OFFICAL DOCUMENTATION FOR CEIT PROJECT

Date Modified: <who cares>

### **NOTE: THE QUOTES LISTED IN THIS DOCUMENT MIGHT FAIL IN PHPMYADMIN AND HAS TO BE MANUALLY REAPPLIED IN THE CONSOLE AS MS WORD CONVERTS THEM INTO A FORMAT THAT PHPMYADMIN DOES NOT RECOGNIZE.**

Project Description

This will be a project taken by a few students from APTECH to create a better process to allow students and interested individuals to apply for courses online.

The website will have two (2) page; the front page and the admin page

The front page will display a grid showing pending and ongoing course batches and will allow users to register for said batches. The requirements for the front page are as follow:

·**Display pending and ongoing batches in grid by days and timeslots Ϡ (displays pending and out going::*more information needs to be sent to understand “timeslots” sorting by date is not implemented as yet)***

**·View details about batch by click the respective batch in the grid Ϡ - Completed**

**·Allow user to register for a batch by submitting their name and email address** **Ϡ - Completed**

The admin page will allow the administrator to manage course batches, view registrations and send emails to registered students. The requirements for the admin page are as follows:

·**List pending and ongoing batches with the option to also view closed batches (displayCourseBatches() & displayAllCourseBatches())**

**·Allow admin to cancel a pending batch and close an ongoing batch Ϡ (Compelted)**

**·Allow admin to create new batches-Ϡ (Compelted)**

**·View details batch by click the respective list item (?Need to clarifiy)**

**·View interested students by batch**

**·Allow admin to send email to interested students**

**SeverPages**

Server pages for the site serve one of three main purposes: displaying a UI, providing data in JSON format for client side processing and processing submissions from users. Pages and their roles are listed below.

Index: entry page to the site. Displays the main public UI which is the batch grid.

Batches: retrieves batch information for pending and ongoing batches, as well as batch-timeslot pairings. Data will reflect below.

*{id:1,status:1,summary:"short description",courseid:1,coursename:"course 1",availseats:10,maxseats:20,startdate:'2017-06-20',enddate:'2017-08-20',teacher:'Subra'}*

*{timeslot:0,batchId:1}*

Register: receives and processes student registration. POST parameters will reflect below.

username, useremail, batchId

# DATABASE TABLE DIAGRAMS

Format: PNG

File Name: sql2.png

# PROCEDURE LIST OF INSERTING DATA FROM THE FRONT END TO THE BACK END

We will use the following MYSQL Procedures that the Front End Designer will call to acquire and modify the desired tables. An over view of the procedures are as follows:

|  |  |  |
| --- | --- | --- |
| Procedure | Data Type | Tables called |
| insertNewCourse( **in** courseCode(varchar 6),**in** courseName(varchar 100),**in** courseSum (text) ) | Inserts new Records into the database for just an academic field. CourseCode will represent the abbriv code for each subject such as eg. CMP112. Unique and cant be null | subjectcourses |
| insertNewTeacher(**in** teacherName,**in**  gender enum(‘Male’,’Female’,’Unknown’),**in**  email varchar(100)) | Inserts new records of teachers into the database. Caters for teachers with the same names but the email addresses are unique for each record | ceiteachers |
| insertNewBatch(**in** subjectId (smallint),**in**  coursestatus(enum),**in**  teacher( varchar 100),**in**  schoolYear(smallint 5),**in**  start\_date,**in**  end\_date,**in**  availableSeats(smallint),**in**  max\_Seats(smallint)) | Inserts the new course batches that will be displayed eventually to the users on the current status of a course | coursebatches,  ceiteachers,  subjectcourses |
| insertNewRegisteredStudent(**in**sName(varchar 100),**in** sEmail(varchar 100), **in** desiredBatchId (smallint)) | Record used to store all interested students who were successful or failed to get a seat. Email Address is unique | registeredstudents |
| updateAvailableSeats**in** courseBatchId (smallint),  **in** studentEmailQ (varchar 100) | Increments the availableseats col until it is less or equal to the max seats | coursebatches |
| displayCourseBatches(); | Table used to display all the courses,seats etc with the course status of **PENDING OR ONGOING** | coursebatches  subjectcourses   ceiteachers |
| insertStudentCourseBatches(courseBatchId (smallint),  studentRefId(smallint),  conditionResult (bit)) | Used to insert the status of student attempting to register for a course if they were successful or not | studentcoursebatches |
| createDatabaseTables() | Used to create all the tables in the database **DOES NOT CREATE THE INDEXES AND RELATIONSHIPS YET** | <ALL THE TABLES> |
| displaySelectedBatchDetails(in batchId smallint) | Displays all the details about a particular batch via its batch id. | Coursebatches  subjectcourses  ceiteachers |
| displayAllCourseBatches() | Displays all the batches and their details including closed batches |  |
|  |  |  |

## Creating a User and Setting Access Rights

Teachers & SubAdmins

**Database username:** ceitdbuser

**Database userPassword:** 4E5F9888F0BB1EC597E58BC341053660F

***Create user ceitdbuser;***

***grant all on ceitdominicaregister\_dm\_db.\* to 'ceitdbuser'@'localhost' identified by '4E5F9888F0BB1EC597E58BC341053660F';***

<!>*User Account that will be used by the submins (teachers) to gain access to the database and said tables*</!>

**//TODO: PREVENT ACCOUNT FROM DOING THE FOLLOWING:**

1. **Deleting Database and Tables**
2. **Review account Access Rights**

Front End Users on the Website (Public Domain)

**Database username: i**nternetUser

**Database userPassword:** 331A11AC61113EDDC283C5BEC8996

***Create user internetUser;***

<!>This user account will be the account that will be used directly on the website. As a result the access rights to this account has been limited to the following:

·         INSERT::registeredstudents

·         SELECT ::coursebatches,registeredstudents,ceiteachers(lecturer),subjectcourse

·         UPDATE::registeredstudents

All of this will be done on the following tables:

1.       registeredstudents

2.       coursebatches

3.       ceiteachers :: will only have access to the coloumn to view the information under this account

4.       subjectcourse

</!>

***use ceitdominicaregister\_dm\_db;***

***grant insert on ceitdominicaregister\_dm\_db.registeredstudents to 'internetUser'@'localhost' identified by '331A11AC61113EDDC283C5BEC8996';***

***grant select on ceitdominicaregister\_dm\_db.coursebatches to 'internetUser'@'localhost';***

***grant select on ceitdominicaregister\_dm\_db.registeredstudents to 'internetUser'@'localhost';***

***grant select on ceitdominicaregister\_dm\_db.subjectcourses to 'internetUser'@'localhost';***

***grant select (lecturer) on ceitdominicaregister\_dm\_db.ceiteachers to 'internetUser'@'localhost';***

***grant insert on ceitdominicaregister\_dm\_db.studentcoursebatches to 'internetUser'@'localhost';***

***grant select on ceitdominicaregister\_dm\_db.studentcoursebatches to 'internetUser'@'localhost';***

***grant execute on PROCEDURE ceitdominicaregister\_dm\_db.insertNewRegisteredStudent to 'internetUser'@'localhost';***

***grant execute on PROCEDURE ceitdominicaregister\_dm\_db.updateAvailableSeats to 'internetUser'@'localhost';***

***grant execute on PROCEDURE ceitdominicaregister\_dm\_db.insertStudentCourseBatches to 'internetUser'@'localhost';***

***grant execute on PROCEDURE ceitdominicaregister\_dm\_db.displayCourseBatches to 'internetUser'@'localhost';***

***grant execute on PROCEDURE ceitdominicaregister\_dm\_db.displaySelectedBatchDetails to 'internetUser'@'localhost';***

**//TODO:REVIEW ACCOUNT ACCESS LIST BY ADDING AND REMOVING MORE PROCEDURE ACCESS TO PUBLIC ACCOUNT**

SUBNOTE(S)

# Important Random Notes

### SQL STATEMENTS FOR CREATING AND SETTING UP THE TABLES FOR THE DATABASE

//revoke user permissions

REVOKE ALL PRIVILEGES, GRANT OPTION FROM <username>@localhost;

//Creating the database

create database if not exists ceitdominicaregister\_dm\_db;

<!>When setting up the foreign an primary key relationship between tables ensure that the UPDATE is set to CASCADE due to the fact that the RESTRICT will also prevent root and subroot accounts from updating any information in those relationships.</!>

**Step # 1 Create the database**

Database Name: **ceitdominicaregister\_dm\_db**

SQL CODE:

create database **ceitdominicaregister\_dm\_db;**

**Step # 2: Select the database for work**

SQL CODE:

use ceitdominicaregister\_dm\_db;

**Step #3: Create the procedures to create the tables**

SQL Code:

use ceitdominicaregister\_dm\_db;

DELIMITER $$

create procedure createDatabaseTables()

begin

create table ceiteachers

(id smallint primary key not null AUTO\_INCREMENT comment 'generic id used for the unique identification of a teacher',

lecturer varchar(100) not null comment 'first Name and last name of the teacher',

teacherGender enum('Male','Female','Unknown') **not null** default 'Unknown' comment 'gender of each teacher',

tEmail varchar(100) UNIQUE **not null** default '@domain.com' comment 'email for each teacher which needs to be unique on an individual level') engine='Innodb';

create table subjectcourses

(id smallint primary key not null AUTO\_INCREMENT comment 'unique row id',

courseID varchar(6) unique not null comment 'internal college course id thingy not important but cannot be null',

courseName varchar(100) not null comment 'name of the course and is required',

courseSummary text comment 'optional information about what the course is about') engine='Innodb';

create table batchschedule

(

batchFK smallint comment 'foreign key used to link to the table coursebatches id',timeslot tinyint default 0

) engine='Innodb';

create table registeredstudents

(

studentid smallint primary key not null AUTO\_INCREMENT comment 'unique id to represent the student record',

studentName varchar(100) not null comment 'name of the student that had a desire to joing the course contains both the first and last name',

studentEmail varchar(100) not null UNIQUE comment 'contact email for the student unique'

) engine='Innodb';

create table studentcoursebatches

(batchesFK smallint comment 'foreign key used to link to the table coursebatches under the col id',studentFK smallint comment 'foreign key used to link to the table registeredstudents to the col student',

submitDate timestamp default current\_timestamp,

seated bit comment 'true or false if this particular student was succesful in getting a seat')engine='Innodb';

create table coursebatches

(id smallint primary key not null AUTO\_INCREMENT comment 'unique primary key used to identify a course batch',

coursesFK smallint not null comment 'foreign key used to link the subject records from the table subjectcourses in col id',

courseStatus enum ('Pending','On-Going','Closed','Cancelled') **not null** default 'Pending' comment 'used to define the status if a course',

lecturerFK varchar(100) not null comment 'foreign key used to link to the table called ceiteachers for teacher FN and LN',

schoolYr smallint unsigned default 2000 comment 'school academic yr',

startDate date,endDate date,

availableSeats tinyint unsigned default 0 comment 'used to display to the students ',

maxSeats tinyint unsigned default 30 comment 'max seats per batch')engine='Innodb';

end $$

DELIMITER ;

**Step #4 Set Relationships (STILL PENDING!!)**

**STEP 5: Assign the relationships (Manually)**

\*\*See the SQL image **sql2.png** for guidance\*\*

**STEP 6: Create Procedure for inserting data into table subjectcourses**

SYNTAX:   
insert into subjectcourses (courseID,courseName,courseSummary) values (….,….,…);  
  
EXAMPLE:

**insert into subjectcourses(courseID,courseName,courseSummary)  
 values   
('Math11','Mathematics','Mathematics is an amazing subject to be taught and trained in'),  
('ENG117','English','Your mothers tongue bro'),('HFLE11','Health and Family Life',''),  
('SC','Speech Communication','');**

PROCEDURE FOR TABLE : subjectcouses

DELIMITER $$  
  
create procedure insertNewCourse( in courseCode varchar (6), in courseName varchar(100), in courseSum text)

Begin

Insert into subjectcourses (courseID,courseName,courseSummary)

Values

(courseCode,courseName,courseSum);

End $$

DELIMITER ;

**STEP #7: Create Procedure to insert a new teacher into the CEIT Organization**

SYNTAX:   
insert into ceiteachers (Lecturer,teacherGender,tEmail) values (….,….,…);  
  
EXAMPLE:

**insert into ceiteachers(Lecturer,teacherGender,tEmail)  
 values   
('Macaully Tavernier’,’Male’,’macaully.tavernier@aol.com’);**

**PROCEDURE FOR TABLE ceiteachers**

DELIMITER $$

create procedure insertNewTeacher(in teacher varchar(100), in gender enum('Male','Female','Unknown'),in email varchar(100))

Begin

Insert into ceiteachers(lecturer,teacherGender,tEmail)

Values

(teacher,gender,email);

End $$

Delimiter ;

**STEP #8: Create Procedure to insert new records in the coursebatches table**

SYNTAX:   
insert into coursebatches (coursesFK,courseStatus,lecturerFK,schoolYr,startDate,endDate,availableSeats,maxSeats) values (….,….,…);  
  
EXAMPLE:

**insert into coursebatches(**coursesFK,courseStatus,lecturerFK,schoolYr,startDate,endDate,availableSeats,maxSeats**)  
 values   
(11,’Pending’,’John Doe’,2017,’2018/07/3’,’2019/07/03’,0,30);**

Procedure

DELIMITER $$

Create procedure insertNewBatch(in subjectId smallInt, coursestatus enum('Pending','On-Going','Closed','Cancelled'), in teacher varchar(100) , in schoolYear smallint(5), in start\_date date, in end\_date date, in availableSeats tinyint, in max\_Seats tinyint)

Begin

Insert into coursebatches(coursesFK,courseStatus,lecturerFK,schoolYr,startDate,endDate,availableSeats,maxSeats)

values

(subjectId,coursestatus,teacher,schoolYear,start\_date,end\_date,availableSeats,max\_Seats);

End $$

Delimiter ;

**STEP 9: Create Procedure that will accept the information from the student and begin to process their request**

The Process for this result are as follows:

1. <User Selects a batch and chooses to submit their details>
2. Evaluate input
3. Send it to the database server (Name,Email Address and Desired Batch Id)
   1. Insert their name and email into the table **registeredstudent**
   2. Update the **availableSeats** counter once there is more ***available seats remaining returning 0 if there is an available email address before or 1 if there is not.***
   3. insert the **course batch id** and the **student record id into the table studentcoursebatches** along with the following**:**
      1. **time stamp of submitted record**
      2. **seated Status 0 meaning successful or 1 false. This will inform us if the student was successful in getting a seat or not**

**SYNTAX:**

Insert into registeredstudents(studentName,studentEmail) values (…,…,..,,)

EXAMPLE:

Insert into registeredstudents (studentName,studentEmail) values (‘John John’,’johntwo@gmail.com’);

**PROCEDURE FOR INSERTING NEW RECORDS INTO TABLE registeredstudents**

#1 a

DELIMITER $$

create procedure insertStudentCourseBatches(in courseBatchId smallint,studentRefId smallint,conditionResult bit)

begin

declare value11 smallint;

set value11 = (select count(\*) from studentcoursebatches where batchesFK = courseBatchid and (studentFK=studentRefId and seated=0));

if value11 = 0 then

insert into studentcoursebatches (batchesFK,studentFK,seated) values (courseBatchId,studentRefId,conditionResult);

end if;

end $$

delimiter ;

Procedure

#1 b.  
DELIMITER $$

Create Procedure updateAvailableSeats(in courseBatchId smallint,in studentEmailQ varchar(100))

Begin

declare avSeats smallint;  
declare maxxSeats smallint;

declare idFromEmail varchar(100);

declare result bit;

declare value11 smallint;

set avSeats = (select availableSeats from coursebatches where id= courseBatchId);  
set maxxSeats= (select maxSeats from coursebatches where id = courseBatchId);  
set idFromEmail=(select studentid from registeredstudents where studentEmail = studentEmailQ);

if avSeats < maxxSeats then

set value11 = (select count(\*) from studentcoursebatches where batchesFK = courseBatchid and (studentFK=idFromEmail and seated=0));

if value11 = 0 then

Update coursebatches set availableSeats = avSeats + 1 where id=courseBatchId;

set result = 0;

call insertStudentCourseBatches(courseBatchId,idFromEmail,result);

end if;

else

set result = 1;

call insertStudentCourseBatches(courseBatchId,idFromEmail,result);

end if;

End $$

Delimiter ;

**Procedure**

#1 c.

DELIMITER $$

Create procedure insertNewRegisteredStudent(in sName varchar(100),sEmail varchar(100),in desiredBatchid smallint)

Begin

declare xx bit;

declare value11 smallint;

set value11 =(select count(studentEmail) from registeredstudents where studentEmail = sEmail);

if value11>0 THEN

set xx = 0;

ELSE

set xx =1;

end if;

if xx=1 then

Insert into registeredstudents(studentName,studentEmail)values(sName,sEmail);

call updateAvailableSeats(desiredBatchid,sEmail);

else

call updateAvailableSeats(desiredBatchid,sEmail);

end if;

End $$

Delimiter ;

**STEP #10: Create Procedure for displaying the batches to the users with the course status of PENDING AND ONGOING**

**Example:**

select

**coursebatches.id,coursebatches.courseStatus,subjectcourses.courseName,subjectcourses.courseSummary,coursebatches.lecturerFK,coursebatches.schoolYr,coursebatches.startDate,coursebatches.endDate,coursebatches.availableSeats,coursebatches.maxSeats from coursebatches join subjectcourses on coursebatches.coursesFK=subjectcourses.id WHERE coursebatches.courseStatus='Pending'** [**or**](file:///C:/wamp64/www/ceitdominica/ceit_site_project/mysql_doc) **coursebatches.courseStatus='On-Going'**

**Procedure for displaying the Pending an Ongoing courses**

DELIMITER $$

create procedure displayCourseBatches()

begin

declare condition1 varchar(20);

declare condition2 varchar(20);

set condition1='pending';

set condition2='on-going';

select

coursebatches.id,coursebatches.courseStatus,subjectcourses.courseName,subjectcourses.courseSummary,coursebatches.lecturerFK,coursebatches.schoolYr,coursebatches.startDate,coursebatches.endDate,coursebatches.availableSeats,coursebatches.maxSeats from coursebatches join subjectcourses on coursebatches.coursesFK=subjectcourses.id WHERE coursebatches.courseStatus=condition1 or coursebatches.courseStatus=condition2;

end $$

DELIMITER ;

Procedure to Display all the details about a particular batch

Delimiter $$

create procedure displaySelectedBatchDetails(in batchId smallint)

begin

select

coursebatches.id,coursebatches.courseStatus,subjectcourses.courseName,subjectcourses.courseSummary,coursebatches.lecturerFK,coursebatches.schoolYr,coursebatches.startDate,coursebatches.endDate,coursebatches.availableSeats,coursebatches.maxSeats from coursebatches join subjectcourses on coursebatches.coursesFK=subjectcourses.id WHERE coursebatches.id = batchId;

end $$

Delimiter ;

Procedure to display all the courses in the batch including closed batches

DELIMITER $$

create procedure displayAllCourseBatches()

begin

select

coursebatches.id,coursebatches.courseStatus,subjectcourses.courseName,subjectcourses.courseSummary,coursebatches.lecturerFK,coursebatches.schoolYr,coursebatches.startDate,coursebatches.endDate,coursebatches.availableSeats,coursebatches.maxSeats from coursebatches join subjectcourses on coursebatches.coursesFK=subjectcourses.id;

end $$

delimiter ;

#1

delimiter $$

create procedure updateRegisteredStudentsDetails(in studentN varchar(100),in studentE varchar(100),in recordId smallint)

begin

update registeredstudents set studentName =studentN ,studentEmail = studentE where studentid = recordId;

end $$

delimiter ;

TEST

call updateRegisteredStudentsDetails('Macaully Venus Tavernier','lastspartan009@gmail.com',11);

#2

DELIMITER $$

create procedure updateTeacherDetails(in lecturerName varchar(100),tGender enum('Male','Female','Unknown'),teachEmail varchar(100),in teacherRecordId smallint)

begin

update ceiteachers set lecturer=lecturerName,teacherGender=tGender,tEmail=teachEmail where id = teacherRecordId;

end $$

Delimiter ;

TEST

call updateTeacherDetails("Charissa Vigilant","Female","charricharri@aol.com",3);

#3

DELIMITER $$

create procedure updateCourseDetails(in courseCodeID varchar(6),in course\_Name varchar(100),in course\_summ text,in courseRecordId smallint)

begin

update subjectcourses set courseID=courseCodeID,courseName = course\_Name,courseSummary = course\_summ where id = courseRecordId;

end $$

DELIMITER ;

TEST

call updateCourseDetails("JJ117","Introduction into Jew-Hood","They were the ones who killed Jesus but it is not too bad to get to know the culture and history of the Jews",5);

#4

DELIMITER $$

create procedure updateBatchDetails(courseRefId smallint,courseStatusQ enum('Pending','On-Going','Closed','Canceled'),lecturerRefId varchar(100),schoolYear smallint,start\_Date date,end\_Date date,availableSeatss tinyint,maxSeatss tinyint,in desiredRecord smallint)

begin

update coursebatches set coursesFK = courseRefId,courseStatus=courseStatusQ,lecturerFK=lecturerRefId,schoolYr = schoolYear,startDate = start\_Date,endDate = end\_Date,availableSeats=availableSeatss,maxSeats=maxSeatss where id =desiredRecord;

end $$

DELIMITER ;

TEST

call updateBatchDetails(5,"Pending","Charissa Vigilant",2020,"2020/04/25","2022/07/10",16,32,5);

#5

//TODO :: UNDERSTAND WHAT IS BATCHSCHEDULE PURPOSE

####DELETING RECORDS##################################

#1

DELIMITER $$

create procedure deleteRegisteredStudent(in emailAddress varchar(100),in recordId smallint)

begin

delete from registeredstudents where studentEmail = emailAddress and studentid = recordId;

end $$

DELIMITER ;

call deleteRegisteredStudent("macaully.tavernier@gmail.com",'2');

//add ondelete => cascade on studentcousebatches table

#2

DELIMITER $$

create procedure deleteAssignedTeacher(in emailAddress varchar(100),in recordId smallint)

begin

delete from ceiteachers where tEmail = emailAddress and id = recordId;

end $$

DELIMITER ;

/\*

by default it is blocking it from deleting a record that has a relationship but on thinking of such,we will over ride that to cascade instead.

As a result, it will delete both the parent and child record relationships

\*/

//cascaded lecturerFK from restrict to cascade

call deleteAssignedTeacher("dylanbart@gmail.com",'2');

#3

DELIMITER $$

create procedure deleteSubjectCourse(in referenceID smallint)

begin

delete from subjectcourses where id = referenceID;

end $$

DELIMITER ;

TEST

call deleteSubjectCourse(4);

/\*

have to change the state from ondelete restrict to cascade on coloumn coursesFK.

be aware that it will remove all the relationships that have been referenced to it.\*/

/\*

deleting the parent will as a result remove the child. but deleting the child will not affect the parent\*/

#4

DELIMITER $$

create procedure deleteBatchDetails(in idRecordReference smallint)

begin

delete from coursebatches where id = idRecordReference;

end $$

DELIMITER ;

call deleteBatchDetails(5);

#5

DELIMITER $$

create procedure updateBatchStatus(in currentState enum('Pending','On-Going','Cancelled','Closed'),in referenceRecord smallint)

begin

update coursebatches set courseStatus = currentState where id =referenceRecord;

end $$

DELIMITER ;

TEST

call updateBatchStatus("Cancelled",4);