



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SQL Language

Contents

- Part 1. SQL Language – Basic
- Part 2. SQL Language – Advanced

Reference: <https://www.tutorialspoint.com/sql/index.htm>

PART 1. SQL LANGUAGE - BASIC

- Datatypes
- Create, Drop, Select Database
- Create, Insert Table
- Where
- And & Or
- Update
- Delete
- Like
- Top
- Order by
- Group by
- Distinct

Data Types

- **char(*n*)**. Fixed length character string, with user-specified length *n*.
- **varchar(*n*)**. Variable length character strings, with maximum length *n*.
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(*p,d*)**. Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex., **numeric**(3,1), allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(*n*)**. Floating point number, with user-specified precision of at least *n* digits.
- **date**: A calendar date, containing four digit year, month, and day of the month.
- **time**: The time of the day in hours, minutes, and seconds.

Data Types

Data Type	From	To
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
numeric	$-10^{38} + 1$	$10^{38} - 1$
real	$-3.40E + 38$	$3.40E + 38$
float	$-1.79E + 308$	$1.79E + 308$
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M	

CHAR and VARCHAR Data types

- **char(n).**

- Fixed length character string, with user-specified length n .
- Values are right-padded with **spaces** to the specific length

- **varchar(n).**

- Variable length character strings, with maximum length n .
- **1-byte or 2-byte length** prefix is used to indicate the number of bytes in the value

Value	CHAR (4)	Storage Required	VARCHAR (4)	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

Create Database

- **CREATE DATABASE** statement is used to create a new SQL database.
- Syntax

```
CREATE DATABASE DatabaseName;
```

- Note: database name should be unique within the RDBMS.
- Example:

```
SQL> CREATE DATABASE testDB;
```

```
SQL> SHOW DATABASES;
```

Drop Database

- **DROP DATABASE** statement is used to drop an existing database in SQL schema
- Syntax

DROP DATABASE DatabaseName;

Note: database name should be unique within the RDBMS.

- Example:

SQL> DROP DATABASE testDB;

SQL> SHOW DATABASES;

Select Database

- **USE** statement is used to select any existing database in the SQL schema.
- Syntax

USE DatabaseName;

Note: database name should be unique within the RDBMS.

- Example:

SQL> SHOW DATABASES;

SQL> USE mysql;

Create Table

- SQL **CREATE TABLE** statement is used to create a new table.
- Syntax

```
CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    .....  
    columnN datatype,  
    PRIMARY KEY( one or more columns )  
);
```

- Example:

```
SQL> CREATE TABLE CUSTOMERS(  
    ID      INT                NOT NULL,  
    NAME    VARCHAR (20)       NOT NULL,  
    AGE     INT                NOT NULL,  
    ADDRESS CHAR (25) ,  
    SALARY  DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

Desc Table

```
SQL> DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI		
NAME	varchar(20)	NO			
AGE	int(11)	NO			
ADDRESS	char(25)	YES		NULL	
SALARY	decimal(18,2)	YES		NULL	

```
5 rows in set (0.00 sec)
```

Drop Table

- DROP TABLE statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

- Syntax

```
DROP TABLE table_name;
```

- Example:

```
SQL> DROP TABLE CUSTOMERS;
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
SQL> DESC CUSTOMERS;
```

```
ERROR 1146 (42S02): Table 'TEST.CUSTOMERS' doesn't exist
```

Insert Query

- **INSERT INTO** Statement is used to add new rows of data to a table in the database.

- Syntax

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

- Example:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
```

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES
    (2, 'Peb', 25, 'Delhi', 1500.00),
    (3, 'kaushik', 23, 'Kota', 2000.00);
```

Sample data

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Select Query

- **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table
- Syntax

SELECT column1, column2, columnN FROM table_name;

SELECT ID, NAME, SALARY FROM CUSTOMERS;

ID	NAME	SALARY
1	Ramesh	2000.00
2	Khilan	1500.00
3	kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

*SELECT * FROM CUSTOMERS;*

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Where Clause

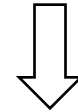
- **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables
- Syntax

```
SELECT column1,..., columnN  
FROM table_name  
WHERE [condition]
```

- Operators: >, <, =, LIKE, NOT, etc.
- Example

```
SQL> SELECT ID, NAME, SALARY  
FROM CUSTOMERS  
WHERE SALARY > 2000;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	SALARY
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

AND and OR Conjunctive Operators

- **AND & OR** operators are used to combine multiple conditions to narrow data in an SQL statement.

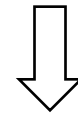
- **Syntax**

```
SELECT column1, ,columnN
FROM table_name
WHERE [condition1] AND...AND
      [conditionN];
```

- **Example**

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	SALARY
6	Komal	4500.00
7	Muffy	10000.00

Update Query

- **UPDATE** Query is used to modify the existing records in a table.
- Syntax

```
UPDATE table_name  
SET column1 = value1, column2 = value2..., columnN = valueN  
WHERE [condition];
```

- Example:

```
UPDATE CUSTOMERS  
SET ADDRESS = 'Pune'  
WHERE ID = 6;
```

Delete Query

- **DELETE Query** is used to delete the existing records from a table.

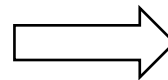
- Syntax

```
DELETE FROM table_name  
WHERE [condition];
```

- Example

```
DELETE FROM CUSTOMERS  
WHERE ID = 6;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Like Clause

- **LIKE** clause is used to compare a value to similar values using wildcard operators.
 - %: zero, one or multiple characters
 - _: single number or character

- Example syntax

```
SELECT column FROM table_name
WHERE column LIKE '_XXXX%'
```

- Example

```
SELECT * FROM CUSTOMERS
WHERE SALARY LIKE '200%'
```

- Other conditions

```
WHERE SALARY LIKE '%200%'
WHERE SALARY LIKE '%2'
WHERE SALARY LIKE '_00%'
WHERE SALARY LIKE '2_ _ %'
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00

Top Clause

- **TOP** clause is used to fetch a TOP N number or X percent records from a table.

- Syntax

```
SELECT TOP number|percent  
column_name(s)  
FROM table_name  
WHERE [condition]
```

- Example

```
SELECT TOP 3 * FROM CUSTOMERS;
```

- MySQL

```
SELECT * FROM CUSTOMERS LIMIT 3;
```

- Oracle

```
SELECT * FROM CUSTOMERS  
WHERE ROWNUM <= 3;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00

Order By

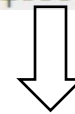
- **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the results in an *ascending order* by default.
- Syntax

```
SELECT column-list  
FROM table_name  
[WHERE condition]  
[ORDER BY column1, column2, ..  
columnN] [ASC | DESC];
```

- Example:

```
SELECT * FROM CUSTOMERS  
ORDER BY NAME DESC;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
7	Muffy	24	Indore	10000.00
6	Komal	22	MP	4500.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
4	Chaitali	25	Mumbai	6500.00

Group By

- **GROUP BY** clause is used in collaboration with the **SELECT** statement to arrange identical data into groups.

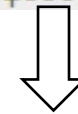
- **Syntax**

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
```

- **Example:**

```
SELECT NAME, SUM(SALARY) FROM
CUSTOMERS
GROUP BY NAME;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



NAME	SUM(SALARY)
Chaitali	6500.00
Hardik	8500.00
kaushik	2000.00
Khilan	1500.00
Komal	4500.00
Muffy	10000.00
Ramesh	2000.00

Distinct

- **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records.

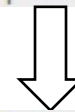
- Syntax

```
SELECT DISTINCT column1,...columnN
FROM table_name
WHERE [condition]
```

- Example:

```
SELECT DISTINCT SALARY
FROM CUSTOMERS
ORDER BY SALARY;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



SALARY
1500.00
2000.00
4500.00
6500.00
8500.00
10000.00

Excercise 1

- Give a relational database containing 4 tables:
 - customers(customerid, firstname, lastname, address, city, phone, gender)
 - orders(orderid, customerid, totalamount, orderdate)
 - orderlines(orderlineid, orderid, productid, quantity)
 - products(productid, color, title, price)
- Write SQL queries to perform following requirements:
 1. Define the database and tables above.
 2. Insert data into tables. Each table should contain at least 5 rows

Note: default format of date value is 'YYYY-MM-DD' e.g., '2021-10-20'

Part 2. SQL LANGUAGE - ADVANCED

- ALTER TABLE
- CONSTRAINTS
- JOIN
- UNION, MINUS, INTERSECT
- ALIAS
- Sub Queries
- Functions

Alter Table

- **ALTER TABLE** command is used to add, delete or modify columns in an existing table.

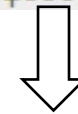
- Add new column:

```
ALTER TABLE CUSTOMERS  
ADD SEX char(1);
```

- Delete a column:

```
ALTER TABLE CUSTOMERS  
DROP SEX;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY	SEX
1	Ramesh	32	Ahmedabad	2000.00	NULL
2	Ramesh	25	Delhi	1500.00	NULL
3	kaushik	23	Kota	2000.00	NULL
4	kaushik	25	Mumbai	6500.00	NULL
5	Hardik	27	Bhopal	8500.00	NULL
6	Komal	22	MP	4500.00	NULL
7	Muffy	24	Indore	10000.00	NULL

Alter Table

- Modify data type
- SQL Server

```
ALTER TABLE table_name  
ALTER COLUMN column_name datatype;
```

- MySQL

```
ALTER TABLE table_name  
MODIFY COLUMN column_name datatype;
```

- Example

```
ALTER TABLE CUSTOMERS  
MODIFY COLUMN ADDRESS VARCHAR(30);
```

Alter Table

- Rename column in MySQL

- Syntax

```
ALTER TABLE table_name  
CHANGE old_column_name new_column_name data_type(length)
```

- MySQL [from version 8]

```
ALTER TABLE table_name  
RENAME COLUMN old_column_name TO new_column_name
```

- Example

```
ALTER TABLE CUSTOMERS  
CHANGE ADDRESS Diachi varchar(30);
```

Alter Table

- Rename table in MySQL

```
ALTER TABLE old_table_name  
RENAME new_table_name
```

- Example

```
ALTER TABLE CUSTOMERS  
RENAME KhachHang;
```

Constraint

- Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table
 1. *NOT NULL Constraint* - Ensures that a column cannot have NULL value.
 2. *DEFAULT Constraint* - Provides a default value for a column
 3. *UNIQUE Constraint* - Ensures that all values in a column are different.
 4. *PRIMARY Key* - Uniquely identifies each row/record in a table.
 5. *FOREIGN Key* - Uniquely identifies a row/record in any of the given database table.
 6. *CHECK Constraint* - The CHECK constraint ensures that all the values in a column satisfies certain conditions.
 7. *INDEX* - Used to create and retrieve data from the database very quickly

NULL

- **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.
- **Define the constraint in a new table**

```
CREATE TABLE CUSTOMERS(  
    ID            INT            NOT NULL,  
    NAME          VARCHAR (20)   NOT NULL,  
    AGE           INT            NOT NULL,  
    ADDRESS       VARCHAR (25) ,  
    SALARY        DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	

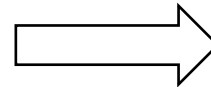
NULL

- Example

```
SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;
```

- Try with: WHERE SALARY IS NULL;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

NULL

- **Alter an existing table**

```
ALTER TABLE CUSTOMERS
```

```
MODIFY AGE int NULL;
```

```
[mysql> DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
Name	varchar(20)	NO		NULL	
AGE	int(11)	<u>NO</u>		NULL	
Address	varchar(30)	YES		NULL	
Salary	decimal(18,2)	YES		NULL	



```
[mysql> DESC CUSTOMERS;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
Name	varchar(20)	NO		NULL	
AGE	int(11)	<u>YES</u>		NULL	
Address	varchar(30)	YES		NULL	
Salary	decimal(18,2)	YES		NULL	

```
ALTER TABLE CUSTOMERS
```

```
MODIFY SALARY int NOT NULL;
```

ERROR 1265 (01000): Data truncated for column 'SALARY' at row 6

ID	Name	AGE	Address	Salary
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	keushilk	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	NULL
7	Muffy	24	Indore	NULL

DEFAULT

- **Define the constraint in a new table**

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(30),  
    City varchar(255) DEFAULT 'Sandnes'  
);
```

- **Alter an existing table**

```
ALTER TABLE CUSTOMERS  
ALTER AGE SET DEFAULT 30;
```

- **Insert data**

```
INSERT INTO CUSTOMERS (ID, NAME)  
VALUE (8, 'KITTY');
```

- **Drop a DEFAULT constraint**

```
ALTER TABLE CUSTOMERS  
ALTER AGE DROP DEFAULT;
```

UNIQUE

- **Define the constraint in a new table**

- [MySQL]

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(30) NOT NULL,  
    UNIQUE (ID)  
);
```

- SQLServer

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    Name varchar(30) NOT NULL  
);
```

UNIQUE

- **Alter an existing table**

```
ALTER TABLE CUSTOMERS  
ADD UNIQUE (Name) ;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
Name	varchar(20)	NO	UNI	NULL	
Age	int(11)	NO		NULL	
Address	varchar(30)	YES		NULL	
Salary	decimal(18,2)	YES		NULL	

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	
7	Muffy	24	Indore	

```
INSERT INTO CUSTOMERS
```

```
VALUES (8, 'Muffy', 30, NULL, NULL)
```

```
ERROR 1062 (23000): Duplicate entry 'Muffy' for key 'Name'
```

UNIQUE

- Show Constraints

```
SHOW CREATE TABLE CUSTOMERS;
```

```
| CUSTOMERS | CREATE TABLE `CUSTOMERS` (  
  `ID` int(11) NOT NULL,  
  `Name` varchar(20) NOT NULL,  
  `Age` int(11) NOT NULL,  
  `Address` varchar(30) DEFAULT NULL,  
  `Salary` decimal(18,2) DEFAULT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `Name` (`Name`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
```

- Drop a UNIQUE Constraint

```
ALTER TABLE CUSTOMERS  
DROP INDEX Name
```



UNIQUE

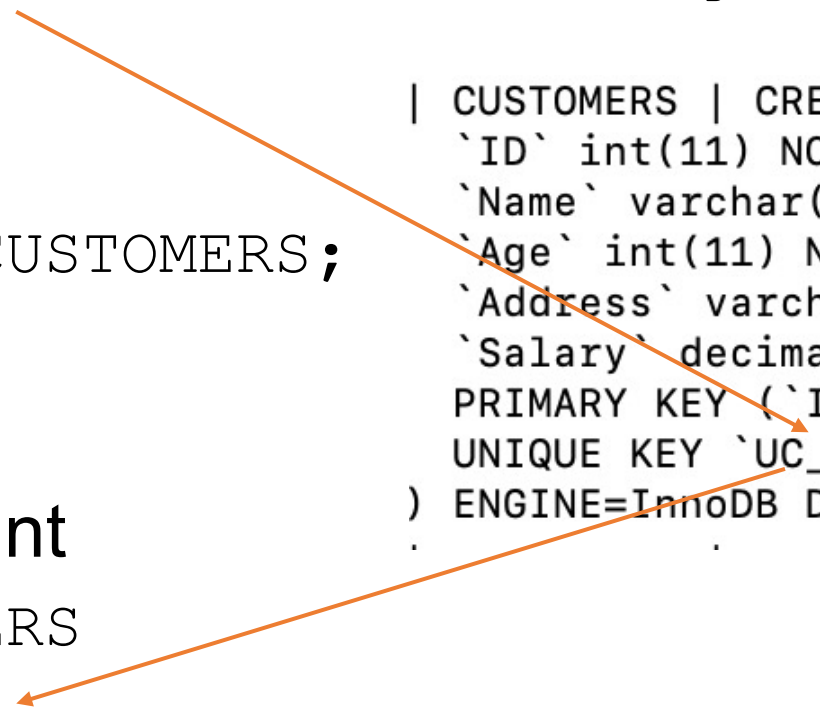
- **Alter an existing table**

```
ALTER TABLE CUSTOMERS  
ADD CONSTRAINT UC_CUST UNIQUE (Name, Age);
```

- **Show all Constraints**

```
SHOW CREATE TABLE CUSTOMERS;
```

```
| CUSTOMERS | CREATE TABLE `CUSTOMERS` (  
  `ID` int(11) NOT NULL,  
  `Name` varchar(20) NOT NULL,  
  `Age` int(11) NOT NULL,  
  `Address` varchar(30) DEFAULT NULL,  
  `Salary` decimal(18,2) DEFAULT NULL,  
  PRIMARY KEY (`ID`),  
  UNIQUE KEY `UC_CUST` (`Name`,`Age`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |  
.
```



- **Drop a UNIQUE Constraint**

```
ALTER TABLE CUSTOMERS  
DROP INDEX UC_CUST;
```


Primary Key

- The PRIMARY KEY constraint uniquely identifies each record in a table.
 - Primary keys must contain UNIQUE values, and cannot contain NULL values.
 - A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
-
- **Define the constraint in a new table**

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(30),  
    PRIMARY KEY (ID)  
);
```

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(30),  
    CONSTRAINT PK_Person  
    PRIMARY KEY (ID)  
);
```


Primary Key

- **Alter an existing table**

```
ALTER TABLE Persons  
ADD PRIMARY KEY (ID) ;
```

OR

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

- **Drop a Primary key constraint**

```
ALTER TABLE Persons  
DROP PRIMARY KEY;
```

Foreign Key

A field (or collection of fields) in one table, that refers to the **Primary key** in another table.

- **Example:**

```
CREATE TABLE CUSTOMERS (  
  ID ← INT NOT NULL,  
  NAME VARCHAR (20) NOT NULL,  
  AGE INT NOT NULL,  
  ADDRESS CHAR (25) ,  
  SALARY DECIMAL (18, 2),  
  PRIMARY KEY (ID)  
);
```

```
CREATE TABLE ORDERS (  
  ID INT NOT NULL,  
  DATE DATETIME,  
  CUSTOMER_ID INT,  
  AMOUNT double,  
  PRIMARY KEY (ID)  
);
```

Foreign Key

- **Define the constraint in a new table**

```
CREATE TABLE Orders (  
    ID int NOT NULL,  
    DATE datetime,  
    CUSTOMER_ID int,  
    AMOUNT double,  
    PRIMARY KEY (ID),  
    CONSTRAINT FK_CustomerID FOREIGN KEY (CUSTOMER_ID)  
    REFERENCES Customers (ID)  
);
```

- **Alter an existing table**

```
ALTER TABLE Orders  
ADD CONSTRAINT FK_CustomerID  
FOREIGN KEY (CUSTOMER_ID) REFERENCES Customers (ID);
```

- **Drop a Foreign key constraint**

```
ALTER TABLE Orders  
DROP FOREIGN KEY FK_CustomerID;
```

On Delete/Update Cascade

- Delete/update the rows from the child table **automatically**, when the rows from the parent table are deleted/updated
- **Define the constraint in a new table**

```
CREATE TABLE Orders2 (  
    ID int NOT NULL,  
    DATE datetime,  
    CUSTOMER_ID int,  
    AMOUNT double,  
    PRIMARY KEY (ID),  
    CONSTRAINT FK_CustomerID2 FOREIGN KEY (CUSTOMER_ID)  
    REFERENCES Customers (ID)  
    ON DELETE CASCADE  
    ON UPDATE CASCADE  
);
```

On Delete/Update Cascade

- **Alter an existing table**

- Drop the foreign key
- Add constraint

```
SHOW CREATE TABLE Orders;
```

```
ALTER TABLE Orders  
DROP FOREIGN KEY FK_CustomerID
```

```
ALTER TABLE Orders  
ADD CONSTRAINT FK_CustomerID FOREIGN KEY (CUSTOMER_ID)  
REFERENCES Customers(ID)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

CHECK

- Define the constraint in a new table

SQLServer

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(30) NOT NULL,  
    Age int CHECK (Age>=18)  
);
```

MySQL

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    Name varchar(255) NOT NULL,  
    Age int,  
    CHECK (Age>=18)  
);
```

Note: in MySQL 8.0.15 and before, CHECK clause is ignored by all storage engines

- **Alter an existing table**

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

- **Alter an existing table with a constraint name**

```
ALTER TABLE CUSTOMERS  
ADD CONSTRAINT CHK_Age CHECK (Age >= 18) ;
```

- **DROP a CHECK constraint**

```
ALTER TABLE CUSTOMERS  
DROP CHECK CHK_Age ;
```


Auto Increment

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. Default starting value is 1

- **Define the constraint in a new table**

```
CREATE TABLE Persons (  
    Personid int NOT NULL AUTO_INCREMENT,  
    Name varchar(255),  
    PRIMARY KEY (Personid)  
);
```

- **Alter an existing table**

```
ALTER TABLE table_name
```

```
MODIFY column_name INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
```

- **Change the starting value**

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

- **Insert values**

```
INSERT INTO Persons (Name) VALUES ('Anh'), ('Hanh');
```

Join

- **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Table 1 – CUSTOMERS Table

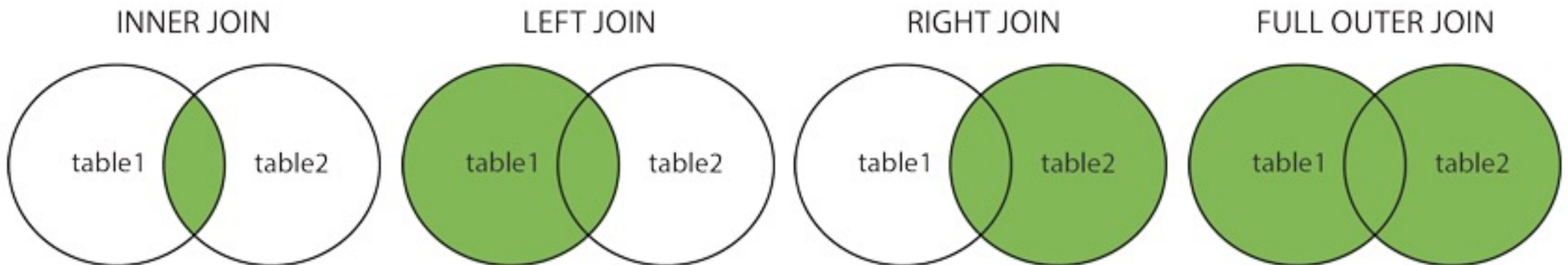
ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Join Types

- (Inner) Join: return records that have matching values in both tables
- Left (Outer) Join: return all records from the left table, and the matched records from the right table
- Right (Outer) Join: return all records from the right table, and the matched records from the left table
- Full (Outer) Join: return all records when there is a match in either left or right table



Inner Join

- Return records that have matching values in **both** tables

- Syntax

```
SELECT table1.column1, table2.column2...
```

```
FROM table1
```

```
INNER JOIN table2
```

```
ON table1.common_field =  
table2.common_field
```

- Example

```
SELECT CUSTOMERS.ID, NAME, AMOUNT, DATE
```

```
FROM CUSTOMERS INNER JOIN ORDERS
```

```
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

Left Join

- Left (Outer) Join: return all records from the **left** table, and the matched records from the **right** table

- Syntax

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

- Example

```
SELECT CUSTOMERS.ID, NAME, AMOUNT, D  
FROM CUSTOMERS LEFT JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

Right Join

- Right (Outer) Join: return all records from the **right** table, and the matched records from the **left** table

- Syntax

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

- Example

```
SELECT CUSTOMERS.ID, NAME, AMOU  
FROM CUSTOMERS RIGHT JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

Full Join

- Full (Outer) Join: return all records when there is a match in either left or right table

- Syntax

```
SELECT table1.column1, table2.column2...  
FROM table1 FULL JOIN table2  
ON table1.common_field = table2.common_field;
```

- MySQL does not support Full Join

```
SELECT CUSTOMERS.ID, NAME, AMOUNT, DATE  
FROM CUSTOMERS LEFT JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID  
UNION  
SELECT CUSTOMERS.ID, NAME, AMOUNT, DATE  
FROM CUSTOMERS RIGHT JOIN ORDERS  
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

id	name	amount	date
3	keushilk	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-10-20 00:00:00
3	keushilk	3000	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
1	Ramesh	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

CARTESIAN JOIN

- Returns the Cartesian product of the set of records from two or more tables. It equates to an inner join without any join-condition

- Syntax

```
SELECT table1.column1,  
       table2.column2...  
FROM table1, table2 [, table3 ]
```

- Example

```
SELECT CUSTOMERS.ID, NAME, AMOUNT, DATE  
FROM CUSTOMERS, ORDERS;
```

ID	Name	Amount	Date
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-10-20 00:00:00
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-10-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	keushilk	1500	2009-10-08 00:00:00
3	keushilk	1560	2009-10-20 00:00:00
3	keushilk	3000	2009-10-08 00:00:00
3	keushilk	2060	2008-05-20 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-10-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
6	Komal	1500	2009-10-08 00:00:00
6	Komal	1560	2009-10-20 00:00:00
6	Komal	3000	2009-10-08 00:00:00
6	Komal	2060	2008-05-20 00:00:00
7	Muffy	1500	2009-10-08 00:00:00
7	Muffy	1560	2009-10-20 00:00:00
7	Muffy	3000	2009-10-08 00:00:00
7	Muffy	2060	2008-05-20 00:00:00

Join

- Example:

```
SELECT CUSTOMERS.ID, NAME, AGE, AMOUNT
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.ID =
ORDERS.CUSTOMER_ID;
```

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Self Join

- Join a table to itself; temporarily renaming at least one table in the SQL statement.

- Syntax

```
SELECT a.column_name, b.column_name...  
FROM table1 a, table1 b  
WHERE a.common_field = b.common_field;
```

- Example

```
SELECT a.ID, b.NAME, a.SALARY  
FROM CUSTOMERS a, CUSTOMERS b  
WHERE a.SALARY < b.SALARY;
```

Union

- **UNION** clause/operator is used to combine the results of two or more SELECT statements **without returning any duplicate rows**. To use this UNION clause, each SELECT statement must have:
 - The same number of columns selected
 - The same data type and
 - Have them in the same order
- **UNION ALL**: allows duplicate rows in the result

- Syntax

```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ] [WHERE condition]
```

UNION

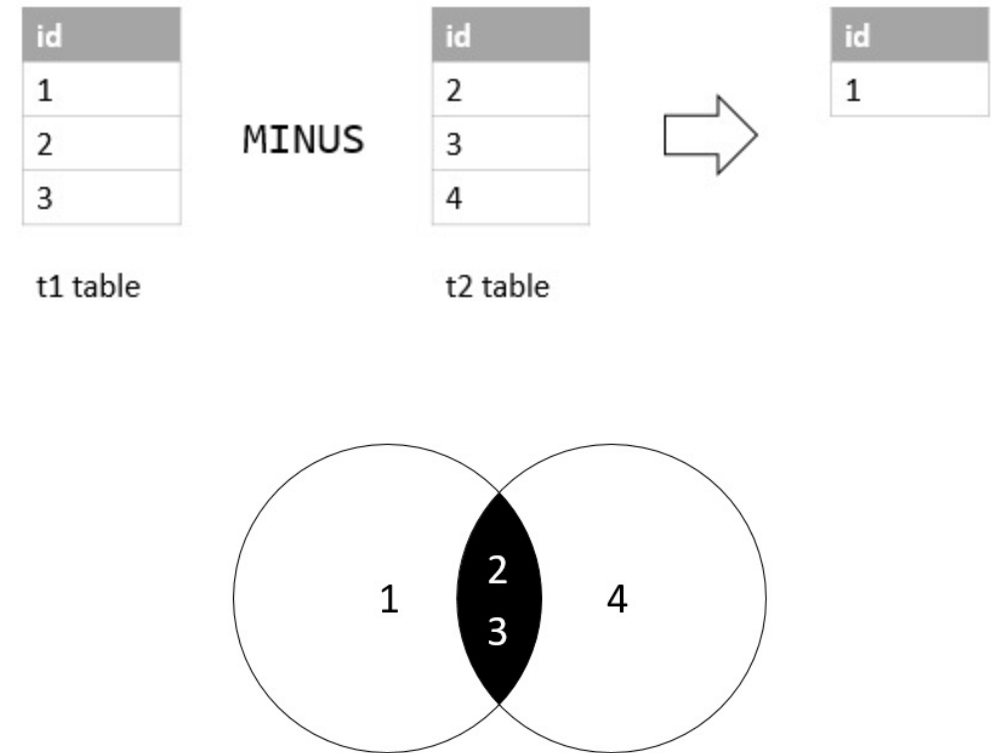
```
SELECT column1 [, column2 ]  
FROM table1 [, table2 ] [WHERE condition]
```

Union

```
SELECT ID, NAME FROM CUSTOMERS  
WHERE AGE >= 25  
  
UNION  
  
SELECT ID, NAME FROM CUSTOMERS  
WHERE SALARY >= 5000
```

MINUS, INTERSECT

- **MINUS** returns all records/rows in the first SELECT statement that are not returned by the second SELECT statement
 - Oracle supports MINUS
 - SQL Server, PostgreSQL support EXCEPT
 - MySQL does not support MINUS
- **INTERSECT** returns all records/rows in both SELECT statements
 - Remove duplicate records/rows
 - MySQL does not support INTERSECT



ALIAS

- We can rename a table or a column temporarily by giving another name known as **Alias**. The actual table name does not change in the database.

- Syntax

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

- Example

```
SELECT ID AS CUSTOMER_ID, NAME AS
CUSTOMER_NAME
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;
```

CUSTOMER_ID	CUSTOMER_NAME
1	Ramesh
2	Khilan
3	kaushik
4	Chaitali
5	Hardik
6	Komal
7	Muffy

Sub Queries

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
 - An ORDER BY command cannot be used in a subquery
 - A Subquery typically returns only one column
- Syntax

```
SELECT column_name...
```

```
FROM table...
```

```
WHERE column_name OPERATOR
```

```
(SELECT column_name [, column_name ]
```

```
FROM table [WHERE])
```

Operators

- **IN:** is used to check whether a specific value matches any value in a list

- Syntax:

```
SELECT column1, column2,...  
FROM table1, table2,...  
WHERE column1 IN ('value1', 'value2',...);
```

- **NOT IN:** is opposite IN
- **EXISTS:** is used to check whether the subquery returns any record/row

```
WHERE EXISTS (subquery)
```

- **NOT EXISTS:** is opposite EXISTS

Functions

- Some aggregate functions
 - MAX/MIN
 - SUM
 - AVG
 - COUNT
- Customers have the highest salary

```
SELECT Name FROM CUSTOMERS
```

```
WHERE Salary ≥ ALL(SELECT Salary FROM CUSTOMERS)
```

- Count the number of Customers

```
SELECT COUNT(ID) FROM CUSTOMERS
```

- Highest salary

```
SELECT MAX(Salary) FROM CUSTOMERS
```

Table 1 – CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

DATE functions

- `CURTIME ()` : the current time as a value in 'HH:MM:SS'
 - **Example:** `SELECT CURTIME ();`
 - **Result:** 23:50:26
- `CURDATE ()` : the current date as a value in 'YYYY-MM-DD'
 - **Example:** `SELECT CURDATE ();`
 - **Result:** 2019-09-30
- `DATEDIFF (expr1,expr2)` : the difference between expr1 and expr2.
 - **Example:** `SELECT DATEDIFF ('2020-10-30', '2020-10-01') AS 'Result';`
 - **Result**

```
+-----+
| Result |
+-----+
|      29 |
+-----+
```

Sub Queries - Example

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

```
SELECT *  
FROM CUSTOMERS  
WHERE ID IN (SELECT CUSTOMER_ID  
FROM ORDERS) ;
```

Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Structure of a query

```
SELECT [DISTINCT] <column 1>, <column 2>, ...  
FROM    <table 1>,<table 2>, ...  
[WHERE    <condition>]  
[GROUP BY <column 1>, <column 2>, ...  
[HAVING <condition>]]  
[ORDER BY <column 1> [ASC|DESC]]
```