EDUDatabase	Doc Number	EDUDatabase-02		
	Version	2.0		
	Print Date	6/10/20		
	Page	1 of 14		

Baseline Project Report

Customer	
Project Name	
Project Manager	

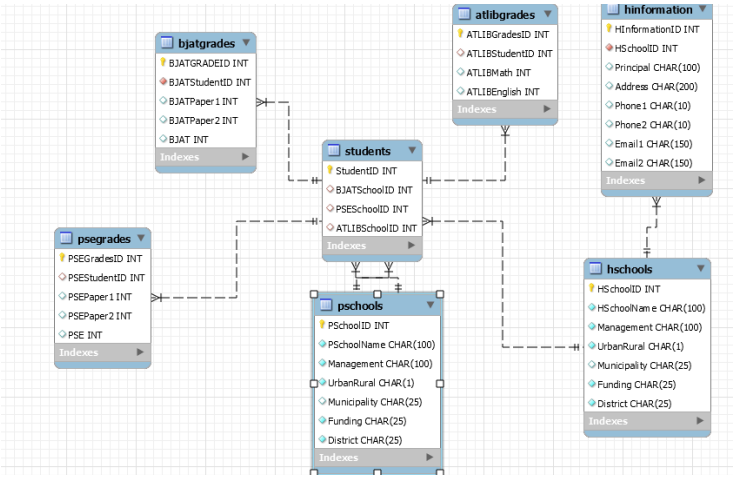
1.0 Introduction	A. Project Overview - This is an introduction to databases project designed to introduce novices to database management systems, database design and data manipulation languages. B. The rationale behind this project is to achieve course intended learning outcomes(CILO). The CILO is as follows. This course is designed to provide fundamental techniques and knowledge for data storage and retrieval. Topics include: Fundamentals of Database Management System and Database Design which encompasses Relational Model and Normalization, Database Design Using Normalization, Data Modeling and the Entity-Relationship Model, and Transforming Data Models in Database Designs. C. The data for this project was put into tables that were based on entity relationships. The reason data is put into tables is two-fold. Firstly data redundancy, if all fields are put into the same table data is heavily repeated. Secondly empty slots, not all data has the same fields in common therefore those fields remain empty. If the fields relate most rows will be complete with data in every field. Overall splitting data into entity based relational tables makes the data easier to apprehend, wastes less space, and improves data integrity. D. This database was made the way it is due to normalization. Normalization is the process used to put the data into tables. This process reduces data redundancy and eliminates data integrity anomalies that occur when data isn't properly placed into tables. There are three common anomalies that occur when data isn't properly placed into tables. Firstly the insertion anomaly which occurs when the user is required to enter unnecessary data in the table for which the user doesn't have access. The data is unnecessary in that it doesn't directly relate to what the table should be based on. Secondly the deletion anomaly occurs when too much information is stored in one field resulting in loss of information when it is deleted with the intention of removing only a couple of columns. Finally the update anomaly occurs when a table has attributes that should be put into separate tables due to them being too unrelated, then when an update is made on an attribute in a particular field that attribute will be different in other fields when it shouldn't be different.

	E. The aforementioned tables were then implemented using MySQL which is an open-source relational database management system (DBMS) developed by Oracle Corporation,notably the most popular DBMS in the world. F. The project began at UB during scheduled class times. A linux server with MySQL was set up for the students to practice structured query language(SQL) using MySQL. The tables were then implemented in MySQL using data definition language (DDL) using the information from the entity relationship diagrams (ERD). The ERD describes how the tables will relate, their keys, data types and attributes. It's like the database's blueprint, which is important because making changes on the fly leads to poor documentation and is actually quite tedious. G. A pandemic caused UB to move their lectures online. This is when students turned their personal devices into servers when they installed MySQL. The database and its tables were then created as per the ERD specifications.
2.0 System Description	A. This database project was created with the intention of being full stack, that is both frontend and backend. The backend database system is where SQL is written to create the database and its tables on the machine. It is where data can be imported and distributed to the tables on a massive scale.The backend is unseen by the user for security and user friendliness. Meanwhile the frontend is what is seen by the user. Using front end technologies the backend can be viewed graphically with user interaction. Databases are everywhere, the entire world wide web is simply a myriad of databases connected by a search engine. Fantastic interaction features allow even small children to search a database using the mic search option despite the fact they can't read nor write. Therefore creating a full stack project we can gain experience on what it takes to create a working database and how to interact with it graphically. B. The backend of the system utilizes a MySQL database with InnoDB as its storage engine. The backend was created on a personal computer after the aforementioned DBMS was installed. The backend fulfills the database description and purpose. However the frontend doesn't interact with a MySQL database but instead with a PostgreSQL for the purposes of easier compatibility. This was an easy transition as the semantics are similar. C. The middleware is the stack technology that connects the frontend to the backend. The middleware for this project was Node JS and Express JS. Node JS is an open source JavaScript runtime environment that executes JavaScript code outside of a web browser. Express JS is a web

	application framework made for Node JS it may be considered the de facto standard server framework for Node JS. As the web application framework, Express was used to build and deploy the database to a website. D. The frontend for this project was created using HTML and CSS. HTML stands for Hypertext Markup Language and is used to display information on the screen. Meanwhile CSS, Cascading Sheet Styles, is used alongside HTML to make the web page more graphic. CSS adds font, color, contour, shape, and much more to the website making it a real graphical interface. The frontend can become an intuitive interaction with the database when done correctly. E. Data for the system was provided by the course lecturer and consisted of BJAT, PSE, and ATLIB test scores along with district and school information.. School contact and address information was provided by MoE online. However it was processed for this project's data requirements because it was too raw and had more data than necessary.
3.0 Database Design ER Diagram	A. The ER diagram blueprints the database. It gives information on the table name, it's attributes, attributes' data type, and the tables' keys. It also shows a table's relation to other tables. Hence it is the perfect database documentation in diagram form. The diagram is also error free because it is created using MySQL Workbench reverse engineer option. Reverse engineer simply takes the database and creates its blueprint based on the DDL used to create the database. Workbench is a visual design tool for MySQL that integrates SQL development, administration, database design, creation and maintenance into a single IDE. B. The database is designed to have entity related tables. This means that it is categorized by similar content. The process of putting the data into tables is called normalization, the tables in the diagram are in Boyce-Codd Normal Form. This means that every column that determines related attributes gets to be the primary key in its very own table. We can see that 'StudentID' in 'students' determines every other attribute. That means the student determines the PSE, BJAT and ATLIB school identification that specific student attended. Meanwhile these latter attributes each get to be the foreign key in other tables. These tables are 'pschools' and 'hschools', they provide information on the schools name, management, urban or rural status, municipality, funding and district. We can see that because their foreign key exists in 'students' the students ID can be used to trace school information. Meanwhile 'studentID' in 'students' is the foreign key in the test grades tables. Again we can see that using information within 'students' we can use it as a key to search the corresponding data in a different table. The crow's feet and the line indicates whether the relationship between the table is one to one, one to many or many to

many. The crow's feet means its many as a maximum and the line indicates that its minimum and/or maximum is one. If there were an oval sign it would indicate an optional relationship. Overall this ER diagram depicts how the problem of redundant data and blank spaces is overcome, by separating the attributes appropriately.

C.



4.0
SQL DDL

:

The DDL code looks different because of table and column names however the semantics are alike. Every DDL chunk produces a table its columns and links it to another table using a foreign key.

This code creates a table for primary schools and the information related to the school like it's management, funding, etc.

```
CREATE TABLE PSCHOOLS(
    PSchoolID INT NOT NULL,
    PSchoolName CHAR(100) NOT NULL,
    Management CHAR(100) NOT NULL,
    UrbanRural CHAR(1) NOT NULL,
    Municipality CHAR(25),
    Funding CHAR(25) NOT NULL,
    District CHAR(25) NOT NULL,
    PRIMARY KEY (PSchoolID)
);
```

This code creates the high school information table with attributes similar to the previous table but for high schools.

```
CREATE TABLE HSCHOOLS(
   HSchoolID INT NOT NULL,
   HSchoolName CHAR(100) NOT NULL,
   Management CHAR(100) NOT NULL,
   UrbanRural CHAR(1) NOT NULL,
   Municipality CHAR(25),
   Funding CHAR(25) NOT NULL,
   District CHAR(25) NOT NULL,
   PRIMARY KEY (HSchoolID)
);
```

Next up is the core of the entire database. This table is on students which contains foreign keys which are used to correspond student data to data in every other table. This way not all student related data is contained within a single table.

```
CREATE TABLE STUDENTS(
   StudentID INT NOT NULL,
   BJATSchoolID INT,
   PSESchoolID INT,
   ATLIBSchoolID INT,
   PRIMARY KEY (StudentID),
   FOREIGN KEY (BJATSchoolID)
   REFERENCES PSCHOOLS(PSchoolID),
   FOREIGN KEY (PSESchoolID)
   REFERENCES PSCHOOLS(PSchoolID),
   FOREIGN KEY (ATLIBSchoolID)
   REFERENCES HSCHOOLS(HSchoolID)
);
```

Table created to contain data on BJAT grades with 'BJATStudentID' as the foreign key referencing 'StudentID' in 'students'.

```
CREATE TABLE BJATGRADES(
   BJATGradeID INT NOT NULL,
   BJATStudentID INT NOT NULL,
   BJATPaper1 INT,
   BJATPaper2 INT,
   BJAT INT,
```

```
    PRIMARY KEY (BJATGradeID),
      FOREIGN KEY (BJATStudentID)
      REFERENCES STUDENTS(StudentID)
    );
```

This table contains PSE grades the first attribute is the table ID the second attribute, ‘PSEStudentID’, is of type INT and is a foreign key the references ‘StudentID’ of ‘students’. Then there are two more attributes for the two different PSE papers, followed by an attribute ‘PSE’ that is the average of the two papers.

```
CREATE TABLE PSEGRADES(
   PSEGradesID INT,
   PSEStudentID INT,
   PSEPaper1 INT,
   PSEPaper2 INT,
   PSE INT,
   PRIMARY KEY (PSEGradesID),
   FOREIGN KEY (PSEStudentID)
      REFERENCES STUDENTS(StudentID)
);
```

The following table follows the logic of the previous table but for ATLIB however it doesn’t have a test total average.

```
CREATE TABLE ATLIBGRADES(
   ATLIBGradesID INT NOT NULL,
   ATLIBStudentID INT,
   ATLIBMath INT,
   ATLIBEnglish INT,
   PRIMARY KEY (ATLIBGradesID),
   FOREIGN KEY (ATLIBStudentID)
      REFERENCES STUDENTS(StudentID)

);
```

This DDL code is used to import the data into ‘PSchools’. The data is in a csv file therefore fields are terminated by ‘,’. A row is ignored because it is the attribute names row.

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server
8.0/Uploads/PSCHOOLS.csv'
INTO TABLE PSCHOOLS
FIELDS TERMINATED BY ','
```

ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

This DDL and the following work exactly the same except they are meant for a different table and therefore load data from a different file. For example this DDL imports 'HSCHOOLS.csv' to 'HSchools'.

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/HSCHOOLS.csv'
INTO TABLE HSCHOOLS
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

Imports data from 'STUDENTS.csv' to the 'students' table.

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/STUDENTS.csv'
INTO TABLE STUDENTS
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

Imports data from 'BJATGRADES.csv' to the 'BJATGrades' table.

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/BJATGRADES.csv'
INTO TABLE BJATGRADES
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

Imports data from 'PSEGRADES.csv' to the 'PSEGrades' table.

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/PSEGRADES.csv'
INTO TABLE PSEGRADES
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

Imports data from ‘ATLIBGRADES.csv’ to the ‘ATLIBGrades’ table.

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ATLIBGRADES.csv'
INTO TABLE ATLIBGRADES
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;
```

I wrote the following DDL that creates a table, ‘HInformation’, that holds data on high schools’ contact information and address. The primary key auto increments. The second attribute is a foreign key referencing HSchools. Meanwhile the other attributes contain high school data.

```
CREATE TABLE HInformation(
HInformationID INT NOT NULL  AUTO_INCREMENT ,
HSchoolID INT NOT NULL ,
Principal CHAR(100),
Address CHAR(200),
Phone1 CHAR(10),
Phone2 CHAR(10),
Email1 CHAR(150),
Email2 CHAR(150),
Primary Key (HInformationID),
Foreign Key (HSchoolID)
REFERENCES HSchools (HSchoolID)
);
```

The following DDL populates ‘HInformation’ with data from ‘db-hs-data.csv’.

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/db-hs-data.csv'
INTO TABLE HInformation
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
```

5.0
SQL DML

This data manipulation language (DML) is used to gain information from the database. It's the means of querying the database. The DML documented here are the basic queries done in the course and also used to produce information for the front end. It is also important to note that much of it is the same, BJAT, PSE, and ATLIB school averages get grouped by an attribute in 'pschools' for BJAT and PSE and 'hschools' for ATLIB. The average grade and a school description are selected from a joint table of the grades table and the school table then grouped by the school description and ranked by the average.

Description of SQL DML goes here. Also explanations of INTERESTING queries.

Information is the answer to the question asked of data. Therefore to get useful information from this database we need to query the appropriate data for the answer.

Districts by BJAT Scores

```
SELECT District, AVG(BJAT)
FROM BJATGRADES B
JOIN STUDENTS S ON B.BJATStudentID = S.StudentID
JOIN PSCHOOLS PS ON S.BJATSchoolID = PS.PSchoolID
GROUP BY District
ORDER BY AVG(BJAT) DESC;
```

Municipality by BJAT Scores

```
SELECT Municipality, AVG(BJAT)
FROM BJATGRADES B
JOIN STUDENTS S ON B.BJATStudentID = S.StudentID
JOIN PSCHOOLS PS ON S.BJATSchoolID = PS.PSchoolID
GROUP BY Municipality
ORDER BY AVG(BJAT) DESC;
```

Management by PSE scores.

```
SELECT Management, AVG(PSE)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
GROUP BY Management
ORDER BY AVG(PSE) DESC;
```

Funding by PSE scores.

```
SELECT Funding, AVG(PSE)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
GROUP BY Funding
ORDER BY AVG(PSE) DESC;
```

Urban/Rural by ATLIB scores.

```
SELECT UrbanRural, AVG(ATLIBMATH)
FROM ATLIBGrades B
JOIN Students S ON B.ATLIBStudentID = S.StudentID
JOIN HSchools HS ON S.ATLIBSchoolID = HS.HSchoolID
GROUP BY UrbanRural
ORDER BY AVG(ATLIBMATH) DESC;
```

Schools by ATLIB score.

```
SELECT HSchoolName, AVG(ATLIBMATH)
FROM ATLIBGrades B
JOIN Students S ON B.ATLIBStudentID = S.StudentID
JOIN HSchools HS ON S.ATLIBSchoolID = HS.HSchoolID
GROUP BY HSchoolName
ORDER BY AVG(ATLIBMATH) DESC;
```

Some semi-interesting queries

This query is different in that it ranks schools by PSE averages but only those in the Cayo district.
Ranking of Cayo Primary Schools by average PSE.

```
SELECT PSchoolName, AVG(PSE)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
WHERE DISTRICT = "Cayo"
GROUP BY PSchoolName
ORDER BY AVG(PSE) DESC;
```

This query shows the BJAT averages for Belmopan and Benque.
BJAT grades in Belmopan vs Benque

```
SELECT Municipality,  AVG(BJAT)
FROM BJATGRADES B
JOIN STUDENTS S ON B.BJATStudentID = S.StudentID
```

```
JOIN PSCHOOLS PS ON S.BJATSchoolID = PS.PSchoolID
where municipality IN ("Belmopan", "Benque")
group by municipality
order by avg(bjat) desc;

Belize City High Schools by ATLIB averages

SELECT HSchoolName, AVG(ATLIBMATH)
FROM ATLIBGrades B
JOIN Students S ON B.ATLIBStudentID = S.StudentID
JOIN HSchools HS ON S.ATLIBSchoolID = HS.HSchoolID
GROUP BY HSchoolName
ORDER BY AVG(ATLIBMATH) DESC;

ATLIB average for a specific school

SELECT  HschoolName ,AVG(ATLIBMATH)
FROM ATLIBGrades B
JOIN Students S ON B.ATLIBStudentID = S.StudentID
JOIN HSchools HS ON S.ATLIBSchoolID = HS.HSchoolID
Where hschoolname = "Cayo Christian Academy";

Largest primary school by PSEs taken

SELECT PSchoolName, count(psestudentid)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
GROUP BY PSchoolName
ORDER BY count(PSEstudentid) DESC;

Smallest highschool based on ATLIBs taken

SELECT hschoolname, count(atlibstudentid)
FROM atlibGRADES B
JOIN Students S ON B.atlibStudentID = S.StudentID
JOIN hSchools PS ON S.atlibSchoolID = PS.hSchoolID
group by hschoolname
ORDER BY count(atlibstudentid) ASC;

Primary School Rural Urban Count

SELECT count(pschoolid)
FROM pschools
group by urbanrural;

School Management Ranked by Size
```

```
SELECT Management, count(studentid)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
GROUP BY Management
ORDER BY count(studentid) DESC;
```

Number of schools per funding type

```
SELECT Funding, count(pschoolName)
FROM PSEGRADES B
JOIN Students S ON B.PSEStudentID = S.StudentID
JOIN PSchools PS ON S.PSESchoolID = PS.PSchoolID
GROUP BY Funding
ORDER BY count(pschoolName) DESC;
```

Now the following is an interesting query based on United Evergreen Primary School (UEPS). It shows the effect that repeating failing students has on their grades. This is done by showing the average BJAT and PSE. Then showing the average BJAT and PSE for students that didn't repeat. Finally the average BJAT score for students that failed standard III, then the average PSE for students who had at some point failed a year from standard III to standard V. The evidence would be a lot more conclusive if the averages from the repeating students were from the same set of students. However, the query still gives some insight.

This DML displays the average BJAT score for all students from UEPS. Its result is 56.79 which is lower than the average BJAT for non repeating students.

```
select avg(bjat)
from bjatgrades b
join students s on b.bjatstudentid = s.studentid
join pschools ps on s.bjatschoolid = ps.pschoolid
where pschoolname = 'UTD EVERGREEN CY';
```

This DML displays the PSE average for all the students at UEPS. This PSE average is higher than the students that had previously repeated but lower than students who had not repeated. The result of this query is 73.90

```
select avg(pse)
from psegrades b
join students s on b.psestudentid = s.studentid
join pschools ps on s.pseschoolid = ps.pschoolid
```

where pschoolname = 'UTD EVERGREEN CY';

This DML outputs the average test scores for non repeating students. The output is 59.16 and 75.49 for BJAT and PSE respectively.

```
select avg(bjat), avg(pse)
from psegrades b
join bjatgrades p on b.psestudentid = p.bjatstudentid
join students s on p.bjatstudentid = s.studentid
join pschools ps on s.pseschoolid = ps.pschoolid
where pschoolname = 'UTD EVERGREEN CY';
```

This DML shows the BJAT average for students who took the BJAT but whose student ID can't be found in the PSE grades table implying that they failed. The result is 50.25 which is lower than non repeating students.

```
select avg(bjat)
from bjatgrades b
join students s on b.bjatstudentid = s.studentid
join pschools ps on s.bjatschoolid = ps.pschoolid
where pschoolname = 'UTD EVERGREEN CY'
and bjatstudentid NOT IN(
select psestudentid
from psegrades);
```

This DML shows the PSE average for students who took the PSE but whose student ID doesn't appear in the BJAT table implying that they failed at some point in between the two tests. This average is 70.16 which is approximately a 5 point difference to non repeating students' average meanwhile the BJAT difference between the two groups of students has a nine point difference. Therefore we consider repeating students over promoting everyone regardless of grades.

```
select avg(pse)
from psegrades p
join students s on p.psestudentid = s.studentid
join pschools ps on s.pseschoolid = ps.pschoolid
where pschoolname = 'UTD EVERGREEN CY'
and psestudentid NOT IN(
select bjatstudentid
from bjatgrades);
```
