Unit -3

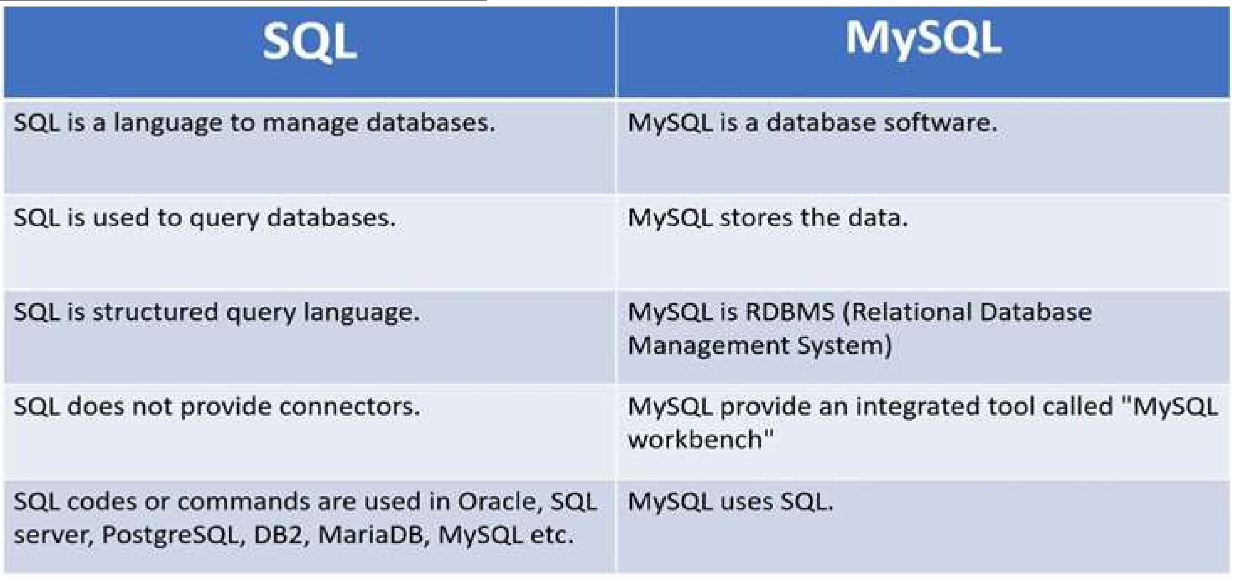
Basics of MySQL

- MySQL is a widely used relational database management system (RDBMS).
- MySQL is free and open-source.
- MySQL is ideal for both small and large applications.
- MySQL is named after the daughter of co-founder Michael Widenius whose name is "My".
- MySQL is cross-platform
- MySQL is compliant with the ANSI SQL standard
- MySQL was first released in 1995
- MySQL is developed, distributed, and supported by Oracle Corporation

Who Uses MySQL?

- Huge websites like Facebook, Twitter, Airbnb, Booking.com, Uber, GitHub, YouTube, etc.
- Content Management Systems like WordPress, Drupal, Joomla!, Contao, etc.
- A very large number of web developers around the world.

Difference between SQL & MYSQL



SQL	MySQL
SQL is a language to manage databases.	MySQL is a database software.
SQL is used to query databases.	MySQL stores the data.
SQL is structured query language.	MySQL is RDBMS (Relational Database Management System)
SQL does not provide connectors.	MySQL provide an integrated tool called "MySQL workbench"
SQL codes or commands are used in Oracle, SQL server, PostgreSQL, DB2, MariaDB, MySQL etc.	MySQL uses SQL.

Difference between DBMS & RDBMS

No.	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	**Normalization is not** present in DBMS.	**Normalization is** present in RDBMS.
4)	DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
8)	DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.
9)	Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc.

What is a Database Table?

- A table is a collection of related data entries, and it consists of columns and rows.
- A column holds specific information about every record in the table.
- A record (or row) is each individual entry that exists in a table.

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

What is a Relational Database?

A relational database defines database relationships in the form of tables. The tables are related to each other - based on data common to each.

Look at the following three tables "Customers", "Orders", and "Shippers" from the Northwind database:

Customers Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The relationship between the "Customers" table and the "Orders" table is the CustomerID column:

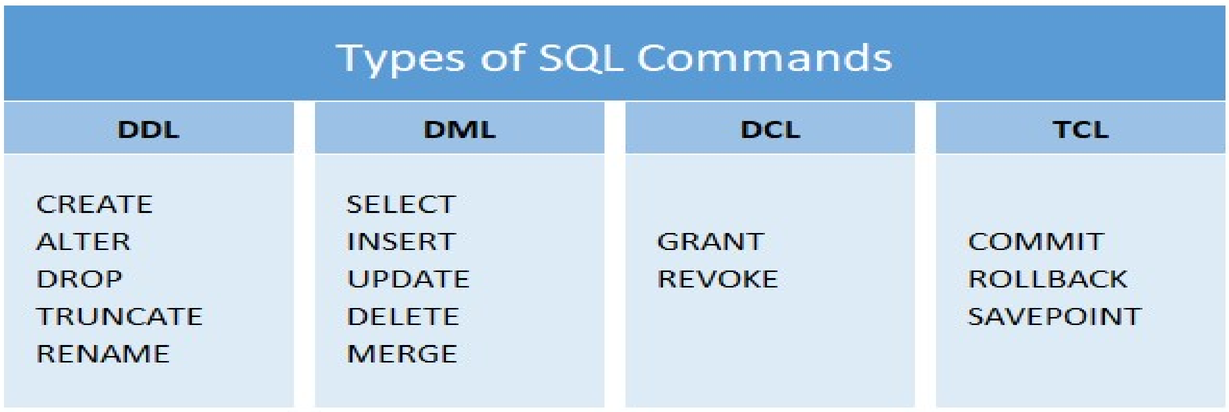
Orders Table

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10278	5	8	1996-08-12	2
10280	5	2	1996-08-14	1
10308	2	7	1996-09-18	3
10355	4	6	1996-11-15	1
10365	3	3	1996-11-27	2
10383	4	8	1996-12-16	3
10384	5	3	1996-12-16	3

The relationship between the "Orders" table and the "Shippers" table is the ShipperID column:

Shippers Table

ShipperID	ShipperName	Phone
1	Speedy Express	(503) 555-9831
2	United Package	(503) 555-3199
3	Federal Shipping	(503) 555-9931



Types of SQL Commands

DDL	DML	DCL	TCL
CREATE	SELECT		
ALTER	INSERT	GRANT	COMMIT
DROP	UPDATE	REVOKE	ROLLBACK
TRUNCATE	DELETE		SAVEPOINT
RENAME	MERGE		

<u>Some of The Most Important SQL Commands</u>

- `SELECT` - extracts data from a database
- `UPDATE` - updates data in a database
- `DELETE` - deletes data from a database
- `INSERT INTO` - inserts new data into a database
- `CREATE DATABASE` - creates a new database
- `ALTER DATABASE` - modifies a database
- `CREATE TABLE` - creates a new table
- `ALTER TABLE` - modifies a table
- `DROP TABLE` - deletes a table
- `CREATE INDEX` - creates an index (search key)
- `DROP INDEX` - deletes an index

Query Building

MySQL Query Builder is an advanced tool that facilitates the generation of MySQL queries of any complexity on a visual diagram without any SQL knowledge. Build, group, and arrange complicated tables, create JOINs, add, and update conditions in an intuitive drag-and-drop interface.

MySQL Queries

A list of commonly used MySQL queries to create database, use database, create table, insert record, update record, delete record, select record, truncate table and drop table are given below.

1. MySQL Create Database

MySQL create database is used to create database.

For example : create database db1;

2. MySQL Select/Use Database

MySQL use database is used to select database.

For example: use db1;

3. MySQL Create Query

MySQL create query is used to create a table, view, procedure and function.

For example:

```
CREATE TABLE customers
(id int(10),
 name varchar(50),
 city varchar(50),
 PRIMARY KEY (id )
);
```

4. MySQL Alter Query

MySQL alter query is used to add, modify, delete or drop colums of a table. Let's see a query to add column in customers table:

```
ALTER TABLE customers
ADD age varchar(50);
```

5. MySQL Insert Query

MySQL insert query is used to insert records into table.

For example:`insert into customers values(101,'rahul','delhi');`

6. MySQL Update Query

MySQL update query is used to update records of a table.

For example:`update customers set name='bob', city='london' where id=101;`

7. MySQL Delete Query

MySQL update query is used to delete records of a table from database.

For example:`delete from customers where id=101;`

8. MySQL Select Query

   Oracle select query is used to fetch records from database. For example:SELECT * from customers;

9. MySQL Truncate Table Query

   MySQL update query is used to truncate or remove records of a table. It doesn't remove structure. For example:truncate table customers;

10. MySQL Drop Query

    MySQL drop query is used to drop a table, view or database. It removes structure and data of a table if you drop table. For example:drop table customers;

Node.Js Create Connection with MySQL

- Node.js can be used in database applications.
- One of the most popular databases is MySQL.

MySQL Database

To be able to experiment with the code examples, you should have MySQL installed on your computer.
You can download a free MySQL database at https://www.mysql.com/downloads/.

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.

To download and install the "mysql" module, open the Command Terminal and execute the following:

```
C:\Users\Your Name>npm install mysql
```

Now you have downloaded and installed a mysql database driver.
Node.js can use this module to manipulate the MySQL database:

```
var mysql = require('mysql');
```

Create Connection (Database connectivity using NodeJS)

Start by creating a connection to the database.
Use the username and password from your MySQL database.

demo_db_connection.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

Save the code above in a file called "demo_db_connection.js" and run the file:

Run "demo_db_connection.js"

```
C:\Users\Your Name>node demo_db_connection.js
```

Which will give you this result:

```
Connected!
```

Now you can start querying the database using SQL statements.

Query a Database

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database.
The connection object created in the example above, has a method for querying the database:

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Result: " + result);
  });
});
```

The query method takes an sql statements as a parameter and returns the result.

Learn how to read, write, delete, and update a database .

A. Node.js MySQL Create Database

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

Example: Create a database named "mydb":

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Save the code above in a file called "demo_create_db.js" and run the file:

Run "demo_create_db.js"

```
C:\Users\Your Name>node demo_create_db.js
```

Which will give you this result:

```
Connected!
Database created
```

B. Node.js MySQL Create Table

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.
Make sure you define the name of the database when you create the connection:

Example:**Create a table named "customers":**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

Save the code above in a file called "demo_create_table.js" and run the file:

Run "demo_create_table.js"

```
C:\Users\Your Name>node demo_create_table.js
```

Which will give you this result:

```
Connected!
Table created
```

Primary Key

When creating a table, you should also create a column with a unique key for each record.
This can be done by defining a column as "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example:**Create primary key when creating the table:**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255),
address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

If the table already exists, use the ALTER TABLE keyword:

Example

Create primary key on an existing table:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY KEY";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table altered");
  });
});
```

C. Node.js MySQL Insert Into

Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Example:**Insert a record in the "customers" table:**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES ('Company Inc', 'Highway 37')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});
```

Save the code above in a file called "demo_db_insert.js", and run the file:

Run "demo_db_insert.js"

```
C:\Users\Your Name>node demo_db_insert.js
```

Which will give you this result:

```
Connected!
1 record inserted
```

Insert Multiple Records

To insert more than one record, make an array containing the values, and insert a question mark in the sql, which will be replaced by the value array:

INSERT INTO customers (name, address) VALUES ?

Example:**Fill the "customers" table with data:**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES ?";
  var values = [
    ['John', 'Highway 71'],
    ['Peter', 'Lowstreet 4'],
    ['Amy', 'Apple st 652'],
    ['Hannah', 'Mountain 21'],
    ['Michael', 'Valley 345'],
    ['Sandy', 'Ocean blvd 2'],
    ['Betty', 'Green Grass 1'],
   ['Richard', 'Sky st 331'],
    ['Susan', 'One way 98'],
    ['Vicky', 'Yellow Garden 2'],
    ['Ben', 'Park Lane 38'],
    ['William', 'Central st 954'],
    ['Chuck', 'Main Road 989'],
    ['Viola', 'Sideway 1633']
  ];
  con.query(sql, [values], function (err, result) {
    if (err) throw err;
    console.log("Number of records inserted: " + result.affectedRows);
  });
});
```

Save the code above in a file called "demo_db_insert_multple.js", and run the file:
Run "demo_db_insert_multiple.js"

```
C:\Users\Your Name>node demo_db_insert_multiple.js
```

Which will give you this result:

```
Connected!
Number of records inserted: 14
```

D. Node.js MySQL Select From

Selecting From a Table
To select data from a table in MySQL, use the "SELECT" statement.

Example:**Select all records from the "customers" table, and display the result object:**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

SELECT * will return *all* columns

Save the code above in a file called "demo_db_select.js" and run the file:

Run "demo_db_select.js"

```
C:\Users\Your Name>node demo_db_select.js
```

Which will give you this result:

```
[
  { id: 1, name: 'John', address: 'Highway 71'},
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},
  { id: 3, name: 'Amy', address: 'Apple st 652'},
  { id: 4, name: 'Hannah', address: 'Mountain 21'},
  { id: 5, name: 'Michael', address: 'Valley 345'},
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},
  { id: 7, name: 'Betty', address: 'Green Grass 1'},
  { id: 8, name: 'Richard', address: 'Sky st 331'},
  { id: 9, name: 'Susan', address: 'One way 98'},
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},
  { id: 11, name: 'Ben', address: 'Park Lane 38'},
  { id: 12, name: 'William', address: 'Central st 954'},
  { id: 13, name: 'Chuck', address: 'Main Road 989'},
  { id: 14, name: 'Viola', address: 'Sideway 1633'}
]
```

E. Node.js MySQL Delete

Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

Example:**Delete any record with the address "Mountain 21":**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "DELETE FROM customers WHERE address = 'Mountain 21'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Number of records deleted: " + result.affectedRows);
  });
});
```

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Save the code above in a file called "demo_db_delete.js" and run the file:

Run "demo_db_delete.js"

```
C:\Users\Your Name>node demo_db_delete.js
```

Which will give you this result:

```
Number of records deleted: 1
```

F. Node.js MySQL Drop Table

Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

Example:**Delete the table "customers":**

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "DROP TABLE customers";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table deleted");
  });
});
```

Save the code above in a file called "demo_db_drop_table.js" and run the file:

Run "demo_db_drop_table.js"

```
C:\Users\Your Name>node demo_db_drop_table.js
```

Which will give you this result:

```
Table deleted
```

G. Node.js MySQL Update

Update Table

You can update existing records in a table by using the "UPDATE" statement:

Example:Overwrite the address column from "Valley 345" to "Canyon 123":

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result.affectedRows + " record(s) updated");
  });
});
```

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Save the code above in a file called "demo_db_update.js" and run the file:

Run "demo_db_update.js"

```
C:\Users\Your Name>node demo_db_update.js
```

Which will give you this result:

```
1 record(s) updated
```

H. Node.js MySQL Limit

Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement:

Example:Select the 5 first records in the "customers" table:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT * FROM customers LIMIT 5";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

Save the code above in a file called "demo_db_limit.js" and run the file:

Run "demo_db_limit.js"

```
C:\Users\Your Name>node demo_db_limit.js
```

Which will give you this result:

```
[
  { id: 1, name: 'John', address: 'Highway 71'},
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},
  { id: 3, name: 'Amy', address: 'Apple st 652'},
  { id: 4, name: 'Hannah', address: 'Mountain 21'},
  { id: 5, name: 'Michael', address: 'Valley 345'}
]
```

I. Node.js MySQL Join

Join Two or More Tables

You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.

Consider you have a "users" table and a "products" table:

users

```
[
  { id: 1, name: 'John', favorite_product: 154},
  { id: 2, name: 'Peter', favorite_product: 154},
  { id: 3, name: 'Amy', favorite_product: 155},
  { id: 4, name: 'Hannah', favorite_product:},
  { id: 5, name: 'Michael', favorite_product:}
]
```

products

```
[
  { id: 154, name: 'Chocolate Heaven' },
  { id: 155, name: 'Tasty Lemons' },
  { id: 156, name: 'Vanilla Dreams' }
]
```

These two tables can be combined by using users' favorite_product field and products' id field.

Example:Select records with a match in both tables:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT users.name AS user, products.name AS favorite FROM users JOIN products ON
users.favorite_product = products.id";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

Note: You can use INNER JOIN instead of JOIN. They will both give you the same result.

Save the code above in a file called "demo_db_join.js" and run the file:

Run "demo_db_join.js"

```
C:\Users\Your Name>node demo_db_join.js
```

Which will give you this result:

```
[
  { user: 'John', favorite: 'Chocolate Heaven' },
  { user: 'Peter', favorite: 'Chocolate Heaven' },
  { user: 'Amy', favorite: 'Tasty Lemons' }
]
```

As you can see from the result above, only the records with a match in both tables are returned.