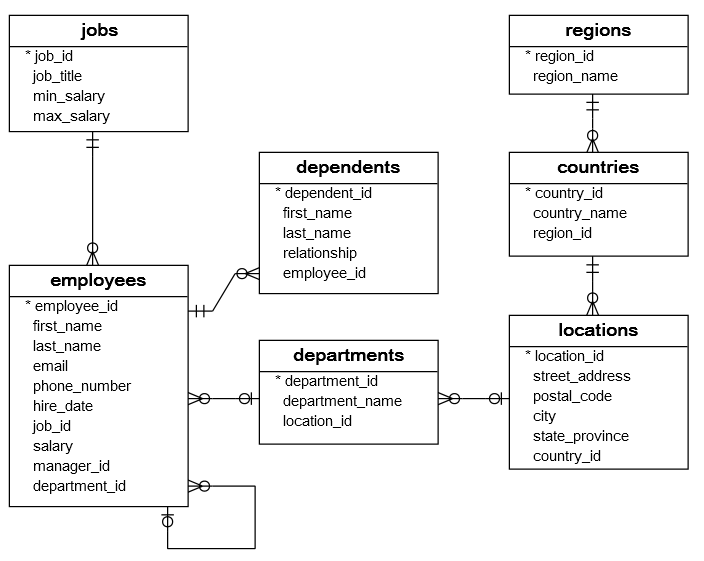


SQL Mega Project

**HR DATABASE MANAGEMENT SYSTEM**



Overview of Project:--

The HR sample database has seven tables:

1. The employees table stores the data of employees.
2. The jobs table stores the job data including job title and salary range.
3. The departments table stores department data.
4. The dependents table stores the employee’s dependents.
5. The locations table stores the location of the departments of the company.
6. The countries table stores the data of countries where the company is doing business.
7. The regions table stores the data of regions such as Asia, Europe, America, and the Middle East and Africa. The countries are grouped into regions.

The following picture shows the table names and their records.

| **Table** | **Rows** |
| --- | --- |
| employees | 40 |
| dependents | 30 |
| departments | 11 |
| jobs | 11 |
| locations | 7 |
| countries | 25 |
| regions | 4 |

Create the tables given below or else you can Copy the below table and paste it into your SSMS(SQL Server Management Studio)

CREATE TABLE regions (

region\_id INT IDENTITY(1,1) PRIMARY KEY,

region\_name VARCHAR (25) DEFAULT NULL

);

CREATE TABLE countries (

country\_id CHAR (2) PRIMARY KEY,

country\_name VARCHAR (40) DEFAULT NULL,

region\_id INT NOT NULL,

FOREIGN KEY (region\_id) REFERENCES regions (region\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE locations (

location\_id INT IDENTITY(1,1) PRIMARY KEY,

street\_address VARCHAR (40) DEFAULT NULL,

postal\_code VARCHAR (12) DEFAULT NULL,

city VARCHAR (30) NOT NULL,

state\_province VARCHAR (25) DEFAULT NULL,

country\_id CHAR (2) NOT NULL,

FOREIGN KEY (country\_id) REFERENCES countries (country\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE jobs (

job\_id INT IDENTITY(1,1) PRIMARY KEY,

job\_title VARCHAR (35) NOT NULL,

min\_salary DECIMAL (8, 2) DEFAULT NULL,

max\_salary DECIMAL (8, 2) DEFAULT NULL

);

CREATE TABLE departments (

department\_id INT IDENTITY(1,1) PRIMARY KEY,

department\_name VARCHAR (30) NOT NULL,

location\_id INT DEFAULT NULL,

FOREIGN KEY (location\_id) REFERENCES locations (location\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE employees (

employee\_id INT IDENTITY(1,1) PRIMARY KEY,

first\_name VARCHAR (20) DEFAULT NULL,

last\_name VARCHAR (25) NOT NULL,

email VARCHAR (100) NOT NULL,

phone\_number VARCHAR (20) DEFAULT NULL,

hire\_date DATE NOT NULL,

job\_id INT NOT NULL,

salary DECIMAL (8, 2) NOT NULL,

manager\_id INT DEFAULT NULL,

department\_id INT DEFAULT NULL,

FOREIGN KEY (job\_id) REFERENCES jobs (job\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (department\_id) REFERENCES departments (department\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (manager\_id) REFERENCES employees (employee\_id)

);

CREATE TABLE dependents (

dependent\_id INT IDENTITY(1,1) PRIMARY KEY,

first\_name VARCHAR (50) NOT NULL,

last\_name VARCHAR (50) NOT NULL,

relationship VARCHAR (25) NOT NULL,

employee\_id INT NOT NULL,

FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id) ON DELETE CASCADE ON UPDATE CASCADE

);

After Creating All tables insert the given records and execute it in your SSMS

**/\*Data for the table regions \*/**

INSERT INTO regions(region\_id,region\_name)

VALUES (1,'Europe');

INSERT INTO regions(region\_id,region\_name)

VALUES (2,'Americas');

INSERT INTO regions(region\_id,region\_name)

VALUES (3,'Asia');

INSERT INTO regions(region\_id,region\_name)

VALUES (4,'Middle East and Africa');

**/\*Data for the table countries \*/**

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('AR','Argentina',2);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('AU','Australia',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('BE','Belgium',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('BR','Brazil',2);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('CA','Canada',2);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('CH','Switzerland',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('CN','China',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('DE','Germany',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('DK','Denmark',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('EG','Egypt',4);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('FR','France',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('HK','HongKong',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('IL','Israel',4);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('IN','India',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('IT','Italy',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('JP','Japan',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('KW','Kuwait',4);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('MX','Mexico',2);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('NG','Nigeria',4);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('NL','Netherlands',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('SG','Singapore',3);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('UK','United Kingdom',1);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('US','United States of America',2);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('ZM','Zambia',4);

INSERT INTO countries(country\_id,country\_name,region\_id)

VALUES ('ZW','Zimbabwe',4);

**/\*Data for the table locations \*/**

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (1400,'2014 Jabberwocky Rd','26192','Southlake','Texas','US');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (1500,'2011 Interiors Blvd','99236','South San Francisco','California','US');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (1700,'2004 Charade Rd','98199','Seattle','Washington','US');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (1800,'147 Spadina Ave','M5V 2L7','Toronto','Ontario','CA');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (2400,'8204 Arthur St',NULL,'London',NULL,'UK');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (2500,'Magdalen Centre, The Oxford Science Park','OX9 9ZB','Oxford','Oxford','UK');

INSERT INTO locations(location\_id,street\_address,postal\_code,city,state\_province,country\_id)

VALUES (2700,'Schwanthalerstr. 7031','80925','Munich','Bavaria','DE');

**/\*Data for the table jobs \*/**

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (1,'Public Accountant',4200.00,9000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (2,'Accounting Manager',8200.00,16000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (3,'Administration Assistant',3000.00,6000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (4,'President',20000.00,40000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (5,'Administration Vice President',15000.00,30000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (6,'Accountant',4200.00,9000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (7,'Finance Manager',8200.00,16000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (8,'Human Resources Representative',4000.00,9000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (9,'Programmer',4000.00,10000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (10,'Marketing Manager',9000.00,15000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (11,'Marketing Representative',4000.00,9000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (12,'Public Relations Representative',4500.00,10500.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (13,'Purchasing Clerk',2500.00,5500.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (14,'Purchasing Manager',8000.00,15000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (15,'Sales Manager',10000.00,20000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (16,'Sales Representative',6000.00,12000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (17,'Shipping Clerk',2500.00,5500.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (18,'Stock Clerk',2000.00,5000.00);

INSERT INTO jobs(job\_id,job\_title,min\_salary,max\_salary)

VALUES (19,'Stock Manager',5500.00,8500.00);

**/\*Data for the table departments \*/**

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (1,'Administration',1700);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (2,'Marketing',1800);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (3,'Purchasing',1700);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (4,'Human Resources',2400);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (5,'Shipping',1500);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (6,'IT',1400);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (7,'Public Relations',2700);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (8,'Sales',2500);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (9,'Executive',1700);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (10,'Finance',1700);

INSERT INTO departments(department\_id,department\_name,location\_id)

VALUES (11,'Accounting',1700);

**/\*Data for the table employees \*/**

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (100,'Steven','King','steven.king@sqltutorial.org','515.123.4567','1987-06-17',4,24000.00,NULL,9);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (101,'Neena','Kochhar','neena.kochhar@sqltutorial.org','515.123.4568','1989-09-21',5,17000.00,100,9);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (102,'Lex','De Haan','lex.de haan@sqltutorial.org','515.123.4569','1993-01-13',5,17000.00,100,9);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (103,'Alexander','Hunold','alexander.hunold@sqltutorial.org','590.423.4567','1990-01-03',9,9000.00,102,6);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (104,'Bruce','Ernst','bruce.ernst@sqltutorial.org','590.423.4568','1991-05-21',9,6000.00,103,6);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (105,'David','Austin','david.austin@sqltutorial.org','590.423.4569','1997-06-25',9,4800.00,103,6);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (106,'Valli','Pataballa','valli.pataballa@sqltutorial.org','590.423.4560','1998-02-05',9,4800.00,103,6);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (107,'Diana','Lorentz','diana.lorentz@sqltutorial.org','590.423.5567','1999-02-07',9,4200.00,103,6);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (108,'Nancy','Greenberg','nancy.greenberg@sqltutorial.org','515.124.4569','1994-08-17',7,12000.00,101,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (109,'Daniel','Faviet','daniel.faviet@sqltutorial.org','515.124.4169','1994-08-16',6,9000.00,108,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (110,'John','Chen','john.chen@sqltutorial.org','515.124.4269','1997-09-28',6,8200.00,108,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (111,'Ismael','Sciarra','ismael.sciarra@sqltutorial.org','515.124.4369','1997-09-30',6,7700.00,108,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (112,'Jose Manuel','Urman','jose manuel.urman@sqltutorial.org','515.124.4469','1998-03-07',6,7800.00,108,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (113,'Luis','Popp','luis.popp@sqltutorial.org','515.124.4567','1999-12-07',6,6900.00,108,10);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (114,'Den','Raphaely','den.raphaely@sqltutorial.org','515.127.4561','1994-12-07',14,11000.00,100,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (115,'Alexander','Khoo','alexander.khoo@sqltutorial.org','515.127.4562','1995-05-18',13,3100.00,114,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (116,'Shelli','Baida','shelli.baida@sqltutorial.org','515.127.4563','1997-12-24',13,2900.00,114,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (117,'Sigal','Tobias','sigal.tobias@sqltutorial.org','515.127.4564','1997-07-24',13,2800.00,114,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (118,'Guy','Himuro','guy.himuro@sqltutorial.org','515.127.4565','1998-11-15',13,2600.00,114,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (119,'Karen','Colmenares','karen.colmenares@sqltutorial.org','515.127.4566','1999-08-10',13,2500.00,114,3);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (120,'Matthew','Weiss','matthew.weiss@sqltutorial.org','650.123.1234','1996-07-18',19,8000.00,100,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (121,'Adam','Fripp','adam.fripp@sqltutorial.org','650.123.2234','1997-04-10',19,8200.00,100,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (122,'Payam','Kaufling','payam.kaufling@sqltutorial.org','650.123.3234','1995-05-01',19,7900.00,100,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (123,'Shanta','Vollman','shanta.vollman@sqltutorial.org','650.123.4234','1997-10-10',19,6500.00,100,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (126,'Irene','Mikkilineni','irene.mikkilineni@sqltutorial.org','650.124.1224','1998-09-28',18,2700.00,120,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (145,'John','Russell','john.russell@sqltutorial.org',NULL,'1996-10-01',15,14000.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (146,'Karen','Partners','karen.partners@sqltutorial.org',NULL,'1997-01-05',15,13500.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (176,'Jonathon','Taylor','jonathon.taylor@sqltutorial.org',NULL,'1998-03-24',16,8600.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (177,'Jack','Livingston','jack.livingston@sqltutorial.org',NULL,'1998-04-23',16,8400.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (178,'Kimberely','Grant','kimberely.grant@sqltutorial.org',NULL,'1999-05-24',16,7000.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (179,'Charles','Johnson','charles.johnson@sqltutorial.org',NULL,'2000-01-04',16,6200.00,100,8);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (192,'Sarah','Bell','sarah.bell@sqltutorial.org','650.501.1876','1996-02-04',17,4000.00,123,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (193,'Britney','Everett','britney.everett@sqltutorial.org','650.501.2876','1997-03-03',17,3900.00,123,5);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (200,'Jennifer','Whalen','jennifer.whalen@sqltutorial.org','515.123.4444','1987-09-17',3,4400.00,101,1);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (201,'Michael','Hartstein','michael.hartstein@sqltutorial.org','515.123.5555','1996-02-17',10,13000.00,100,2);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (202,'Pat','Fay','pat.fay@sqltutorial.org','603.123.6666','1997-08-17',11,6000.00,201,2);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (203,'Susan','Mavris','susan.mavris@sqltutorial.org','515.123.7777','1994-06-07',8,6500.00,101,4);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (204,'Hermann','Baer','hermann.baer@sqltutorial.org','515.123.8888','1994-06-07',12,10000.00,101,7);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (205,'Shelley','Higgins','shelley.higgins@sqltutorial.org','515.123.8080','1994-06-07',2,12000.00,101,11);

INSERT INTO employees(employee\_id,first\_name,last\_name,email,phone\_number,hire\_date,job\_id,salary,manager\_id,department\_id)

VALUES (206,'William','Gietz','william.gietz@sqltutorial.org','515.123.8181','1994-06-07',1,8300.00,205,11);

**/\*Data for the table dependents \*/**

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (1,'Penelope','Gietz','Child',206);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (2,'Nick','Higgins','Child',205);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (3,'Ed','Whalen','Child',200);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (4,'Jennifer','King','Child',100);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (5,'Johnny','Kochhar','Child',101);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (6,'Bette','De Haan','Child',102);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (7,'Grace','Faviet','Child',109);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (8,'Matthew','Chen','Child',110);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (9,'Joe','Sciarra','Child',111);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (10,'Christian','Urman','Child',112);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (11,'Zero','Popp','Child',113);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (12,'Karl','Greenberg','Child',108);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (13,'Uma','Mavris','Child',203);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (14,'Vivien','Hunold','Child',103);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (15,'Cuba','Ernst','Child',104);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (16,'Fred','Austin','Child',105);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (17,'Helen','Pataballa','Child',106);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (18,'Dan','Lorentz','Child',107);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (19,'Bob','Hartstein','Child',201);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (20,'Lucille','Fay','Child',202);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (21,'Kirsten','Baer','Child',204);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (22,'Elvis','Khoo','Child',115);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (23,'Sandra','Baida','Child',116);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (24,'Cameron','Tobias','Child',117);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (25,'Kevin','Himuro','Child',118);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (26,'Rip','Colmenares','Child',119);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (27,'Julia','Raphaely','Child',114);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (28,'Woody','Russell','Child',145);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (29,'Alec','Partners','Child',146);

INSERT INTO dependents(dependent\_id,first\_name,last\_name,relationship,employee\_id)

VALUES (30,'Sandra','Taylor','Child',176);

Based on Above Database Solve the following Task:--

**TASK 1:🡪**

**1)WRITE A QUERY FOR SELECT STATEMENTS :-**

**Syntax of SELECT STATEMENT:-**

**SELECT**

**select\_list**

**FROM**

**table\_name;**

1. To get data from all the rows and columns in the employees table:

Answer- SELECT \*

FROM employees;

1. select data from the employee id, first name, last name, and hire date of all rows in the employees table:

Answer- SELECT employee\_id, first\_name, last\_name, hire\_date

FROM employees;

1. to get the first name, last name, salary, and new salary:

Answer- SELECT

first\_name,

last\_name,

salary,

salary \* 1.1 AS new\_salary

FROM

employees

1. Increase the salary two times and named as New\_SALARY from employees table

Answer- SELECT

first\_name,

last\_name,

salary,

salary \* 2 AS New\_SALARY

FROM

employees

**2)WRITE A QUERY FOR ORDER BY STATEMENTS :-**

**Syntax of ORDER BY Statements:-**

**SELECT**

**select\_list**

**FROM**

**table\_name**

**ORDER BY**

**sort\_expression1 [ASC | DESC],**

**sort\_expression 2[ASC | DESC];**

1. returns the data from the employee id, first name, last name, hire date, and salary column of the employees table:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

employee\_id

1. to sort employees by first names in alphabetical order:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

first\_name

1. to sort the employees by the first name in ascending order and the last name in descending order:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

first\_name ASC,

last\_name DESC

1. to sort employees by salary from high to low:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

salary DESC

1. to sort the employees by values in the hire\_date column from:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

hire\_date ASC

1. sort the employees by the hire dates in descending order:

Answer- SELECT

employee\_id,

first\_name,

last\_name,

hire\_date,

salary

FROM

employees

ORDER BY

hire\_date DESC

**3)WRITE A QUERY FOR DISTINCT STATEMENTS :-**

**Syntax of DISTINCT Statements:-**

SELECT DISTINCT

column1, column2, ...

FROM

table1;

1. selects the salary data from the salary column of the employees table and [sorts](https://www.sqltutorial.org/sql-order-by/) them from high to low.

Answer-SELECT DISTINCT

salary

FROM

employees

ORDER BY

salary DESC

1. select unique values from the salary column of the employees table:

Answer- SELECT DISTINCT

salary

FROM

employees

1. selects the job id and salary from the employees table:

Answer- SELECT DISTINCT

job\_id,

salary

FROM

Employees

1. to remove the duplicate values in job id and salary:

Answer- SELECT DISTINCT

job\_id,

salary

FROM

employees

1. returns the distinct phone numbers of employees:

Answer- SELECT DISTINCT

phone\_number

FROM

employees

**4)WRITE A QUERY FOR TOP N STATEMENTS :-**

**Syntax of TOP N Statements(N=Will be any nos)**

**SELECT TOP N**

**column\_list**

**FROM**

**table1**

**ORDER BY column\_list**

1. returns all rows in the employees table sorted by the first\_name column.

Answer-SELECT TOP (SELECT COUNT(\*) FROM employees)

\*

FROM

employees

ORDER BY

first\_name

1. to return the first 5 rows in the result set returned by the SELECT clause:

Answer- SELECT TOP 5

\*

FROM

employees

1. to return five rows starting from the 4th row:

Answer- SELECT

\*

FROM

employees

ORDER BY

employee\_id

OFFSET 3 ROWS

FETCH NEXT 5 ROWS ONLY

1. gets the top five employees with the highest salaries.

Answer- SELECT TOP 5

\*

FROM

employees

ORDER BY

salary DESC

1. to get employees who have the 2nd highest salary in the company

Answer- SELECT TOP 1

\*

FROM

employees

WHERE

salary < (SELECT MAX(salary) FROM employees)

ORDER BY

salary DESC

**5)WRITE A QUERY FOR WHERE CLAUSE and COMPARISON OPERATORS :-**

**Syntax of WHERE CLAUSE and COMPARISON OPERATORS:--**

**SELECT**

**column1, column2, ...**

**FROM**

**table\_name**

**WHERE**

**condition;**

The WHERE clause appears immediately after the FROM clause. The WHERE clause contains one or more logical expressions that evaluate each row in the table. If a row that causes the condition evaluates to true, it will be included in the result set; otherwise, it will be excluded.

Note that SQL has three-valued logic which are TRUE, FALSE, and UNKNOWN. It means that if a row causes the condition to evaluate to FALSE or NULL, the row will not be returned.

Note that the logical expression that follows the WHERE clause is also known as a predicate. You can use various operators to form the row selection criteria used in the WHERE clause.

| **Operator** | **Meaning** |
| --- | --- |
| = | Equal to |
| <> (!=) | Not equal to |
| < | Less than |
| > | Greater than |
| <= | Less than or equal |
| >= | Greater than or equal |

1. query finds employees who have salaries greater than 14,000 and sorts the results sets based on the salary in descending order.

Answer- SELECT

\*

FROM

employees

WHERE

salary > 14000

ORDER BY

salary DESC

1. query finds all employees who work in the department id 5.

Answer- SELECT

\*

FROM

employees

WHERE

department\_id = 5

1. query finds the employee whose last name is Chen

Answer- SELECT

\*

FROM

employees

WHERE

last\_name = 'Chen'

1. To get all employees who joined the company after January 1st, 1999

Anwer- SELECT

\*

FROM

employees

WHERE

hire\_date > '1999-01-01'

1. to find the employees who joined the company in 1999,

Answer- SELECT

\*

FROM

employees

WHERE

YEAR(hire\_date) = 1999

1. statement finds the employee whose last name is Himuro

Answer- SELECT

\*

FROM

employees

WHERE

last\_name = 'Himuro'

1. the query searches for the string Himuro in the last\_name column of the employees table.

Answer- SELECT

\*

FROM

employees

WHERE

last\_name LIKE '%Himuro%'

1. to find all employees who do not have phone numbers:

Answer- SELECT

\*

FROM

employees

WHERE

phone\_number IS NULL

1. returns all employees whose department id is not 8.

Answer- SELECT

\*

FROM

employees

WHERE

department\_id <> 8

1. finds all employees whose department id is not eight and ten.

Answer- SELECT

\*

FROM

employees

WHERE

department\_id NOT IN (8, 10)

1. to find the employees whose salary is greater than 10,000,

Answer- SELECT

\*

FROM

employees

WHERE

salary > 10000

1. finds employees in department 8 and have the salary greater than 10,000:

Answer- SELECT

\*

FROM

employees

WHERE

department\_id = 8

AND salary > 10000

1. the statement below returns all employees whose salaries are less than 10,000:

Answer- SELECT

\*

FROM

employees

WHERE

salary < 10000

1. finds employees whose salaries are greater than or equal 9,000:

Answer- SELECT

\*

FROM

employees

WHERE

salary >= 9000

1. finds employees whose salaries are less than or equal to 9,000

Answer- SELECT

\*

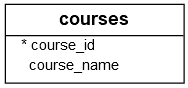
FROM

employees

WHERE

salary <= 9000

**6)WRITE A QUERY FOR:-**



1. adds a new column named credit\_hours to the courses table.

Answer- ALTER TABLE courses

ADD credit\_hours INT

1. adds the fee and max\_limit columns to the courses table and places these columns after the course\_name column.

Answer- ALTER TABLE courses

ADD fee DECIMAL(10, 2) AFTER course\_name,

ADD max\_limit INT AFTER fee

1. changes the attribute of the fee column to NOT NULL.

Answer- ALTER TABLE courses

MODIFY fee DECIMAL(10, 2) NOT NULL

1. to remove the fee column of the courses table

Answer- ALTER TABLE courses

DROP COLUMN fee

1. removes the max\_limit and credit\_hours of the courses table.

Answer- ALTER TABLE courses

DROP COLUMN max\_limit,

DROP COLUMN credit\_hours

**6)WRITE A QUERY FOR:-**

## SQL foreign key constraint

A foreign key is a column or a group of columns that enforces a link between the data in two tables. In a foreign key reference, the primary key column (or columns) of the first table is referenced by the column (or columns) of the second table. The column (or columns) of the second table becomes the foreign key.

You use the FOREIGN KEY constraint to create a foreign key when you create or alter table. Let’s take a simple example to get a better understanding.

## SQL FOREIGN KEY constraint examples

## See the following projects and project\_assignments tables:

CREATE TABLE projects (

project\_id INT PRIMARY KEY,

project\_name VARCHAR(255),

start\_date DATE NOT NULL,

end\_date DATE NOT NULL

);

CREATE TABLE project\_milestones(

milestone\_id INT PRIMARY KEY,

project\_id INT,

milestone\_name VARCHAR(100)

);

Each project may have zero or more milestones while one milestone must belong to one and only one project. The application that uses these tables must ensure that for each row in the project\_milestones table there exists the corresponding row in the projects table. In other words, a milestone cannot exist without a project.

Unfortunately, users may edit the database using client tool or if there is a bug in the application, a row might be added to the project\_milestones table that does not correspond to any row in the projects table. Or user may delete a row in the projects table, leaving orphaned rows in the project\_milestones table. This causes the application not to work properly.

Write a Query

1. to add an SQL FOREIGN KEY constraint to the project\_milestones table to enforce the relationship between the projects and project\_milestones tables.

Answer- ALTER TABLE project\_milestones

ADD CONSTRAINT fk\_project\_milestones\_project\_id

FOREIGN KEY (project\_id) REFERENCES projects(project\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

1. Suppose the project\_milestones already exists without any predefined foreign key and you want to define a FOREIGN KEY constraint for the project\_id column so write a Query to add a FOREIGN KEY constraint to existing table

Answer- ALTER TABLE project\_milestones

ADD CONSTRAINT fk\_project\_milestones\_project\_id

FOREIGN KEY (project\_id) REFERENCES projects(project\_id)

ON DELETE CASCADE

ON UPDATE CASCADE

**TASK 2:🡪**

**Logical Operators and Special Operators**

A logical operator allows you to test for the truth of a condition ,a logical operator returns a value of true, false, or unknown.

The following table illustrates the SQL logical operators:

| **Operator** | **Meaning** |
| --- | --- |
| AND | Return true if both expressions are true |
| NOT | Reverse the result of any other Boolean operator. |
| OR | Return true if either expression is true |

**OTHER SPECIAL OPERATORS:--**

|  |  |
| --- | --- |
| ANY | Return true if any one of the comparisons is true. |
| BETWEEN | Return true if the operand is within a range |
| EXISTS | Return true if a subquery contains any rows |
| IN | Return true if the operand is equal to one of the value in a list |
| LIKE | Return true if the operand matches a pattern |
| ALL | Return true if all comparisons are true |

**1)WRITE A QUERY FOR LOGICAL OPERATORS and OTHER ADVANCED OPERATORS:-**

**Part 1:-**

1. finds all employees whose salaries are greater than 5,000 and less than 7,000:

Answer-SELECT \*

FROM employees

WHERE salary BETWEEN 5000 AND 7000

1. finds employees whose salary is either 7,000 or 8,000:

Answer- SELECT \*

FROM employees

WHERE salary IN (7000, 8000)

1. finds all employees who do not have a phone number:

Answer- SELECT \*

FROM employees

WHERE phone\_number IS NULL

1. finds all employees whose salaries are between 9,000 and 12,000.

Answer- SELECT \*

FROM employees

WHERE salary BETWEEN 9000 AND 12000

1. finds all employees who work in the department id 8 or 9.

Answer- SELECT \*

FROM employees

WHERE department\_id IN (8, 9)

1. finds all employees whose first name starts with the string jo

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE 'jo%'

1. finds all employees with the first names whose the second character is  h

Answer- SELECT \*

FROM employees

WHERE SUBSTRING(first\_name, 2, 1) = 'h'

1. finds all employees whose salaries are greater than all salaries of employees in the department 8:

Answer- SELECT \*

FROM employees

WHERE salary > (SELECT MAX(salary) FROM employees WHERE department\_id = 8)

**Part 2:-**

1. finds all employees whose salaries are greater than the average salary of every department:

Answer- SELECT \*

FROM employees emp

WHERE salary > (SELECT AVG(salary) FROM employees WHERE department\_id = emp.department\_id)

1. finds all employees who have dependents:

Answer- SELECT DISTINCT employees.\*

FROM employees

JOIN dependents d ON employees.employee\_id = dependents.employee\_id

1. to find all employees whose salaries are between 2,500 and 2,900:

Answer- SELECT \*

FROM employees

WHERE salary BETWEEN 2500 AND 2900

1. to find all employees whose salaries are not in the range of 2,500 and 2,900:

Answer- SELECT \*

FROM employees

WHERE salary NOT BETWEEN 2500 AND 2900

1. to find all employees who joined the company between January 1, 1999, and December 31, 2000:

Answer- SELECT \*

FROM employees

WHERE hire\_date BETWEEN '1999-01-01' AND '2000-12-31'

1. to find employees who have not joined the company from January 1, 1989 to December 31, 1999:

Answer- SELECT \*

FROM employees

WHERE hire\_date NOT BETWEEN '1989-01-01' AND '1999-12-31'

1. to find employees who joined the company between 1990 and 1993:

Answer- SELECT \*

FROM employees

WHERE YEAR(hire\_date) BETWEEN 1990 AND 1993

**Part 3:-**

1. to find all employees whose first names start with Da

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE 'Da%'

1. to find all employees whose first names end with er

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE '%er'

1. to find employees whose last names contain the word an:

Answer- SELECT \*

FROM employees

WHERE last\_name LIKE '%an%'

1. retrieves employees whose first names start with Jo and are followed by at most 2 characters:

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE 'Jo\_\_'

1. to find employees whose first names start with any number of characters and are followed by at most one character:

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE '\_%\_'

1. to find all employees whose first names start with the letter S but not start with Sh:

Answer- SELECT \*

FROM employees

WHERE first\_name LIKE 'S%'

AND first\_name NOT LIKE 'Sh%'

**Part 4:-**

1. retrieves all employees who work in the department id 5.

Answer- SELECT \*

FROM employees

WHERE department\_id = 5

1. To get the employees who work in the department id 5 and with a salary not greater than 5000.

Answer- SELECT \*

FROM employees

WHERE department\_id = 5

AND salary <= 5000

1. statement gets all the employees who are not working in the departments 1, 2, or 3.

Answer- SELECT \*

FROM employees

WHERE department\_id NOT IN (1, 2, 3)

1. retrieves all the employees whose first names do not start with the letter D.

Answer- SELECT \*

FROM employees

WHERE first\_name NOT LIKE 'D%'

1. to get employees whose salaries are not between 5,000 and 1,000.

Answer- SELECT \*

FROM employees

WHERE salary NOT BETWEEN 5000 AND 10000

**Part 5:-**



1. Write a query to get the employees who do not have any dependents by above image

Answer- SELECT \*

FROM employees

LEFT JOIN dependents ON employees.Employee\_ID = dependents.Employee\_ID

WHERE dependents.Dependent\_ID IS NULL

1. To find all employees who do not have the phone numbers

Answer- SELECT \*

FROM employees

WHERE Phone\_Number IS NULL

1. To find all employees who have phone numbers

Answer- SELECT \*

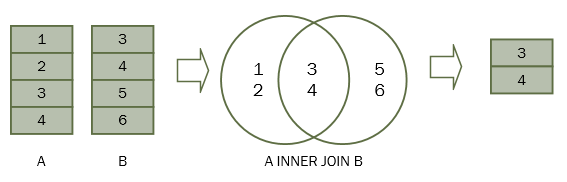
FROM employees

WHERE Phone\_Number IS NOT NULL

**TASK 3:🡪**

**JOINS:-**

SQL INNER JOIN clause



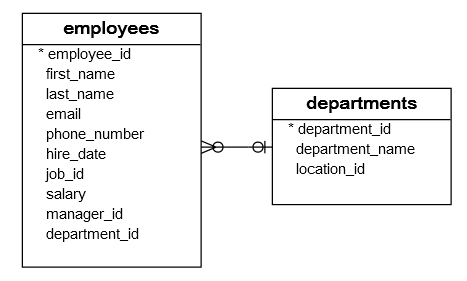
For each row in table A, the inner join clause finds the matching rows in table B. If a row is matched, it is included in the final result set.

Suppose the columns in the A and B tables are a and b. The following statement illustrates the inner join clause:

SELECT a

FROM A

INNER JOIN B ON b = a;



1. Write a Query to
2. To get the information of the department id 1,2, and 3

Answer- SELECT first\_name, last\_name, department\_id

FROM employees

WHERE department\_id IN (1, 2, 3)

ORDER BY department\_id

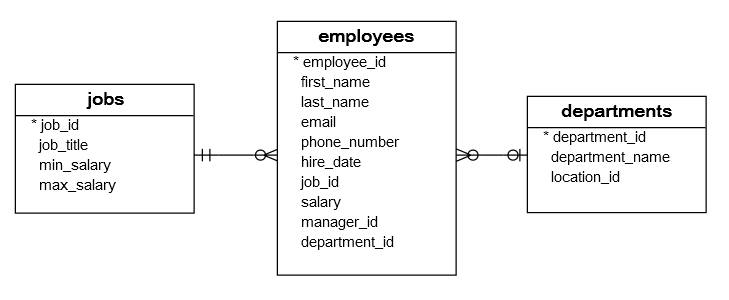
1. To get the information of employees who work in the department id 1, 2 and 3

Answer- SELECT first\_name, last\_name, department\_id

FROM employees

WHERE department\_id IN (1, 2, 3)

ORDER BY department\_id



Write a Query to get the first name, last name, job title, and department name of employees who work in department id 1, 2, and 3.

Answer- SELECT e.first\_name, e.last\_name, j.job\_title, d.department\_name

FROM employees e

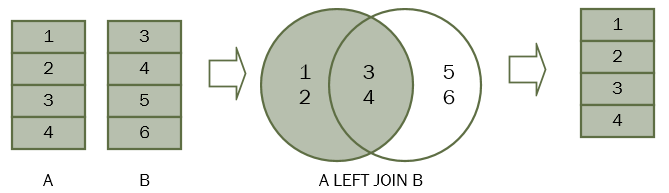
JOIN jobs j ON e.job\_id = j.job\_id

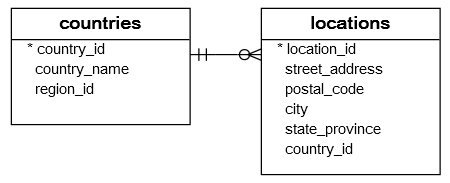
JOIN departments d ON e.department\_id = d.department\_id

WHERE e.department\_id IN (1, 2, 3)

ORDER BY e.department\_id;

## SQL LEFT JOIN clause





Write a Query :--

1. To query the country names of US, UK, and China

Answer- SELECT c.country\_name

FROM countries c

LEFT JOIN locations l ON c.country\_id = l.country\_id

WHERE c.country\_id IN ('US', 'UK', 'CN')

1. query retrieves the locations located in the US, UK and China

Answer- SELECT \* FROM locations

JOIN countries ON locations.country\_id = countries.country\_id

WHERE countries.country\_name IN ('US', 'UK', 'China')

1. To join the countries table with the locations table

Answer- SELECT c.country\_name, c.country\_id, l.country\_id, l.street\_address, l.city

FROM countries c

LEFT JOIN locations l ON l.country\_id = c.country\_id

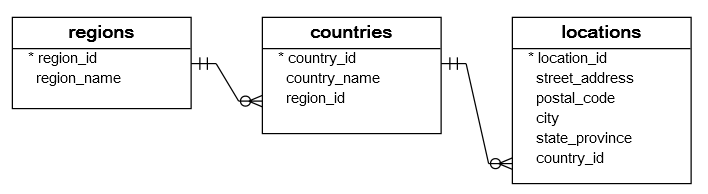
WHERE c.country\_id IN ('US', 'UK', 'CN')

1. to find the country that does not have any locations in the locations table

Answer-SELECT country\_name

FROM countries

WHERE country\_id NOT IN (SELECT country\_id FROM locations)



Write a query to join 3 tables: regions, countries, and locations

Answer- SELECT r.region\_name, c.country\_name, l.street\_address, l.city

FROM regions r

LEFT JOIN countries c ON r.country\_code = c.country\_code

LEFT JOIN locations l ON c.country\_id = l.country\_id

## SQL self-join

SELECT

column1,

column2,

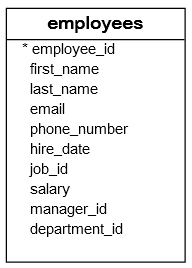
column3,

...

FROM

table1 A

INNER JOIN table1 B ON B.column1 = A.column2;



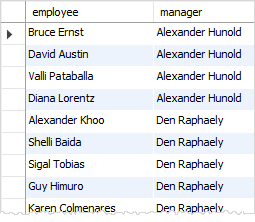
Questions:-

*The manager\_id column specifies the manager of an employee. Write a query statement to joins the employees table to itself to query the information of who reports to whom.*

*Answer-SELECT e1.employee AS Employee, e2.employee AS Manager*

*FROM employees e1*

*INNER JOIN employees e2 ON e1.manager\_id = e2.employee\_id;*

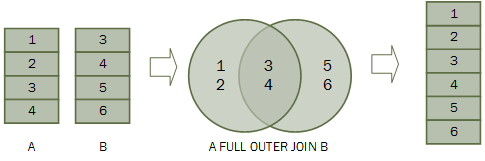


The president does not have any manager. In the employees table, the manager\_id of the row that contains the president is NULL.

Because the inner join clause only includes the rows that have matching rows in the other table, therefore the president did not show up in the result set of the query above.

*Now write a Query To include the president in the result set:-*

## SQL FULL OUTER JOIN clause



Let’s take an example of using the FULL OUTER JOIN clause to see how it works.

First, create two new tables: baskets and fruits for the demonstration. Each basket stores zero or more fruits and each fruit can be stored in zero or one basket.

CREATE TABLE fruits (

fruit\_id INT PRIMARY KEY,

fruit\_name VARCHAR (255) NOT NULL,

basket\_id INTEGER

);

CREATE TABLE baskets (

basket\_id INT PRIMARY KEY,

basket\_name VARCHAR (255) NOT NULL

);

Second, insert some sample data into the baskets and fruits tables.

INSERT INTO baskets (basket\_id, basket\_name)

VALUES

(1, 'A'),

(2, 'B'),

(3, 'C');

INSERT INTO fruits (

fruit\_id,

fruit\_name,

basket\_id

)

VALUES

(1, 'Apple', 1),

(2, 'Orange', 1),

(3, 'Banana', 2),

(4, 'Strawberry', NULL);

Question:-

1. Write a query to returns each fruit that is in a basket and each basket that has a fruit, but also returns each fruit that is not in any basket and each basket that does not have any fruit.

Answer- SELECT fruits.fruit\_name, bAsket.basket\_name

FROM fruits

FULL OUTER JOIN baskets ON fruits.basket\_id = bAsket.basket\_id

1. Write a query to find the empty basket, which does not store any fruit

Answer- SELECT basket.basket\_name

FROM baskets

LEFT JOIN fruits ON basket.basket\_id = fruits .basket\_id

WHERE f.fruit\_id IS NULL

1. Write a query  which fruit is not in any basket

Answer- SELECT f.fruit\_name

FROM fruits f

LEFT JOIN baskets b ON f.basket\_id = b.basket\_id

WHERE b.basket\_id IS NULL

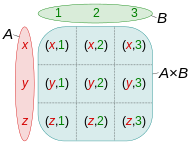
## SQL CROSS JOIN clause

A cross join is a join operation that produces the Cartesian product of two or more tables.

In Math, a Cartesian product is a mathematical operation that returns a product set of multiple sets.

For example, with two sets A {x,y,z} and B {1,2,3}, the Cartesian product of A x B is the set of all ordered pairs (x,1), (x,2), (x,3), (y,1) (y,2), (y,3), (z,1), (z,2), (z,3).

The following picture illustrates the Cartesian product of A and B:



Similarly, in SQL, a Cartesian product of two tables A and B is a result set in which each row in the first table (A) is paired with each row in the second table (B). Suppose the A table has n rows and the B table has m rows, the result of the cross join of the A and B tables have n x m rows.

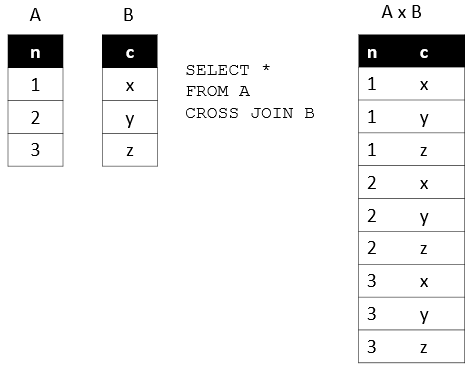
The following illustrates syntax of the CROSS JOIN clause:

SELECT column\_list

FROM A

CROSS JOIN B;

The following picture illustrates the result of the cross join between the table A and table B. In this illustration, the table A has three rows 1, 2 and 3 and the table B also has three rows x, y and z. As the result, the Cartesian product has nine rows:



We will create two new tables for the demonstration of the cross join:

* sales\_organization table stores the sale organizations.
* sales\_channel table stores the sales channels.

The following statements create the sales\_organization and sales\_channel tables:

CREATE TABLE sales\_organization (

sales\_org\_id INT PRIMARY KEY,

sales\_org VARCHAR (255)

);

CREATE TABLE sales\_channel (

channel\_id INT PRIMARY KEY,

channel VARCHAR (255)

);

Suppose the company has two sales organizations that are Domestic and Export, which are in charge of sales in the domestic and international markets.

The following statement inserts two sales organizations into the sales\_organization table:

INSERT INTO sales\_organization (sales\_org\_id, sales\_org)

VALUES

(1, 'Domestic'),

(2, 'Export');

The company can distribute goods via various channels such as wholesale, retail, eCommerce, and TV shopping. The following statement inserts sales channels into the sales\_channel table:

INSERT INTO sales\_channel (channel\_id, channel)

VALUES

(1, 'Wholesale'),

(2, 'Retail'),

(3, 'eCommerce'),

(4, 'TV Shopping');

**Question:--**

Write a Query To find the all possible sales channels that a sales organization

Answer- SELECT sales\_organization .sales\_org, sales\_channel .channel

FROM sales\_organization

CROSS JOIN sales\_channel

**TASK 4:🡪**

## SQL GROUP BY clause

The GROUP BY is an optional clause of the SELECT statement. The GROUP BY clause allows you to group rows based on values of one or more columns. It returns one row for each group.

The following shows the basic syntax of the GROUP BY clause:

SELECT

column1,

column2,

aggregate\_function(column3)

FROM

table\_name

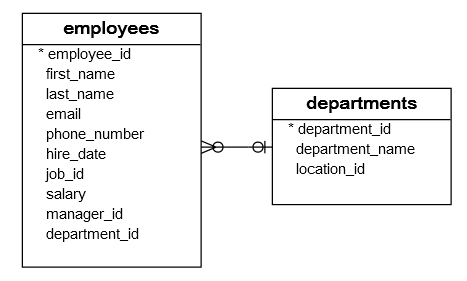
GROUP BY

column1,

column2;

In practice, you often use the GROUP BY clause with an aggregate function such as MIN, MAX, AVG, SUM, or COUNT to calculate a measure that provides the information for each group.

We will use the employees and departments tables  to demonstrate how the GROUP BY clause works.



**Questions:-**

Write a Query

1. to group the values in department\_id column of the employees table:

Answer- SELECT

column1,

column2,

aggregate\_function(column3)

FROM

table\_name

GROUP BY

column1,

column2

1. to count the number of employees by department:

Answer- SELECT department\_id, COUNT(\*) AS total\_employees

FROM employees

GROUP BY department\_id

1. returns the number of employees by department

Answer- SELECT Department.Department\_ID, Department.Department\_Name, COUNT(Employees.Employee\_ID) AS Num\_Employees

FROM Department

LEFT JOIN Employees ON Department.Department\_ID = Employees.Department\_ID

GROUP BY Department.Department\_ID, Department.Department\_Name

1. to sort the departments by headcount:

Answer- SELECT Department.Department\_ID, Department.Department\_Name, COUNT(\*) AS Num\_Employees

FROM Employees

INNER JOIN Department ON Employees.Department\_ID = Department.Department\_ID

GROUP BY Department.Department\_ID, Department.Department\_Name

ORDER BY Num\_Employees DESC, Department.Department\_Name

1. to find departments with headcounts are greater than 5:

Answer- SELECT Department.Department\_ID, Department.Department\_Name, COUNT(Employees.Employee\_ID) AS Num\_Employees

FROM Department

LEFT JOIN Employees ON Department.Department\_ID = Employees.Department\_ID

GROUP BY Department.Department\_ID, Department.Department\_Name

HAVING COUNT(Employees.Employee\_ID) > 5

1. returns the minimum, maximum and average salary of employees in each department.

Answer- SELECT

Department.Department\_ID,

Department.Department\_Name,

MIN(Employees.Salary) AS Min\_Salary,

MAX(Employees.Salary) AS Max\_Salary,

AVG(Employees.Salary) AS Avg\_Salary

FROM

Department

INNER JOIN

Employees ON Department.Department\_ID = Employees.Department\_ID

GROUP BY

Department.Department\_ID, Department.Department\_Name

1. To get the total salary per department,

Answer- SELECT

Department.Department\_ID,

Department.Department\_Name,

SUM(Employees.Salary) AS Total\_Salary

FROM

Department

INNER JOIN

Employees ON Department.Department\_ID = Employees.Department\_ID

GROUP BY

Department.Department\_ID, Department.Department\_Name

1. groups rows with the same values both department\_id and job\_id columns in the same group then return the rows for each of these groups

Answer- SELECT

Department\_ID,

Job\_ID,

COUNT(\*) AS Num\_Employees

FROM

Employees

GROUP BY

Department\_ID, Job\_ID

## SQL HAVING clause

To specify a condition for groups, you use the HAVING clause.

The HAVING clause is often used with the GROUP BY clause in the SELECT statement. If you use a HAVING clause without a GROUP BY clause, the HAVING clause behaves like the WHERE clause

The following illustrates the syntax of the HAVING clause:

SELECT

column1,

column2,

AGGREGATE\_FUNCTION (column3)

FROM

table1

GROUP BY

column1,

column2

HAVING

group\_condition;



Questions:-

Write a Query

1. To get the managers and their direct reports, and to group employees by the managers and to count the direct reports.

Answer- SELECT

Manager\_ID,

COUNT(\*) AS Num\_Direct\_Reports

FROM

Employees

GROUP BY

Manager\_ID

1. To find the managers who have at least five direct reports

Answer- SELECT

Manager\_ID,

COUNT(\*) AS Num\_Direct\_Reports

FROM

Employees

GROUP BY

Manager\_ID

HAVING

COUNT(\*) >= 5

1. calculates the sum of salary that the company pays for each department and selects only the departments with the sum of salary between 20000 and 30000.

Answer- SELECT

Department\_ID,

SUM(Salary) AS Total\_Salary

FROM

Employees

GROUP BY

Department\_ID

HAVING

SUM(Salary) BETWEEN 20000 AND 30000

1. To find the department that has employees with the lowest salary greater than 10000

Answer- SELECT

Department\_ID,

MIN(Salary) AS Lowest\_Salary

FROM

Employees

WHERE

Salary > 10000

GROUP BY

Department\_ID

ORDER BY

Lowest\_Salary

1. To find the departments that have the average salaries of employees between 5000 and 7000

Answer- SELECT

Department\_ID,

AVG(Salary) AS Avg\_Salary

FROM

Employees

GROUP BY

Department\_ID

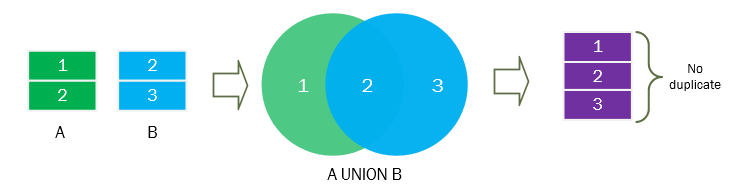
HAVING

AVG(Salary) BETWEEN 5000 AND 7000

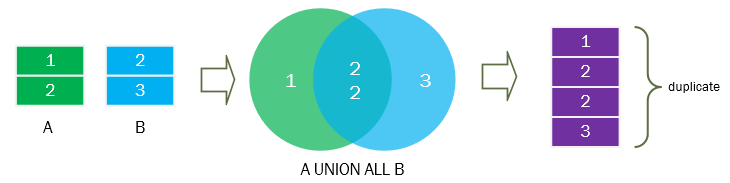
**TASK 5 (Other Queries)**

## 1)SQL UNION operator

Suppose, we have two result sets A(1,2) and B(2,3). The following picture illustrates A UNION B:



And the following picture illustrates A UNION ALL B





*Quetsion:-*

*Write a Query to combine the first name and last name of employees and dependents*

*Answer-* *SELECT First\_Name, Last\_Name FROM Employees*

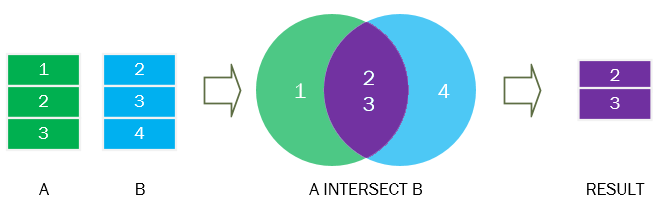
*UNION*

*SELECT First\_Name, Last\_Name FROM Dependents*

## 2)SQL INTERSECT operator

Suppose, we have two tables: A(1,2) and B(2,3).

The following picture illustrates the intersection of A & B tables.



**Question :-**

*Write a Query to  Applies the INTERSECT operator to the A and B tables and sorts the combined result set by the id column in descending order.*

*Answer-SELECT ID FROM A*

*INTERSECT*

*SELECT ID FROM B*

*ORDER BY ID DESC*

## 3)SQL EXISTS operator

We will use the  employees and dependents tables in the sample database for the demonstration.



**Write a Query**

1. finds all employees who have at least one dependent.

Answer-SELECT \*

FROM Employees

WHERE EXISTS (

SELECT 1

FROM Dependents

WHERE Dependents.Employee\_ID = Employees.Employee\_ID

)

B . finds employees who do not have any dependents:

Answer- SELECT \*

FROM Employees

WHERE NOT EXISTS (

SELECT 1

FROM Dependents

WHERE Dependents.Employee\_ID = Employees.Employee\_ID

)

## 4) SQL CASE expression

**CASE expression**

**WHEN when\_expression\_1 THEN**

**result\_1**

**WHEN when\_expression\_2 THEN**

**result\_2**

**WHEN when\_expression\_3 THEN**

**result\_3**

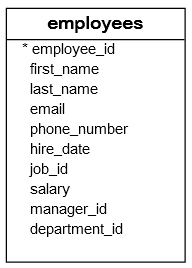
**...**

**ELSE**

**else\_result**

**END**

Let’s take a look at the employees table.



Questions:-

1. Suppose the current year is 2000. How to use the simple CASE expression to get the work anniversaries of employees by

Answer-SELECT

Employee\_ID,

First\_Name,

Last\_Name,

Hire\_Date,

CASE

WHEN YEAR(Hire\_Date) = 2000 THEN 'New Employee'

WHEN YEAR(Hire\_Date) = 1999 THEN '1 Year Anniversary'

WHEN YEAR(Hire\_Date) = 1998 THEN '2 Year Anniversary'

WHEN YEAR(Hire\_Date) = 1997 THEN '3 Year Anniversary'

ELSE 'More than 3 Years'

END AS Work\_Anniversary

FROM

Employees

1. Write a Query If the salary is less than 3000, the CASE expression returns “Low”. If the salary is between 3000 and 5000, it returns “average”. When the salary is greater than 5000, the CASE expression returns “High”.

Answer- SELECT

Employee\_ID,

First\_Name,

Last\_Name,

Salary,

CASE

WHEN Salary < 3000 THEN 'Low'

WHEN Salary BETWEEN 3000 AND 5000 THEN 'Average'

ELSE 'High'

END AS Salary\_Category

FROM

Employees

## 5) SQL UPDATE statement



Suppose the employee id 192 Sarah Bell changed her last name from Bell to Lopez and you need to update her record in the  employees table.

SQL UPDATE example

Write a Query to update Sarah’s last name from  Bell to Lopez

Answer-UPDATE Employees

SET Last\_Name = 'Lopez'

WHERE Employee\_ID = 192 AND First\_Name = 'Sarah'

How to make sure that the last names of children are always matched with the last name of parents in the  employees table,

1. Answer- **Foreign Key Constraint**: Create a foreign key constraint on the Dependents table that references the Employees table. This constraint will ensure that each dependent must have a corresponding parent (employee).
2. **Trigger**: Create a trigger on the Dependents table that checks whether the last name of the dependent matches the last name of the parent (employee) when a new record is inserted or when the last name is updated. If the last names do not match, the trigger can raise an error or take appropriate action to ensure consistency.

-- Create a foreign key constraint on Dependents table referencing Employees table

ALTER TABLE Dependents

ADD CONSTRAINT fk\_dependent\_employee

FOREIGN KEY (Employee\_ID)

REFERENCES Employees(Employee\_ID)

ON DELETE CASCADE; -- Optional: Define action on deletion

-- Create a trigger to ensure last names match

CREATE OR REPLACE TRIGGER trg\_dependent\_last\_name\_check

BEFORE INSERT OR UPDATE OF Last\_Name ON Dependents

FOR EACH ROW

DECLARE

v\_parent\_last\_name VARCHAR2(100);

BEGIN

-- Retrieve the last name of the parent (employee) based on the Employee\_ID

SELECT Last\_Name INTO v\_parent\_last\_name

FROM Employees

WHERE Employee\_ID = :NEW.Employee\_ID;

-- Check if the last name of the dependent matches the last name of the parent

IF :NEW.Last\_Name != v\_parent\_last\_name THEN

-- Raise an error or take appropriate action

RAISE\_APPLICATION\_ERROR(-20001, 'Dependent last name must match parent last name.');

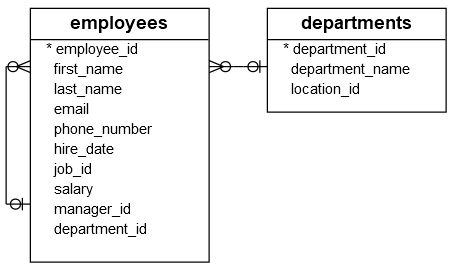
END IF;

END

**FINAL TASK (Advanced Queries)**

**SQL SUBQUERY**

Consider the following employees and departments tables from the sample database



Suppose you have to find all employees who locate in the location with the id 1700. You might come up with the following solution.

First, find all departments located at the location whose id is 1700:

SELECT

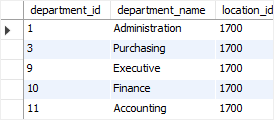
\*

FROM

departments

WHERE

location\_id = 1700;



Second, find all employees that belong to the location 1700 by using the department id list of the previous query:

SELECT

employee\_id, first\_name, last\_name

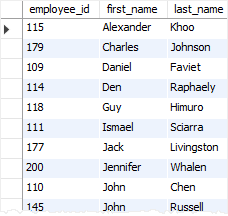
FROM

employees

WHERE

department\_id IN (1 , 3, 8, 10, 11)

ORDER BY first\_name , last\_name;



This solution has two problems. To start with, you have looked at the departments table to check which department belongs to the location 1700. However, the original question was not referring to any specific departments; it referred to the location 1700.

Because of the small data volume, you can get a list of department easily

.

However, in the real system with high volume data, it might be problematic .Another problem was that you have to revise the queries whenever you want to find employees who locate in a different location

A much better solution to this problem is to use a subquery. By definition, a subquery is a query nested inside another query such as SELECT, INSERT, UPDATE, or DELETE statement. In this tutorial, we are focusing on the subquery used with the SELECT statement.

Question:-

Write a Query :-

1. *Combine Above two queries using subquery inorder find all departments located at the location whose id is 1700 and find all employees that belong to the location 1700 by using the department id list of the previous query*

*Answer-* *SELECT*

*employee\_id, first\_name, last\_name*

*FROM*

*employees*

*WHERE*

*department\_id IN (SELECT department\_id FROM departments WHERE location\_id = 1700)*

*ORDER BY first\_name, last\_name*

1. to find all employees who do not locate at the location 1700:

SELECT

employee\_id, first\_name, last\_name

FROM

employees

WHERE

department\_id NOT IN (SELECT department\_id FROM departments WHERE location\_id = 1700)

ORDER BY first\_name, last\_name

1. finds the employees who have the highest salary:

Answer- SELECT

employee\_id, first\_name, last\_name, salary

FROM

employees

WHERE

salary = (SELECT MAX(salary) FROM employees)

1. finds all employees who salaries are greater than the average salary of all employees:

Answer- SELECT

employee\_id, first\_name, last\_name, salary

FROM

employees

WHERE

salary > (SELECT AVG(salary) FROM employees)

1. finds all departments which have at least one employee with the salary is greater than 10,000:

Answer- SELECT

department\_id, department\_name

FROM

departments

WHERE

department\_id IN (SELECT DISTINCT department\_id FROM employees WHERE salary > 10000)

1. finds all departments that do not have any employee with the salary greater than 10,000:

Answer- SELECT

department\_id, department\_name

FROM

departments

WHERE

department\_id NOT IN (SELECT DISTINCT department\_id FROM employees WHERE salary > 10000)

1. to find the lowest salary by department:

Answer- SELECT

department\_id,

MIN(salary) AS lowest\_salary

FROM

employees

GROUP BY

department\_id

1. finds all employees whose salaries are greater than the lowest salary of every department:

Answer- SELECT

employee\_id, first\_name, last\_name, salary

FROM

employees

WHERE

salary > ALL (SELECT MIN(salary) FROM employees GROUP BY department\_id)

1. finds all employees whose salaries are greater than or equal to the highest salary of every department

Answer- SELECT

employee\_id, first\_name, last\_name, salary

FROM

employees

WHERE

salary >= ALL (SELECT MAX(salary) FROM employees GROUP BY department\_id)

1. returns the average salary of every department

Answer- SELECT

department\_id,

AVG(salary) AS average\_salary

FROM

employees

GROUP BY

department\_id

1. to calculate the average of average salary of departments :

Answer- SELECT

AVG(average\_salary) AS average\_of\_average\_salary

FROM (

SELECT

AVG(salary) AS average\_salary

FROM

employees

GROUP BY

department\_id

) AS department\_avg

1. finds the salaries of all employees, their average salary, and the difference between the salary of each employee and the average salary.

Answer- SELECT

e.employee\_id,

e.first\_name,

e.last\_name,

e.salary,

AVG(e.salary) OVER() AS average\_salary,

e.salary - AVG(e.salary) OVER() AS salary\_difference

FROM

employees e

**THANK YOU**