

ASSIGNMENT COVERSHEET

Student Name: Andrew Mitchell Mullen (W9304694)	
Class: Advanced Databases	
Assignment: 1	
Lecturer: Patrick Scherer	Semester: 1904
Due Date: 1/5/2020	Actual Submission Date: 29/4/2020

Evidence Produced (List separate items)	Location (Choose one)	
	Y	Uploaded to the Learning Center (Moodle)
		Submitted to reception

Note: Email submissions to the lecturer are not valid.

Student Declaration	
I declare that the work contained in this assignment was researched and prepared by me, except where acknowledgement of sources is made. I understand that the college can and will test any work submitted by me for plagiarism. Note: The attachment of this statement on any electronically submitted assignments will be deemed to have the same authority as a signed statement	
Date: 5/4/2020	Student Signature: Mullen

A separate feedback sheet will be returned to you after your work has been graded.
Refer to your Student Manual for the Appeals Procedure if you have concerns about the grading decision.

Student Comment (Optional)
Was the task clear? If not, how could it be improved?
Was there sufficient time to complete the task? If not, how much time should be allowed?
Did you need additional assistance with the assignment?
Was the lecturer able to help you?
Were there sufficient resources available?
How could the assignment be improved?
<i>For further comments, please use the reverse of this page.</i>

Advanced Database Systems (FT)

Andrew Mitchell Mullen (W9303332)

Prague College

Advanced Database Systems

Patrick Scherer, David Petryca

Technical Report

May 1, 2020

Semester 2001

Word Count: 3704

Table of Contents

1	Introduction	4
2	Design and Implement a Database in MySQL.....	4
2.1	Requirements of a Relational Database.....	4
2.2	Database Implementation	4
3	SQL Programming	6
4	Advanced Features of MySQL.....	8
4.1	Transactions	8
4.2	Triggers.....	8
4.3	Users and Permissions	9
4.4	PHP Functionality	10
5	A Research Topic.....	12
5.1	SQL vs NoSQL.....	12
5.1.1	Normalization and Flexibility	13
5.1.2	Scalability	13
5.1.3	Structure	13
5.1.4	Performance	14
5.1.5	Development Experience.....	14
5.2	CAP Theorem.....	14
5.3	Table of Differences	15
5.4	Summary	16
6	Critical Review.....	17
6.1	Design	17
6.2	Queries	17
6.3	Documentation	17
7	Self-Improvement Evaluation	17
	References	19

List of Figures

Figure 1: dbdiagram.io Building Design	5
Figure 2: System Diagram	6
Figure 3: Child Events View.....	7
Figure 4: Child Attendance Procedure.....	7
Figure 5: Trigger/Transaction.....	9
Figure 6: Users	10
Figure 7: PHP Initialization	11
Figure 8: PHP Implementation.....	11
Figure 9: Rendered PHP	12

List of Tables

Table 1: SQL vs NoSQL	16
-----------------------------	----

1 Introduction

This report will show how the author created and implemented a database. It will show the advanced features of a database server software and will contain research about different database solutions.

The database that will be worked on is called “skolka”. It will hold information about children, guardians, teachers, events, classes, and attendance. The tables will be connected to each other so that data is normalized and can be easily accessed.

2 Design and Implement a Database in MySQL

2.1 Requirements of a Relational Database

“The system should hold information about children and their parents or legal custodians. For each child, the system must contain the information about a class the child attends (classes are identified by numbers, 1, 2, 3 etc.) and whether the child stays by default for the afternoon or leaves after lunch. Each class is supervised by two teachers which take turns in days depending on a schedule which is imposed on a per-week basis (for instance teacher 1 has Monday and Wednesday, teacher 2 has Thursday and Friday). The information about the days when the teacher 1 or 2 has their duty should be recorded in the system. The information about staying for the afternoon should be updatable if parents call and change the default arrangement or wish to excuse their children because they are ill. From time to time the kindergarten plans social events such as theatre visits or swimming pool outings; it should be possible to create such an event and register students for the event and indicate whether they paid an event fee.” (Prague College, 2020)

2.2 Database Implementation

The database design was first implemented on paper as a rough draft. Then it was implemented on a web application tool called “dbdiagram.io”. This allowed for easy small incremental changes for design improvements. A figure of the how the diagram looked when building the design can be found below.



Figure 1: dbdiagram.io Building Design

Dbdiagram.io also allows the database design to be exported into MySQL as an SQL file. The given file was executed in the Kindergarten database that will be used for this project. Bellow you can find what the database looks like after being implemented on MySQL.

Building it this way helped ensure that the implemented database is normalized. For following normalization there is a set of best practices for avoiding certain anomalies known which is known as Normal Forms. The Normal Form that our database design needs to achieve is the Boyce-Codd Normal Form. To understand this the 1st, 2nd, and 3rd normal forms must be discussed as well.

- The **First Normal Form (1NF)** addresses the structure of a given table.
- The **Second (2NF)**, **Third (3NF)** and **Boyce-Codd Normal Forms** (it is also known as the 3.5NF) address, “one to one” and “many to one” relationships between tables. (Janert, 2003)

To follow 1NF all values must be atomic, and columns must have unique names. For a table to be in the 2NF it should not have partial dependency. This happens when a non-prime attribute

is functionally reliant on a section of a candidate key. For a table to be in the 3NF the design must not have any transitive dependencies. This is when attributes in a table create an indirect relationship. BCNF does not allow non-prime attributes to derive a prime attribute. (Janert, 2003) The design was implemented satisfying these requirements and a figure below proves it by showing the final database design.

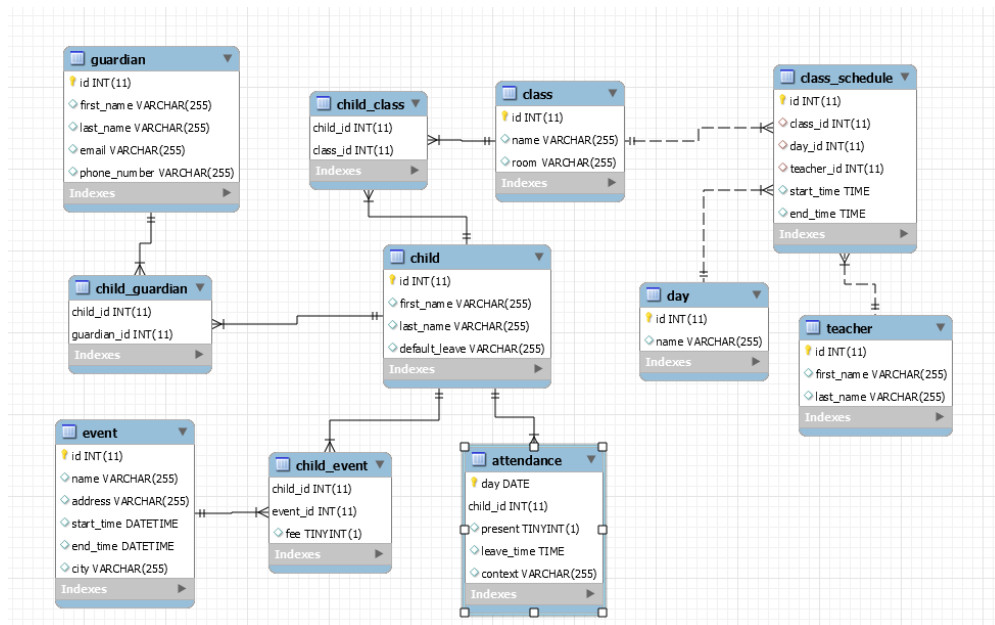


Figure 2: System Diagram

Data was added for each table using a web application that generates SQL insert queries to add data to tables. The focus of that data is not to be realistic but to prove that the database design works. For example, a child might have seven parents or a teacher's class might go from 5PM to 2AM. This does not mean the data is useless, it is simply there for practical use of observing if the database behaves in the correct manner.

3 SQL Programming

Seven queries were created for the given database. All the queries were turned into views and stored procedures. Here are some figures as proof:

	child name	child surname	event
1	Astra	Walters	felis,at
2	Bell	Cabrera	Nunc
3	Billy	Johnson	Orientation
4	Blair	White	mauris
5	Bob	Johnson	Orientation
6	Buckminster	Odom	mauris,vitae,
7	Castor	Dominguez	Fusce
8	Cedric	Bruce	magna.
9	Dieter	Rasmussen	vestibulum.
10	Donovan	Sykes	mauris
11	Garth	Miller	vestibulum.
12	Gil	Foster	Fusce
13	Hilel	George	hendrerit
14	Hiroko	Navarro	Nam
15	Indigo	Whitehead	mauris,fermentum

```

[2020-04-29 09:12:21] Connected
skoika> use skolka
[2020-04-29 09:12:21] completed in 6 ms
skoika> SELECT t.*
          FROM skolka.child_events t
          LIMIT 501
[2020-04-29 09:12:22] 33 rows retrieved starting from

```

Figure 3: Child Events View

```

# counts the number of days a child was at the kindergarten based off of the child's ID
START TRANSACTION ;
CREATE PROCEDURE child_attendance (
  in id_child int(10)
)
SELECT child.first_name AS 'child', count(attendance.present) AS 'days present'
FROM child, attendance
where  child.id = id_child
and child.id = attendance.child_id
group by (child.id);

CALL child_attendance( id_child: 20);
COMMIT ;

```

Output		Result 8	
1 row			
child	days present		
1 Scarlet	2		

Figure 4: Child Attendance Procedure

All the queries and their results can be found in the “SQL_programming” folder of this project.

4 Advanced Features of MySQL

4.1 Transactions

Transactions are used for testing/implementing SQL queries in a fail safe manner. The way it works is that you start with running the “START TRANSACTION” command in the database. After this command any change to the database can be reverted or committed with the commands “ROLLBACK” or “COMMIT”. (Teesside University, 2010) This allows the database developer to test queries to the database with minimized risk. Common practice is to run the transaction command, then run the query you are testing followed by queries to check that the change was successful.

Save points can also be set within a transaction, so that anything added after it can be rolled back. This is useful because it prevents the whole command from being rolled back which allows for a better development experience. The commands used for save points include “SAVE TRANSACTION savepoint_name” and “ROLLBACK TRANSACTION savepoint_name”. (MySQL, 2020a)

After implementing the design into the database transactions were used for editing tables structure, testing triggers, and updating data. The next section will show an example of a transaction that was used on the database.

4.2 Triggers

A trigger is a defined SQL query that is executed when a specific operation is performed on the database. A trigger can be executed with the use of the following statements “INSERT”, “UPDATE”, and “DELETE” on a given table. (MySQL, 2020b) Triggers are useful for automating actions in response to certain queries. For example, in the case of our “skolka” database the following transaction was implemented.

```

START TRANSACTION ;
DELIMITER $$
CREATE
    TRIGGER orientation_meeting AFTER INSERT
    on child
    FOR EACH ROW BEGIN
        INSERT INTO child_event (child_id, event_id, fee) VALUES(NEW.id, 21, 1);
    END $$
DELIMITER ;

INSERT INTO child (first_name, last_name) VALUES ('Billy', 'Johnson');
ROLLBACK ;|
COMMIT ;

```

Figure 5: Trigger/Transaction

For this trigger when a child is created it automatically adds the child into the orientation meeting/event for that semester and marks the fee as paid (every new semester the event id must be updated). Otherwise every new child would have to manually added to the orientation meeting event. The trigger was also added into a transaction to test and see if the trigger works by adding a new child. Since it the trigger was successful after testing it with the child insert, the trigger transaction was committed.

4.3 Users and Permissions

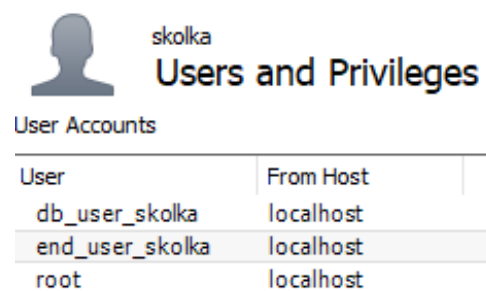
MySQL allows for user creation with different privileges. Depending on which privileges a user is assigned it determines what queries the user can perform. Generally, there are three levels of privileges: administrative, database designer/maintainer, and database collaborator. (MySQL, 2020a) Implementing these users in extremely important for the database in terms of security and preventing unnecessary mistakes. Here are some commands used for managing users and what they do:

- CREATE USER: creates user, the username, password, and host also must be specified.
- GRANT/REVOKE: grants or revokes privileges from a specific user.

Users with administrative privileges can perform almost every if not every query, this includes creating other users/database. The database/maintainer privileges are assigned to user developing and working with a given database. These users can create and edit tables, triggers,

routes, procedures, and views. Lastly there are the database collaborator users. These users are the ones working with the database by performing select, insert, update, and delete queries on their given database. A common use case for this kind of user is that it is given to the backend for interaction with the database.

The three tiers of users were implemented into the project with their given privileges listed above. The users were created and assigned privileges using MySQL Workbench. A figure of the given implemented users can be found bellow.

The image shows a screenshot of the 'Users and Privileges' window in MySQL Workbench. At the top, there is a user icon and the text 'skolka Users and Privileges'. Below this, the title 'User Accounts' is displayed. A table lists the user accounts and their host information.

User	From Host
db_user_skolka	localhost
end_user_skolka	localhost
root	localhost

Figure 6: Users

4.4 PHP Functionality

MySQL has been extremely popular with PHP for several reasons one being server bundles (are used for development). Server bundles such as MAMP, LAMP, and XAMPP include both PHP and with MySQL. This has made the two technologies an infamous combination in the sphere of web development. (phptpoint, n.d.)

PHP comes with seventy-four different MySQLi functions. (w3schools, 2020) These functions include features like connecting the database, checking for connection errors, query execution, query error messages, getting the number of affected rows and so on. Common uses of PHP with MySQL are backend, setting up a testing environment, and managing MySQL. With PHP you can create a database and insert, update, delete, and select data. PHP can even be used to manage users.

The implemented PHP for this project shows how PHP connects to MySQL which can be seen in the figure bellow.

```

1  <?php
2  $servername = "localhost";
3  $username = "end_user_skolka";
4  $password = "7~?f,nCsE!QBvU,$<C32";
5  $dbname = "skolka";
6
7  // Create connection
8  $conn = new mysqli($servername, $username, $password, $dbname);
9
10 // Check connection
11 if ($conn->connection_error) {
12     die("Connection failed: " . mysqli_connect_error());
13 }
14 echo "Connected successfully";
15

```

Figure 7: PHP Initialization

Then it executes a query and displays it on localhost for the browser. The implemented PHP uses the “end_user_skolka” collaborator user due to its limited permissions. The limited permissions provide enough access for the PHP to operate but prevent the good amount of damage as SQL injection could cause. Below are two figures showing how PHP can work with MySQL.

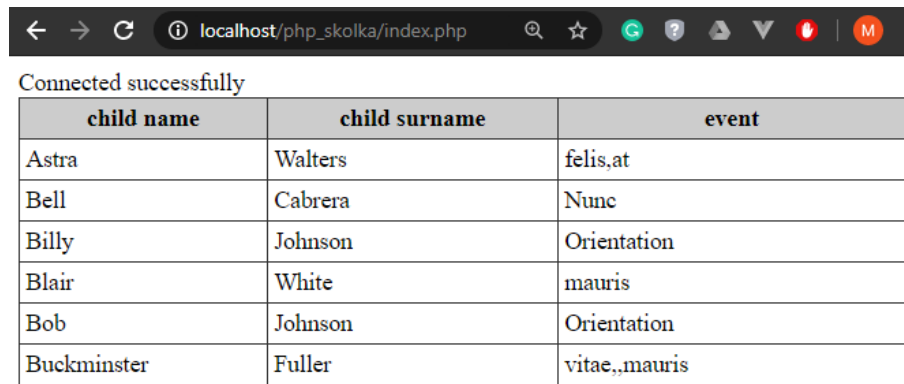
```

16 function select($sql) {
17
18     global $conn;
19
20     if ($result =
21         $conn->query($sql)) {
22
23         echo "<table style='border-collapse:collapse;width:100%;'\>\n";
24
25         $first = $result->fetch_assoc();
26
27         echo "<tr>\n";
28         foreach ($first as $key => $value) {
29             echo "<th style='border:1px solid #333;padding: 4px;background:#ccc;'\>$key</th>\n";
30         }
31         echo "</tr>\n";
32
33         $result->data_seek(0);
34
35         while ($row = $result->fetch_assoc()) {
36             echo "<tr>\n";
37             foreach ($row as $key => $value) {
38                 echo "<td style='border:1px solid #333;padding: 4px;'\>$value</td>\n";
39             }
40             echo "</tr>\n";
41         }
42         echo "</table>\n";
43     } elseif ($conn->error) {
44         die("Query error: " . $conn->error);
45     } else {
46         echo 'No result';
47     }
48 }
49
50 }
51
52 $sql = "SELECT child.first_name AS 'child name', child.last_name AS 'child surname', GROUP_CONCAT(event.name) AS 'event'
53 FROM child
54 JOIN child_event ON child.id = child_event.child_id
55 JOIN event ON event.id = child_event.event_id
56 GROUP BY (child.first_name)";
57 select($sql);
58

```

Figure 8: PHP Implementation

The following code above takes an SQL query and creates a table in the html with the given data inside. A stored procedure could have been called but the query was left in so that the reader can see what is exactly selected. The following figure shows the rendered result of the query in the PHP code.



The screenshot shows a web browser window with the address bar displaying 'localhost/php_skolka/index.php'. Below the address bar, the text 'Connected successfully' is visible. A table with three columns is displayed: 'child name', 'child surname', and 'event'. The table contains six rows of data.

child name	child surname	event
Astra	Walters	felis,at
Bell	Cabrera	Nunc
Billy	Johnson	Orientation
Blair	White	mauris
Bob	Johnson	Orientation
Buckminster	Fuller	vitae,,mauris

Figure 9: Rendered PHP

When using PHP as a backend such as an MVC framework with MySQL, some security issues could be exploited. If there is an input that a user can interact with, a hacker can inject SQL into it, thus manipulating the query commands allowing them to wreak havoc on your database. That is why the database user with the lowest privileges is given to the PHP code. Measure need to be taken to filter out SQL injections. The password of the users should be saved as a system variable, so in the case that a hacker gets the PHP code, they cannot access the database. Another important thing to implement in PHP for security with MySQL is hashing certain data such as passwords. This is to protect your user's information in case a hacker somehow obtains the data.

5 A Research Topic

5.1 SQL vs NoSQL

Structured Query Language (SQL) is an established programming language that is used for relational databases. A NoSQL database does not use tables or relations for the use of storing and extracting data. Instead NoSQL data models store data in a single data structure, this can also include relations to other data within the structure. (Schaefer, 2020a) The differences

between the two solutions will be compared in terms of normalization, flexibility, scalability, structure, and performance.

5.1.1 Normalization and Flexibility

SQL schemas are normalized which prevents a lot of possible errors and is more efficient because it makes it so that data is only stored once. This does cause it to be less flexible because to save a new type of data in a table the whole table structure must be changed. This can be a long and complicated process.

For NoSQL the only structure you need to follow is the syntax, therefore the developer must check and uphold the normalization. NoSQL is therefore less efficient with data but luckily data is much cheaper to store today than it was back in 1970. This weakness is also its biggest strength because it allows it to be extremely flexible. If changes are made to what data should be saved, it should be easy to implement.

5.1.2 Scalability

SQL databases usually scale vertically meaning that when you reach the limit of your server you must migrate to a larger more expensive one. While with NoSQL databases scale horizontally meaning as stated by Schaefer (2020b) “you can add cheaper, commodity servers whenever you need to”.

5.1.3 Structure

SQL databases are comprised of table like structures (rows/columns model) for data storage which is ideal for multi-row transactions. NoSQL databases can use a variety of structures such as key-value pairs, column oriented graph, graphs based, and document oriented. (bmc, n.d.)

The structures will be explained with their use cases.

- **Document based** structures act like a JSON (JavaScript Object Notation) document with fields and values that can be a variety of types such as strings, integers, booleans, arrays, or objects. This is a good solution for a general purpose NoSQL database. MongoDB and CouchDB are both examples of solutions that use document based structures.
- **Key-value pairs** are a simpler structure where each element has a key with a value. The key is used to extract the value. This solution is good for storing large amounts of data that does not need complex queries. This could include user preference or caching.

(Schaefer, 2020a) Redis and DynamoDB are both examples of solutions that use the key-value pair structure.

- Schaefer states that (2020a) **Column oriented graphs** “store data in tables, rows, and dynamic columns... provide a lot of flexibility over relational databases because each row is not required to have the same columns.” This solution is good for large amounts of data where the query pattern can be predicted such as profile data. Cassandra and HBase are both examples of solutions that use column oriented graph structures.
- **Graph based** databases use nodes and edges to store data. The node store data about things such as people and locations while the edges store information about the relationships between the nodes. (Schaefer, 2020a) They are ideal for relationships networks. Neo4j and Amazon Neptune are both examples of solutions that use graph base structures.

5.1.4 Performance

Query performance generally depends on the amount of data being stored. For small amounts of data joining tables is very quick. However, joining two tables requires a lot of expense when dealing with large amounts of data. In NoSQL data retrieval typically does not require joins resulting in faster and more efficient query performance.

5.1.5 Development Experience

SQL database are good for SQL serves that are used by multiple different teams with multiple database developers with different roles/permissions. This is because SQL servers can be easily managed by controlling what given users can do. NoSQL is better for cloud solutions due to its easy integration and mapping. Cloud computing is more resilient and scalable compared to traditional server hosting.

SQL is very rigid, so it works well with rigid SDLC models such as the Waterfall or V-Model.

NoSQL can easily adapt to changing requirements, no need to change the database model. This flexibility makes it a very good tool to work within the Agile model.

5.2 CAP Theorem

CAP theorem is a strategy for managing distributed systems. A distributed system stores data in a network of machines all at the same time. This makes distributed systems a great pair with

NoSQL databases because they scale horizontally, so instead of buying a better server you can just add another node (machine) to your distributed system. When working with distributed systems CAP theorem can deliver two out of it's three desired characteristics: consistency, availability, and partition tolerance. (IMB Cloud Education, 2019) The characteristics will be explained in the following bullet points.

- **Consistency** means the data is always consistent when looking at the data it no matter which node the user is connected to. This is achieved with a method where whenever data is updated in one node the other nodes are forwarded the change.
- **Availability** means that whenever a request for data is made the client always gets a response even if a node is down without exception. (IMB Cloud Education, 2019)
- IBM states that (2019) “A *partition* is a communications break within a distributed system—a lost or temporarily delayed connection between two nodes.” **Partition Tolerance** means that the system must continue to operate even if any connections break down between nodes in the cluster.

NoSQL databases are split into three different types of two CAP characteristics which include: CP, AP and CA databases. MongoDB is a CP database meaning it remains consistent by resolving network partitions but lacks availability. Another example is Cassandra which is an AP database so it delivers availability and partition tolerance but cannot always deliver total consistency. AP databases like Calandra provide **eventual consistency** which is the principle that after an update eventually all the nodes will be consistent after all the nodes have been updated. During the updating process is where the inconsistency can occur. This solution is good for fault-tolerant systems. (IMB Cloud Education, 2019) This solution would be bad for systems where immediate consistence in necessary such as ecommerce or banking systems.

5.3 Table of Differences

Name	SQL	NoSQL
History	Old (1970s)	New (2000s)
Schemas	Good Normalization / Bad Flexibility	Bad Normalization / Good Flexibility

Scaling	Expensive to scale, vertical	Cheaper to scale, horizontal
Structure	Tables with fixed rows and columns	Key-value pairs, column oriented graph, graphs based, and document oriented
Examples	MySQL, MS SQL, and PostgreSQL	“Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune” (Schaefer, 2020b)
Transactions	ACID (Atomicity, Consistency, Isolation, Durability)	CAP (consistency, availability, and partition tolerance)
Queries	SQL	REST/GET
Concurrency	Pessimistic (COMMIT, ROLLBACK)	Optimistic (copies, timestamping)

Table 1: SQL vs NoSQL (Scherer, 2020)

5.4 Summary

When choosing a database, you must look at what you need and then decide not only SQL vs NoSQL but also all the different technologies and solutions that come with it. For situations where the database needs to be highly relational, consistent, and will not hold large amounts of data, SQL is the better choice in most situations. SQL also has the advantage of being well established so finding SQL developers should be easy. NoSQL databases should be used for cheaper faster big storage solutions, with easy scalability, where only two of the three CAP theorem properties are present. It is a newer technology but has become a major player in the database landscape today.

6 Critical Review

6.1 Design

Overall the author is satisfied with the design. The only changes that would be made is to make the many to many relationship tables consistent since one of them uses an ID. It would be better if the design had limitation on how many parents' children can have because it is possible for a child to have more than two parents. This also includes how many teachers a class can have since there is no limitation. And the last improvement would be to add the cascade feature in tables/relationships where it is needed. Since the design follows BCNF the prospect of future extensions and enhancements should not be difficult to implement. Things like student grades and school building and their locations, classes and students should be easy to implement.

6.2 Queries

All the queries/views/procedures that were created worked well and as expected. The possibility of query optimization may be possible, but the section was not covered by the Author.

6.3 Documentation

There is not any room for improvement that the author could think of. Everything was covered and was cited with reliable sources.

7 Self-Improvement Evaluation

During this class I learned the following:

- The Normal Forms to better design relational databases.
- What triggers, transactions, views and stored procedures are and used for.
- SQL database user permission levels.
- NoSQL structure types and CAP theorem with their given solutions.
- Better understanding of when to use SQL vs NoSQL.

All the information I have gain has been and will be extremely valuable especially the information about NoSQL since I believe it will become more popular than SQL in the future. I need to investigate query optimization and study the rest of the normal forms (4NF and 5NF) if I will continue working with SQL. I want to learn more about NoSQL and implement an example project for practical knowledge. The knowledge I have gain has made me better informed developer that can make wiser decisions when creating a project with database. This will immensely help with my job prospects in the future as well as being a better developer/employee.

References

- bmc (n.d.) *SQL vs NoSQL Databases: What's The Difference?* [Online] Available at: <https://www.bmc.com/blogs/sql-vs-nosql/> (Accessed: 25 April 2020).
- IMB Cloud Education (2019) *CAP Theorem* [Online] Available at: <https://www.ibm.com/cloud/learn/cap-theorem> (Accessed: 26 April 2020).
- Janert, P. (2003) *Practical database design, Part 2* [Online] Available at: <https://www.ibm.com/developerworks/library/wa-dbdsgn2/index.html> (Accessed: 28 April 2020).
- MySQL (2020a) *Privileges Provided by MySQL* [Online] Available at: <https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html> (Accessed: 22 April 2020).
- MySQL (2020b) *Using Triggers* [Online] Available at: <https://dev.mysql.com/doc/refman/5.7/en/triggers.html> (Accessed: 22 April 2020).
- phptpoint (n.d.) *MySQL Tutorial* [Online] Available at: <https://www.phptpoint.com/mysql-tutorial/> (Accessed: 28 April 2020).
- Prague College (2020) *Advanced Database Systems (FT)* [Online] Available at: The system should hold information about children and also their parents or legal (Accessed: 28 April 2020).
- Schaefer, L. (2020a) *NoSQL Databases Explained* [Online] Available at: <https://www.mongodb.com/nosql-explained> (Accessed: 26 April 2020).
- Schaefer, L. (2020b) *NoSQL vs SQL Databases* [Online] Available at: <https://www.mongodb.com/nosql-explained/nosql-vs-sql> (Accessed: 25 April 2020).
- Scherer, P. (2020) *SQL-NoSQL comparison* [Online] Available at: <https://docs.google.com/spreadsheets/d/1F5NMKAvg8IBirVPSW-2cBY96bel2ell1tKwW1KLOVo/edit#gid=0> (Accessed: 15 April 2020).
- Teesside University (2010) *Database Development* [Online] Available at: https://moodle.praguecollege.cz/pluginfile.php/44388/mod_resource/content/1/slides_transactions.pdf (Accessed: 21 April 2020).
- w3schools (2020) *PHP MySQLi Functions* [Online] Available at: https://www.w3schools.com/php/php_ref_mysqli.asp (Accessed: 27 April 2020).

