Hyperiondev

TASK

Introduction to Java Database Programming

Visit our website

Introduction

Welcome to the Introduction to the Java Database Programming Task!

In this task you will be introduced to the database language SQL as well as the popular open-source relational database MySQL.

Get in touch

Connect for support

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

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor 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

Structured Query Language (SQL) is a database language that is composed of commands that enable users to create database or table structures, perform various types of data manipulation and data administration, and query the database to extract useful information. SQL is supported by all relational DBMS software and is portable, which means that a user does not have to relearn the basics when moving from one RDBMS to another.

SQL is easy to learn since its vocabulary is relatively simple. It's 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 is to 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.

SQL functions fit into two general categories:

1. SQL is a data definition language (DDL): It includes commands to create database objects (such as tables, indexes and views) as well as commands to define access rights to the to those database objects.

2. SQL is a data manipulation language (DML): It includes commands to insert, update, delete and retrieve data within the database tables.

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

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 based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attributes 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 their original values
Comparison Operators	
=, <, >, <=, >=, <>	Used in conditional expressions
Logical Operators	
AND, OR, NOT	Used in conditional expressions
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
AVG	Returns the average of all values for a given column

Creating Tables

To create new tables in SQL you use the CREATE TABLE statement. As it's arguments it expects all the columns we want in the table, as well as the their data types. The syntax of the CREATE TABLE statement is shown below:

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

Note that the constraints are optional and are used specify rules for data in a table. Constraints that are commonly used in SQL are:

- NOT NULL: Ensures that a column cannot have a NULL value
- UNIQUE: Ensures that all values in a column are different
- PRIMARY KEY: Uniquely identifies each row in a table
- FOREIGN KEY: Uniquely identifies a row in another table
- CHECK: Ensures that all values in a column satisfies a specific condition
- DEFAULT: Sets a default value for a column when no value is specified
- INDEX: 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 do the following:

```
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 or multiple columns.

You can add a primary key to 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 (ID,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. Do not 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, ...);
```

So, for example, to add an entry to the Employee table you would do the following:

```
INSERT INTO Employee
VALUES (1234, Smith, John, 25 Oak Rd, 0837856767);
```

2. The other way 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 way you would do the following:

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

Select

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, ... are the column names of the table you want to select data from. 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 however, you use the following syntax:

```
SELECT * FROM table_name;
```

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

You can also use the ORDER BY command to sort the results 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 by 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:

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 those of them 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 commands, that allows you a quick way to check commonly used queries. These are:

- 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;
```

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;
```

Joins

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

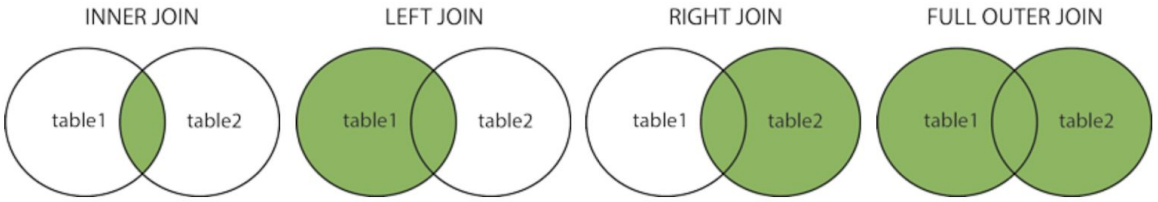
The VideoGame table contains information about various video games while the Game Developer table contains information about the developers of the games. The VideoGame table has a DeveloperID column, that holds the game developer ID which represents the id of the respective developer from the GameDeveloper table. This 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;
```

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 the SQL JOINS (w3schools.com)

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 aliases, for the Name column from GameDeveloper table:

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

This SQL statement shortens the query drastically by setting aliases to the table names:

```
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;
```

Update

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 wanted 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 that you need to do is select the right table and row 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 relational database that is fast and easy to use. MySQL is used by many businesses, both big and small. It is developed, marketed and supported by the Swedish company, MySQL AB.

Some of the reasons why MySQL is so popular are:

- It is an open-source database, which means you don't have to pay anything to use it.

- It is secure. MySQL is globally renowned for being the most secure and reliable database management system. It is used in popular web applications like WordPress, Drupal, Joomla, Facebook and Twitter.

- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.

- It is customisable because it is an open source database and the open-source GPL license allows programmers to modify the SQL software according to their own specific environment.

- It is quicker than other databases so it works well even with a large data set.

- It supports many operating systems with many languages such as PHP, Perl, C, C++, Java etc.

- It uses a standard form of the well-known SQL data language.

- It supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this to a theoretical limit of 8 million terabytes (TB).

Compulsory Task

Answer the following questions:

- Write the SQL code to create a table called Employee. The table structure is summarised in the table below (Note that EMP_NUM is the primary key):

Attribute Name	Datatype
EMP_NUM	CHAR(3)

EMP_LNAME	VARCHAR(15)
EMP_FNAME	VARCHAR(15)
EMP_INITIAL	CHAR(1)
EMP_HIREDATE	DATE
JOB_CODE	CHAR(3)
PROJ_NUM	INT

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

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	PROJ_NUM
101	News	John	G	08-Nov-16	502	25
102	Senior	David	H	12-Jul-17	501	

- Assuming all the data in the Employee table has been entered as shown below, write the SQL code that will list all attributes for a JOB_CODE of 502.

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	PROJ_NUM
101	News	John	G	08-Nov-16	502	25
102	Senior	David	H	12-Jul-17	501	
103	Arbaugh	June	E	01-Dec-16	500	18
104	Ramoras	Anne	K	15-Nov-15	501	
105	Johnson	Alice	K	01-Feb-15	502	25
106	Smithfield	William		22-Jun-17	500	18
107	Alonzo	Maria	D	10-Oct-16	500	18

- Write the SQL code to change the job code to 501 for the person whose employee number is 107.

- Write the SQL code to delete the row of the person named William Smithfield, who was hired on 22 June 2017, whose job code is 500 and project number is 18. 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 employees who were hired before 1 January 2016 and whose job code is at least 501.

- Write the SQL code that will delete all of data inside a table, but not the table itself.

- Write the SQL code that will delete the Employee table entirely.

Rate us

Share your thoughts

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

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

Click here to share your thoughts anonymously.