MCOMD3ADB

Assignment 2 – Advanced database and big data

Audrey Smith and Ben Fowler

GitHub: https://github.com/BF118/DatabaseAssignment
Video: https://streamable.com/nezkog

Contribution: Audrey worked on the front end of the application and the demo video. Ben worked on the creation/insertion data and scripts.

Report Background

The intent of this report is to discuss the process and results of the implementation of a PHP and MySQL application using XAMPP. The data presented was a schema showing a department table and an employee table. See the image below for the given schema.

EMP (EMPNO, ENAME, JOB, MGR#, HIREDATE, SAL, COMM, DEPTNO#)

DEPT (DEPTNO, DNAME, LOC)

Using this schema, a database will be created with two tables that correspond to each of the two presented. EMP is an Employee table that shows relevant information about each employee within the company, including their number, name, their manager's number, their hire date, salary, commission, and department number. The second table, DEPT, is the Department table, which has the department number, department name, and location. These tables are connected by the department number, which acts as a primary key for the Department table and is referenced by the Employee table.

The following sections of the report will detail how the creation and insertion data was written and formatted, as well as the PHP page that inserts data and runs SQL queries.

Database Preparation (backend)

To start with this project the database needs to be set up with the tables that are needed in order for the data from the front end to be stored into it. For this there needs to be 2 tables: one for Departments and the other for Employees. These are the tables that are going to be populated from the front end of the project.

The first table that needs to be created is the Department table, and in this case it'll be shortened to DEPT. In this table there are three attributes, DEPTNO, DNAME and LOC. Starting with DEPTNO, this is going to be the primary key for this table. This attribute needs a name, attribute type and whether it can be null or not. In this case the name would DEPTNO, the attribute would be int and it will be not null. As this is the primary key for the table it has to be no number as a department needs to have a number associated with it. Next is DNAME, which is the column that'll include the names of all the departments in the business. This is similar to the DEPTNO with it being NOT null but its attribute type is VARCHAR(20) instead of int. VARCHAR(20) was used as it allows for much longer department names if they are added in the future. Lastly is LOC which has the same attribute type to DNAME and also can't be null.

```
CREATE TABLE DEPT
(
    DEPTNO int NOT NULL,
    DNAME varchar(20) NOT NULL,
    LOC varchar(20) NOT NULL,
    PRIMARY KEY(DEPTNO)
 );
```

With the DEPT table created, the next thing to do is make the EMP. This table is a bit more complicated with it having foreign keys in it as well as more attributes. Firstly, the primary key which will be EMPNO with int as its attribute type and it being not null. The next two columns are for ENAME and JOB, both of which have attributes as VARCHAR(20) having a limit on 20 characters allows for any rather large names or jobs but isn't too big where it could impact on the performance of the database. These are also not null. The next column is MGR which just has an attribute type of int and can be null. This will also be one of the foreign keys for the table. To set up the foreign key it needs to have a reference, which in this case will be EMPNO from the EMP table. After this is the HIREDATE which has the attribute type of date, this again can not be null as everyone will have a date in which they are hired. The last two normal attributes in this table are SAL and COMM, which both have attribute types of decimal as they are dealing with money. In terms of being not null for these columns it is only SAL as

someone who is employed has to be paid. Finally is the foreign key for DEPTNO. For this, there is a reference to the DEPTNO in the department table which will put the DEPTNO data from the DEPT table into this table.

```
CREATE TABLE EMP
(
    EMPNO        int NOT NULL,
    ENAME    varchar(20) NOT NULL,
    JOB      varchar(20) NOT NULL,
    MGR          int             ,
    FOREIGN KEY (MGR) REFERENCES EMP(EMPNO),
    HIREDATE DATE        NOT NULL,
    SAL      DECIMAL     NOT NULL,
    COMM     DECIMAL     ,
    DEPTNO       int             ,
    PRIMARY KEY(EMPNO),
    FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO)
    );
```

Inserting Data:

With the database tables created, the data that has been provided needs to be inserted into the database and populate the right attributes with the right data. The data needs to be formatted so that it can be inserted.

Starting with the department table, there are three attributes in this table. There are two different attribute types being varchar and int, both of which need to be formatted slightly differently. To get the data into the table, the script needs to know which table to input the data into. In this case it'll be DEPT. From here the column names are then listed in brackets. The command VALUES is used, then the data which in this case for the first value would be formatted as (10, 'ACCOUNTING', 'NEW-YORK'). For anything that uses varchar, quotation marks have to be used for it to be inserted properly.

```
INSERT INTO DEPT(DEPTNO, DNAME, LOC)
VALUES
(10, 'ACCOUNTING', 'NEW-YORK'),
(20, 'RESEARCH', 'DALLAS'),
(30, 'SALES', 'CHICAGO'),
(40, 'OPERATIONS', 'BOSTON')
```

There is also the Employee table that needs to have its data inserted into. This script is a bit more complicated than the database one as it has a lot more attributes with varying

attribute types, meaning some data has to be inserted in different ways, similarly to the department table. This table has additional attribute types being date and decimal. To insert “date” into the table, the formatting has to be specific. The layout for this for inputting values is ‘YYYY-MM-DD’, while for decimal it can be any number but if it has no decimal value it has to be set to .00. Like before, to insert the data the table and attributes need to be selected before the command VALUES then the data that is wanted. The formatting for this is each column value separated by a value; anything that is varchar and date needs to have quotation marks before it. Inserting data will look like the figure below.

```
INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO)
VALUES
(7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800.00, NULL, 20),
(7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600.00, 300.00,30),
(7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250.00, 500.00, 30),
(7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975.00, NULL, 20),
(7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250.00, 1400.00, 30),
(7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850.00, NULL, 30),
(7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450.00, NULL, 10),
(7839, 'KING', 'PRESIDENT', NULL, '1981-11-08', 5000.000, NULL, 10),
(7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08' , 1500.00, 0.00, 30),
(7876,'ADAMS', 'CLERK', 7788, '1987-09-23', 1100.00, NULL, 20),
(7900, 'JAMES','CLERK', 7698, '1981-12-03', 950.00, NULL, 30),
(7902, 'FORD','ANALYST', 7566, '1981-12-03', 3000.00, NULL, 20),
(7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300.00, NULL, 10)
```

Technique (frontend)

The front end of the application consists of a PHP page that has two main sections. First, however, a connection needs to be created to the database in the backend. PHP Data Objects, or PDO, was used to create this connection. The database properties are set using variables $dsn (data source name), $user, and $pass. There is no username or password, so these were left as ‘root’ and blank, respectively. Next, the connection variable was created using these properties. See the image below for the code.

```
// Set database properties
$dsn = 'mysql:host=localhost;dbname=assignment-db';
$user = 'root';
$pass = '';

// Create connection
$connection = new PDO($dsn, $user, $pass);
```

The first section of the page is the insertion of a new entry into each table. The Department table insertion came first because the new entry in the Employee table will need to have the department number ready to reference. The insertion scripts can be seen below.

```
// Insert into DEPT
echo "<h2>Inserting into DEPT table:</h2>";

$sql = "INSERT INTO DEPT(DEPTNO, DNAME, LOC) VALUES (50, 'HUMAN-RESOURCES', 'LONDON')";
$result = $connection->prepare($sql);
$result->execute();
if ($result) {
        echo "<p>New values inserted into dept table.</p>";
}
else {
        echo "<p>Error inserting new values into dept table.</p>";
}


// Insert into EMP
echo "<h2>Inserting into EMP table:</h2>";

$sql = "INSERT INTO EMP(EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO) VALUES (7741, 'JONES', 'SUPERVISOR', 7839, '1985-11-04', 3000.00, NULL, 50)";
$result = $connection->prepare($sql);
$result->execute();
if ($result) {
        echo "<p>New values inserted into emp table.</p>";
}
else {
        echo "<p>Error inserting new values into emp table.</p>";
}
```

The insertion code for each table is highly similar to each other. A variable, $sql, is created to house the string of SQL that will be sent to the database. The connection variable is instructed to prepare and execute $sql, and then the result is displayed to the user. The data inserted into the Department table is a Human Resources department located in London with the department number 50. In the Employee table, the new entry is an employee named Jones with number 7741 with the role Supervisor, managed by employee #7839, born 4th November 1985, salary of £3000, no commission, and in the department number 50.

The next section in the PHP page is the three queries. Each query is written into a variable and executed, similar to the insertion SQL. The first is to name all employees who work in sales. Name and job are selected from the Employee table on the condition that the associated department number correlates to “SALES” in the Department table.

```
// Query 1
echo "<h2>Query 1:</h2>";
echo "<h3>Name all employees who work in sales.</h3>";

$query1 = "SELECT ENAME AS name, JOB AS job FROM EMP E INNER JOIN DEPT D ON D.DEPTNO=E.DEPTNO WHERE D.DNAME='SALES'";
$result1 = $connection->prepare($query1);
$result1->execute();
while ($line = $result1->fetch(PDO::FETCH_ASSOC)) {
        echo "<p>" . $line['name'] . " the " . $line['job'] . "</p>";
}
```

The second query shows the name and number of each employee who has a salary over £2000. The query selects the number, name, job, and salary, all from the Employee table, and checks that the salary is greater than 2000.

```
// Query 2
echo "<h2>Query 2:</h2>";
echo "<h3>Show employees' names and employee numbers who have a salary over £2000.</h3>";

$query2 = "SELECT EMPNO AS empno, ENAME AS name, JOB AS job, SAL AS salary FROM EMP WHERE SAL>2000";
$result2 = $connection->prepare($query2);
$result2->execute();
while ($line = $result2->fetch(PDO::FETCH_ASSOC)) {
        echo "<p>" . $line['empno'] . " " . $line['name'] . " the " . $line['job'] . " makes £" . $line['salary'] . "</p>";
}
```

The third and final query shows the number of employees that work in the Dallas department. From the Employee table, the name of the employee is inside of a COUNT() function. Instead of showing each name, it will just show the total number of entries. The location is selected from the Department table, and the condition is that the location is “DALLAS”.

```
// Query 3
echo "<h2>Query 3:</h2>";
echo "<h3>How many employees work in the Dallas department?</h3>";

$query3 = "SELECT COUNT(E.ENAME) AS name, D.LOC AS location FROM EMP E INNER JOIN DEPT D ON D.DEPTNO=E.DEPTNO WHERE D.LOC='DALLAS'";
$result3 = $connection->prepare($query3);
$result3->execute();
while ($line = $result3->fetch(PDO::FETCH_ASSOC)) {
        echo "<p>" . $line['name'] . " employees work in " . $line['location'] . "</p>";
}
```

At the end of the PHP page, as a safety measure the connection is closed.

Reflection

What are the limitations of the relational model in handling big data? Do you think the noSQL models and technologies have succeeded in overcoming those limitations?

Relational databases are used for many applications and have a range of benefits that can make them an excellent choice for storing data for example, ease of use user can easily access and receive their data in seconds, they are very accurate as they are strictly defined meaning there isn't any duplication of data and finally they are very secure with only authorized users being able to access data. However relational databases have their downsides when it comes to handling big data. As the number of tables and columns grow the performance of the system will decrease over time as they begin to fill with data, this can cause slow query times or even complete failures depending on the number of users accessing it at that time. With more and more data being put into a relational database there gets to a point where all those rows and columns take up a considerable space in terms of physical storage, as more data is added this storage begins to fill up which requires expansion not only to the storage but also the memory of the system.

Relation databases aren’t the only way to store data, noSQL models such as document databases and key value databases can be used to store large amounts of data in some way better than a relational database. They have a large amount of scalability without having as much of a performance impact over relational databases. In terms of big data they can handle large volumes of data at high speeds with them also being able to handle lots of queries with quick response times. With them being scalable, noSQL databases are high in flexibility and can adapt to any requirement changes as their schemas aren’t fixed, this can allow for easier development in the long run.

Comparing noSQL models to relational models, noSQL has developed to overcome the issues that relational models have in terms of scalability, flexibility and performance, however they are limited in terms of the storage that they take up when there is lots of data with noSQL being worse for it as there isn’t anything in place for duplicated data they are usually built for query speed.

References:

Advantages Of noSQL [WWW Document], n.d. . MongoDB. URL https://www.mongodb.com/noSQL-explained/advantages (accessed 5.13.22).

noSQL Vs SQL Databases [WWW Document], n.d. . MongoDB. URL https://www.mongodb.com/noSQL-explained/noSQL-vs-sql (accessed 5.13.22).

Relational Database Benefits and Limitations (Advantages & Disadvantages) - DatabaseTown, 2021. URL https://databasetown.com/relational-database-benefits-and-limitations/ (accessed 5.13.22).