Databases, Express Sessions

CS 390 – Web Application Development

J. Setpal

November 15, 2023

Outline

Outline

WIWYT – Databases

- Persistent sessions!

WIWYT – Databases

- Persistent sessions! Does not require you to store the entire user data in memory.

WIWYT – Databases

- Persistent sessions! Does not require you to store the entire user data in memory.
- Scalable, reliable data updates and queries.

WIWYT – Sessions (Recap)

- Enables state-persistent client-server communication.

WIWYT – Sessions (Recap)

- Enables state-persistent client-server communication.
- Prevents unnecessary / repititive user-inputs.

Outline

What is a Database?

From Oracle[1]: An organized collection of structured information.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

They are not Javascript-native.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

They are not Javascript-native. Node modules allow us to interface using:

a. **Native Language:** Use the query language of the database through a mediator.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

They are not Javascript-native. Node modules allow us to interface using:

a. **Native Language:** Use the query language of the database through a mediator.

b. **Object Data Model (ODM) / Object Relational Model (ORM):** Create a JavaScript object of the database and interface with it like a native variable.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

They are not Javascript-native. Node modules allow us to interface using:

a. **Native Language:** Use the query language of the database through a mediator.

b. **Object Data Model (ODM) / Object Relational Model (ORM):** Create a JavaScript object of the database and interface with it like a native variable.

Today, we'll focus on using Native Language to interact with the database.

[1] creators of MySQL

What is a Database?

From Oracle[1]: An organized collection of structured information.

We can interact with them using **Database Management Systems**.

They are not Javascript-native. Node modules allow us to interface using:

a. **Native Language:** Use the query language of the database through a mediator.

b. **Object Data Model (ODM) / Object Relational Model (ORM):** Create a JavaScript object of the database and interface with it like a native variable.

Today, we'll focus on using Native Language to interact with the database.

There are two main languages we will discuss today: **SQL** and **NoSQL**.

[1] creators of MySQL

Structured Query Language (SQL)

SQL stands for **Structured Query Language**. It's a language primarily used for **relational** database management systems (RDBMS).

Structured Query Language (SQL)

SQL stands for **Structured Query Language**. It's a language primarily used for **relational** database management systems (RDBMS).

Idea: Everything is stored in tables.

Structured Query Language (SQL)

SQL stands for **Structured Query Language**. It's a language primarily used for **relational** database management systems (RDBMS).

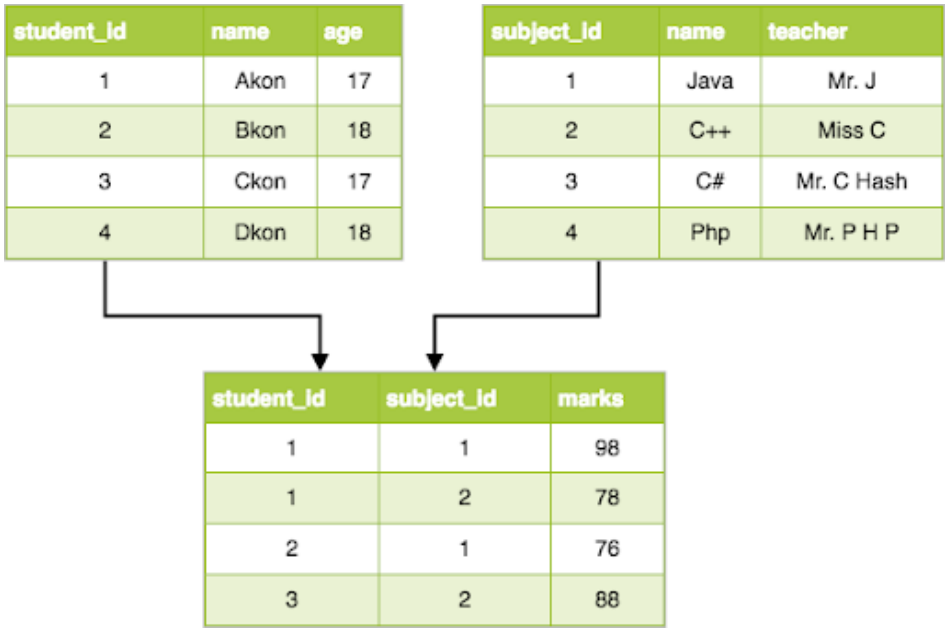
Idea: Everything is stored in tables. Rows represent *data samples*, columns represent *properties*.

Structured Query Language (SQL)

SQL stands for **Structured Query Language**. It's a language primarily used for **relational** database management systems (RDBMS).

Idea: Everything is stored in tables. Rows represent *data samples*, columns represent *properties*.

This database is structured to store **relationships between properties**.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

MariaDB – Setup

Post installation:

```
# mariadb-install-db --user=mysql --basedir=/usr  \
--datadir=/var/lib/mysql
```

initializes the database.

MariaDB – Setup

Post installation:

```
# mariadb-install-db --user=mysql --basedir=/usr  \
--datadir=/var/lib/mysql
```

initializes the database.

Next, we run the daemon:

```
# mysqld_safe / mariadb-safe
```

MariaDB – Setup

Post installation:

```
# mariadb-install-db --user=mysql --basedir=/usr  \
--datadir=/var/lib/mysql
```

initializes the database.

Next, we run the daemon:

```
# mysqld_safe / mariadb-safe
```

Finally, we setup the user:

```
# mariadb
MariaDB> CREATE USER ‘cs390’@‘localhost’ IDENTIFIED BY  \
‘password’;
MariaDB> GRANT ALL PRIVILEGES ON *.* TO ‘cs390’@‘localhost’;
```

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

We then select the database we want to work with, and then can run SQL commands:

1. **Creations:** `CREATE <table>(<schema>);`

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

We then select the database we want to work with, and then can run SQL commands:

1. **Creations:** `CREATE <table>(<schema>);`
2. **Insertions:** `INSERT INTO <table> VALUES(<val_1>, <val_2>, ..., <val_n>);`

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

We then select the database we want to work with, and then can run SQL commands:

1. **Creations:** `CREATE <table>(<schema>);`
2. **Insertions:** `INSERT INTO <table> VALUES(<val_1>, <val_2>, ..., <val_n>);`
3. **Reads:** `SELECT <columns> from <table> WHERE <condition>;`

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

We then select the database we want to work with, and then can run SQL commands:

1. **Creations:** `CREATE <table>(<schema>);`
2. **Insertions:** `INSERT INTO <table> VALUES(<val_1>, <val_2>, ..., <val_n>);`
3. **Reads:** `SELECT <columns> from <table> WHERE <condition>;`
4. **Deletions:** `DROP <tables>;`

MySQL – Data Operations

From here, we can log into the user using `mariadb -u cs390 -p` and enter the password.

We then select the database we want to work with, and then can run SQL commands:

1. **Creations:** `CREATE <table>(<schema>);`
2. **Insertions:** `INSERT INTO <table> VALUES(<val_1>, <val_2>, ..., <val_n>);`
3. **Reads:** `SELECT <columns> from <table> WHERE <condition>;`
4. **Deletions:** `DROP <tables>;`
5. **Lists:** `SHOW tables;`

Integration with Express

We can integrate a MySQL database with Node using the `mysql2` module.

Integration with Express

We can integrate a MySQL database with Node using the `mysql2` module.

Within our application, we setup a database connection, and query the connection.

Integration with Express

We can integrate a MySQL database with Node using the `mysql2` module.

Within our application, we setup a database connection, and query the connection.

Important:

Queries made to the server are **asynchronous**, and returned as promises.

Let's Setup an SQL Database! (+ attendance)

If you can view this screen, I am making a mistake.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**. DBMSs uses various ways to define structure: NoSQL uses **documents**.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**. DBMSs uses various ways to define structure: NoSQL uses **documents**.

Each item / record is a document, stored as JSON / BSON / XML.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**. DBMSs uses various ways to define structure: NoSQL uses **documents**.

Each item / record is a document, stored as JSON / BSON / XML. These are grouped into **collections**.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**. DBMSs uses various ways to define structure: NoSQL uses **documents**.

Each item / record is a document, stored as JSON / BSON / XML. These are grouped into **collections**.

NoSQL also has a **flexible schema**.

Not Only SQL (NoSQL)

NoSQL stands for **Not Only SQL**. DBMSs uses various ways to define structure: NoSQL uses **documents**.

Each item / record is a document, stored as JSON / BSON / XML. These are grouped into **collections**.

NoSQL also has a **flexible schema**. This results in records being localized, making *reads* incredibly fast at scale.

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`
4. **Deletion:** `collection.findOneAndDelete(filterObj);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`
4. **Deletion:** `collection.findOneAndDelete(filterObj);`
5. **Updates:** `collection.findOneAndUpdate(filterObj, updateFieldsValues);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`
4. **Deletion:** `collection.findOneAndDelete(filterObj);`
5. **Updates:** `collection.findOneAndUpdate(filterObj, updateFieldsValues);`
6. **Replaces:** `collection.findOneAndReplace(filterObj, replacementItem);`

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`
4. **Deletion:** `collection.findOneAndDelete(filterObj);`
5. **Updates:** `collection.findOneAndUpdate(filterObj, updateFieldsValues);`
6. **Replaces:** `collection.findOneAndReplace(filterObj, replacementItem);`

MongoDB's lack of schema means that failures are *not obvious*.

MongoDB – Data Operations

The setup process is relatively straightforward: `# mongod --dbpath data/ ; mongosh`

Useful methods:

1. **Find One:** `collection.findOne(filterObj);`
2. **Find All:** `collection.find(filterObj);`
3. **Insertion:** `collection.insertOne(newItem);`
4. **Deletion:** `collection.findOneAndDelete(filterObj);`
5. **Updates:** `collection.findOneAndUpdate(filterObj, updateFieldsValues);`
6. **Replaces:** `collection.findOneAndReplace(filterObj, replacementItem);`

MongoDB's lack of schema means that failures are *not obvious*.
Mongoose introduces optional schema to correct this.

Outline

Sessions – Recap

Cookies have a significant vulnerability: they are visible to the client.

Sessions – Recap

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

Sessions – Recap

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

This exposes sensitive information; both to the client and malicious actors.

Sessions – Recap

Cookies have a significant vulnerability: they are visible to the client. They also have **size limitations**.

This exposes sensitive information; both to the client and malicious actors.

Sessions solve this! The relevant information is stored **server-side**, with an identifier cookie that enables the client-server association.

Session Store

Sessions are not persistent by default. When the node server is restarted, the server is reset.

Session Store

Sessions are not persistent by default. When the node server is restarted, the server is reset.

We can setup persistence using a **Session Store**.

Session Store

Sessions are not persistent by default. When the node server is restarted, the server is reset.

We can setup persistence using a **Session Store**. The default implementation of `MemoryStore` includes persistence, but is not meant for production.

Session Store

Sessions are not persistent by default. When the node server is restarted, the server is reset.

We can setup persistence using a **Session Store**. The default implementation of `MemoryStore` includes persistence, but is not meant for production.

It only runs a *single* thread, *leaks* memory, and does *not* scale well.

Session Store

Sessions are not persistent by default. When the node server is restarted, the server is reset.

We can setup persistence using a **Session Store**. The default implementation of `MemoryStore` includes persistence, but is not meant for production.

It only runs a *single* thread, *leaks* memory, and does *not* scale well.

Instead, we can use **MongoDB** as the session store using `connect-mongo`.

Let's Setup a NoSQL Database!

If you can view this screen, I am making a mistake.

Outline

Thank you!

Have an awesome rest of your day!

Slides:

`https://cs.purdue.edu/homes/jsetpal/slides/databases.pdf`

If anything's incorrect or unclear, please ping `jsetpal@purdue.edu`
I'll patch it ASAP.

Note:

No class next Monday. Enjoy the Thanksgiving break!