Hyperiondev

TASK

Working with SQL

Visit our website

Introduction

WELCOME TO THE WORKING WITH SQL TASK!

In this task, you will learn a language used for manipulating and managing databases: Structured Query Language (SQL). In addition, you will be introduced to a popular open-source relational database: MySQL.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INTRODUCTION TO SQL

SQL is a database language that is composed of commands that enable users to create databases or table structures, perform various types of data manipulation and data administration as well as query the database to extract useful information. SQL is supported by all relational database management system (RDBMS) software. SQL is portable, which means that a user does not have to relearn the basics when moving from one RDBMS to another because all RDBMSs will use SQL in almost the same way.

SQL is easy to learn since its vocabulary is relatively simple. Its basic command set has a vocabulary of fewer than 100 words. It is also a non-procedural language, which means that the user specifies what must be done and not how it should be done. Users do not need to know the physical data storage format or the complex activities that take place when a SQL command is executed in order to issue a command.

According to Rob and Coronel (2009), SQL functions fit into two general categories:

1. **A data definition language (DDL):** it includes commands that allow one to define or create database objects (such as tables, indexes, and views). SQL also includes commands to define access rights to the database objects created.

2. **A data manipulation language (DML):** it includes commands that allow one to manipulate the data stored in the database. For example, commands to insert, update, delete, and retrieve data.

The commands in each category are given in the tables below. We will explore some of these commands in the rest of this task

The table below lists the SQL data definition commands:

Command	Description
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column when no value is given
CHECK	Used to validate data in an attribute
CREATE INDEX	Creates an index for the table
CREATE VIEW	Creates a dynamic subset of rows or columns from one or more tables
ALTER TABLE	Modifies a table (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

Table source: (Rob & Coronel, 2009, p. 225)

The table below lists the SQL data manipulation commands:

Command	Description
INSERT	Inserts rows into a table
SELECT	Select attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more tables rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to its original values
Comparison Operators	=, <, >, <=, >=, <>
Logical Operators	AND, OR, NOT
Special Operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate Functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

Table source: Adapted from (Rob & Coronel, 2009, p. 225)

CREATING TABLES

To create new tables in SQL, you use the `CREATE TABLE` statement. Pass all the columns you want in the table, as well as their data types, as arguments to the `CREATE TABLE` function. The syntax of the `CREATE TABLE` statement is shown below:

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
   ....
);
```

Note the optional constraint arguments. They are used to specify rules for data in a table. Constraints that are commonly used in SQL include:

- `NOT NULL`: Ensures that a column cannot have a null value
- `UNIQUE`: Ensures that all values in a column are different
- `DEFAULT`: Sets a default value for a column when no value is specified
- `INDEX`: Creates an index which is used to create and retrieve data from the database very quickly

To create a table called Employee, for example, that contains five columns (EmployeeID, LastName, FirstName, Address, and PhoneNumber), you would use the following SQL:

```
CREATE TABLE Employee (
    EmployeeID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    PhoneNumber varchar(255)
);
```

The EmployeeID column is of type `int` and will, therefore, hold an integer value. The LastName, FirstName, Address and PhoneNumber columns are of type `varchar` and will, therefore, hold characters. The number in brackets indicates the maximum number of characters, which in this case is 255.

The `CREATE TABLE` statement above will create an empty Employee table that will look like this:

EmployeeID	LastName	FirstName	Address	PhoneNumber

When creating tables, it's advisable to add a primary key to one of the columns as this will help keep entries unique and will speed up select queries. Primary keys must contain unique values and cannot contain null values. A table can only contain one primary key; however, the primary key may consist of a single column or multiple columns.

You can add a primary key when creating the Employee table as follows:

```
CREATE TABLE Employee (
    EmployeeID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    PhoneNumber varchar(255),
    PRIMARY KEY (EmployeeID)
);
```

To name a primary key constraint and define a primary key constraint on multiple columns, you use the following SQL syntax:

```
CREATE TABLE Employee (
    EmployeeID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    PhoneNumber varchar(255),
    CONSTRAINT PK_Employee PRIMARY KEY (EmployeeID, LastName)
);
```

In the example above, there is only one primary key named PK_Employee. However, the value of the primary key is made up of two columns: ID and LastName.

INSERTING ROWS

The table that we have just created is empty and needs to be populated with rows or records. We can add entries to a table using the `INSERT INTO` command.

There are two ways to write the `INSERT INTO` command:

1. You do not have to specify the column names where you intend to insert the data. This can be done if you are adding values for all of the columns of the table. However, you should ensure that the order of the values is in the same order as the columns in the table. The syntax will be as follows:

```
INSERT INTO table_name
      VALUES (value1, value2, value3, ...);
```

   For example, to add an entry to the Employee table (created previously with SQL), you would do the following:

```
INSERT INTO employee
      VALUES (1, 'Smith', 'John', '25 Oak Rd', '0837856767');
```

2. The other way to insert data into a table is to specify both the column names and the values to be inserted. The syntax will be as follows:

```
INSERT INTO table_name (column1, column2, column3, ...)
      VALUES (value1, value2, value3, ...);
```

Therefore, to add an entry to the Employee table using this method, you would do the following:

```
INSERT INTO Employee (EmployeeID, LastName, FirstName, Address, PhoneNumber)
      VALUES (1234, 'Smith', 'John', '25 Oak Rd', '0837856767');
```

The second method for inserting data into a table is often used when you don't want to add data to all the columns in the table.

RETRIEVING DATA FROM A TABLE

The `SELECT` statement is used to fetch data from a database. The data returned is stored in a result table, known as the result-set. The syntax of a `SELECT` statement is as follows:

```
SELECT column1, column2, ...
FROM table_name;
```

`column1, column2, ...` refer to the column names of the table from which you want to select data. The following example selects the FirstName and LastName columns from the Employee table:

```
SELECT FirstName, LastName
FROM Employee;
```

If you want to select all the columns in the table, you use the following syntax:

```
SELECT * FROM table_name;
```

The asterisk (`*`) indicates that we want to fetch all of the columns, without excluding any of them.

You can also order and filter the data that is returned when using the `SELECT` statement using the `ORDER BY` and `WHERE` commands.

ORDER BY

You can use the `ORDER BY` command to sort the results returned in ascending or descending order. The `ORDER BY` command sorts the records in ascending order by default. You need to use the `DESC` keyword to sort the records in descending order.

The `ORDER BY` syntax is as follows:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

The example below selects all Employees in the Employee table and sorts them in descending order, based on the values in the FirstName column:

```
SELECT * FROM Employee
ORDER BY FirstName DESC;
```

WHERE

The `WHERE` clause allows us to filter data depending on a specific condition. The syntax of the `WHERE` clause is as follows:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following SQL statement selects all the employees with the first name 'John', in the Employee table:

```
SELECT * FROM Employee
WHERE FirstName='John';
```

Note that SQL requires single quotes around text values; however, you do not need to enclose numeric fields in quotes.

You can use logical operators (`AND`, `OR`) and comparison operators (`=`,`<`,`>`,`<=`,`>=`,`<>`) to make `WHERE` conditions as specific as you like.

For example, suppose you have the following table which contains the most sold albums of all time:

albums

Artist	Album	Released	Genre	sales_in_millions
Michael Jackson	Thriller	1982	pop	70
AC/DC	Back in Black	1980	rock	50
Pink Floyd	The Dark Side of the Moon	1973	rock	45
Whitney Houston	The Bodyguard	1992	soul	44

You can select the records that are classified as rock and have sold under 50 million copies by simply using the `AND` operator as follows:

```
SELECT *
FROM albums
WHERE Genre = 'rock' AND sales_in_millions <= 50
ORDER BY Released
```

`WHERE` statements also support some special operators to further customise queries:

- `IN`: compares the column to multiple possible values and returns true if it matches at least one
- `BETWEEN`: checks if a value is within an inclusive range
- `LIKE`: searches for a pattern

For example, if we want to select the pop and soul albums from the table above, we can use:

```
SELECT * FROM albums
WHERE Genre IN ('pop', 'soul');
```

Or, if we want to get all the albums released between 1975 and 1985, we can use:

```
SELECT * FROM albums
WHERE Released BETWEEN 1975 AND 1985;
```

AGGREGATE FUNCTIONS

SQL has many functions that do all sorts of helpful stuff. Some of the most regularly used ones are:

- `COUNT()`: returns the number of rows
- `SUM()`: returns the total sum of a numeric column
- `AVG()`: returns the average of a set of values
- `MIN()` / `MAX()`: gets the minimum or maximum value from a column

For example, to get the most recent year in the Album table we can use:

```
SELECT MAX(Released)
FROM albums;
```

Or to get the number of albums released between 1975 and 1985 we can use:

```
SELECT COUNT(album)
WHERE Released BETWEEN 1975 AND 1985;
```

RETRIEVING DATA ACROSS MULTIPLE TABLES

In complex databases, there are often several tables connected to each other in some way. A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Look at the two tables below:

VideoGame

ID	Name	DeveloperID	Genre
1	Super Mario Bros.	2	platformer
2	World of Warcraft	1	MMORPG
3	The Legend of Zelda	2	adventure

GameDeveloper

ID	Name	Country
1	Blizzard	USA
2	Nintendo	Japan

The VideoGame table contains information about various video games, while the GameDeveloper table contains information about the developers of the games. The VideoGame table has a DeveloperID column that holds the game developer ID, which represents the ID of the respective developer from the GameDeveloper table. DeveloperID is a foreign key that points to a specific developer with a matching ID (primary key) in the GameDeveloper table. Therefore, from the tables above we can see that a developer called “Blizzard” from the USA developed the game, “World of Warcraft”. Therefore, the foreign key logically links the two tables and allows us to use the information stored in both of them at the same time.

If we want to create a query that returns everything we need to know about the games, we can use `INNER JOIN` to acquire the columns from both tables.

```
SELECT VideoGame.Name, VideoGame.Genre, GameDeveloper.Name,
GameDeveloper.Country
FROM VideoGame
INNER JOIN GameDeveloper
ON VideoGame.DeveloperID = GameDeveloper.ID;
```

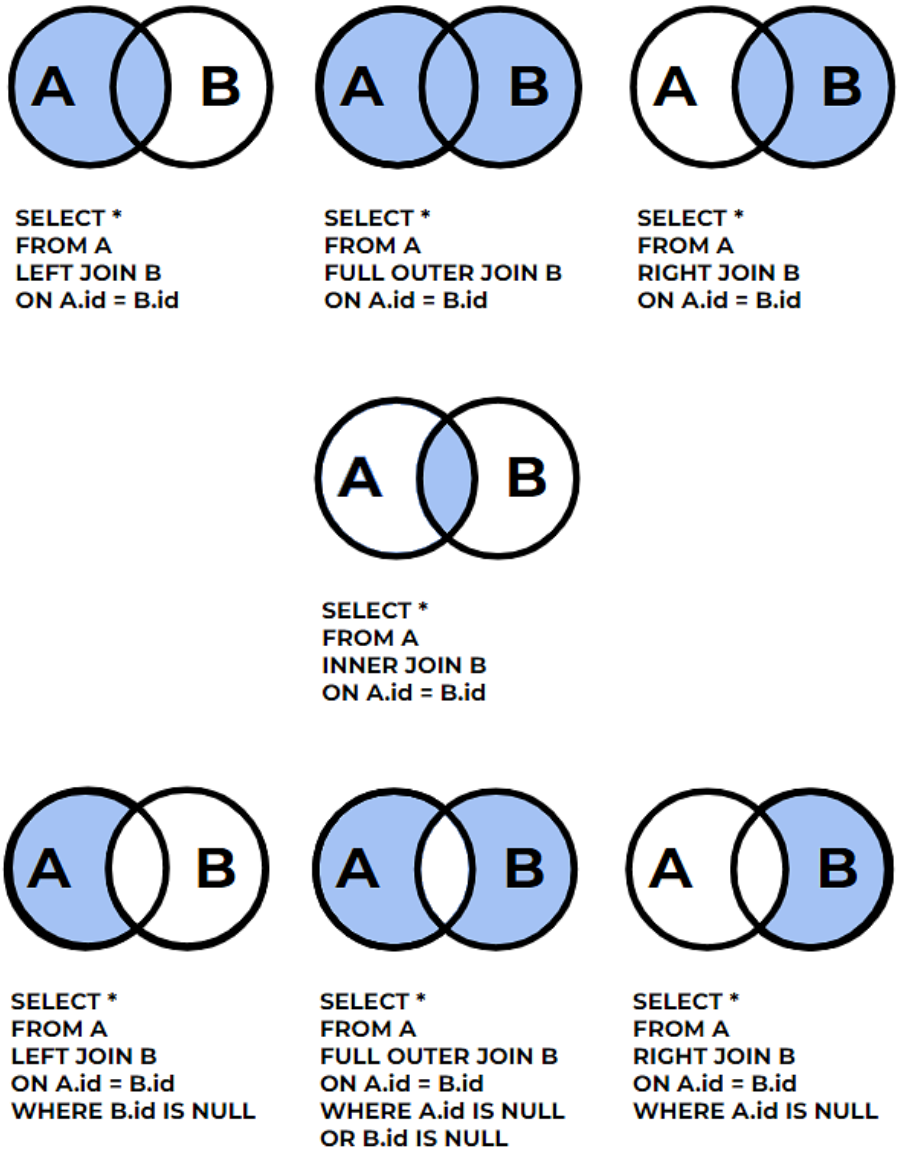
Notice that in the `SELECT` statement above, we specify the name of the table and the column from which we want to retrieve information, and not just the name of the column as we have done previously. This is because we are getting information from more than just one table, and it is possible that tables may have columns with the same names. In this case, both the table VideoGame and the table GameDeveloper contain columns called Name. In the next section, you will see how to use aliases to further address this issue.

Also, notice that we use the `ON` clause to specify how we link the foreign key in one table to the corresponding primary key in the other table. The query above would result in the following data set being returned:

VideoGame.Name	VideoGame.Genre	GameDeveloper.Name	GameDeveloper.Country
Super Mario Bros.	platformer	Nintendo	Japan
World of Warcraft	MMORPG	Blizzard	USA
The Legend of Zelda	adventure	Nintendo	Japan

The `INNER JOIN` is the simplest and most common type of `JOIN`. However, there are many other different types of joins in SQL, namely:

- `INNER JOIN`: returns records that have matching values in both tables
- `LEFT JOIN`: returns all records from the left table, and the matched records from the right table
- `RIGHT JOIN`: returns all records from the right table, and the matched records from the left table
- `FULL JOIN`: returns all records when there is a match in either left or right table



Graphical representation of common SQL joins ***(Mattila, 2012)***

ALIASES

Notice that in the VideoGame and GameDeveloper tables there are two columns called Name. This can become confusing. Aliases are used to give a table or column a temporary name. An alias only exists for the duration of the query and is often used to make column names more readable.

The alias column syntax is:

```
SELECT column_name AS alias_name
FROM table_name;
```

The alias table syntax is:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

The following SQL statement creates an alias for the Name column from the GameDeveloper table:

```
SELECT Name AS Developer
FROM GameDeveloper;
```

See how aliases have been used below:

```
SELECT games.Name, games.Genre, devs.Name AS Developer, devs.Country
FROM VideoGame AS games
INNER JOIN GameDeveloper AS devs
ON games.DeveloperID = devs.ID;
```

UPDATING DATA

The `UPDATE` statement is used to modify the existing rows in a table.

To use the `UPDATE` statement you:

- Choose the table where the row you want to change is located.
- Set the new value(s) for the desired column(s).
- Select which of the rows you want to update using the `WHERE` statement. If you omit this, all rows in the table will change.

The syntax for the update statement is:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Take a look at the following Customer table:

CustomerID	CustomerName	Address	City
1	Maria Anderson	23 York St	New York
2	Jackson Peters	124 River Rd	Berlin
3	Thomas Hardy	455 Hanover Sq	London
4	Kelly Martins	55 Loop St	Cape Town

The following SQL statement updates the first customer (CustomerID = 1) with a new address and a new city:

```
UPDATE Customer
SET Address = '78 Oak St', City= 'Los Angeles'
WHERE CustomerID = 1;
```

DELETING ROWS

Deleting a row is a simple process. All you need to do is select the right table and row that you want to remove. The **DELETE** statement is used to delete existing rows in a table.
The **DELETE** statement syntax is as follows:

```
DELETE FROM table_name
WHERE condition;
```

The following statement deletes the customer 'Jackson Peters' from the Customer table:

```
DELETE FROM Customer
WHERE CustomerName='Jackson Peters';
```

You can also delete all rows in a table without deleting the table:

```
DELETE * FROM table_name;
```

DELETING TABLES

The `DROP TABLE` statement is used to remove every trace of a table in a database. The syntax is as follows:

```
DROP TABLE table_name;
```

For example, if we what to delete the table Customer, we do the following:

```
DROP TABLE Customer;
```

If you want to delete the data inside a table, but not the table itself, you can use the `TRUNCATE TABLE` statement:

```
TRUNCATE TABLE table_name;
```

MYSQL

MySQL is a popular open-source RDBMS owned by MySQL AB. While it is written in C and C++, it is based on SQL. It also supports many different languages and operating systems. MySQL makes using SQL easy because It has an intuitive interface that allows you to add, remove, modify, search, and join tables in a database.

Extra resource

If you'd like to know more about SQL, we highly recommend reading the book, "**Database Design - 2nd Edition**" by Adrienne Watt. Chapters 15 & 16, and Appendix C of this book provide more detail regarding what has been covered in this task.

Compulsory Task 1

Answer the following questions:

- Go to the **w3schools website's SQL browser IDE**. This is where you can write and test your SQL code using their databases. Once you are happy with it, paste your code in a text file named **Student.txt** and save it in your task folder.
- Write the SQL code to create a table called Student. The table structure is summarised in the table below (Note that STU_NUM is the primary key):

Attribute Name	Data Type
STU_NUM	CHAR(6)
STU_SNAME	VARCHAR(15)
STU_FNAME	VARCHAR(15)
STU_INITIAL	CHAR(1)
STU_STARTDATE	DATE
COURSE_CODE	CHAR(3)
PROJ_NUM	INT(2)

- After you have created the table in question 1, write the SQL code to enter the first two rows of the table as below:

STU_NUM	STU_SNAME	STU_FNAME	STU_INITIAL	STU_STARTDATE	COURSE_CODE	PROJ_NUM
01	Snow	John	E	05-Apr-14	201	6
02	Stark	Arya	C	12-Jul-17	305	11

- Assuming all the data in the Student table has been entered as shown below, write the SQL code that will list all attributes for a COURSE_CODE of 305.

STU_ NUM	STU_SNAME	STU_FNAME	STU_INITIAL	STU_START DATE	COURSE_ CODE	PROJ_ NUM
01	Snow	Jon	E	05-Apr-14	201	6
02	Stark	Arya	C	12-Jul-17	305	11
03	Lannister	Jamie	C	05-Sep-12	101	2
04	Lannister	Cercei	J	05-Sep-12	101	2
05	Greyjoy	Theon	I	9-Dec-15	402	14
06	Tyrell	Margaery	Y	12-Jul-17	305	10
07	Baratheon	Tommen	R	13-Jun-19	201	5

- Write the SQL code to change the course code to 304 for the person whose student number is 07.
- Write the SQL code to delete the row of the person named Jamie Lannister, who started on 5 September 2012, whose course code is 101 and project number is 2. Use logical operators to include all of the information given in this problem.
- Write the SQL code that will change the PROJ_NUM to 14 for all those students who started before 1 January 2016 and whose course code is at least 201.
- Write the SQL code that will delete all of the data inside a table, but not the table itself.
- Write the SQL code that will delete the Student table entirely.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved? Do you think we've done a good job?

Click here to share your thoughts anonymously.

.

REFERENCES

Patel, J. (2012). *PHP and MySQL: Practice It Learn It*. Jitendra Patel.

Rob, P., & Coronel, C. (2009). *Database Systems: Design, Implementation, and Management*, (8th ed.). Boston, Massachusetts: Course Technology.

W3Schools.com. (n.d.). *SQL Joins*. Retrieved April 15, 2019, from W3Schools.com: **https://www.w3schools.com/sql/sql_join.asp**