL_QUANTITY	PAYMENTID	PAYMENTDATE
94908631679	85764031654	928	2	31863280913	11/25/2021
89264121725	70701609058	238	13	38387495295	12/15/2021
81921795835	99251300121	68	8	92384433306	11/17/2021
6771139299	90817183713	54	9	49871562602	11/17/2021
50345881058	21942566549	48	1	57773141099	10/12/2021
12714972662	07261862587	177	6	26660945790	10/20/2001

INNER JOIN



INNER JOIN JOINS TWO TABLE ON THE BASIS OF THE COLUMN WHICH IS EXPLICITLY SPECIFIED IN THE ON CLAUSE. THE RESULTING TABLE WILL CONTAIN ALL THE ATTRIBUTES FROM BOTH THE TABLES INCLUDING COMMON COLUMN ALSO.

```
select Buyers.BuyerID,Orders.ShipDate from  
Buyers inner join Orders on  
Orders.BuyerID = Buyers.BuyerID
```

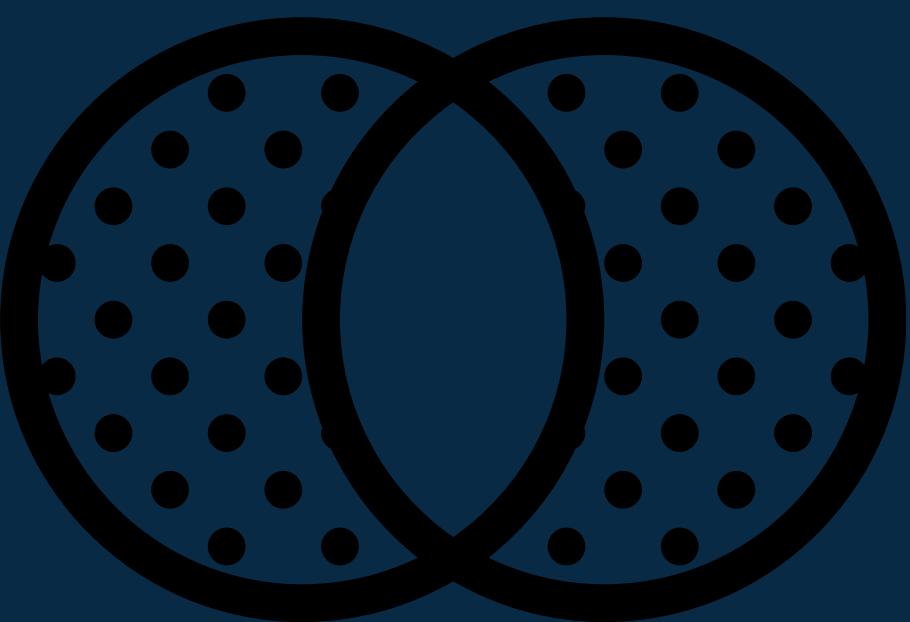
OUTPUT ON NEXT PAGE

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History
BUYERID				SHIPDATE
12714972662				10/27/2021
50345881058				10/15/2021
42711312958				10/31/2021
6771139299				11/25/2021
81921795835				11/22/2021
72330070747				11/16/2021
94908631679				11/28/2021
89264121725				12/20/2021
75521050479				01/05/2021
26350935579				11/28/2021



LEFT JOIN



INNER JOIN JOINS TWO TABLE ON THE BASIS OF THE COLUMN WHICH IS EXPLICITLY SPECIFIED IN THE ON CLAUSE. THE RESULTING TABLE WILL CONTAIN ALL THE ATTRIBUTES FROM BOTH THE TABLES INCLUDING COMMON COLUMN ALSO.

OUTPUT ON NEXT PAGE

```
select ShipCountry,count(*) as bestsell from Buyers left  
join Orders on Buyers.BuyerID = Orders.BuyerID group by ShipCountry;
```

HERE IS THE OUTPUT :

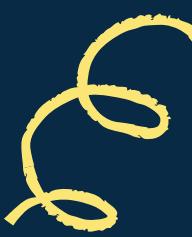
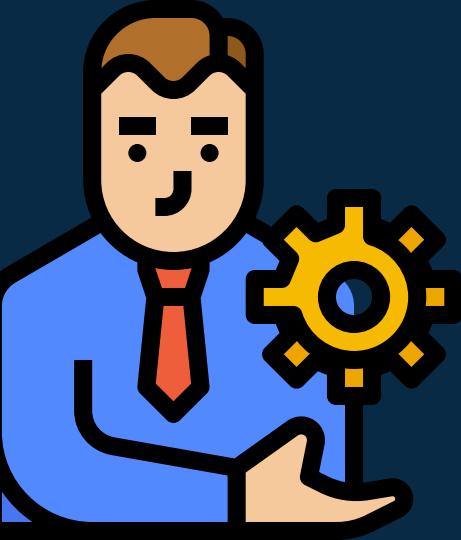
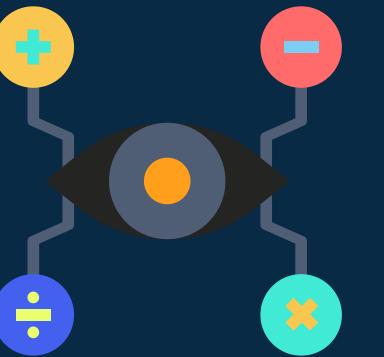
Results Explain Describe Saved SQL History

SHIPCOUNTRY	BESTSELL
Finland	1
Liberia	1
Morocco	1
Singapore	2
Marshall Islands	1
Christmas Island	1
Guinea-Bissau	1
Seychelles	1
Mongolia	1

9 rows returned in 0.04 seconds [Download](#)

$f(x)$

FUNCTIONS



I). .STRING FUNCTIONS

Functions	Description	Syntax
CONCAT	Adds two or more Strings together.	SELECT CONCAT (string1, string2, ..., string_n)
LEN	Returns the length of the string.	SELECT LEN (string)
REVERSE	Reverses a string and returns the result.	SELECT REVERSE (string);
SUBSTRING	Extracts some characters from a string.	SELECT SUBSTRING (string, start, length);

LTRIM	Remove left white spaces of the string.	SELECT LTRIM (string)
RTRIM	Remove right white spaces of the string.	SELECT RTRIM (string)
TRIM	Remove both left and right white spaces from the string.	SELECT TRIM ([characters] FROM)string)
UPPER	Changes all letters of string to upper case.	SELECT UPPER (text)
LOWER	Changes all letters of string to lower case.	SELECT LOWER (text)
REPLACE	Remove all occurrences of a substring within a string, with a new substring.	SELECT REPLACE (string, old_string, new_string)

EXAMPLES ↴

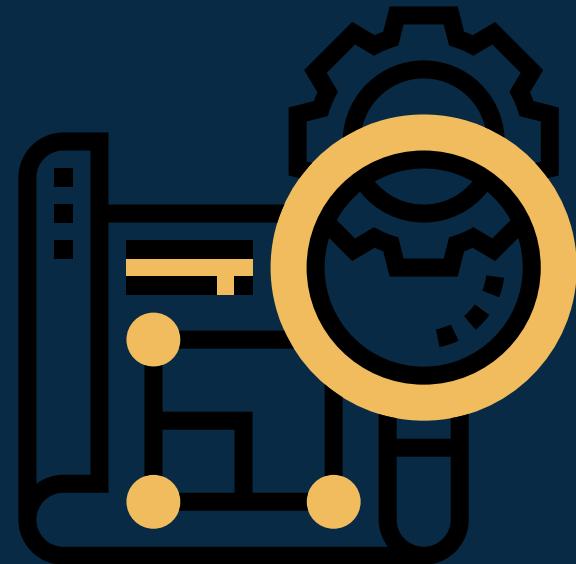
query:

```
////string function  
SELECT ProductName, LENGTH(ProductName) AS LENGTHOFNAME FROM Products;
```

HERE IS THE OUTPUT :

Output

PRODUCTNAME	LENGTHOFNAME
Lotion	6
Remover	7
Beauty	6
Beauty	6
Sets	4
Lotion	6
Shampoo	7
Moisturizer	11
Sets	4
Primer	6



EXAMPLES 2>

```
SELECT UPPER(USERNAME) AS UpperFirstName FROM buyers;
```

HERE IS THE OUTPUT :

	UPPERFIRSTNAME
CLARK	
NOBLE	
TASHYA	
ARETHA	
ABBOT	
VANCE	
MATTHEW	
DOMINIC	
LYNN	

EXAMPLES 3>

Query:

```
SELECT LOWER(USERNAME) AS LowerLastName From buyers;
```

Output:

LOWELASTNAME
clark
noble
tashya
aretha
abbot
vance
matthew
dominic
lynn
dustin

Numeric Functions

Functions	Description	Syntax
ABS()	It returns the absolute value of a number.	SELECT ABS(num);
ACOS()	It returns the arc cosine of a number.	SELECT ACOS(num);
ASIN()	It returns the arc sine of a number.	SELECT ASIN(num);
SQRT()	It returns the square root of a number	SELECT SQRT(num);
ROUND()	It returns a number rounded to a certain number of decimal places.	SELECT ROUND(num);
TRUNCATE()	This doesn't work for SQL Server. It returns 7.53635 truncated to 2 places right of the decimal point.	SELECT TRUNCATE(num,trunc_places);

ATAN()	It returns the arc tangent of a number.	SELECT ATAN(num);
CEIL() / CEILING()	It returns the smallest integer value that is greater than or equal to a number.	SELECT CEIL(num);
COS()	It returns the cosine of the integer.	SELECT COS(num);
SIN()	It returns the sine of the integer.	SELECT SIN(num);
TAN()	It returns the tangent of the integer.	SELECT TAN(num);
FLOOR()	It returns the largest integer value that is less than or equal to a number.	SELECT FLOOR(num);
GREATEST()	It returns the greatest value in a list of expressions.	SELECT GREATEST(list_of_numbers);
LEAST()	It returns the smallest value in a list of expressions	SELECT LEAST(list_of_numbers);

EXAMPLES

Query:

```
select avg(Total_amount) from orders;
```

HERE IS THE OUTPUT :

Output:

```
238.7
```



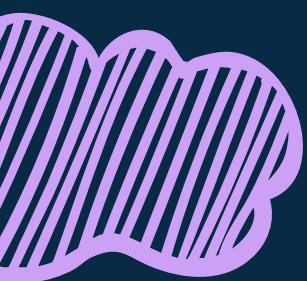


AGGREGATE FUNCTIONS



SQL AGGREGATE FUNCTIONS RETURN A SINGLE VALUE, CALCULATED FROM VALUES IN A COLUMN. USEFUL AGGREGATE FUNCTIONS:

- AVG() - RETURNS THE AVERAGE VALUE
- COUNT() - RETURNS THE NUMBER OF ROWS
- FIRST() - RETURNS THE FIRST VALUE
- LAST() - RETURNS THE LAST VALUE
- MAX() - RETURNS THE LARGEST VALUE
- MIN() - RETURNS THE SMALLEST VALUE
- SUM() - RETURNS THE SUM



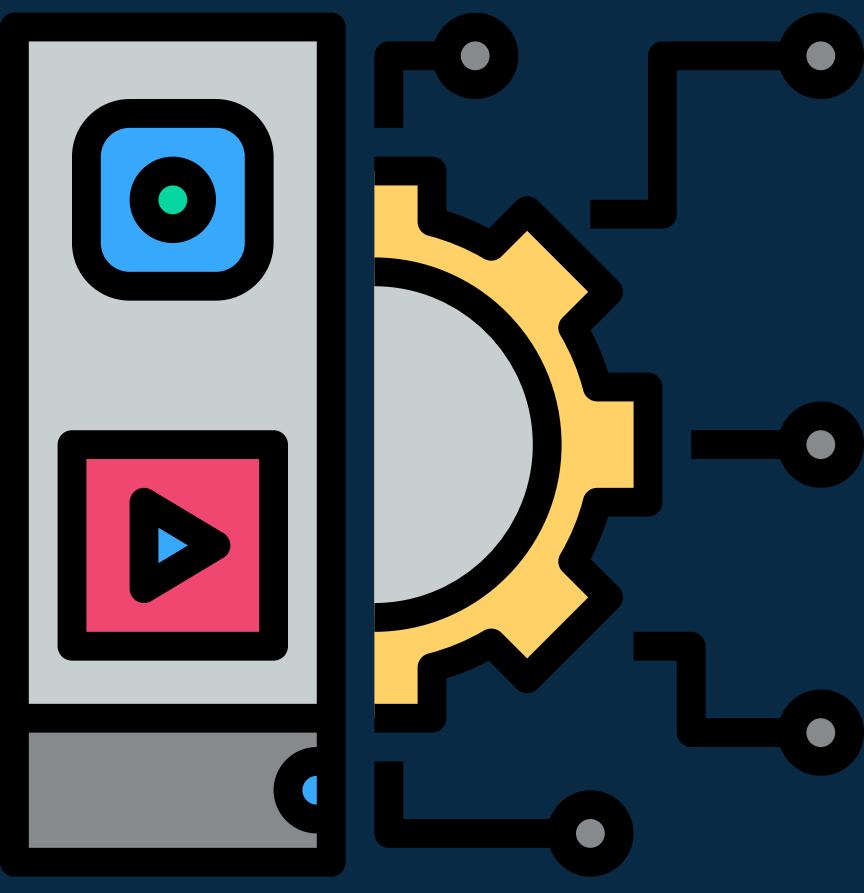
EXAMPLES

```
//////sum  
select sum(Total_amount) from orders;
```

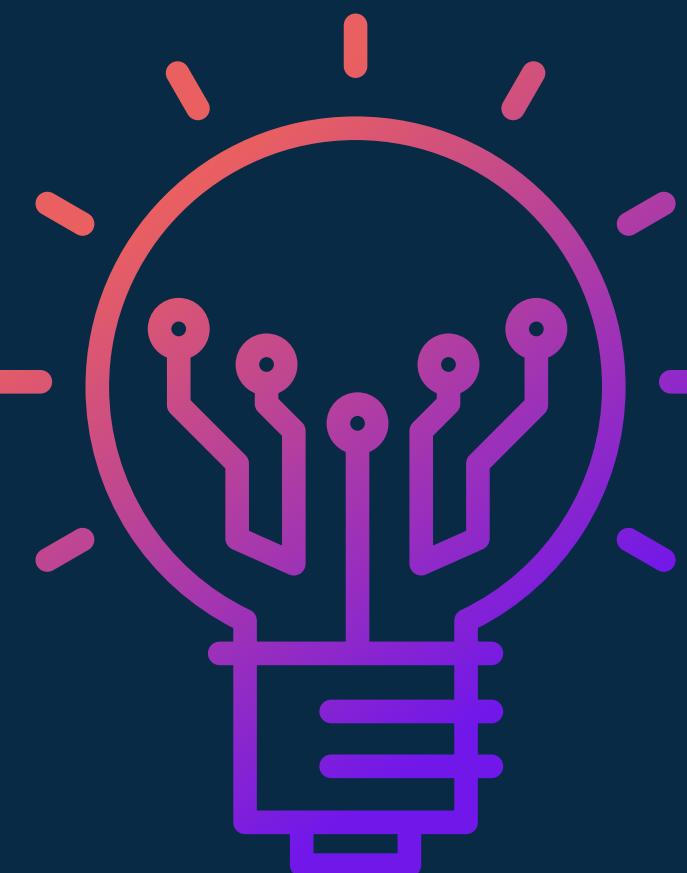
HERE IS THE OUTPUT :

Results Explain Describe Saved SQL History

2387 rows returned in 0.00 seconds [Download](#)



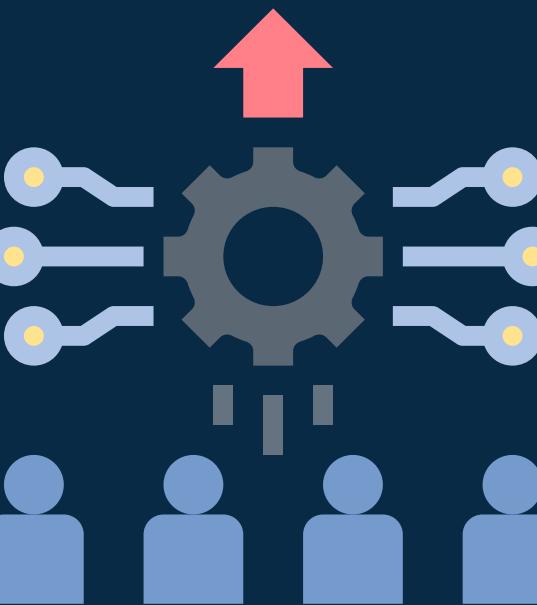
```
select avg(Total_amount) from orders;
```



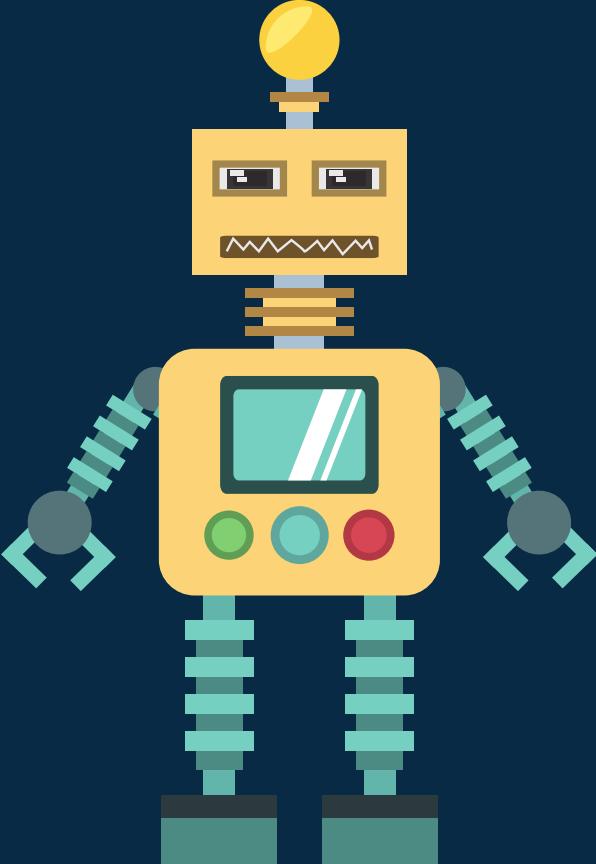
HERE IS THE OUTPUT :

Output:

AVG(TOTAL_AMOUNT)
238.7



```
/////////current date  
SELECT CURRENT_DATE FROM DUAL;
```



HERE IS THE OUTPUT :

OUTPUT::

Results Explain Describe Saved SQL History

	CURRENT_DATE
01/17/2021	2021-01-17

1 rows returned in 0.01 seconds [Download](#)

Order By Group By and Having Statement



THE ORDER BY KEYWORD IS USED TO SORT THE RESULT-SET IN ASCENDING OR DESCENDING ORDER. THE ORDER BY KEYWORD SORTS THE RECORDS IN ASCENDING ORDER BY DEFAULT. TO SORT THE RECORDS IN DESCENDING ORDER, USE THE DESC KEYWORD.

SYNTAX:-

```
SELECT COLUMN1, COLUMN2, ... FROM  
TABLE_NAME
```

EXAMPLE

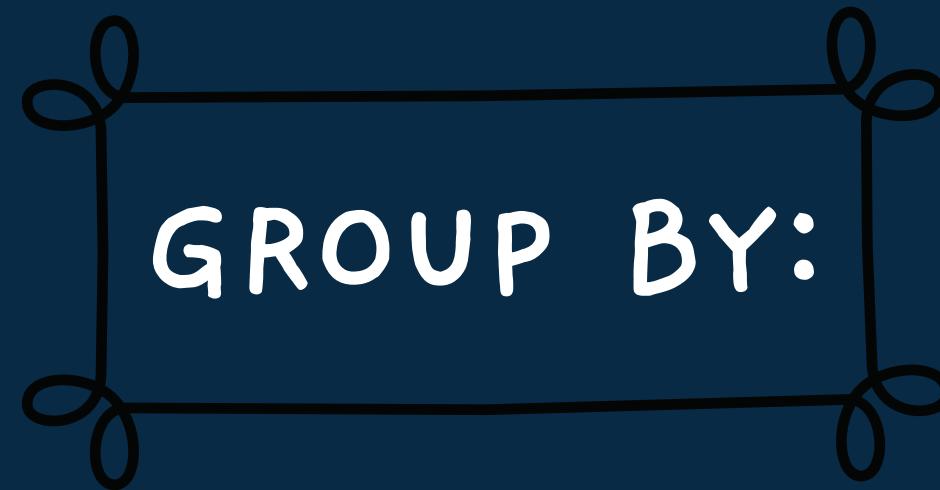
```
//////ORDER BY  
SELECT OrderID, BuyerID, Total_quantity FROM Orders ORDER BY Total_quantity;
```

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History	
	ORDERID		BUYERID		TOTAL_QUANTITY
21942566549		50345881058		1	
85764031654		94908631679		2	
96945305380		42711312958		4	
47217024154		26350935579		5	
07261862587		12714972662		6	
05553373345		75521050479		7	
99251300121		81921795835		8	
90817183713		67711139299		9	
89878325779		72330070747		11	
70701609058		89264121725		13	

10 rows returned in 0.01 seconds

[Download](#)



GROUP BY:

THE GROUP BY STATEMENT GROUPS ROWS THAT HAVE THE SAME VALUES INTO SUMMARY ROWS, LIKE "FIND THE NUMBER OF CUSTOMERS IN EACH COUNTRY". THE GROUP BY STATEMENT IS OFTEN USED WITH AGGREGATE FUNCTIONS (COUNT, MAX, MIN, SUM, AVG) TO GROUP THE RESULT-SET BY ONE OR MORE COLUMNS.

SYNTAX:-

```
SELECT COLUMN_NAME(S) FROM TABLE_NAME  
WHERE CONDITION GROUP BY COLUMN_NAME(S)  
ORDER BY COLUMN_NAME(S);
```

GROUP BY: EXAMPLE

Query:

```
||||||||| ALIAS / GROUP BY |
select CardType, count(*) as popular from Payment group by CardType;
```

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History
				CARDTYPE
				POPULAR
				America Express
				2
				Visa
				3
				JCR
				3
				Discover
				2
4 rows returned in 0.03 seconds		Download		



HAVING:

THE HAVING CLAUSE WAS ADDED TO SQL BECAUSE THE WHERE KEYWORD COULD NOT BE USED WITH AGGREGATE FUNCTIONS.

SYNTAX:-

```
SELECT COLUMN_NAME(S) FROM TABLE_NAME  
WHERE CONDITION GROUP BY COLUMN_NAME(S)  
HAVING CONDITION ORDER BY COLUMN_NAME(S);
```

HAVING EXAMPLE

```
SELECT TOTAL_QUANTITY  
FROM orders  
WHERE TOTAL_QUANTITY>7  
GROUP BY TOTAL_QUANTITY  
HAVING TOTAL_QUANTITY>7  
ORDER BY TOTAL_QUANTITY;
```

HERE IS THE OUTPUT :

Results	Explain	Describe	Saved SQL	History	TOTAL_QUANTITY
					8
					9
					11
					13

4 rows returned in 0.00 seconds [Download](#)

SUBQUERIES

A SUBQUERY IS A QUERY WITHIN ANOTHER SQL QUERY AND EMBEDDED WITHIN THE WHERE CLAUSE. A SUBQUERY IS USED TO RETURN DATA THAT WILL BE USED IN THE MAIN QUERY AS A CONDITION TO FURTHER RESTRICT THE DATA TO BE RETRIEVED.

- SUBQUERIES MUST BE ENCLOSED WITHIN PARENTHESES.
- A SUBQUERY CAN HAVE ONLY ONE COLUMN IN THE SELECT CLAUSE, UNLESS MULTIPLE COLUMNS ARE IN THE MAIN QUERY FOR THE SUBQUERY TO COMPARE ITS SELECTED COLUMNS.
- AN ORDER BY COMMAND CANNOT BE USED IN A SUBQUERY, ALTHOUGH THE MAIN QUERY CAN USE AN ORDER BY. THE GROUP BY COMMAND CAN BE USED TO PERFORM THE SAME FUNCTION AS THE ORDER BY IN A SUBQUERY.
- SUBQUERIES THAT RETURN MORE THAN ONE ROW CAN ONLY BE USED WITH MULTIPLE VALUE OPERATORS SUCH AS THE IN OPERATOR.
- THE SELECT LIST CANNOT INCLUDE ANY REFERENCES TO VALUES THAT EVALUATE TO A BLOB, ARRAY, CLOB, OR NCLOB.
- A SUBQUERY CANNOT BE IMMEDIATELY ENCLOSED IN A SET

(I) NESTED QUERIES

IN NESTED QUERIES, A QUERY IS WRITTEN INSIDE A QUERY. THE RESULT OF THE INNER QUERY IS USED IN THE EXECUTION OF THE OUTER QUERY.

SYNTAX:-

SINGLE

```
SELECT COLUMN(S) FROM TABLE_NAME WHERE  
COLUMN IN/NOT IN(QUERY);
```

MULTIPLE

```
SELECT COLUMN(S) FROM TABLE_NAME WHERE  
COLUMN IN/NOT IN(QUERY2 IN/NOT IN(QUERY(S)))
```

EXAMPLES

////////// NESTED QUERIES

```
select max(UnitPrice) as second_highest_price  
from Products  
where Products.UnitPrice<(select max(UnitPrice)  
                           from Products);
```

HERE IS THE OUTPUT :

Results Explain Describe Saved SQL History

\$73.53

1 rows returned in 0.02 seconds

[Download](#)

(II) CORRELATED QUERIES

CORRELATED SUBQUERIES ARE USED FOR ROW-BY-ROW PROCESSING. EACH SUBQUERY IS EXECUTED ONCE FOR EVERY ROW OF THE OUTER QUERY. A CORRELATED SUBQUERY IS EVALUATED ONCE FOR EACH ROW PROCESSED BY THE PARENT STATEMENT. THE PARENT STATEMENT CAN BE A SELECT, UPDATE, OR DELETE STATEMENT.

SYNTAX:-

```
SELECT COLUMN1, COLUMN2, .... FROM TABLE1 OUTER  
WHERE COLUMN1 OPERATOR (SELECT COLUMN1 FROM  
TABLE2 WHERE EXPRI = OUTER.EXPR2);
```

EXAMPLE

```
1 SELECT *
2 FROM Products
3 WHERE QuantityOnHand >
4 (SELECT AVG(QuantityOnHand)
5 FROM Products
6 );
```

HERE IS THE OUTPUT :

Results Explain Describe Saved SQL History

PRODUCTID	PRODUCTNAME	SUPPLIERID	CATEGORY
206908-3232	Remover	31704489846	Lip
096419-4823	Beauty	30918186609	Cheek
197710-3876	Lotion	70268860023	Eye, Face
533169-3365	Shampoo	30751277561	Body, Hair
998757-5991	Sets	43636501698	-

rows returned in 0.01 seconds [Download](#)

SEQUENCE

SEQUENCE IS A SET OF INTEGERS 1, 2, 3, . . . THAT ARE GENERATED AND SUPPORTED BY SOME DATABASE SYSTEMS TO PRODUCE UNIQUE VALUES ON DEMAND.

A SEQUENCE IS A USER DEFINED SCHEMA BOUND OBJECT THAT GENERATES A SEQUENCE OF NUMERIC VALUES. SEQUENCES ARE FREQUENTLY USED IN MANY DATABASES BECAUSE MANY APPLICATIONS REQUIRE EACH ROW IN A TABLE TO CONTAIN A UNIQUE VALUE AND SEQUENCES PROVIDE AN EASY WAY TO GENERATE THEM.

CREATE/DROP SEQUENCE:

CREATE :

SYNTAX TO CREATE A SEQUENCE IS,

```
CREATE SEQUENCE sequence-name  
    START WITH initial-value  
    INCREMENT BY increment-value  
    MAXVALUE maximum-value  
    CYCLE | NOCYCLE;
```

THE INITIAL-VALUE SPECIFIES THE STARTING VALUE FOR THE SEQUENCE.

THE INCREMENT-VALUE IS THE VALUE BY WHICH SEQUENCE WILL BE INCREMENTED.

THE MAXIMUM-VALUE SPECIFIES THE UPPER LIMIT OR THE MAXIMUM VALUE UPTO WHICH SEQUENCE WILL INCREMENT ITSELF.

THE KEYWORD CYCLE SPECIFIES THAT IF THE MAXIMUM VALUE EXCEEDS THE SET LIMIT, SEQUENCE WILL RESTART ITS CYCLE FROM THE BEGINING.

AND, NO CYCLE SPECIFIES THAT IF SEQUENCE EXCEEDS MAXVALUE VALUE, AN ERROR WILL BE THROWN.

EXAMPLE:
LET'S START BY CREATING A SEQUENCE, WHICH WILL
START FROM 1, INCREMENT BY 1 WITH A MAXIMUM
VALUE OF 999.

```
CREATE SEQUENCE seq_1
START WITH 1
INCREMENT BY 1
MAXVALUE 999
CYCLE;
```

DROP SEQUENCE

Syntax:

```
DROP SEQUENCE schema_name.sequence_name;
```

```
DROP SEQUENCE [ IF EXISTS ] { database_name.schema_name.sequence_name | schema_
[ ; ] }
```

ALTER SEQUENCE

ALTER SEQUENCE STATEMENT CAN BE USED TO CHANGE THE INCREMENT, MINIMUM AND MAXIMUM VALUES, CACHED NUMBERS, AND BEHAVIOR OF AN EXISTING SEQUENCE. THIS STATEMENT AFFECTS ONLY FUTURE SEQUENCE NUMBERS.

Syntax:

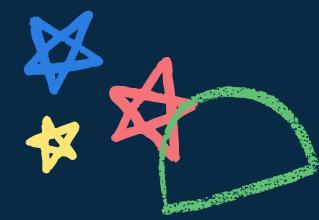
```
ALTER SEQUENCE [ IF EXISTS ] name  
[ AS data_type ]  
[ INCREMENT [ BY ] increment ]  
[ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]  
[ START [ WITH ] start ]  
[ RESTART [ [ WITH ] restart ] ]  
[ CACHE cache ] [ [ NO ] CYCLE ]  
[ OWNED BY { table_name.column_name | NONE } ]
```

```
CREATE TABLE ordprod ( o_id NUMBER NOT NULL, name VARCHAR2(20) );
--Creating a sequence to insert values in o_id
CREATE SEQUENCE sequence_1
START WITH 1
INCREMENT BY 1
MINVALUE 0
MAXVALUE 100 CYCLE;
--Inserting into ordprod
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order1');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order2');
INSERT INTO ordprod VALUES(sequence_1.nextval, 'Order3');
--Displaying the ordprod table
SELECT * FROM ordprod;
--Altering the sequence 'sequence_1'
ALTER SEQUENCE sequence_1
INCREMENT BY 2;
--Inserting into ordprod using the altered sequence
INSERT INTO ordprod VALUES (sequence_1.nextval, 'Order3');
INSERT INTO ordprod VALUES (sequence_1.nextval, 'Order4');
--Displaying ordprod table
SELECT * FROM ordprod;
```

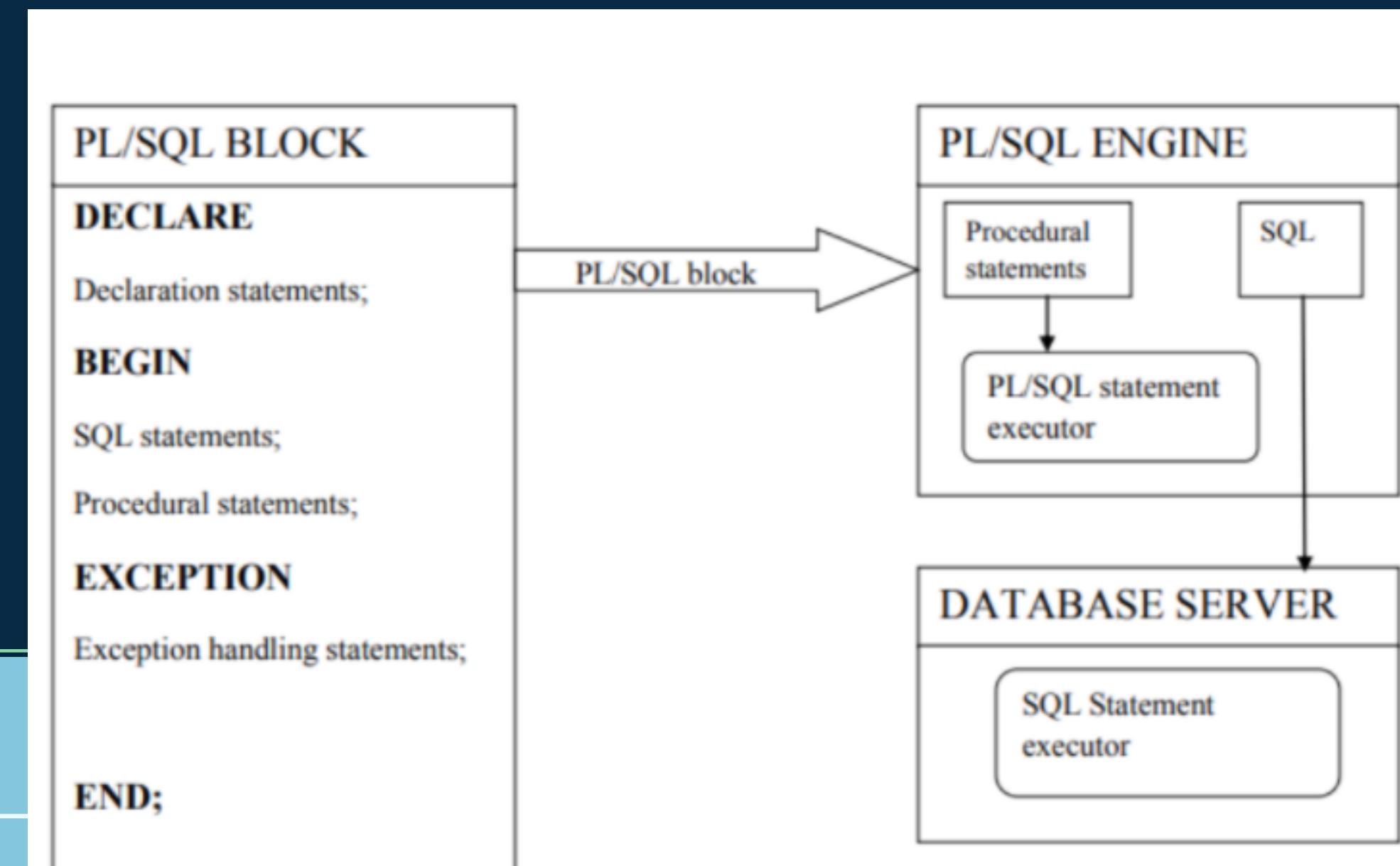
ALTER SEQUENCE QUERIES

o_id	NAME
22	Order5
1	Order1
2	Order2
3	Order5
62	Order4

INTRODUCTION TO PL/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.



MM

SYNTAX OF PL/SQL:



PL/SQL IS A BLOCK-STRUCTURED LANGUAGE; THIS MEANS THAT THE PL/SQL PROGRAMS ARE DIVIDED AND WRITTEN IN LOGICAL BLOCKS OF CODE. EACH BLOCK CONSISTS OF THREE SUB-PARTS -

1. DECLARATIONS

This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

2. EXECUTABLE COMMANDS

- This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program.

3. EXCEPTION HANDLING

This section starts with the keyword **EXCEPTION**. This optional section contains exception(s) that handle errors in the program. Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END.



TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:-

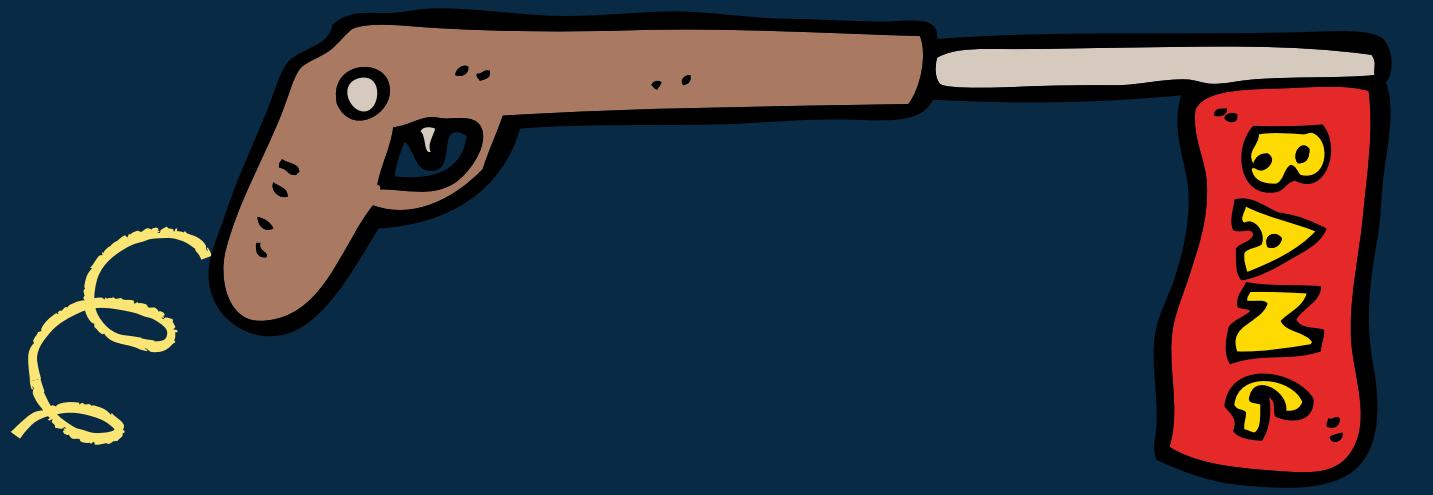
1. A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
2. A database definition (DDL) statement (CREATE, ALTER, or DROP)
3. A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN) Triggers can be defined on the table, view, schema, or database with which the event is associated

ADVANTAGES

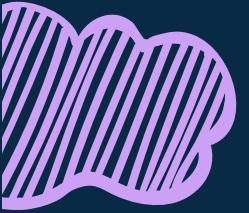
- 1) It helps in maintaining the integrity constraints in the database tables, especially when the primary key and foreign key constraints are not defined.
- 2) It sometimes also helps in keeping the SQL codes short and simple as I show in the real-life example.
- 3) It helps in maintaining the track of all the changes (update, deletion and insertion) occurs in the tables through inserting the changes values in the audits tables.

DISADVANTAGES

- 1) Hard to maintain since this may be a possibility that the new developer isn't able to know about the trigger defined in the database and wonder how data is inserted, deleted or updated automatically.
- 2) They are hard to debug since they are difficult to view as compared to stored procedures, views, functions, etc.
- 3) If complex code is written in the triggers, then it will slow down the performance of the applications.



TYPES OF TRIGGERS



1.) Before event

Triggers: BEFORE triggers run the trigger action before the triggering statement is run.

2.) After event

Triggers: AFTER triggers run the trigger action after the triggering statement is run.

3.) Row Trigger:

A row trigger is fired each time the table is affected by the triggering statement.

4.) Statement Trigger:

A STATEMENT TRIGGER IS FIRED ONCE ON BEHALF OF THE TRIGGERING STATEMENT, REGARDLESS OF THE NUMBER OF ROWS IN THE TABLE THAT THE TRIGGERING STATEMENT AFFECTS, EVEN IF NO ROWS ARE AFFECTED

USE OF TRIGGER IN OUR PROJECT

```
create table Products_1 as (select * from Products);
create table Stockcheck (ProductID varchar2(11), Stocklevel varchar2(100));
```

```
Create or Replace trigger stocklevel
BEFORE insert on Products_1
FOR EACH ROW
begin
case when (:new.QuantityOnHand <= 300)
then insert into Stockcheck (ProductID, Stocklevel)
values(:new.ProductID, 'Low-stock');
END case;
END;
```

HERE IS THE OUTPUT :

Trigger created.

PRODUCTID	STOCKLEVEL
96419-4825	Low-stock

CURSORS

Implicit cursor - Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it. Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected. In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK ROWCOUNT and %BULK EXCEPTIONS, designed for use with the FORALL statement. Any SQL cursor attribute will be accessed as sql%attribute name as shown below in the example

II. EXPLICIT CURSOR - EXPLICIT CURSORS ARE PROGRAMMER-DEFINED CURSORS FOR GAINING MORE CONTROL OVER THE CONTEXT AREA. AN EXPLICIT CURSOR SHOULD BE DEFINED IN THE DECLARATION SECTION OF THE PL/SQL BLOCK. IT IS CREATED ON A SELECT STATEMENT WHICH RETURNS MORE THAN ONE ROW. THE SYNTAX FOR CREATING AN EXPLICIT CURSOR IS:-

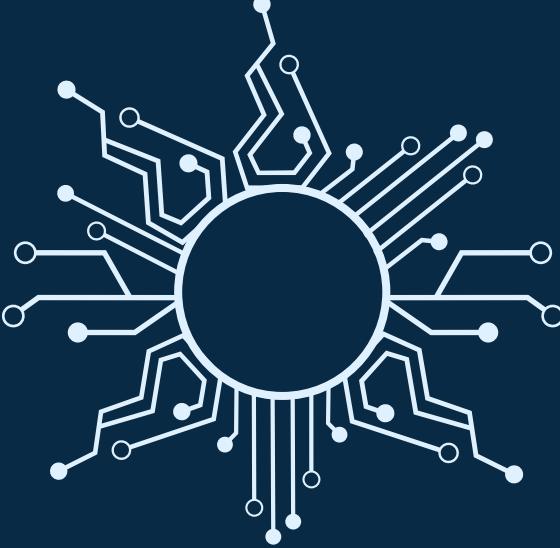
```
1 CURSOR cursor_name IS select_statement;
```

EXPLICIT CURSORS CAN BE DECLARED, OPENED, FETCHED AND CLOSED AS IN THE PRECEDING SECTION.



USE OF CURSOR IN OUR PROJECT

```
--cursor
DECLARE
    p_id Products.ProductID%type;
    p_name Products.ProductName%type;
    p_catg Products.Category%type;
    CURSOR p_products is
        SELECT ProductID, ProductName, Category FROM Products;
BEGIN
    OPEN p_products;
    LOOP
        FETCH p_products into p_id, p_name, p_catg;
        EXIT WHEN p_products%notfound;
        dbms_output.put_line(p_id || ' ' || p_name || ' ' || p_catg);
    END LOOP;
    CLOSE p_products;
END;
```



HERE IS THE OUTPUT :

```
748130-5105 Lotion Hair  
206908-3232 Remover Lip  
332338-2345 Beauty Face  
096419-4823 Beauty Cheek  
585230-0077 Sets Cheek  
197710-3876 Lotion Eye, Face  
533169-3365 Shampoo Body, Hair  
588681-0034 Moisturizer Lip  
998757-5991 Sets  
375163-6428 Primer
```

Statement processed.



VIEWS

VIEWS IN SQL ARE A KIND OF VIRTUAL TABLE. A VIEW ALSO HAS ROWS AND COLUMNS AS THEY ARE IN A REAL TABLE IN THE DATABASE. WE CAN CREATE A VIEW BY SELECTING FIELDS FROM ONE OR MORE TABLES PRESENT IN THE DATABASE. A VIEW CAN EITHER HAVE ALL THE ROWS OF A TABLE OR SPECIFIC ROWS BASED ON CERTAIN CONDITIONS. WE CAN PERFORM VARIOUS TYPES OF OPERATION LIKE INSERT/UPDATE/DELETE IN A ROW IN VIEWS.

Syntax:

```
INSERT INTO view_name(column1, column2 , column3..) VALUES(value1,  
value2, value3..);
```

This will insert a row in a view.

Syntax:

```
DELETE FROM view_name WHERE condition;
```

This will delete a row from the view.

TYPES OF VIEWS

materialized view

Oracle Database creates one internal table and at least one index, and may create one view, all in the schema of the materialized views. Oracle Database uses these objects to maintain the materialized views in SQL data..

view resolution

When you read from a view, or join a view to an existing query the SQL server will then execute the query in the view and join it to your data set. When it does that, that would be view resolution.

features /advantages of views

* SECURITY

Each user can be given permission to access the database only through a small set of views

*CONSISTENCY

A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.

* QUERY SIMPLICITY

A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view

*DATA INTEGRITY

If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.

* STRUCTURAL SIMPLICITY

Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

*LOGICAL DATA INDEPENDENCE

Views can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.

CREATE VIEW

TO CREATE A NEW VIEW

Query:

```
/////////// VIEW  
create view pop as  
select ProductName,ItemQuantity from Products inner join Order_details on Products.ProductID = Order_Details.ProductID;  
select ProductName, sum(ItemQuantity) from pop group by ProductName;
```

Syntax :

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

Output:

The screenshot shows a database query results interface. At the top, there are tabs for 'Results' (which is selected), 'Explain', 'Describe', 'Saved SQL', and 'History'. Below this is a table with two columns: 'PRODUCTNAME' and 'SUM(ITEMQUA...'. The data rows are: Lotion (86), Moisturizer (9), Primer (81), Shampoo (11), Scrub (59), Remover (76), and Beauty (91).

PRODUCTNAME	SUM(ITEMQUA...
Lotion	86
Moisturizer	9
Primer	81
Shampoo	11
Scrub	59
Remover	76
Beauty	91

DROP VIEW

TO DROP A VIEW

Query:

```
DROP view pop;
```

Results

Explain

Describe

View dropped.

0.08 seconds

Syntax 2:

```
DROP VIEW view_name;
```

TECHNIK

