CareerScale IT Consulting LLP

http://careerscale.in info@careerscale.in +91 040 42100276

Contact us for classroom/corporate/customized/online trainings.

SQL training handout ver. 1.2

mImportant:

1. **This class is not a replacement for regular database course taught in a semester or a year. This is only a quick refresher.**
2. **Follow the instructor through out the session. This handout is just a supplementary for the training careerscale is organizing.**
3. **Optional parameters, conditions are mentioned in [ ] in the queries or the text in this handout.**
4. **If you do not understand something, please raise questions with the trainer and get the clarifications.**
5. **We try our best to provide correct information. If you find any mistakes, please notify us. We will rectify in subsequent releases**

Database: A *database* is a collection of data that is organized so that its contents can easily be accessed, managed, and updated, usually in digital form.
A **Database Management System** (**DBMS**) is a software package with computer programs that control the creation, maintenance, and the use of a database.

RDBMS : Relational DataBase Managerment System is a DBMS in which data is stored in tables and the relationships among the data are also stored in tables. An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table. E. F. Codd introduced the relational database model. Many modern DBMS do not conform to the Codd's definition of a RDBMS, but nonetheless they are still considered to be RDBMS.
The most popular RDBMS are MS SQL Server, DB2, Oracle and MySQL.

SQL: SQL (pronounced "ess-que-el") stands for Structured Query Language. It is the standard language for accessing and manipulating databases. **SQL:2008** is the sixth revision of the ISO and ANSI standard for the SQL database query language. It was formally adopted in July 2008. Though all the databases implement the SQL, they also provided additional custom functionality.

Table basics:

Table is the database structure holding the data in rows and columns. columns represent the "attributes" and rows represent "records".

Student Id	Name	Class
1001	Vivek	9th std
1002	Raghu	10th std

Unique key: A unique key can uniquely identify each row in a table. A unique key comprises a single column or a set of columns. No two distinct rows in a table can have the same value (or combination of values) in those columns if NULL values are not used. Depending on its design, a table may have arbitrarily many unique keys but at most one primary key.

Primary key: A primary key is a field that uniquely identifies each record in a table. As it uniquely identifies each entity, it cannot contain null value and duplicate value. A table will have at most one primary key.

Candidate key: A candidate key is a combination of attributes that can be uniquely used to identify a database record. Each table may have one or more candidate keys. One of these candidate keys is selected as the table primary key.

Super Key: If we add additional attributes to a primary key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called super key. A primary key is therefore a minimum super key.

Foreign Key: A foreign key is a field in a relational table that matches primary key of another table. The foreign key can be used to cross-reference tables.

Query types: Queries can be categorized into two**. 1. DDL – Data Definition Language 2. DML – Data Manipulation Language**.
All DDL queries are auto commit statements. Commit is implicit with DDL statements. For DML we need to execute commit explicitly.

Data Definition	Data Manipulation
CREATE TABLE Adds a new table to the database DROP TABLE Removes a table from the database ALTER TABLE Changes the structure of an existing table CREATE VIEW Adds a new view to the database DROP VIEW Removes a view from the database	SELECT Retrieves data from the database INSERT Adds new rows of data to the database DELETE Removes rows of data from the database UPDATE Modifies existing database data

CREATE SCHEMA Adds a new schema to the database DROP SCHEMA Removes a schema from the database TRUNCATE TABLE deletes all rows in the table.	

ANSI/ISO SQL Data Types

Data Type	Description	Data type	Description
CHAR(len)	Fixed-length character strings	INTEGER INT	Integer numbers
VARCHAR(len)	Variable-length character strings*	SMALLINT	Small integer numbers
CHARACTER(len)		BIT(len)	Fixed-length bit string*
CHAR VARYING(len)		BIT VARYING(len) NUMERIC(precision,scale) DECIMAL(precision,scale2q) DEC(precision,scale) FLOAT(precision) Floating point numbers REAL Low-precision floating point numbers DOUBLE PRECISION High-precision floating numbers DATE Calendar date* TIME(precision) Clock time* TIMESTAMP(precision) Date and time* INTERVAL Time interval*	Variable-length bit string*
CHARACTER VARYING(len)			
NCHAR(len) NATIONAL CHAR(len) NATIONAL CHARACTER(len)	Fixed-length national character strings*		
NCHAR VARYING(len) NATIONAL CHAR VARYING(len) NATIONAL CHARACTER VARYING(len)	Variable-length national character strings*		

* added with latest SQL standards, May not be available with older version of databases.

Data types for Oracle 8 - Oracle 10g + PL/SQL

Datatype	**Description**	**Max Size: Oracle 8**	**Max Size: Oracle 9i/10g**	**Max Size: PL/SQL**
VARCHAR2(size)	Variable length character string having maximum length *size* bytes. You must specify size	**4000** bytes minimum is 1	**4000** bytes minimum is 1	32767 bytes minimum is 1
NVARCHAR2(size)	Variable length national character set string having maximum length *size* bytes. You must specify size	4000 bytes minimum is 1	4000 bytes minimum is 1	32767 bytes minimum is 1

SQL training handout ver. 1.2

CHAR(size)	Fixed length character data of length size bytes. This should be used for fixed length data. Such as codes A100, B102...	**2000** bytes Default and minimum size is 1 byte.	**2000** bytes Default and minimum size is 1 byte.	32767 bytes Default and minimum size is 1 byte.
NUMBER(p,s)	Number having precision p and scale s.	The precision p can range from 1 to 38. The scale s can range from -84 to 127.	The precision p can range from 1 to 38. The scale s can range from -84 to 127.	Magnitude 1E-130 .. 10E125 maximum precision of 126 binary digits, which is roughly equivalent to 38 decimal digits The scale s can range from -84 to 127.
LONG	Character data of variable length (A bigger version the VARCHAR2 datatype)	2 Gigabytes	2 Gigabytes - but now deprecated (provided for backward compatibility only).	32760 bytes Note this is smalller than the maximum width of a LONG column
DATE	Valid date range	from January 1, 4712 BC to December 31, **9999** AD.	from January 1, 4712 BC to December 31, **9999** AD.	from January 1, 4712 BC to December 31, **9999** AD. (in Oracle7 = 4712 AD)
TIMESTAMP (fractional_seconds_precision)	the number of digits in the fractional part of the SECOND datetime field.	-	Accepted values of fractional_seconds_precision are 0 to 9. (default = 6)	
TIMESTAMP (fractional_seconds_precision) WITH {LOCAL} TIMEZONE	As above with time zone displacement value	-	Accepted values of fractional_seconds_precision are 0 to 9. (default = 6)	
RAW(size)	Raw binary data of length size bytes. You must specify size for a RAW value.	Maximum size is **2000** bytes	Maximum size is **2000** bytes	32767 bytes
LONG RAW	Raw binary data of variable length. (not intrepreted by PL/SQL)	2 Gigabytes.	2 Gigabytes - but now deprecated (provided for backward compatibility only)	32760 bytes Note this is smalller than the maximum width of a LONG RAW column
ROWID	Hexadecimal string representing the unique address of a row in its table. (primarily for values returned by the ROWID pseudocolumn.)	10 bytes	10 bytes	Hexadecimal string representing the unique address of a row in its table. (primarily for values returned by the ROWID pseudocolumn.)
CLOB	Character Large Object	4Gigabytes	8 TB	

BLOB	Binary Large Object	4Gigabytes	8 TB	
BFILE	pointer to binary file on disk	4Gigabytes	8 TB	
XMLType	XML data	-	4 Gigabytes	Populate with XML from a CLOB or VARCHAR2. or query from another XMLType column.

CREATE table

The **create table** statement is used to create a new table. Here is the format of a simple **create table.**
create table "tablename" ("column1" "data type" [constraint], "column2" "data type" [constraint], "column3" "data type" [constraint]);
CREATE TABLE employee (id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10),location char(10));

CREATE view:

a view is a virtual table based on the result-set of an SQL statement.
CREATE VIEW “view_name” AS SELECT “column_name(s)” FROM “table_name” WHERE “condition”

Update view:

CREATE OR REPLACE VIEW “view_name “ AS SELECT “column_name(s)” FROM “table_name” WHERE “condition”

CREATE sequence

CREATE SEQUENCE “sequence_name” MINVALUE “number_value” MAXVALUE “number_value” START WITH “number_value” INCREMENT BY “number_value” CACHE “number_value”;
E.g. CREATE SEQUENCE employee_seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 20;

If you omit the **MAXVALUE option**, your sequence will automatically default to: MAXVALUE 999999999999999999999999999

DROP table:

The **drop table** command is used to delete a table and all rows in the table.
DROP TABLE "tablename".
DROP VIEW view_name

SELECT Query.

The **select** statement is used to query the database and retrieve selected data that match the criteria that you specify. Here is the format of a simple select statement:
SELECT “column1" [,"column2",etc] from "tablename" [WHERE "condition"];
The column names that follow the select keyword determine which columns will be returned in the results. You can select as many column names that you'd like, or you can use a "*" to select all columns.

The table name that follows the keyword **from** specifies the table that will be queried to retrieve the desired results.

The **where** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria described after the keyword **where**.

Conditional selections used in the **where** clause:

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

SQL training handout ver. 1.2

LIKE

The **LIKE** pattern matching operator can also be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "%" can be used as a wild card to match any possible character that might appear before or after the characters specified.
e.g. **:** select first, last, city from empinfo where first LIKE 'Er%'; This SQL statement will match any first names that start with 'Er'. **Strings must be in single quotes.**

SQL constraints:

NOT NULL	FOREIGN KEY
UNIQUE	CHECK
PRIMARY KEY	DEFAULT

E.g:

1. CREATE TABLE address (id INTEGER, address VARCHAR (255) **NOT NULL**, **PRIMARY KEY** (id));

2. CREATE TABLE person
(id INTEGER **PRIMARY KEY** , first_name VARCHAR(255) NOT NULL, last_name VARCHAR(255),
passport_number VARCHAR(25) **UNIQUE** , address_id INTEGER **NOT NULL**, City varchar(255) **DEFAULT** 'Hyderabad',
CONSTRAINT chk_id **CHECK** (id>0),
CONSTRAINT fk_person_address_id **FOREIGN KEY** (address_id) REFERENCES address(id)) ;

UPDATE query

UPDATE <table_name> SET <column_name> = <value>, [<column_name> = <value>] WHERE <condition>

E.g. UPDATE employee SET manager_id = 1 WHERE department_id = 2;

DELETE query

DELETE FROM <table_name> WHERE <condition>;

E.g. DELETE FROM employee WHERE department_id=1;

Normalization

Normalization is the process of splitting large table into smaller set tables to ensure data integrity and eliminating data redundancy. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations.

- Data integrity - all of the data in the database are consistent, and satisfy all integrity constraints.
- Data redundancy – if data in the database can be found in two different locations (direct redundancy) or if data can be calculated from other data items (indirect redundancy) then the data is said to contain redundancy.

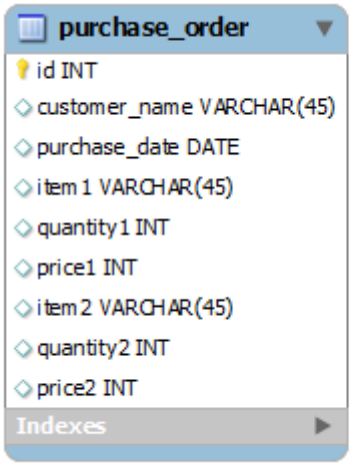
We try will try to understand various normalization methods using an example.
Assumption: A customer can have multiple orders and an order can include multiple products.

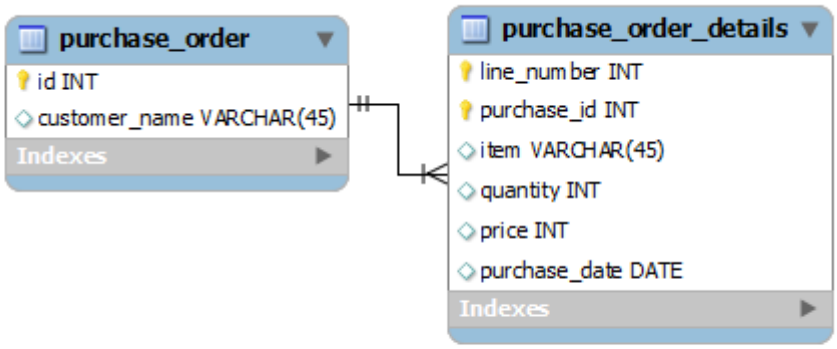
1st Normal Form (1NF)

There are no duplicated rows in the table.
Each cell is single-valued (i.e., there are no repeating groups or arrays).
Entries in a column (attribute, field) are of the same kind.

Remove multi valued attributes

SQL training handout ver. 1.2

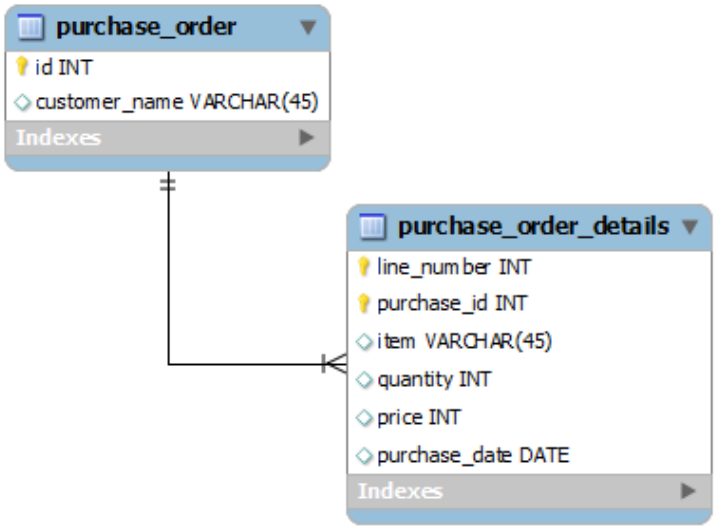


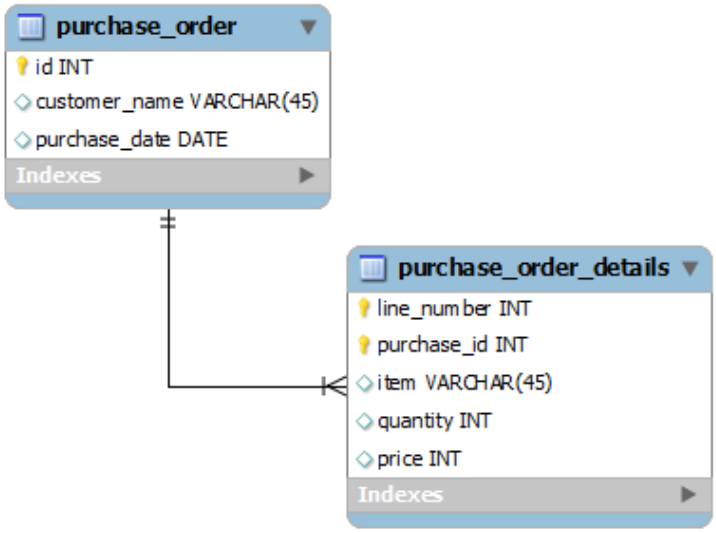


2nd Normal Form (2NF)

A table is in 2NF if it is in 1NF and if all non-key attributes are dependent on all of the key.

Remove partial dependencies

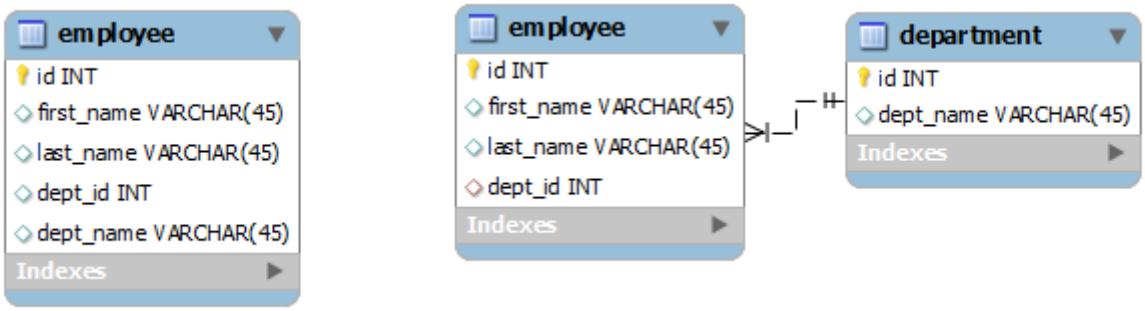




3rd Normal Form (3NF)

A table is in 3NF if it is in 2NF and if it has no transitive dependencies.

Remove transitive dependencies



Transaction:

A transaction is an atomic unit of work that must be completed in its entirety. The transaction succeeds if it committed and fails if it is aborted.

ACID properties:

Atomicity: All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are. For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency: Data is in a consistent state when a transaction starts and when it ends. For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation: The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability: After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

CareerScale IT Consulting LLP

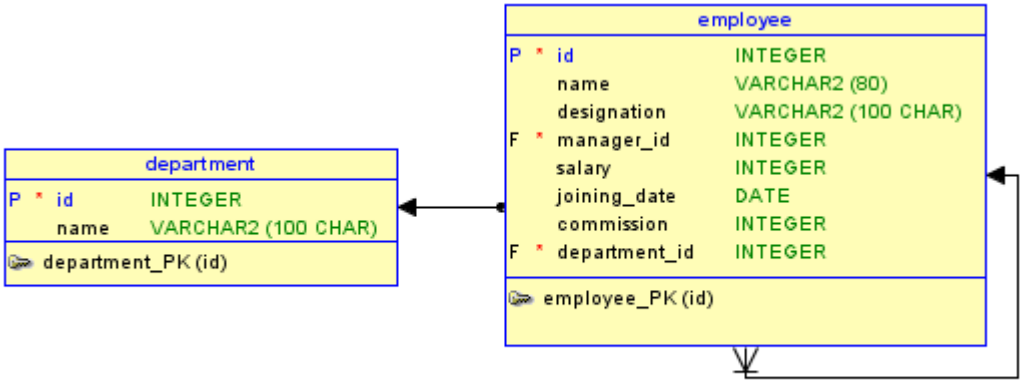
http://careerscale.in info@careerscale.in +91 040 42100276

Contact us for classroom/corporate/customized/online trainings.

SQL training handout ver. 1.2

Important queries:

All the queries below are based on 2 simple tables "department" and "employee".



SCHEMA definition:

-- Use the drop statements only if you have created the objects in database already, otherwise jump to create statements --directly

-- DROP TABLE department;

-- DROP TABLE employee;

-- DROP SEQUENCE department_id_SEQ;

-- DROP SEQUENCE employee_id_SEQ;

1. CREATE TABLE department(id INTEGER NOT NULL,name VARCHAR2(100)) ;

2. ALTER TABLE department ADD CONSTRAINT department_PK PRIMARY KEY (id) ;

3. CREATE TABLE employee(id INTEGER NOT NULL,name VARCHAR2(80),designation VARCHAR2(100),manager_id INTEGER,salary INTEGER,joining_date DATE,commission INTEGER,department_id INTEGER);

4. ALTER TABLE employee ADD CONSTRAINT employee_PK PRIMARY KEY (id) ;

5. ALTER TABLE employee ADD CONSTRAINT employee_department_FK FOREIGN KEY (department_id) REFERENCES department(id);

6. ALTER TABLE employee ADD CONSTRAINT employee_manager_FK FOREIGN KEY (manager_id) REFERENCES employee(id);

7. CREATE SEQUENCE department_id_SEQ START WITH 1 INCREMENT BY 1 ;

8. CREATE OR REPLACE TRIGGER department_id_TRG

BEFORE INSERT ON department

```
FOR EACH ROW
WHEN (NEW.id IS NULL)
BEGIN
   SELECT department_id_SEQ.NEXTVAL INTO :NEW.id FROM DUAL;
END;
/
```

9. CREATE SEQUENCE employee_id_SEQ START WITH 1 INCREMENT BY 1;

```
10 .CREATE OR REPLACE TRIGGER employee_id_TRG
BEFORE INSERT ON employee
FOR EACH ROW
WHEN (NEW.id IS NULL)
BEGIN
   SELECT employee_id_SEQ.NEXTVAL INTO :NEW.id FROM DUAL;
END;
/
```

11. INSERT INTO department(name) VALUES('Human Resources');
12. INSERT INTO department(name) VALUES('Software Development');
13. INSERT INTO department(name) VALUES('QA');
14. INSERT INTO department(name) VALUES('Marketing');

15. INSERT INTO employee(name, designation,department_id, manager_id,salary, joining_date, commission)
    VALUES('Vijay','Managing Director',2, NULL, 100000,to_date('2003/05/03', 'yyyy/mm/dd'), 20000);
16. INSERT INTO employee(name, designation,department_id, manager_id,salary, joining_date, commission)
    VALUES('Srikanth','Manager',2, 1, 50000,to_date('2003/05/03', 'yyyy/mm/dd'), 200);
17. INSERT INTO employee(name, designation,department_id, manager_id,salary, joining_date, commission)
    VALUES('Gopalan','Developer',2, 2, 35000,to_date('2003/05/03', 'yyyy/mm/dd'), 200);
18. INSERT INTO employee(name, designation,department_id, manager_id,salary, joining_date, commission)
    VALUES('Sree Vidya','Receptionist',NULL , 2, 35000,to_date('2010/05/02', 'yyyy/mm/dd'), 200);

```
-- Common queries without any table
```

19. SELECT 10+2 FROM dual;

20. SELECT SYSDATE FROM dual;

21. SELECT To_Char(SYSDATE,'hh:MM:ss') FROM dual;

-- Display your age in days

22. SELECT to_date(sysdate) - To_Date('01-sep-85') AS age_in_days FROM dual;

-- Display your age in months

23. SELECT Months_Between(SYSDATE,'01-sep-85') AS age_in_months FROM dual;

24. SELECT Length('CareerScale') FROM dual;

--String operations

25. SELECT Upper(name) FROM employee;

SELECT Lower(name) FROM employee;

26. SELECT InitCap(name) FROM employee;

--Get only the first word from CareerScale

27. SELECT SubStr('CareerScale',1,6) FROM dual;

-- Get all the employees in the company

28. SELECT * FROM employee;

-- Get all the employees in the department 'Software Development'

29. SELECT e.id,e.name,e.designation FROM employee e, department d WHERE e.department_id = d.id AND d.name= 'Software Development'

-- Get all the employee names in ascending order

30. SELECT * FROM employee ORDER BY name ASC;

-- TOP N query , only N th ranker

-- Solution 1. Note: Replace 0 with N-1, N being actual rank needed

31a) SELECT * FROM employee a WHERE 0 = (SELECT Count(distinct(salary)) FROM employee b WHERE b.salary >a.salary);

-- Solution 2. Note: replace 1 with actual rank neeed

31b) SELECT id, name, salary, salary_rank FROM (SELECT id, name,salary, RANK() OVER (ORDER BY salary Desc NULLS LAST) AS salary_rank FROM employee) WHERE salary_rank = 1;

--Top N results Note replace 3 with required N value.

32. SELECT * FROM employee WHERE ROWNUM < 3 ORDER BY salary DESC;

-- Max salary in each in each department

33. SELECT department_id,Max(salary) AS max_salary FROM employee GROUP BY department_id;

-- Employees with maximum salary in each department

34. SELECT e.name, e.salary, e.department_id FROM employee e, (SELECT department_id,Max(salary) AS max_salary FROM employee GROU
department_id) max_salary WHERE e.department_id = max_salary.department_id AND e.salary = max_salary.max_salary;

-- Show all employees whose salary is equals or greater than that of Mr. Srikanth

35. SELECT e.name, e.salary FROM employee e WHERE e.salary >= (SELECT salary FROM employee WHERE name='Srikanth');

-- Show employee and his manager name

36. SELECT e.name, m.name FROM employee e, employee m WHERE e.manager_id = m.id;

-- show employee names who are not managing anyone else

37. SELECT id, name FROM employee WHERE id NOT IN (SELECT manager_id FROM
employee WHERE manager_id IS NOT NULL);

-- show only manager names

38. SELECT DISTINCT(m.name) FROM employee e, employee m WHERE m.id = e.manager_id;

show names OF employees who are NOT managers

39. SELECT name FROM employee WHERE name NOT IN (SELECT DISTINCT(m.name) FROM employee e, employee m WHERE m.id = e.manager_id);

-- select employees who ids are not in any of the 1, 2, 4

40. SELECT * FROM employee WHERE id NOT IN (1,2,4);

-- Get total salaries of employees in the company

41. SELECT Sum(salary) FROM employee;

-- Get Average salary in the company

42. SELECT Avg(salary) FROM employee;

-- Get department wise total salaries

43. SELECT department_id,Sum(salary) FROM employee GROUP BY department_id;

--Get average salary per department

44. SELECT department_id,Avg(salary) FROM employee GROUP BY department_id;

-- Get count of employees in each department

45. SELECT department_id, Count(id) FROM employee GROUP BY department_id;

-- select employees whose employee id is between 1 and 3

Joins

The JOIN keyword is used in an SQL statement to query data from two or more tables, based on a relationship between certain columns in these tables.

Tables in a database are often related to each other with keys. Joins are explained with employee and department tables that are given above.

Inner Join

-- inner join results only the matching rows from both the tables. Also called as equi join

46. SELECT e.id, e.name AS employee_name, d.name AS department_name
FROM **employee e JOIN department d ON e.department_id = d.id;**

Outer Joins

Left outer join:

-- left outer join, let us get all the employees (including employees having no departments)

47. SELECT e.id, e.name AS employee_name, d.name AS department_name

FROM **employee e LEFT JOIN department d ON e.department_id = d.id;**

Right outer join:

-- right outer join, let us get all the departments

48. SELECT e.id, e.name AS employee_name, d.name AS department_name

FROM **employee e RIGHT JOIN department d ON e.department_id = d.id**;

Full Join

49. SELECT e.id, e.name AS employee_name, d.name AS department_name

FROM **employee e FULL JOIN department d ON e.department_id = d.id;**