SQL

CM Suvarna Varma
Assoc Prof

Unit 3

PART 1/2

syllabus

- SQL Commands: DDL, DML, TCL, DCL
- Types of Constraints (Primary, Alternate, Not Null, Check, Foreign),
- Basic form of SQL query, joins, outer joins, set operations, group operations,
- various types of queries,

- PL/SQL (Cursor, Procedures, Functions, Packages, Triggers…)

SQL

- **SQL** (Structured Query Language) is a standardized programming language that's used to manage relational databases and perform various operations on the data in them.

- ... Also known as **SQL** databases, relational systems comprise a set of tables containing data in rows and columns.

History of SQL

- The language, Structured English Query Language (SEQUEL) was developed by IBM Corporation, Inc., to use Codd's model.

- Raymond Boyce and Donald Chamberlin **developed SQL** at IBM in the early 1970s. It was **created** for getting access and modifying data held in databases.

- Initially, it was called SEQUEL (Structured English Query Language) but later needed to change its name because another business claimed that name as a trademark

- SEQUEL later became **SQL** (still pronounced "sequel").

- In 1979, Relational Software, Inc. (now Oracle) introduced the first commercially available implementation of **SQL**.

1. SolarWinds Database Performance Analyzer
2. DbVisualizer
3. ManageEngine Applications Manager
4. Altibase
5. Oracle RDBMS
6. IBM DB2
7. Microsoft SQL Server
8. SAP Sybase ASE
9. Teradata
10. ADABAS
11. MySQL
12. FileMaker
13. Microsoft Access
14. Informix
15. SQLite
16. PostgresSQL
17. AmazonRDS
18. MongoDB
19. Redis
20. CouchDB
21. Neo4j
22. OrientDB
23. Couchbase
24. Toad
25. phpMyAdmin
26. SQL Developer
27. Seqel PRO
28. Robomongo
29. Hadoop HDFS
30. Cloudera
31. MariaDB
32. Informix Dynamic Server
33. 4D (4th Dimension)

Free Databases

- **Best Free Database Software:**
- **MySQL**.
- Microsoft SQL.
- **PostgreSQL**.
- Teradata Database.
- SAP HANA, Express Edition.
- **MongoDB**.
- **CouchDB**.
- **DynamoDB**.

SQL Commands

- DDL - Data Definition Language [Create, Drop, alter, rename, Truncate]
- DML -Data Manipulation language [Select, Insert, Update, Delete]
- DCL -Data Control Language [Grant, Revoke]
- TCL -Transaction control Language [Commit, rollback , savepoint]

DDL

- CREATE TABLE or ALTER TABLE belong to the DDL.
- DDL is about "metadata".
-
- DDL includes commands such as CREATE, ALTER, and DROP statements.
- DDL are used to CREATE, ALTER, OR DROP the database objects (Table, Views, Users).
-
- Data Definition Language (DDL) is used in different statements :
- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

CREATE TABLE

- **Syntax:**

```
CREATE TABLE table_name(
Col_name1 datatype(),
Col_name2 datatype(),…
Col_namen datatype(),
);
```

1. E.g: **CREATE TABLE** DDLTest
   (id **int**, DDL_Type **varchar**(50), DDL_Value **int**
   );

ALTER TABLE -add

- Syntax:

ALTER TABLE table_name

ADD Col_name datatype()...;

E,g: **ALTER TABLE** DDLTest

ADD COLUMN DDL_Example **varchar**(50);

ALTER TABLE - modify

- Syntax:

ALTER TABLE table_name
MODIFY (fieldname datatype()...);

E.g: **ALTER TABLE** DDLTest
MODIFY DDL_Example **BIGINT**;

Decribe & Drop table

- Syntax:

DESCRIBE TABLE NAME;

- E,g: Describe DDLTest;

- Syntax:

DROP TABLE NAME;

- E,g: Drop DDLTest;

Rename

- Syntax:

RENAME table table_name new table_name;

- E,g: RENAME table DDLTest DDLTest1;

CREATE WITH KEYS

```
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) );
```

```
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)
);
```

```
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)
```

DML

- Data Manipulation Language (DML) statements are used for managing data within schema objects DML deals with data manipulation, and therefore includes most common SQL statements such as SELECT, INSERT, etc.
- DML allows adding / modifying / deleting data itself.
-
- DML is used to manipulate the existing data in the database objects (insert, select, update, delete).
-
- **DML Commands (SIDU)**
-
- 1. SELECT
- 2. INSERT
- 3.DELETE
- 4.UPDATE

INSERT

- Syntax:

INSERT INTO Table_Name VALUES();

- [can add multiple values at single time]
- [can use the order or specific insert values]

E.g:
INSERT INTO DDLTest (id, DDL_Type, DDL_Value) **VALUES** (2, 'DML', 123 , ‘ashsg’),
(3, 'DCL', 123, ‘ghgd’);

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Komal', 22, 'MP', 4500.00 );
```

Insert with field names included

```
INSERT INTO CUSTOMERS
VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );
```

Insert without field names

```
INSERT INTO CUSTOMERS VALUES (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 );
```

Insert without field names and multiple values at same time

Insert multiple rows in single query;

```
INSERT INTO CUSTOMERS VALUES (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 );
```

Insert using another table

Populate one table using another table

You can populate the data into a table through the select statement over another table; provided the other table has a set of fields, which are required to populate the first table.

Here is the syntax −

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

SELECT

- SELECT * FROM <table_name>;

- E,g: **SELECT** * **FROM** DDLTest;

UPDATE

- Syntax:

UPDATE <table name> set (to_calculation);

E,g:

```
UPDATE ddlTest  SET DDL_Value = 555 WHERE DDL_Type = '
DML';
```

DELETE

Syntax:

DELETE FROM <table_name>;

E.g: **DELETE FROM** DDLTest
Where id = 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 |
+----+----------+-----+-----------+----------+
```

```
SQL> 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 |
+----+----------+----------+
```

DCL

- DCL is the abstract of Data Control Language.
- Data Control Language includes commands such as GRANT, and is concerned with rights, permissions, and other controls of the database system.
- DCL is used to grant/revoke permissions on databases and their contents.
- DCL is simple, but MySQL permissions are a bit complex.
- DCL is about security.
- DCL is used to control the database transaction.
- DCL statements allow you to control who has access to a specific object in your database.
- 1. GRANT
- 2. REVOKE

GRANT

- **Syntax :**
- Statement permissions:
- GRANT { ALL | statement [,...n] }
  TO security_account [,...n]
-
- Normally, a database administrator first uses CREATE USER to create an account, then GRANT to define its privileges and characteristics.

GRANT

1. **CREATE** USER vatsa@'localhost' IDENTIFIED **BY** 'mypass';
2. **GRANT** ALL **ON** MY_TABLE **TO** vatsa@'localhost';
3. **GRANT SELECT ON** Users **TO** vatsa@'localhost';

REVOKE

-
- The REVOKE statement enables system administrators and to revoke (back permission) the privileges from MySQL accounts.
-
- **Syntax:**
- REVOKE

priv_type [(column_list)]
[, priv_type [(column_list)]] ...
ON [object_type] priv_level
FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [, user] ...

E,g: REVOKE INSERT ON *.* **FROM** 'vatsa'@'localhost';

TCL

TCL commands deal with the transaction within the database.

Examples of TCL commands:

1. **COMMIT**– commits a Transaction.
2. **ROLLBACK**– rollbacks a transaction in case of any error occurs.
3. **SAVEPOINT**–sets a savepoint within a transaction.
4. **SET TRANSACTION**–specify characteristics for the transaction.

COmmit

Syntax:

- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.
- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

Syntax:

```
COMMIT;
```

Rollback

- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

- Syntax:

- Rollback;

SAVEPOINT

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.
- The syntax for a SAVEPOINT command is as shown below.

- SAVEPOINT SAVEPOINT_NAME;
- This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.
- The syntax for rolling back to a SAVEPOINT is as shown below.
- ROLLBACK TO SAVEPOINT_NAME;

Consider the CUSTOMERS table having the following records −

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

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> ROLLBACK;
```

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
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

```
SQL> DELETE FROM CUSTOMERS
   WHERE AGE = 25;
SQL> COMMIT;
```

Savepoint

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2;
1 row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

```
SQL> ROLLBACK TO SP2;
Rollback complete.
```

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  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 |
+----+----------+-----+-----------+----------+
```

SQL Functions

- Queries to facilitate acquaintance of Built-In Functions, String Functions, Numeric
- Functions, Date Functions and Conversion Functions.
- **SQL FUNCTIONS:**
- SQL Functions are used to perform m calculations on data. Manipulate output from groups of rows. It can also format date members for display. It can also used for modifying individual data items. SQL function sometimes takes arguments and always returns value.
- There are two distinct types of functions:
- 1. Single Row functions
- 2. Multiple Row functions

Single Row functions

- Single row functions operate on single rows only and return one result per row.
- The different type single row functions are
- 1. Character functions
- 2. Number Functions
- 3. Date functions.
- 4. Conversion Function
- 5. General Function

Character functions

Upper	Returns char with all letters into upper case
lower	Converts the mixed case or uppercase character strings to lowercase
Initcap	Converts the first letter of each word to upper case and remaining letters to lowercase
Concat	Joins values together you are limited to two arguments with concat
Substr	This extracts a string of determined length
Length	Shows the length of a string as a numeric value
Instr	Finds numeric position of named character
Lpad	Pads the character value right justified
rpad	Pads the character value left justified
Trim	Trims heading or trailing characters from a character string
Raplace	To replace a set of character (String based)
Translat e	Change a character to a new described character(character based)

Examples

```
SQL> select concat('sql','functions') from dual;
CONCAT('SQL'
------------
Sqlfunctions

SQL> select substr('sqlfunctions',1,5) from dual;
SUBST
-----
sqlfu

SQL> select substr('sqlfunctions',2,5) from dual;
SUBST
-----
qlfun
SQL> select substr('sqlfunctions',4,5) from dual;
SUBST
-----
funct

SQL> select length('sqlfunctions') from dual;

LENGTH('SQLFUNCTIONS')
```

Number Functions

Round	Rounds the value to specified decimal
Trunc	Truncates the column, expression, or value to n decimal places
Power	Calculates the power of the given value
Mod	Finds the remainder of value1 divided by value1
Ceil	Takes the height decimal value
Floor	Takes the lowest decimal value

Examples

```
SQL> select round(35.823,2), round(35.823,0), round(35.823,-1) from dual;
ROUND(35.823,2) ROUND(35.823,0) ROUND(35.823,-1)
--------------- --------------- ---------------
          35.82              36              40

SQL> select trunc(35.823,2), trunc(35.823), trunc(35.823,-2) from dual;
TRUNC(35.823,2) TRUNC(35.823) TRUNC(35.823,-2)
--------------- ------------- ---------------
          35.82            35               0

SQL> select mod(5,2) from dual;
 MOD(5,2)
---------
         1

SQL> select mod(sal,2000) from emp where job like 'SALESMAN';
MOD(SAL,2000)
-------------
         1600
         1250
         1250
         1500
```

Date Functions

Months_between	It returns the numeric value. Finds the no. of months between date1 and date2, result may be positive or negative.
Add_months	It returns the date datatype. Adds n number of calendar months to date, n must be an integer and it can be negative
Last_day	It returns the date datatype. Date of the
Next_day	It returns the date datatype. Date of the next specified day of the week following date1, char may be number representing a day, or a character

DATE

- SYSDATE is a pseudo column that returns the current date and time. When we select sysdate it will display in a dummy table called DUAL.

- Oracle date range between 1st jan 4712 BC and 31st Dec 4712 AD.

Examples

```
SQL> select sysdate from dual;

SYSDATE
---------
08-JUL-10

SQL> select months_between(sysdate, hiredate) from emp;

MONTHS_BETWEEN(SYSDATE,HIREDATE)
--------------------------------
                      354.728983
                      352.632208
                      352.567692
                      351.212854
                      345.374144
                      350.245112
                      348.987047
                      278.664466
                      343.728983

SQL> select months_between('01-jan-2010', sysdate) from dual;
MONTHS_BETWEEN('01-JAN-2010',SYSDATE)
-------------------------------------
                         -6.2451325

SQL> select last_day(sysdate) from dual;
LAST_DAY(
---------
31-JUL-10
```

Conversion Functions

To_char(number | date,['fmt']	Converts numbers or date to character format fmt
To_number(char)	Converts char, which contains a number to a NUBER
To_date	Converts the char value representing date, into a date value according to fmt specified. If fmt is omitted, format is DD-MM-YYYY

Examples

```
SQL> select to_char(3000, '$9999.99') from dual;
TO_CHAR(3
---------
 $3000.00

SQL> select to_char(sysdate, 'fmday, ddth month yyyy') from dual;
TO_CHAR(SYSDATE,'FMDAY,DDTHMON
------------------------------
thursday, 8th july 2010

SQL> select to_char(sysdate, 'hh:mi:ss') from dual;
TO_CHAR(
--------
03:04:27

SQL> select to_char(sal,'$9999.99') from emp;
TO_CHAR(S
---------
  $800.00
 $1600.00
```

General Functions

Uid	This function returns the integer value corresponding to the user currently logged in
User	This function returns the login user name, which is in varchar2 datatype
Nvl	This function is used in case where we want to consider null values
Vsize	This function returns the number of bytes in the expression, if expression is null it returns zero.
Case	Case expression let you use IF-THEN-ELSE logic in SQL statements without having invoke procedures
Decode	Decodes and expression in a way similar IF-THEN-ELSE logic. Decodes and expression after comparing it to each search value.

Examples

```
SQL> select uid from dual;
      UID
---------
       59
```

```
SQL> select user from dual;
USER
------------------------------
SCOTT
```

```
SQL>  select ename, nvl(comm,0) from emp;
ENAME      NVL(COMM,0)
---------- -----------
SMITH                0
ALLEN              300
WARD               500
JONES                0
MARTIN            1400
BLAKE                0
CLARK                0
```

```
SQL> select ename,job,sal ,
  decode(job,'CLERK',1.10*sal,'MANAGER',1.15*sal,'SALESMAN',1.20*sal,sal) "revise
emp;

ENAME      JOB           SAL revised salary
---------- --------- ---------- --------------
SMITH      CLERK            800            880
ALLEN      SALESMAN        1600           1920
WARD       SALESMAN        1250           1500
JONES      MANAGER         2975         3421.25
```

MULTIPLE ROW FUNCTIONS

- Multi row function is also called as a group function or Aggregate function.

- Group function operate on a set of rows to give one result per group.

GROUP

A group function returns a result based on a group of rows. Some of these are just purely mathematical functions. This group function operate on sets of rows of rows to give one result per group. These sets may be the whole table or the table split into groups.

Sum	To obtain the sum of a range of values of a record set
Avg	This function will return the average of values of the column specified in the argument of column
Min	This function will give the least value of all values of the column present in the argument.
Max	This function will give the maximum value of all values of the column present in the argument.
Count	This function will return the number of rows contained to the related column

Examples

```
SQL> select sum(sal) from emp;
 SUM(SAL)
---------
    29025U

SQL> select avg(Sal) from emp;
 AVG(SAL)
---------
2073.21429

SQL> select min(sal) from emp;
 MIN(SAL)
---------
      800

SQL> select max(sal) from emp;
 MAX(SAL)
---------
     5000

SQL> select count(*) from emp;
 COUNT(*)
---------
       14
```

SQL Operators

- Arithmetic operators: add + , sub - , mul *, div / , mod %
- Comparison operators:=, != , <, >, <=, >=
- Logical operators: ALL , AND, ANY, BETWEEN, EXISTS, IN , LIKE, NOT, OR, IS NULL, UNIQUE.

Arithmetic operators

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Comparison operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Logical operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Examples

```sql
SELECT ProductName
FROM Products
WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

```sql
SELECT * FROM Customers
WHERE City = "London" AND Country = "UK";
```

```sql
SELECT * FROM Customers
WHERE City = "London" OR Country = "UK";
```

```sql
SELECT * FROM Products
WHERE Price > ANY (SELECT Price FROM Products WHERE Price > 50);
```

```sql
SELECT * FROM Products
WHERE Price BETWEEN 50 AND 60;
```

```sql
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price < 20);
```

```sql
SELECT * FROM Customers
WHERE City IN ('Paris','London');
```

```sql
SELECT * FROM Customers
WHERE City LIKE 's%';
```

```sql
SELECT * FROM Customers
WHERE City NOT LIKE 's%';
```

```sql
SELECT * FROM Products
WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);
```

SQL Constraints

- SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.
- Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `CHECK` - Ensures that the values in a column satisfies a specific condition
- `DEFAULT` - Sets a default value for a column if no value is specified
- `CREATE INDEX` - Used to create and retrieve data from the database very quickly

NOT NULL

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

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

UNIQUE

The `UNIQUE` constraint ensures that all values in a column are different.

Both the `UNIQUE` and `PRIMARY KEY` constraints provide a guarantee for uniqueness for a column or set of columns.

A `PRIMARY KEY` constraint automatically has a `UNIQUE` constraint.

However, you can have many `UNIQUE` constraints per table, but only one `PRIMARY KEY` constraint per table.

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

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).

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

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

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

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

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

```sql
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;
```

Foreign key

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL PRIMARY KEY,
    OrderNumber int NOT NULL,
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

```sql
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

CHECK

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

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

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);
```

DEFAULT

The `DEFAULT` constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

```sql
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

Index

The `CREATE INDEX` statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

```sql
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

```sql
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

```sql
CREATE INDEX idx_lastname
ON Persons (LastName);
```

```sql
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

Auto increment

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

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

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

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

Tables used in this note:

Sailors(sid: integer, sname: string, rating: integer, age: real);

Boats(bid: integer, bname: string, color: string);

Reserves(sid: integer, bid: integer, day: date).

Sailors

Sid	Sname	Rating	Age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	40
85	Art	3	25.5
95	Bob	3	63.5

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Reserves

sid	bid	day
22	101	1998-10-10
22	102	1998-10-10
22	103	1998-10-8
22	104	1998-10-7
31	102	1998-11-10
31	103	1998-11-6
31	104	1998-11-12
64	101	1998-9-5
64	102	1998-9-8
74	103	1998-9-8

EMPNO	ENAME	JOB	HIREDATE	MGR	SAL	COMM	DEPTNO
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30
7566	JONES	MANAGER	1981-04-02	7839	2975	(null)	20
7654	MARTIN	SALESMAN	1981-09-28	7698	1250	1400	30
7698	BLAKE	MANAGER	1981-05-01	7839	2850	(null)	30
7782	CLARK	MANAGER	1981-06-09	7839	2450	(null)	10
7788	SCOTT	ANALYST	1987-04-19	7566	3000	(null)	20
7839	KING	PRESIDENT	1981-11-17	7782	5000	(null)	10
7844	TURNER	SALESMAN	1981-09-08	7698	1500	0	30
7876	ADAMS	CLERK	1987-05-23	7788	1100	(null)	20
7900	JAMES	CLERK	1981-12-03	7698	950	(null)	30
7902	FORD	ANALYST	1981-12-03	7566	3000	(null)	20
7934	MILLER	CLERK	1982-01-23	7782	1300	(null)	10

DEPTNO	DNAME	DLOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Order by

ORDER BY

The `ORDER BY` command is used to sort the result set in ascending or descending order.

The `ORDER BY` command sorts the result set in ascending order by default. To sort the records in descending order, use the `DESC` keyword.

The following SQL statement selects all the columns from the "Customers" table, sorted by the "CustomerName" column:

```
SELECT * FROM Customers
ORDER BY CustomerName;
```

All Customer names are displayed in ascending order

GROUP BY

GROUP BY

The `GROUP BY` command is used to group the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG).

The following SQL lists the number of customers in each country:

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

COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland
11	France
11	Germany

HAVING

The `HAVING` command is used instead of WHERE with aggregate functions.

The following SQL lists the number of customers in each country. Only include countries with more than 5 customers:

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

COUNT(CustomerID)	Country
9	Brazil
11	France
11	Germany

SQL GROUP BY - Having

Syntax:

SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING condition];

HERE

- "SELECT statements..." is the standard SQL SELECT command query.

- "**GROUP BY** *column_name1*" is the clause that performs the grouping based on
- column_name1.
- "[,column_name2,...]" is optional; represents other column names when the grouping is done on more than one column.
- "[HAVING condition]" is optional; it is used to restrict the rows affected by the GROUP BY
clause. It is similar to the WHERE clause.

Group by Example

```
SELECT `gender` FROM `members` ;
```

gender
Female
Female
Male
Female
Male
Male
Male
Male

```
SELECT `gender` FROM `members` GROUP BY `gender`;
```

gender
Female
Male

```
SELECT `gender`,COUNT(`membership_number`)  FROM `members` GROUP BY `gender`;
```

gender	COUNT('membership_number')
Female	3
Male	5

```
SELECT `category_id`,`year_released` FROM `movies` GROUP BY `category_id`,`year_
released`;
```

category_id	year_released
NULL	2008
NULL	2010
NULL	2012
1	2011
2	2008
6	2007
7	1920
8	1920
8	2005
8	2007

Example- Group by having

```
SELECT * FROM `movies` GROUP BY `category_id`,`year_released` HAVING `category_i
d` = 8;
```

movie_id	title	director	year_released	category_id
9	Honey mooners	John Schultz	2005	8
5	Daddy's Little Girls	NULL	2007	8

What is SQL Join?

- JOIN clause combines rows from two or more tables.
- creates a set of rows in a temporary table.

- INNER JOIN
- LEFT JOIN OR LEFT OUTER JOIN
- RIGHT JOIN OR RIGHT OUTER JOIN
- FULL OUTER JOIN
- NATURAL JOIN
- CROSS JOIN
- SELF JOIN

Types of SQL JOIN

EQUI JOIN

- EQUI JOIN is a simple SQL join.
- Uses the equal sign(=) as the comparison operator for the condition

NON EQUI JOIN

- NON EQUI JOIN uses comparison operator other than the equal sign.
- The operators uses like >, <, >=, <= with the condition.

INNER JOIN

- Returns only matched rows from the participating tables.
- Match happened only at the key record of participating tables.

OUTER JOIN

- Returns all rows from one table and
- Matching rows from the secondary table and
- Comparison columns should be equal in both the tables.

Example : INNER JOIN

A	M
1	m
2	n
4	o

table_A

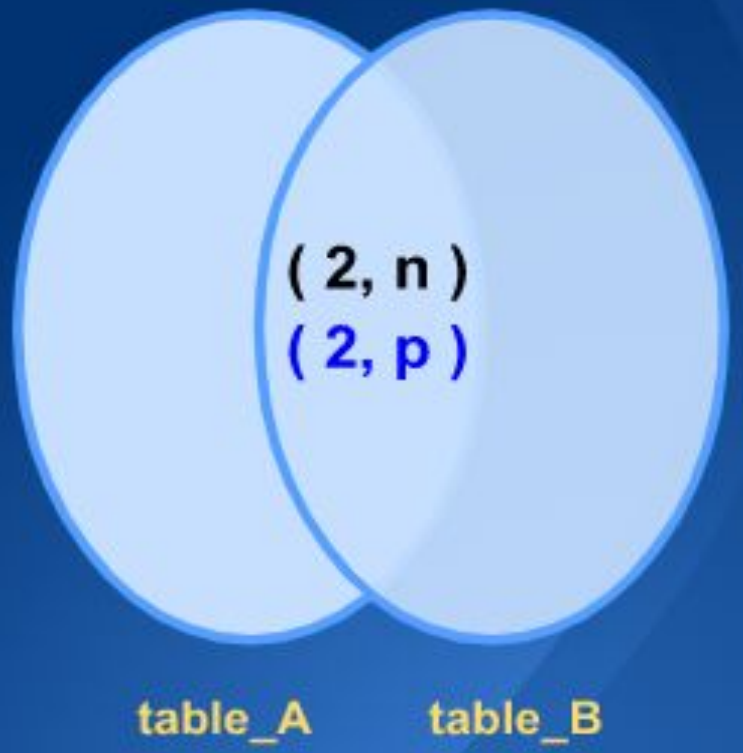
A	N
2	p
3	q
5	r

table_B

```
SELECT * FROM table_A
INNER JOIN table_B
ON table_A.A=table_B.A;
```

A	M	A	N
2	n	2	p

Output



```
select * from emp inner join dept where emp.deptno = dept.deptno;
```

EMPNO	ENAME	JOB	HIREDATE	MGR	SAL	COMM	DEPTNO	DEPTNO	DNAME	DLOC
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30	30	SALES	CHICAGO
7566	JONES	MANAGER	1981-04-02	7839	2975	(null)	20	20	RESEARCH	DALLAS
7654	MARTIN	SALESMAN	1981-09-28	7698	1250	1400	30	30	SALES	CHICAGO
7698	BLAKE	MANAGER	1981-05-01	7839	2850	(null)	30	30	SALES	CHICAGO
7782	CLARK	MANAGER	1981-06-09	7839	2450	(null)	10	10	ACCOUNTING	NEW YORK

Example : LEFT JOIN or LEFT OUTER JOIN

A	M
1	m
2	n
4	o

table_A

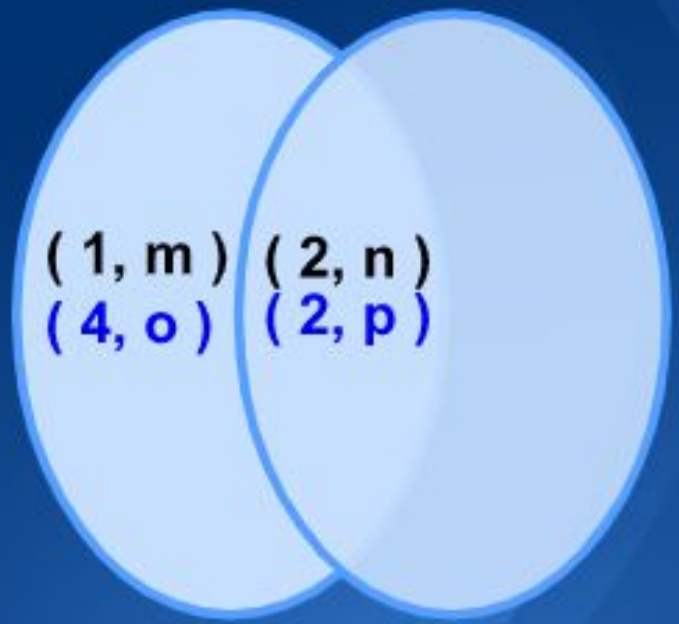
A	N
2	p
3	q
5	r

table_B

```
SELECT * FROM table_A
LEFT JOIN table_B
ON table_A.A=table_B.A;
```

A	M	A	N
2	n	2	p
1	m	null	null
4	o	null	null

Output



```
SELECT A.`title` , B.`first_name` , B.`last_name`
FROM `movies` AS A
LEFT JOIN `members` AS B
ON B.`movie_id` = A.`id`
```

title	first_name	last_name
ASSASSIN'S CREED: EMBERS	Adam	Smith
Real Steel(2012)	Ravi	Kumar
Safe (2012)	Susan	Davidson
Deadline(2009)	Jenny	Adrianna
Marley and me	Lee	Pong
Alvin and the Chipmunks	NULL	NULL
The Adventures of Tin Tin	NULL	NULL
Safe House(2012)	NULL	NULL
GIA	NULL	NULL
The Dirty Picture	NULL	NULL

Example : RIGHT JOIN or RIGHT OUTER JOIN

A	M
1	m
2	n
4	o

table_A

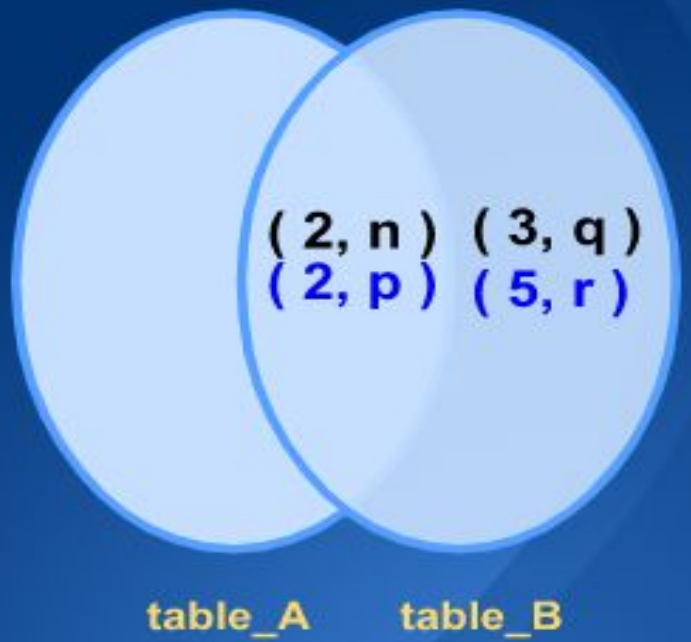
A	N
2	p
3	q
5	r

table_B

```
SELECT * FROM table_A
RIGHT JOIN table_B
ON table_A.A=table_B.A;
```

A	M	A	N
2	n	2	p
null	null	3	q
null	null	5	r

Output



```
SELECT  A.`first_name` , A.`last_name`, B.`title`
FROM `members` AS A
RIGHT JOIN `movies` AS B
ON B.`id` = A.`movie_id`
```

first_name	last_name	title
Adam	Smith	ASSASSIN'S CREED: EMBERS
Ravi	Kumar	Real Steel(2012)
Susan	Davidson	Safe (2012)
Jenny	Adrianna	Deadline(2009)
Lee	Pong	Marley and me
NULL	NULL	Alvin and the Chipmunks
NULL	NULL	The Adventures of Tin Tin
NULL	NULL	Safe House(2012)
NULL	NULL	GIA
NULL	NULL	The Dirty Picture

Example : FULL OUTER JOIN

```
SELECT * FROM table_A
FULL OUTER JOIN table_B
ON table_A.A=table_B.A;
```

A	M
1	m
2	n
4	o

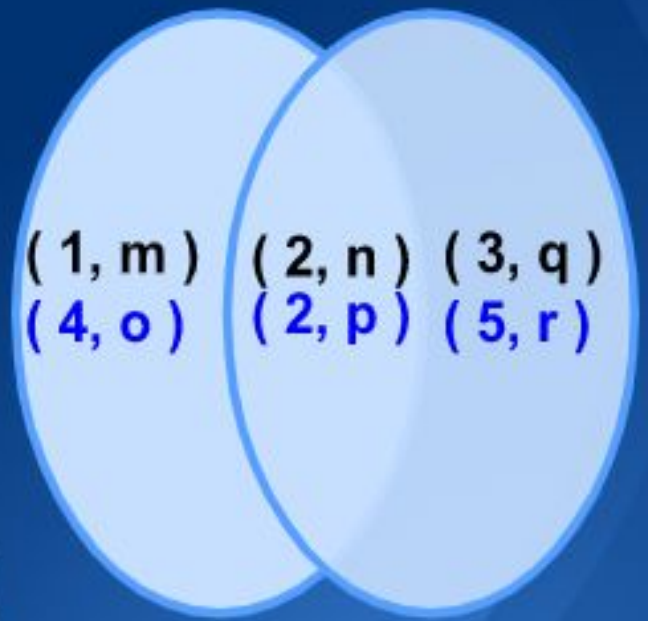
table_A

A	N
2	p
3	q
5	r

table_B

A	M	A	N
2	n	2	p
1	m	null	null
4	o	null	null
null	null	3	q
null	null	5	r

Output



Example : NATURAL JOIN

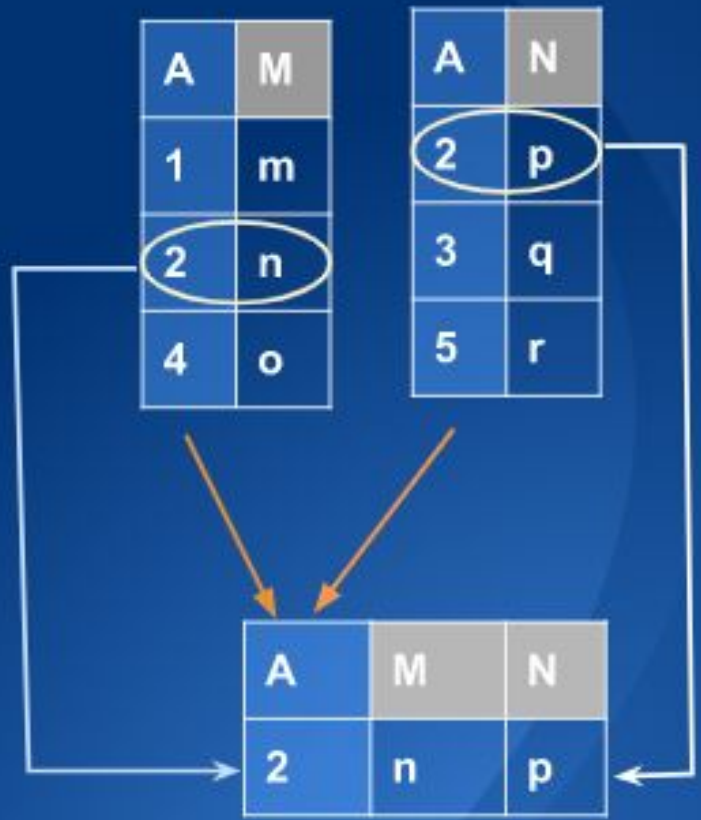
A	M
1	m
2	n
4	o

table_A

A	N
2	p
3	q
5	r

table_B

```
SELECT *
FROM table_A
NATURAL JOIN table_B;
```



Output

```
select * from emp natural join dept;
```

DEPTNO	EMPNO	ENAME	JOB	HIREDATE	MGR	SAL	COMM	DNAME	DLOC
20	7369	SMITH	CLERK	1980-12-17	7902	800	(null)	RESEARCH	DALLAS
30	7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	SALES	CHICAGO
30	7521	WARD	SALESMAN	1981-02-22	7698	1250	500	SALES	CHICAGO
20	7566	JONES	MANAGER	1981-04-02	7839	2975	(null)	RESEARCH	DALLAS
30	7654	MARTIN	SALESMAN	1981-09-28	7698	1250	1400	SALES	CHICAGO
30	7698	BLAKE	MANAGER	1981-05-01	7839	2850	(null)	SALES	CHICAGO
10	7782	CLARK	MANAGER	1981-06-09	7839	2450	(null)	ACCOUNTING	NEW YORK
20	7788	SCOTT	ANALYST	1987-04-19	7566	3000	(null)	RESEARCH	DALLAS
10	7839	KING	PRESIDENT	1981-11-17	7782	5000	(null)	ACCOUNTING	NEW YORK
30	7844	TURNER	SALESMAN	1981-09-08	7698	1500	0	SALES	CHICAGO
20	7876	ADAMS	CLERK	1987-06-23	7788	1100	(null)	RESEARCH	DALLAS

Example : CROSS JOIN

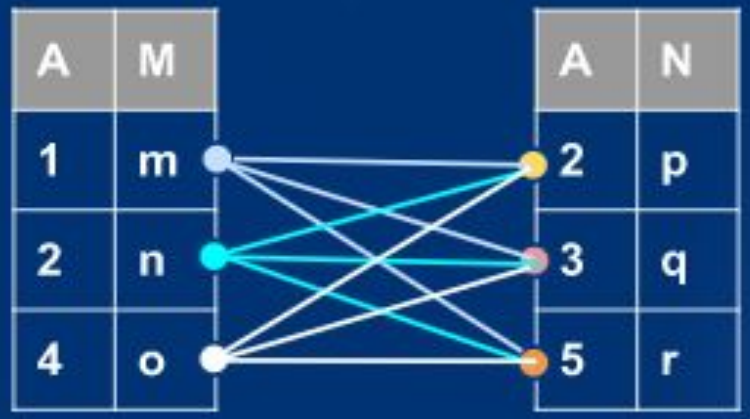
A	M
1	m
2	n
4	o

table_A

A	N
2	p
3	q
5	r

table_B

```
SELECT *
FROM table_A
CROSS JOIN  table_B;
```



A	M	A	N
1	m	2	p
2	n	2	p
4	o	2	p
1	m	3	q
2	n	3	q
4	o	3	q
1	m	5	r
2	n	5	r
4	o	5	r

Output

```
select * from emp cross join dept;
```

EMPNO	ENAME	JOB	HIREDATE	MGR	SAL	COMM	DEPTNO	DEPTNO	DNAME	DLOC
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20	10	ACCOUNTING	NEW YORK
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20	20	RESEARCH	DALLAS
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20	30	SALES	CHICAGO
7369	SMITH	CLERK	1980-12-17	7902	800	(null)	20	40	OPERATIONS	BOSTON
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30	10	ACCOUNTING	NEW YORK
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30	20	RESEARCH	DALLAS
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30	30	SALES	CHICAGO
7499	ALLEN	SALESMAN	1981-02-20	1600	7698	300	30	40	OPERATIONS	BOSTON
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30	10	ACCOUNTING	NEW YORK
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30	20	RESEARCH	DALLAS
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30	30	SALES	CHICAGO
7521	WARD	SALESMAN	1981-02-22	7698	1250	500	30	40	OPERATIONS	BOSTON

Example : SELF JOIN

A	M
1	m
2	n
4	o

table_A

A	M
1	m
2	n
4	o

table_A

```
SELECT *
FROM table_A X, table_A Y
WHERE X.A=Y.A;
```

A	M	A	N
1	m	1	m
2	n	2	n
4	o	4	o

Output

staff_id	first_name	last_name	email	phone	active	store_id	manager_id
1	Fabiola	Jackson	fabiola.jackson@bikes.shop	(831) 555-5554	1	1	NULL
2	Mireya	Copeland	mireya.copeland@bikes.shop	(831) 555-5555	1	1	1
3	Genna	Serrano	genna.serrano@bikes.shop	(831) 555-5556	1	1	2
4	Virgie	Wiggins	virgie.wiggins@bikes.shop	(831) 555-5557	1	1	2
5	Jannette	David	jannette.david@bikes.shop	(516) 379-4444	1	2	1
6	Marcelene	Boyer	marcelene.boyer@bikes.shop	(516) 379-4445	1	2	5
7	Venita	Daniel	venita.daniel@bikes.shop	(516) 379-4446	1	2	5
8	Kali	Vargas	kali.vargas@bikes.shop	(972) 530-5555	1	3	1
9	Layla	Terrell	layla.terrell@bikes.shop	(972) 530-5556	1	3	7
10	Bernardine	Houston	bernardine.houston@bikes.shop	(972) 530-5557	1	3	7

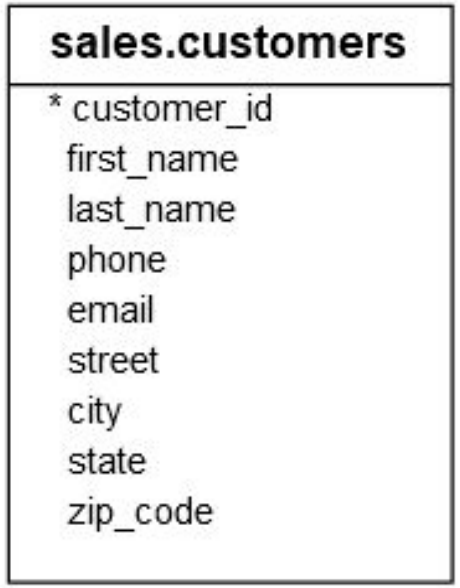
The `staffs` table stores the staff information such as id, first name, last name, and email. It also has a column named `manager_id` that specifies the direct manager. For example, `Mireya` reports to `Fabiola` because the value in the `manager_id` of `Mireya` is `Fabiola`.

To get who reports to whom, you use the self join as shown in the following query:

```
SELECT
    e.first_name + ' ' + e.last_name employee,
    m.first_name + ' ' + m.last_name manager
FROM
    sales.staffs e
INNER JOIN sales.staffs m ON m.staff_id = e.manager_id
ORDER BY
    manager;
```

employee	manager
Mireya Copeland	Fabiola Jackson
Jannette David	Fabiola Jackson
Kali Vargas	Fabiola Jackson
Marcelene Boyer	Jannette David
Venita Daniel	Jannette David
Genna Serrano	Mireya Copeland
Virgie Wiggins	Mireya Copeland
Layla Terrell	Venita Daniel
Bernardine Houston	Venita Daniel

See the following `customers` table:



The following statement uses the self join to find the customers who locate in the same city.

```
SELECT
    c1.city,
    c1.first_name + ' ' + c1.last_name customer_1,
    c2.first_name + ' ' + c2.last_name customer_2
FROM
    sales.customers c1
INNER JOIN sales.customers c2 ON c1.customer_id > c2.customer_id
AND c1.city = c2.city
ORDER BY
    city,
    customer_1,
    customer_2;
```

city	customer_1	customer_2
Albany	Douglass Blankenship	Mi Gray
Albany	Douglass Blankenship	Priscilla Wilkins
Albany	Mi Gray	Priscilla Wilkins
Amarillo	Andria Rivers	Delaine Estes
Amarillo	Andria Rivers	Jonell Rivas
Amarillo	Andria Rivers	Luis Tyler
Amarillo	Andria Rivers	Narcisa Knapp
Amarillo	Delaine Estes	Jonell Rivas
Amarillo	Delaine Estes	Luis Tyler
Amarillo	Delaine Estes	Narcisa Knapp
Amarillo	Jonell Rivas	Luis Tyler
Amarillo	Jonell Rivas	Narcisa Knapp
Amarillo	Narcisa Knapp	Luis Tyler
Amityville	Abby Gamble	Tenisha Lyons
Amityville	Barton Cox	Abby Gamble
Amityville	Barton Cox	Kylee Dickson
Amityville	Barton Cox	Marisa Chambers