MySQL

* RDBMS, open Source, free, Small & large app, fast, reliable, scalable, easy to use
* Cross platform, ANSI SQL Std, 1995 (released), dev; Supported - distributed by ORACLE
* Name after Co-founder Monty Widenius daughter: My
  → FB, GitHub, Youtube, Airbnb, Twitter, Booking.com, Uber (uses)
  → WordPress, Drupal, Joomla, Contao

Have relationships

* Semicolon → Seperate each SQL statement

```
SELECT Column1, ...
FROM tablename ;
```

* → All columns

SELECT DISTINCT ——— (Unique value alone)

SELECT COUNT (DISTINCT Country) FROM Customers;

Note: distinct → apply to RHS eg columns

WHERE Condition =, >, <, <=, >=, <>, BETWEEN, LIKE, IN

AND
OR
NOT

ORDER BY Col1 ——, DESC|ASC ASC → Default.

'After where'

Insert into

INSERT INTO table_name (col1, ...) → Column list good practice!
VALUES (value1, ...)

NULL Values

WHERE col_name IS NULL ;
IS NOT NULL ;

UPDATE

UPDATE table_name
SET col1 = val1, col2 = val2, ...
WHERE Condition

→ without where — All rows affected
'Default: TRUE

---

multiple records - Condition (wherever TRUE)

'UPDATED'

S. Hamprasad

DELETE

```
DELETE FROM table_name WHERE Condition ;
DELETE FROM table_name; ⟶ Delete all
```

LIMIT

```
SELECT Colname FROM table_name
WHERE Condition
LIMIT number;
```

```
SELECT * FROM Customers
LIMIT 3
```

(Top 3 — 0, 1, 2 Rows)

```
SELECT Colname FROM table_name
WHERE Condition (optional)
LIMIT 5 OFFSET 10 ; ⟶ 10, 11, 12, 13, 14
```

Shortcut

```
LIMIT 10,5
      ↓  ↓
   offset  howmany
```

MIN(), MAX(), COUNT(), AVG(), SUM()

```
SELECT COUNT (Price)
WHERE Price > 10;
```

Wildcard %, _ (alone)

```
WHERE Columnname LIKE pattern;
```

- '%a' → ending with a
- 'a%' → start with a
- '%or%' → has or (at any position) eg: oracle, sunorp, sonor
- '_r%' → second letter r (First letter anything)

% → 0/more
_ → exactlyone

- 'a_' → first start with a, 2chars exactly.
- 'a%o' → Start with a, end with o

%, _

```
SELECT Column_name
FROM table_name
WHERE Column_name IN (value1, value2, ...)
```

SELECT Col_name BETWEEN Value1 AND Value2

NOT BETWEEN

NOT IN (_,_,_)

Aliases

SELECT Column_name AS alias_name
FROM table_name

SELECT Col_name
FROM table_name AS col_1

SELECT Col1 AS C1, Col2 AS C2

SELECT O.OrderId, O.orderDate, C.CustomerName
FROM Customers AS C, Orders AS O
WHERE C.CustomerName = 'Around' AND C.CustomerId = O.CustomerId.

JOIN


Inner (Common - both)


Common + rem of table 1


Common + rem of table 2


Common + rem of table 1 + rem of table 2.

table 1 2
Say: 3 rows, 4 rows

SELECT table1, Col1, table2. Col1
(No WHERE condt)
↳ 12 rows — each row of table1 with every other row of table2

INNER JOIN

SELECT Orders. OrderID, Customers. CustomerName
FROM Orders
INNER JOIN Customers ON Orders. CustID = Customers. CustID;

3 Rows

SELECT Orders. OrderID, Customers. CustomerName, Shippers. ShipperName
FROM Orders
INNER JOIN Customers ON Orders. Cust_ID = Customers. Cust_ID
INNER JOIN Shippers ON Orders. ShipperID = Shippers. ShipperID.

LEFT JOIN

SELECT Customers. CustomerName, Orders. OrderID
FROM Customers
LEFT JOIN Orders. CustomerID = Orders. CustomerID
ORDER BY Customers. CustomerName;

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID.
ORDER BY Orders.OrderID;
```

CROSS JOIN 

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
CROSS JOIN Orders
WHERE Customers.CustomerID = Orders.CustomerID;
```

SELF JOIN: (AS) optional in MySQL - good practice.

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName2,
       A.City
FROM Customer AS A, Customer AS B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Customers → ID, Name, ContactName, Address, City, PostalCode, Country

Orders → OrderID, CusID, EmpID, orderdate, ShipperID

OrderID → Orderdetail ID, OrderID, ProductID, Quantity

SHIPPERS → ID, Name, phone.

UNION

```
SELECT Colname FROM table1
UNION                                  ] No duplicates
SELECT Colname FROM table2             ] must: same no. of columns!
ORDER BY ___
```

```
SELECT City FROM Customers
UNION ALL ──────────→ Repitation allowed.
SELECT City FROM Suppliers
ORDER BY City;
```

```
SELECT City FROM Customers
WHERE Country = 'Germany'
UNION ALL
SELECT City FROM Suppliers
WHERE Country = 'Germany'
ORDER BY City;
```

```
SELECT Colname
FROM tablename
WHERE Condition
GROUP BY Col_name
ORDER BY Col_name
```

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

```
SELECT Shippers.ShipperName, COUNT (Orders.OrderID) AS CountTotal
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

→ get data (Common + Rem of table 1) → filter (group) by shipper Name

---

HAVING — (when GROUP BY used - Preread of WHERE)

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

↳ WHERE (only Rowwise) can't used

```
SELECT Employees.LastName, COUNT (Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT (Orders.OrderID) > 25;
```

---

* EXISTS → Test existence of any record in a subquery
* TRUE — if the subquery returns 1/more records.

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE
              Products.SupplierID = Suppliers.supplierID AND
              price = 22);
```

when price != 22 → FALSE → query → won't be executed.

ANY, ALL

ANY → True (Any of the subquery meets the condt)

ALL → True (All Of the " ")

```
SELECT ProdName
FROM Products
WHERE ProdID = Any (SELECT ProdID FROM OrdersDetails
                    WHERE Quantity >99);

SELECT ProdName
FROM Products
WHERE ProdId = ALL (SELECT ProdID FROM OrdersDetails
                    WHERE Quantity = 10);  ---> FALSE (Nothing)
```

INSERT INTO SELECT

```
INSERT INTO table2
SELECT * FROM table1
WHERE Condition;
```

```
INSERT INTO table2 (col1, col2, ....)
SELECT col1, col2, .... FROM table1
WHERE Condition;
```

CASE

```
CASE
    WHEN Condition1 THEN result1
    WHEN Condition2 THEN result2
    WHEN ConditionN THEN resultN
    ELSE result
END;
```

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'Good'
    WHEN Quantity = 30 THEN 'Correct'
    ELSE 'Bad'
END AS QualityText
FROM OrdersDetails;
```

IFNULL(), COALESCE()

```
SELECT ProdName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

(IFNULL: False → 0, True → UnitsOnOrder)

```
-- Comments
/*  */ multiline Comments
```

Operator

+, -, *, /, % ---> Arithmetic

&, |, ^ ---> Bitwise

=, >, <, >=, <=, <> ---> Comparison

Compound operators

+=, -=, *=, /=, %=, &=, ^-=, |*= ---> Bitwise exclusive equals, OR equals.

Logical:

ALL, AND, ANY, BETWEEN, EXISTS, IN,
LIKE, NOT, OR, SOME

create database

```
CREATE DATABASE databaseName;
DROP   DATABASE databasen  ⟶ delete
```

CREATE TABLE

```
CREATE TABLE table_name (
        Col1  datatype,        varchar,
        Col2  datatype,        Integer,
        Col3  datatype,        date
        ....
);
```

```
CREATE TABLE Persons (
        PersonID   int,
        LastName   varchar(255),
        FirstName  varchar(255),
        Address    varchar(255),   ⟶ max length 255 characters.
        City       varchar(255)
);
```

Using another table

```
CREATE TABLE new_table_name AS
    SELECT col1, col2, ...
    FROM existing_table_name
    WHERE ... ;
```

```
CREATE TABLE TestTable AS
SELECT CustName, ContactName
FROM Customers;
```

DROP table

```
DROP TABLE table_num;      ⟶ delete entire table
TRUNCATE TABLE table_num;  ⟶ Just delete values – not table.
```

ALTER TABLE

```
ALTER TABLE table_name
ADD Column_name datatype;
```

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

```
ALTER TABLE Customers
DROP COLUMN Email;
```

```
ALTER TABLE Persons
ADD DateOfBirth date;

ALTER TABLE Persons
MODIFY COLUMN DateOfBirth YEAR;

ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

```
CREATE TABLE table_new (
        Col1 datatype constraint,
        :
        ...
);
```

Constraints: NOT NULL
UNIQUE
PRIMARY KEY
FOREIGN KEY
CHECK
DEFAULT
CREATE INDEX

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

```
CREATE TABLE Persons (                                  Multiple
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int,
    UNIQUE (ID)                  (or) CONSTRAINT UC_Person UNIQUE (ID, LastName)
);
```

ALTER TABLE

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID, LastName);

ALTER TABLE Persons
DROP INDEX UC_Person;
```

PRIMARY Key

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)                CONSTRAINT PK_Person
);                                  PRIMARY KEY (ID, LastName);
```

ALTER TABLE

```
ALTER TABLE Persons                 ALTER TABLE Persons
ADD PRIMARY KEY (ID);               ADD CONSTRAINT PK_Person PRIMARY KEY
                                        (ID, LastName);

                                    ALTER TABLE Persons
                                    DROP PRIMARY KEY;
```

```
CREATE TABLE Orders (                                   Foreign Key → Orders.PersonID
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons (PersonID)
);

FOREIGN KEY (PersonID) REFERENCES Persons (PersonID)
```

```
CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Person (PersonID)
```

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons (PersonID);

ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons (PersonID);
```

DROP

```
ALTER TABLE ORDERS
DROP FOREIGN KEY FK_PersonOrder;
```

CHECK

```
CREATE TABLE Persons (
    ID int NOT NULL,
    ⋮
    CHECK (Age >= 18)
);
```

```
    ⋮
CONSTRAINT CHK_Person CHECK (Age >= 18
    AND City = 'Sandnes');
```

```
ALTER TABLE Persons
ADD CHECK (Age >= 18);
```

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge (Age >= 18 AND
    City = 'Sandnes');
```

DROP

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

CREATE INDEX

* speed up search/queries — Affects update performance
  Users — can't see indices

```
CREATE UNIQUE INDEX index_name ⟶ UNIQUE index!
ON table_name (col1, ...);

CREATE INDEX idx_name
ON Persons (LastName, FirstName);
```

```
ALTER TABLE table_name
DROP INDEX index_name;
```

Auto increment – ID generation.

default: 1

```
CREATE TABLE Persons (
    PersonId int NOT NULL AUTO_INCREMENT,
    ⋮
    PRIMARY KEY (PersonId)
);
```

`AUTO_INCREMENT = 100` Inc by 100.

```
INSERT INTO Persons (FirstName, LastName)
VALUES ('Laura', 'Monsen');
```

Date

DATE – YYYY – MM – DD

DATETIME – YYYY – MM – DD HH:MI:SS

TIMESTAMP – YYYY – MM – DD HH:MI:SS

YEAR YYYY (or) YY

WHERE OrderDate = '2008-11-11' → easy to compare!

Say in DATETIME format – Failed!

VIEW

Virtual table

```
CREATE VIEW view_name AS
SELECT Col1, Col2, ...
FROM table_name
WHERE Condition;
```

```
CREATE VIEW [Brazil Cust] AS
SELECT CustName, Name
FROM Customers
WHERE Country = 'Brazil';
```

```
SELECT * FROM [BRAZIL Cust];
```

```
CREATE VIEW [Prod above Average] AS
SELECT ProdName, Price
FROM Products
WHERE Price > (SELECT AVG (Price) FROM Products);
```

```
SELECT * FROM [Prod above Average];
```

Update/Replace view

```
CREATE OR REPLACE VIEW [Brazil Cust] AS
SELECT CusName, ConkName, City
FROM Customers
WHERE Country = 'Brazil';
```

DROP

```
DROP VIEW [Brazil Customers];
```

Many Functions available – MySQL

Refer: W3Schools

why RDBMS: data redundancy avoid
Suitable for widen range of app
Columns – normalize
Relate two tables
Access to stored info
Data sorting
foreign keys
RDBMS – meant for multiple users
Security
modify data, Transaction managements

RDBMS – must for large org!