SQL Basics Tutorial for Beginners (Practice SQL Queries)

Download Scripts for practice

Download DDL & DML

Click on “Download DDL & DML” link to download all the create table and insert table scripts.

Download SELECT Queries

Click on “Download SELECT Queries” link to download all the SQL SELECT queries.

Introduction

There are 100’s of programming languages today such as Java, JavaScript, Python, C, C++, C#, PHP, Ruby, Swift and so many more but arguably one of the most easiest programming language that you can learn is **SQL**.

SQL is not only easy to learn, but is also one of the most widely used programming language in the world.

SQL is not just used by software developers, but it is also widely used by Quality Analyst, Data Analyst, Business Analyst, Data Engineers, Data Scientist and many more etc.

In this blog, We will cover only the most important and the most basic SQL concepts which we strongly believe are required to get started with SQL. We will also guide you through on how to write simple SQL queries.

What is SQL?

Overview on SQL Commands

DDL & DML

Constraints & Data types

SELECT Queries

Different functionalities associated with SELECT

List of topics covered in this blog

What is SQL?

SQL stands for Structured Query Language. Also referred to as “SEQUEL”

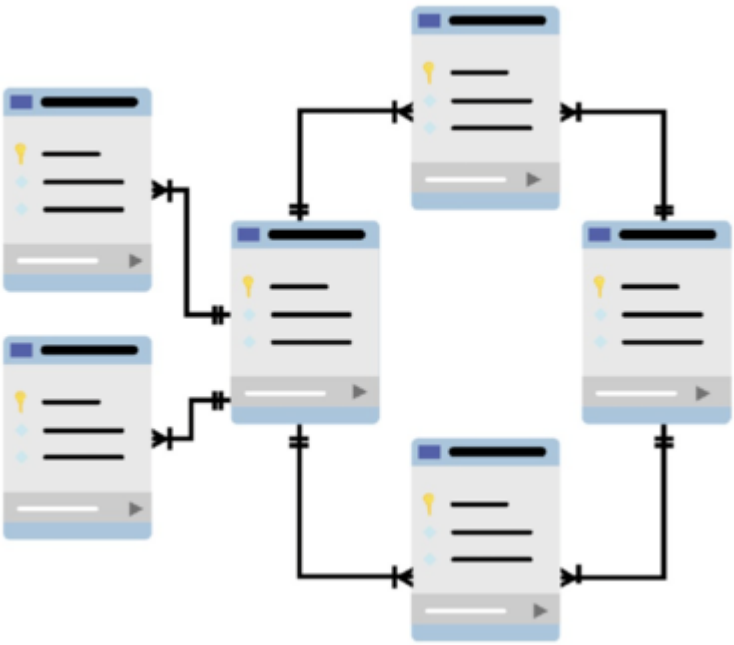
SQL is a programming language which is used to interact with relational database or RDBMS.

Using SQL commands, you can read data from a relational database or write data into a relational database. You can also create, modify and delete database. SQL can also be used to analyze and protect data.

Almost all of the RDBMS available today such as Oracle, MySQL, Microsoft SQL Server, PostgreSQL, IBM DB2, SQLite, MariaDB, Teradata, Hive etc uses SQL to manage data.

Different relational database have created their own version of SQL, however majority of the functionally and syntax are common across all databases.

So no matter which RDBMS you use, the SQL you learn in this blog will be applicable to all.



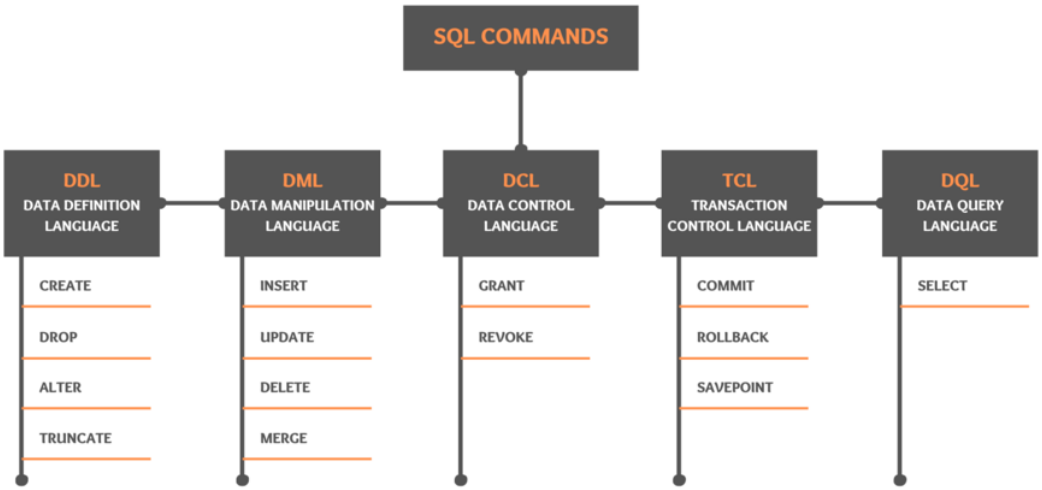
Note:

SQL is primarily used in RDBMS (or Relational Database Management System).

In RDBMS, data is stored in multiple tables.

Each table can have a set of rows and columns. Different tables will be related to each other through certain columns forming relations.

SQL Commands



SQL Commands can be categorized into 5 types, namely:

DDL (Data Definition Language):

Used to define the structure of database objects such as tables, views, functions etc. Using DDL commands, we can create, modify and drop any database objects. The commands include:

- CREATE

  Create a new database object such a tables, views or functions etc.

  Syntax to create a new table is:

  CREATE TABLE IF NOT EXISTS STUDENTS

  (

  ID VARCHAR(20) PRIMARY KEY

  , FIRST_NAME VARCHAR(100) NOT NULL

  , LAST_NAME VARCHAR(100) NOT NULL

  , GENDER VARCHAR(10) CHECK (GENDER IN ('M', 'F', 'Male', 'Female'))

  , AGE INT

  , DOB DATE

, GRADE FLOAT

, IS_ACTIVE BOOLEAN

, CONSTRAINT CH_STUDENTS_AGE CHECK (AGE > 0)

);

Here STUDENTS is the name of the table whereas it has 6 columns namely ID, FIRST_NAME, LAST_NAME, GENDER, AGE and DOB. These columns belong to different data type. There are also few constraints like PRIMARY KEY, NOT NULL, CHECK used in this table.

IF NOT EXISTS before the table name is an optional clause which can be included while creating table which tells the RDBMS to check if the table already exist. If table exists then RDBMS will skip executing this create statement else the statement will be executed and table will be create.

In order to completely understand how to properly create a table, we need to first understand what is Data Type and what are Constraints:

DATA TYPE

Tables in RDBMS consist of rows and columns. Each column has a data type associated to it. Data type is like a data rule applicable to that particular column. Meaning that only the data or values satisfying this data rule can be inserted into this column. There are several types of data types. However, in this blog we will go through the 5 most basic and most commonly used data type across all of RDBMS, they are VARCHAR, INT, DATE, FLOAT and BOOLEAN:

- **VARCHAR:** Stands for Variable Character. If a column is associated to a VARCHAR data type then the values that can be stored in this column are alphabets, numbers, alphanumeric values as well as special characters.
- **INT:** Stands for Integer. As the name suggests, only integer/whole numbers are allowed under INT column.
- **DATE:** DATE data type is used to store DATE values which are in any date format.

- **FLOAT:** Stands for Floating point numbers. It can hold decimal numbers only.
- **BOOLEAN:** It can hold only 2 values either True or False. It's kind of a binary representation of 0 and 1 where True = 1 and False = 0

CONSTRAINTS

Constraints refers to limitation or restriction applied to a column in a table. Constraints are very important to maintain data integrity among tables. If you want to make sure that wrong data is not inserted into your table then these kind of sanity checks can be applied by using CONSTRAINTS. Let's look some of the most widely used constraints in RDBMS:

- **CHECK:** CHECK constraint allows you to control the values that can be inserted into a column. Let's say if you have a column "AGE" and you want to make sure that only positive values are being is inserted into this column then you can use a CHECK constraint on this column to apply this rule. So if anyone tries to insert a negative value into this column then RDBMS will throw an error.
- **NOT NULL:** By applying NOT NULL constraint on a column, you make sure that this column will never have NULL or empty values.
- **UNIQUE:** Unique constraints are used to make sure that values inserted into a column across all the rows have unique or distinct values. It can help you to eliminate any duplicate data in a column. Remember NULL values are allowed in a UNIQUE constraint column. And two NULL values are not same hence multiple rows with NULL values are allowed.
- **PRIMARY KEY:** Primary key constraints is basically a combination of UNIQUE + NOT NULL constraint. It will ensure that all the values in the column are unique and there are no NULL values. A table can only have one primary key constraint. Primary key constraint can either be applied to a single column or to a combination of multiple columns in the table.

    - **FOREIGN KEY:** FOREIGN KEY constraint can be used to form relationship between tables. It basically helps to create a parent child relationship between 2 tables. Such that the child table references a column value from the parent table. So that only the values present in parent table can be inserted into the child table.

- ALTER

  Alter is used to modify the structure of an existing table. Alter can be used to rename a table or rename a column. Alter can also be used to add new column or change the data type of an existing column. Using Alter we can also add new constraints or remove a constraint from a table.

  Syntax to alter table is:

  ALTER TABLE STUDENTS DROP COLUMN GRADE; -- Drop a column.

  ALTER TABLE STUDENTS ADD COLUMN REGISTER_NO VARCHAR(100); -- Add new column.

  ALTER TABLE STUDENTS ALTER COLUMN IS_ACTIVE TYPE VARCHAR(1); -- Change data type of a column.

  ALTER TABLE STUDENTS RENAME COLUMN IS_ACTIVE TO ACTIVE; -- Rename a column.

  ALTER TABLE STUDENTS ADD CONSTRAINT UNQ_STD UNIQUE (REGISTER_NO); -- Add new constraint

  ALTER TABLE STUDENTS DROP CONSTRAINT UNQ_STD; -- Drop a constraint.

  ALTER TABLE STUDENTS RENAME TO STUDENTS123; -- Rename a table.

- DROP

  Drop, as the name suggest, is used to remove a database object such as table, view, functions etc from the database.

  Syntax to drop table is:

  DROP TABLE STUDENTS;

- TRUNCATE:

  Truncate is used to remove all the data from a table at once.

Syntax to truncate table is:

TRUNCATE TABLE STUDENTS;

DML (Data Manipulation Language):

DML commands are used to load, modify and remove data from the database. The commands include:

- INSERT

  Insert command can be used to load data into the table.

  The syntax is: *(considering the original table structure we created above in CREATE command)*

  INSERT INTO STUDENTS (ID, FIRST_NAME, LAST_NAME, GENDER, AGE, DOB, GRADE, IS_ACTIVE)

  VALUES ('STD10251','Minnaminnie','Cleft','Female',8,TO_DATE('2012-02-23', 'YYYY-MM-DD'), 3, TRUE); -- Mention the column names.

  INSERT INTO STUDENTS

  VALUES ('STD10252','Effie','Emlyn','Female',8,TO_DATE('2012-03-28', 'YYYY-MM-DD'), 3, TRUE); -- Do not mention column names.

  INSERT INTO STUDENTS VALUES

  ('STD10253','Kerry','Aysik','Female',8,TO_DATE('2012-01-09', 'YYYY-MM-DD'), 3, TRUE),

  ('STD10254','Jo','Mansfield','Male',8,TO_DATE('2012-03-26', 'YYYY-MM-DD'), 3, TRUE),

  ('STD10255','Elianore','Macon','Female',8,TO_DATE('2012-04-01', 'YYYY-MM-DD'), 3, FALSE); -- Insert multiple records.

- UPDATE

  Update commands is used to modify the data in the table.

  The syntax is:

  UPDATE STUDENTS

  SET FIRST_NAME = 'James'

  WHERE ID = 'STD10253'; -- Update single column.

  UPDATE STUDENTS

SET FIRST_NAME = 'Rohan', GRADE = 4

WHERE ID = 'STD10251'; -- Update multiple columns at once.

- DELETE

  Delete command will remove the data from a table. If you want to delete all records from table then mention the delete statement without the WHERE condition. Else mention WHERE condition to specify the exact records to be deleted.

  The syntax is:

  DELETE FROM STUDENTS WHERE ID = 'STD10251'; -- Removes only one record.

  DELETE FROM STUDENTS; -- Removes all data from table.

DCL (Data Control Language):

Using DCL commands, database objects from a one database or schema can be accessed from another database or schema. DCL includes following commands:

- GRANT

  Provide access for database objects to be accessed from different database or schema.

- REVOKE

  Remove the access for database objects to be accessed from different database or schema.

TCL (Transaction Control Language):

TCL commands are basically used to save or undo DML transactions into the database. It includes:

- COMMIT

  Saves the open transaction to database. Such as Insert, Delete, Update transactions etc.

- ROLLBACK

  Used to undo a transaction which are not yet committed.

- SAVEPOINT

  Can be used to create reference points in between a group of transaction which can be then called upon to either commit or rollback the preceding transactions.

DQL (Data Query Language):

The SELECT statement in SQL falls under the category of DQL. Using SELECT, we can retrieve data from one or more tables. SELECT can also be used to build reports, analyze data and much more.

SELECT

The basic syntax to write SELECT query is:

SELECT column_name

FROM table_name

WHERE join / filter conditions;

SELECT clause - All the columns which needs to be displayed when the query is executed.

FROM clause - All the tables which are required to execute this query.

WHERE clause - All the join conditions or filter conditions to fetch the desired data.

There are two ways to write SELECT queries:

1. Using JOIN keyword between tables in FROM clause.

   SELECT T1.COLUMN1 AS C1, T1.COLUMN2 C2, T2.COLUMN3 AS C3

   FROM TABLE1 T1

   JOIN TABLE2 AS T2 ON T1.C1 = T2.C1 AND T1.C2 = T2.C2;
2. Using comma (,) between tables in FROM clause.

   SELECT T1.COLUMN1 AS C1, T1.COLUMN2 AS C2, T2.COLUMN3 C3

   FROM TABLE1 AS T1, TABLE2 AS T2

WHERE T1.C1 = T2.C1

AND T1.C2 = T2.C2;

Both these method of writing queries are correct and you can follow whichever you are comfortable with. However, I personally prefer the first method since it makes the query more cleaner and easier to understand and debug. Also very useful when writing outer join queries.

In the above query T1 following the table name TABLE1 and T2 following table name TABLE2 are aliases to the name (kind of nick name given to the table). This aliases can be used through out the query to access the actual table. Similarly C1, C2, C3 are aliases given to column names. You can use the keyword "AS" while stating the aliases however it's not mandatory.

SELECT Queries

Best way to learn any programming language is by practicing it hence in the beginning of this blog, I have included all the DDL and DML scripts ("Download DDL & DML") which you can download and execute in your machine.

Once you execute DDL and DML commands, you can then download the "Download SELECT Queries" file which contains all the SQL SELECT queries for you to practice and learn.

-- List of all the tables. Create and Insert script can be downloaded by clicking on the button "Download DDL & DML" in the very beginning of this blog.

-- All these tables are designed to replicate a School Database.

*SELECT * FROM SCHOOL; -- Table contains school name and other school details.*

*SELECT * FROM SUBJECTS; -- Contains all subjects thought in this school.*

*SELECT * FROM STAFF; -- All teaching and non teaching staff details are present here.*

*SELECT * FROM STAFF_SALARY; -- Staff salary can be found in this table.*

*SELECT * FROM CLASSES; -- Total classes in the school (from Grade 1 to Grade 10) along with subjects thought in each class and the teachers teaching these subjects.*

*SELECT * FROM STUDENTS; -- Student details including their name, age, gender etc.*

*SELECT * FROM PARENTS; -- Parents details including their name, address etc.*

*SELECT * FROM STUDENT_CLASSES; -- Students present in each class (or grade).*

*SELECT * FROM STUDENT_PARENT; -- Parent of each student can be found here.*

*SELECT * FROM ADDRESS; -- Address of all staff and students.*

/ Different SQL Operators::: = , <, >, >=, <=, <>, !=, BETWEEN, ORDER BY, IN, NOT IN, LIKE, ALIASE, DISTINCT, LIMIT, CASE:*

Comparison Operators: =, <>, != , >, <, >=, <=

*Arithmetic Operators: +, -, *, /, %*

*Logical Operators: AND, OR, NOT, IN, BETWEEN, LIKE etc. */*

-- Basic queries

*SELECT * FROM STUDENTS; -- Fetch all columns and all records (rows) from table.*

SELECT ID, FIRST_NAME FROM STUDENTS; -- Fetch only ID and FIRST_NAME columns from students table.

-- Comparison Operators

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME = 'Mathematics'; -- Fetch all records where subject name is Mathematics.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME <> 'Mathematics'; -- Fetch all records where subject name is not Mathematics.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME != 'Mathematics'; -- same as above. Both "<>" and "!=" are NOT EQUAL TO operator in SQL.*

*SELECT * FROM STAFF_SALARY WHERE SALARY > 10000; -- All records where salary is greater than 10000.*

*SELECT * FROM STAFF_SALARY WHERE SALARY < 10000; -- All records where salary is less than 10000.*

*SELECT * FROM STAFF_SALARY WHERE SALARY < 10000 ORDER BY SALARY; -- All records where salary is less than 10000 and the output is sorted in ascending order of salary.*

*SELECT * FROM STAFF_SALARY WHERE SALARY < 10000 ORDER BY SALARY DESC; -- All records where salary is less than 10000 and the output is sorted in descending order of salary.*

*SELECT * FROM STAFF_SALARY WHERE SALARY >= 10000; -- All records where salary is greater than or equal to 10000.*

*SELECT * FROM STAFF_SALARY WHERE SALARY <= 10000; -- All records where salary is less than or equal to 10000.*

-- Logical Operators

*SELECT * FROM STAFF_SALARY WHERE SALARY BETWEEN 5000 AND 10000; -- Fetch all records where salary is between 5000 and 10000.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME IN ('Mathematics', 'Science', 'Arts'); -- All records where subjects is either Mathematics, Science or Arts.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME NOT IN ('Mathematics', 'Science', 'Arts'); -- All records where subjects is not Mathematics, Science or Arts.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME LIKE 'Computer%'; -- Fetch records where subject name has Computer as prefixed. % matches all characters.*

*SELECT * FROM SUBJECTS WHERE SUBJECT_NAME NOT LIKE 'Computer%'; -- Fetch records where subject name does not have Computer as prefixed. % matches all characters.*

*SELECT * FROM STAFF WHERE AGE > 50 AND GENDER = 'F'; -- Fetch records where staff is female and is over 50 years of age. AND operator fetches result only if the condition mentioned both on left side and right side of AND operator holds true. In OR operator, atleast any one of the conditions needs to hold true to fetch result.*

*SELECT * FROM STAFF WHERE FIRST_NAME LIKE 'A%' AND LAST_NAME LIKE 'S%'; -- Fetch record where first name of staff starts with "A" AND last name starts with "S".*

*SELECT * FROM STAFF WHERE FIRST_NAME LIKE 'A%' OR LAST_NAME LIKE 'S%'; -- Fetch record where first name of staff starts with "A" OR last name starts with "S". Meaning either the first name or the last name condition needs to match for query to return data.*

*SELECT * FROM STAFF WHERE (FIRST_NAME LIKE 'A%' OR LAST_NAME LIKE 'S%') AND AGE > 50; -- Fetch record where staff is over 50 years of age AND has his first name starting with "A" OR his last name starting with "S".*

-- Arithmetic Operators

SELECT (5+2) AS ADDITION; -- Sum of two numbers. PostgreSQL does not need FROM clause to execute such queries.

SELECT (5-2) AS SUBTRACT; -- Oracle & MySQL equivalent query would be --> select (5+2) as Addition FROM DUAL; --> Where dual is a dummy table.

*SELECT (5*2) AS MULTIPLY;*

SELECT (5/2) AS DIVIDE; -- Divides 2 numbers and returns whole number.

SELECT (5%2) AS MODULUS; -- Divides 2 numbers and returns the remainder

SELECT STAFF_TYPE FROM STAFF ; -- Returns lot of duplicate data.

SELECT DISTINCT STAFF_TYPE FROM STAFF ; -- Returns unique values only.

SELECT STAFF_TYPE FROM STAFF LIMIT 5; -- Fetches only the first 5 records from the result.

-- CASE statement: (IF 1 then print True ; IF 0 then print FALSE ; ELSE print -1)

SELECT STAFF_ID, SALARY

, CASE WHEN SALARY >= 10000 THEN 'High Salary'

WHEN SALARY BETWEEN 5000 AND 10000 THEN 'Average Salary'

WHEN SALARY < 5000 THEN 'Too Low'

END AS RANGE

FROM STAFF_SALARY

ORDER BY 2 DESC;

-- TO_CHAR / TO_DATE:

*SELECT * FROM STUDENTS WHERE TO_CHAR(DOB,'YYYY') = '2014';*

*SELECT * FROM STUDENTS WHERE DOB = TO_DATE('13-JAN-2014','DD-MON-YYYY');*

-- JOINS (Two ways to write SQL queries):

-- #1. Using JOIN keyword between tables in FROM clause.

SELECT T1.COLUMN1 AS C1, T1.COLUMN2 C2, T2.COLUMN3 AS C3 -- C1, C2, C3 are aliase to the column

FROM TABLE1 T1

JOIN TABLE2 AS T2 ON T1.C1 = T2.C1 AND T1.C2 = T2.C2; -- T1, T2 are aliases for table names.

-- #2. Using comma "," between tables in FROM clause.

SELECT T1.COLUMN1 AS C1, T1.COLUMN2 AS C2, T2.COLUMN3 C3

FROM TABLE1 AS T1, TABLE2 AS T2

WHERE T1.C1 = T2.C1

AND T1.C2 = T2.C2;

-- Fetch all the class name where Music is thought as a subject.

SELECT CLASS_NAME

FROM SUBJECTS SUB

JOIN CLASSES CLS ON SUB.SUBJECT_ID = CLS.SUBJECT_ID

WHERE SUBJECT_NAME = 'Music';

-- Fetch the full name of all staff who teach Mathematics.

SELECT DISTINCT (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME --, CLS.CLASS_NAME

FROM SUBJECTS SUB

JOIN CLASSES CLS ON CLS.SUBJECT_ID = SUB.SUBJECT_ID

JOIN STAFF STF ON CLS.TEACHER_ID = STF.STAFF_ID

WHERE SUB.SUBJECT_NAME = 'Mathematics';

-- Fetch all staff who teach grade 8, 9, 10 and also fetch all the non-teaching staff

-- UNION can be used to merge two differnt queries. UNION returns always unique records so any duplicate data while merging these queries will be eliminated.

-- UNION ALL displays all records including the duplicate records.

-- When using both UNION, UNION ALL operators, rememeber that noo of columns and their data type must match among the different queries.

SELECT STF.STAFF_TYPE

```
, (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME

, STF.AGE

, (CASE WHEN STF.GENDER = 'M' THEN 'Male'

WHEN STF.GENDER = 'F' THEN 'Female'

END) AS GENDER

, STF.JOIN_DATE

FROM STAFF STF

JOIN CLASSES CLS ON STF.STAFF_ID = CLS.TEACHER_ID

WHERE STF.STAFF_TYPE = 'Teaching'

AND CLS.CLASS_NAME IN ('Grade 8', 'Grade 9', 'Grade 10')

UNION ALL

SELECT STAFF_TYPE

, (FIRST_NAME||' '||LAST_NAME) AS FULL_NAME, AGE

, (CASE WHEN GENDER = 'M' THEN 'Male'

WHEN GENDER = 'F' THEN 'Female'

END) AS GENDER

, JOIN_DATE

FROM STAFF

WHERE STAFF_TYPE = 'Non-Teaching';

-- Count no of students in each class

SELECT SC.CLASS_ID, COUNT(1) AS "no_of_students"

FROM STUDENT_CLASSES SC
```

```
GROUP BY SC.CLASS_ID

ORDER BY SC.CLASS_ID;

-- Return only the records where there are more than 100 students in each class

SELECT SC.CLASS_ID, COUNT(1) AS "no_of_students"

FROM STUDENT_CLASSES SC

GROUP BY SC.CLASS_ID

HAVING COUNT(1) > 100

ORDER BY SC.CLASS_ID;

-- Parents with more than 1 kid in school.

SELECT PARENT_ID, COUNT(1) AS "no_of_kids"

FROM STUDENT_PARENT SP

GROUP BY PARENT_ID

HAVING COUNT(1) > 1;

--SUBQUERY: Query written inside a query is called subquery.

-- Fetch the details of parents having more than 1 kids going to this school. Also display student details.

SELECT (P.FIRST_NAME||' '||P.LAST_NAME) AS PARENT_NAME

, (S.FIRST_NAME||' '||S.LAST_NAME) AS STUDENT_NAME

, S.AGE AS STUDENT_AGE

, S.GENDER AS STUDENT_GENDER

, (ADR.STREET||', '||ADR.CITY||', '||ADR.STATE||', '||ADR.COUNTRY) AS ADDRESS

FROM PARENTS P
```

```
JOIN STUDENT_PARENT SP ON P.ID = SP.PARENT_ID

JOIN STUDENTS S ON S.ID = SP.STUDENT_ID

JOIN ADDRESS ADR ON P.ADDRESS_ID = ADR.ADDRESS_ID

WHERE P.ID IN ( SELECT PARENT_ID

FROM STUDENT_PARENT SP

GROUP BY PARENT_ID

HAVING COUNT(1) > 1)

ORDER BY 1;

-- Staff details who's salary is less than 5000

SELECT STAFF_TYPE, FIRST_NAME, LAST_NAME

FROM STAFF

WHERE STAFF_ID IN (SELECT STAFF_ID

FROM STAFF_SALARY

WHERE SALARY < 5000);

--Aggregate Functions (AVG, MIN, MAX, SUM, COUNT): Aggregate functions are used to perform calculations on a set of values.

-- AVG: Calculates the average of the given values.

SELECT AVG(SS.SALARY)::NUMERIC(10,2) AS AVG_SALARY

FROM STAFF_SALARY SS

JOIN STAFF STF ON STF.STAFF_ID = SS.STAFF_ID

WHERE STF.STAFF_TYPE = 'Teaching';

SELECT STF.STAFF_TYPE, AVG(SS.SALARY)::NUMERIC(10,2) AS AVG_SALARY
```

FROM STAFF_SALARY SS

JOIN STAFF STF ON STF.STAFF_ID = SS.STAFF_ID

GROUP BY STF.STAFF_TYPE;

/ Note*:

"::NUMERIC" is a cast operator which is used to convert values from one data type to another.

In the above query we use it display numeric value more cleanly by restricting the decimal point to only 2.

Here 10 is precision which is the total no of digits allowed.

2 is the scale which is the digits after decimal point.

**/*

-- SUM: Calculates the total sum of all values in the given column.

SELECT STF.STAFF_TYPE, SUM(SS.SALARY)::NUMERIC(10,2) AS AVG_SALARY

FROM STAFF_SALARY SS

JOIN STAFF STF ON STF.STAFF_ID = SS.STAFF_ID

GROUP BY STF.STAFF_TYPE;

-- MIN: Returns the record with minimun value in the given column.

SELECT STF.STAFF_TYPE, MIN(SS.SALARY)::NUMERIC(10,2) AS AVG_SALARY

FROM STAFF_SALARY SS

JOIN STAFF STF ON STF.STAFF_ID = SS.STAFF_ID

GROUP BY STF.STAFF_TYPE;

-- MAX: Returns the record with maximum value in the given column.

SELECT STF.STAFF_TYPE, MAX(SS.SALARY)::NUMERIC(10,2) AS AVG_SALARY

FROM STAFF_SALARY SS

JOIN STAFF STF ON STF.STAFF_ID = SS.STAFF_ID

GROUP BY STF.STAFF_TYPE;

*/**

SQL Joins: There are several types of JOIN but we look at the most commonly used:

1) Inner Join

- Inner joins fetches records when there are matching values in both tables.

2) Outer Join

- Left Outer Join

- Left join fetches all records from left table and the matching records from right table.

- The count of the query will be the count of the Left table.

- Columns which are fetched from right table and do not have a match will be passed as NULL.

- Right Outer Join

- Right join fetches all records from right table and the matching records from left table.

- The count of the query will be the count of the right table.

- Columns which are fetched from left table and do not have a match will be passed as NULL.

- Full Outer Join

- Full join always return the matching and non-matching records from both left and right table.

*/

-- Inner Join: 21 records returned – Inner join always fetches only the matching records present in both right and left table.

-- Inner Join can be represented as eithe "JOIN" or as "INNER JOIN". Both are correct and mean the same.

```
SELECT COUNT(1)

FROM STAFF STF

JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1;

SELECT DISTINCT (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME,
SS.SALARY

FROM STAFF STF

JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 2;
```

-- 23 records – 23 records present in left table.

-- All records from LEFT table with be fetched irrespective of whether there is matching record in the RIGHT table.

```
SELECT COUNT(1)

FROM STAFF STF

LEFT JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1;

SELECT DISTINCT (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME,
SS.SALARY

FROM STAFF STF
```

LEFT JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 2;

-- 24 records – 24 records in right table.

-- All records from RIGHT table with be fetched irrespective of whether there is matching record in the LEFT table.

SELECT COUNT(1)

FROM STAFF STF

RIGHT JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1;

SELECT DISTINCT (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME, SS.SALARY

FROM STAFF STF

RIGHT JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1;

-- 26 records – all records from both tables. 21 matching records + 2 records from left + 3 from right table.

-- All records from both LEFT and RIGHT table with be fetched irrespective of whether there is matching record in both these tables.

SELECT COUNT(1)

FROM STAFF STF

FULL OUTER JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1;

SELECT DISTINCT (STF.FIRST_NAME||' '||STF.LAST_NAME) AS FULL_NAME, SS.SALARY

FROM STAFF STF

FULL OUTER JOIN STAFF_SALARY SS ON SS.STAFF_ID = STF.STAFF_ID

ORDER BY 1,2;

**** THE END ****