CS2102 Database Systems

Team 16

Topic B: Course Registration System

Huang Xuankun A0185827N
Wei Feng A0185740B
Yang Chenglong A0167107H
Ye Guoquan A0188947A

Table of Contents

1. Introduction

ShadowModreg is a module registration website prototype that allows students to bid lectures and ballot tutorials. This report details the database design and functions of ShadowModreg.

1.1 Job Allocation and Design Specifications

This application consists of a web frontend (including the connector to database) and a database. The job allocation among our team and tools used for developing each component are as follows:

Job Scope	Web Development		SQL Development
Members In-Charge	Yang Chenglong Huang Xuankun		Ye Guoquan Wei Feng
Specifications	Python3.7 ● flask ● flask-login ● flask-SQLAlchemy ● flask-wtform Front-end ● bootstrap ● jquery	VScode Chrome	PostgreSQL 11.1 DBeaver 5.3.4 PgAdmin 4.1

A detailed prototype deployment and configuration description (Readme.md) can be found in the root directory of the project folder. Most of the scripts are written from scratch, with the exception of downloaded scripts for external libraries as listed above in specifications.

1.2 Naming Convention and Presentation

All entity and relationship names begin with a capital letter, although in postgres it will be converted to lower-case. All attributes, functions, triggers use lower-case character and english words connected by '_'.

2. Database Design

2.1 Assumptions

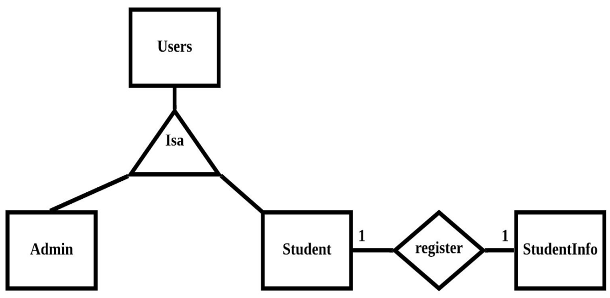
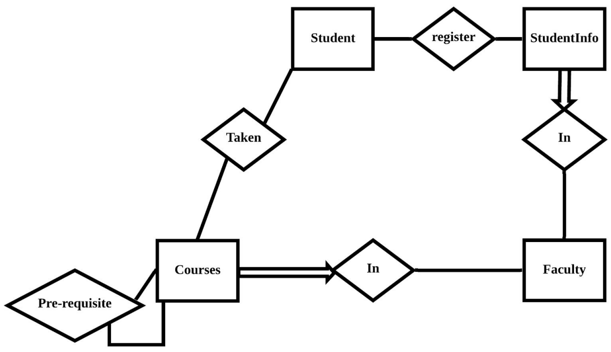
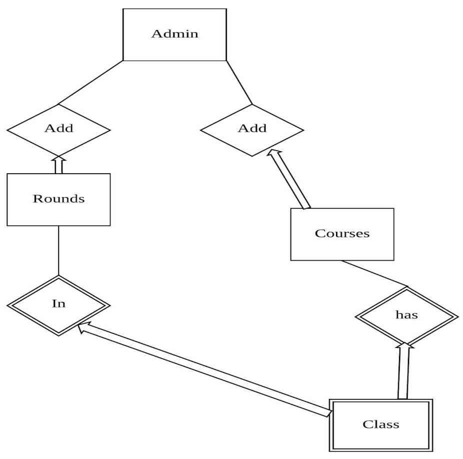
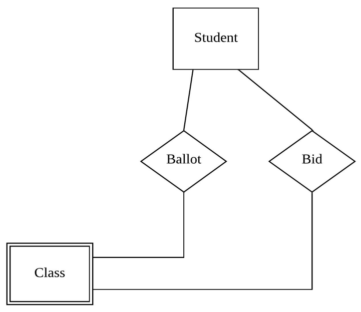
ShadowModreg has the following assumptions for its implementation and operation:

- This module registration system is used in NUS. Every student has unique matriculation number, and this table pre-exists in StudentInfo table.
- Class is classified into lecture and tutorial. Lectures are only for bidding in round 1 and round 2. Tutorials are only for balloting in round 3.
- Class is instantiated from courses every semesters and are then added to different rounds for students to register(bid or ballot)
- Admin has the access right to all parts of database. However, admin can only delete or modify the course and round he created. When there is a tie in selecting successful bid and ballot, admin will manually select the successful one.
- Admin will delete the unsuccessful bid before the start of the next round.
- Security is not our concern for this prototype.

With these assumptions in mind, next section proceeds to illustrate our design decisions for database setup.

2.2 Entity-relationship diagram

Our ER diagram is presented parts by parts, with explanation about constraints and entity sets that may not be intuitive on first glance.

 Users Isa Admin Student 1 register 1 StudentInfo	1. Users can be either admin or student. Admin has more privileges than student. 2. StudentInfo consist of basic information about student, including unique matric no and their real name Relation: Register: New student needs to register an account for the Web using their matric no.
 Student register StudentInfo Taken In Courses In Faculty Pre-requisite	1. StudentInfo has the student's home faculty. Courses has the course's host faculty. Student in the same faculty as a course has priority in breaking-tie in bidding for the course. Relation: Prerequisite: it stores the pre-requisites of a course. Taken: it stores the courses a student has taken.
 Admin Add Add Rounds Courses In has Class	1. Rounds table and Courses table refer to Admin. This is to ensure that an admin can only edit the round and course he created. 2. Rounds table is needed as the rounds at different semester have different information(eg. round start and end time) that need to be set by admin. Similar reasons for Courses table. 3. Class is weak entity to both Rounds and Courses. This is because one round can have multiple same courses and one courses can exist in multiple rounds.
	1. Student has two relationships with Class as two systems are used in selecting class, namely Bid for lectures and Ballot for tutorials. Relation: Bid: students can bid lectures at round 1 and round 2. Ballot: students can ballot tutorials at round 3.

The picture below shows the complete ER diagram.

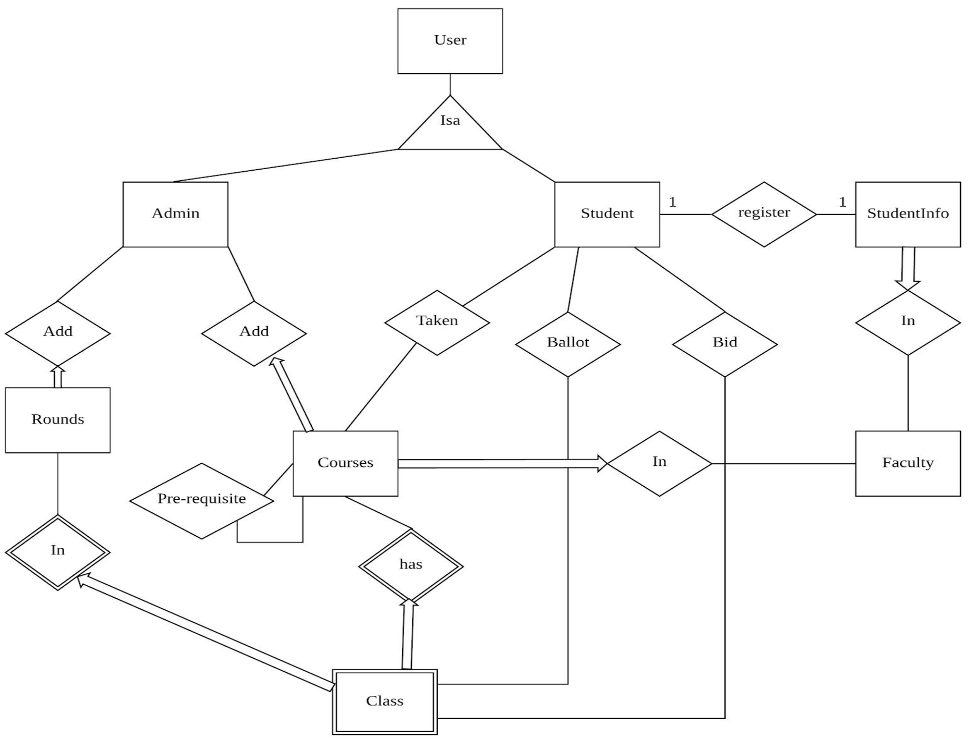


Figure 1: Full ER Diagram

The picture below shows the attributes of each set in ER diagram.

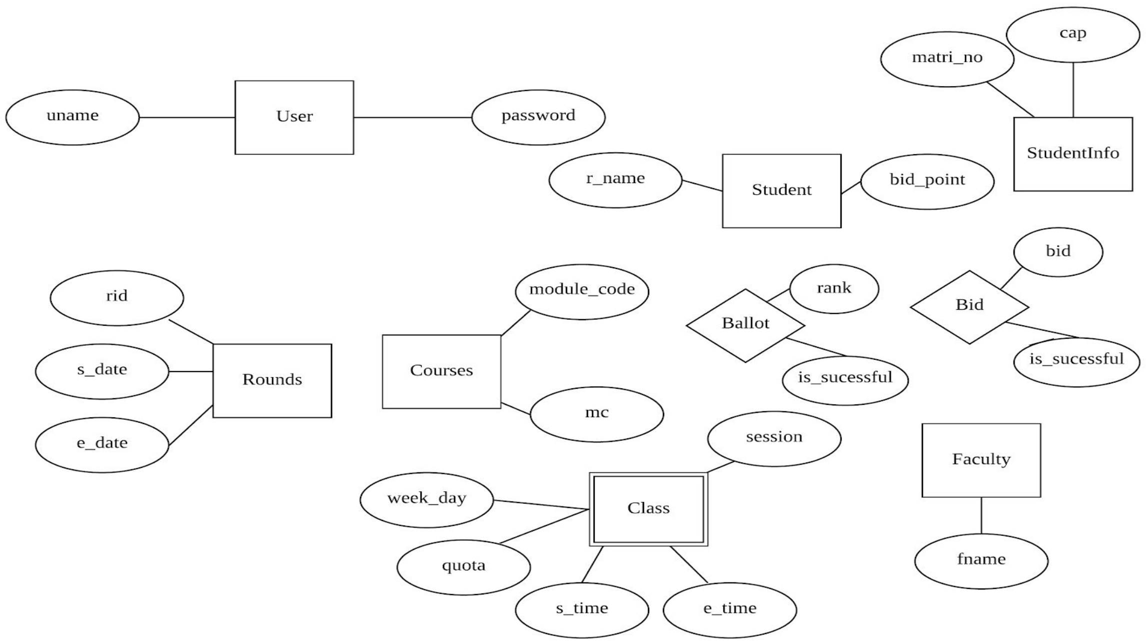


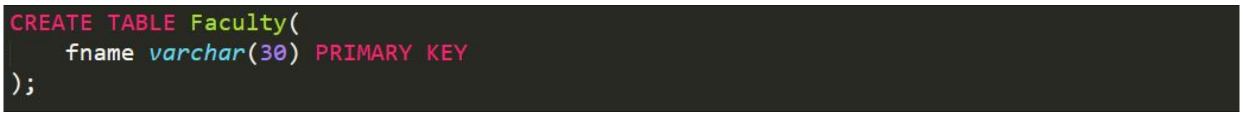
Figure 2: Attributes

2.3 Relational schema

This section shows the sql schema used in the database. Table names are in italic with grey font color, such as *User*.

Faculty

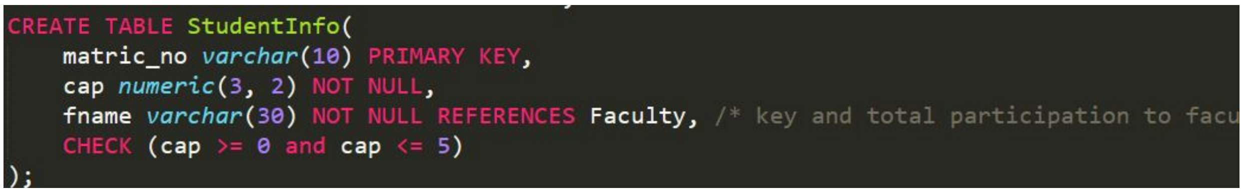
This table captures *Faculty* entity set.



```
CREATE TABLE Faculty(
    fname varchar(30) PRIMARY KEY
);
```

StudentInfo

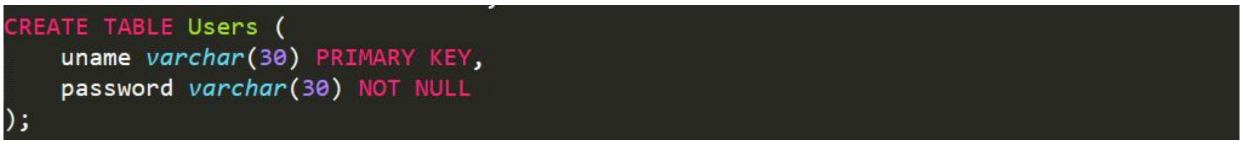
This table captures *StudentInfo* entity set and *In* relationship between *Faculty* and *StudentInfo*.



```
CREATE TABLE StudentInfo(
    matric_no varchar(10) PRIMARY KEY,
    cap numeric(3, 2) NOT NULL,
    fname varchar(30) NOT NULL REFERENCES Faculty, /* key and total participation to facu
    CHECK (cap >= 0 and cap <= 5)
);
```

Users

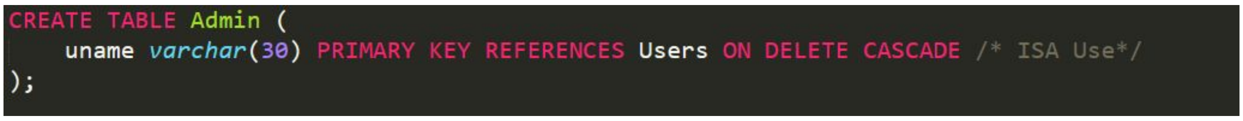
This table captures *Users* entity set.



```
CREATE TABLE Users (
    uname varchar(30) PRIMARY KEY,
    password varchar(30) NOT NULL
);
```

Admin

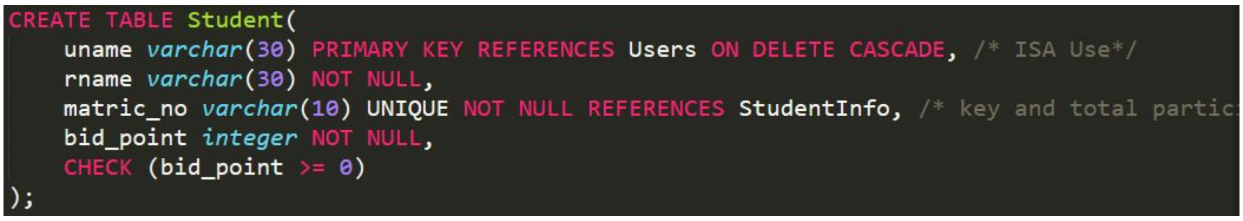
This table captures *Admin* entity set and ISA relationship between *Users* and *Admin*.



```
CREATE TABLE Admin (
    uname varchar(30) PRIMARY KEY REFERENCES Users ON DELETE CASCADE /* ISA Use*/
);
```

Student

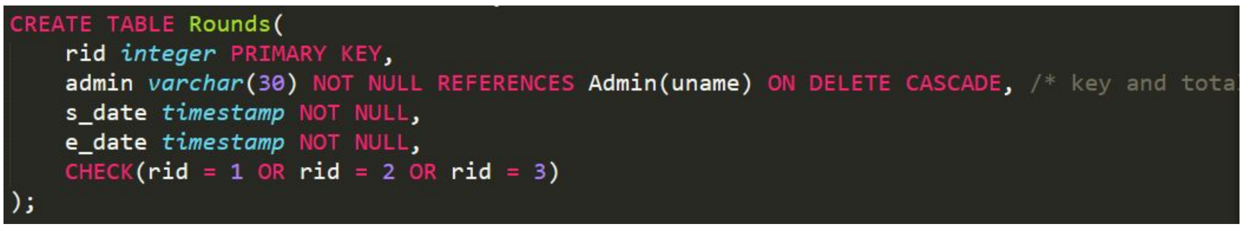
This table captures *Student* entity set, *ISA* relationship between *Users* and *Student*, and *register* relationship between *StudentInfo* and *Student*.



```
CREATE TABLE Student(
    uname varchar(30) PRIMARY KEY REFERENCES Users ON DELETE CASCADE, /* ISA Use*/
    rname varchar(30) NOT NULL,
    matric_no varchar(10) UNIQUE NOT NULL REFERENCES StudentInfo, /* key and total partic
    bid_point integer NOT NULL,
    CHECK (bid_point >= 0)
);
```

Rounds

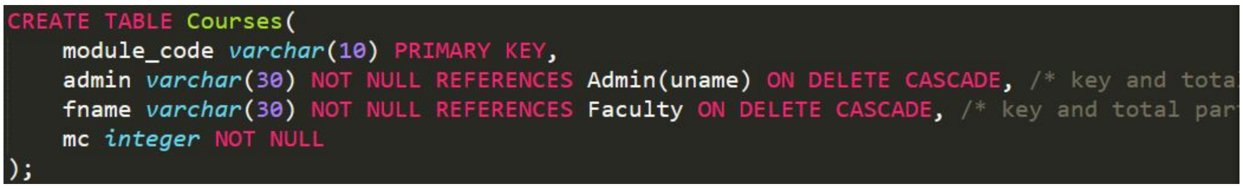
This table captures *Rounds* entity set and *Add* relationship between *Admin* and *Rounds*.



```
CREATE TABLE Rounds(
    rid integer PRIMARY KEY,
    admin varchar(30) NOT NULL REFERENCES Admin(uname) ON DELETE CASCADE, /* key and tota
    s_date timestamp NOT NULL,
    e_date timestamp NOT NULL,
    CHECK(rid = 1 OR rid = 2 OR rid = 3)
);
```

Courses

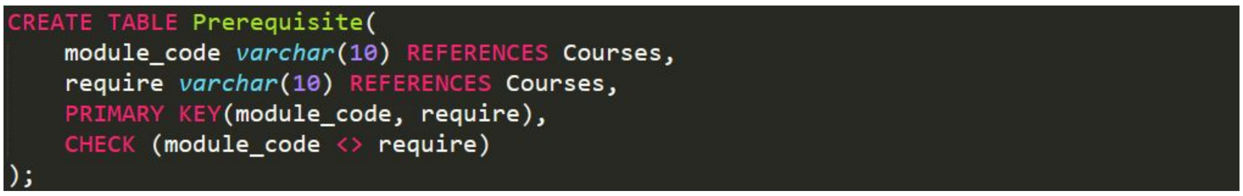
This table captures *Courses* entity set, *Add* relationship between *Admin* and *Courses* and *In* relationship between *Faculty* and *Courses*.



```
CREATE TABLE Courses(
    module_code varchar(10) PRIMARY KEY,
    admin varchar(30) NOT NULL REFERENCES Admin(uname) ON DELETE CASCADE, /* key and tota
    fname varchar(30) NOT NULL REFERENCES Faculty ON DELETE CASCADE, /* key and total par
    mc integer NOT NULL
);
```

Prerequisite

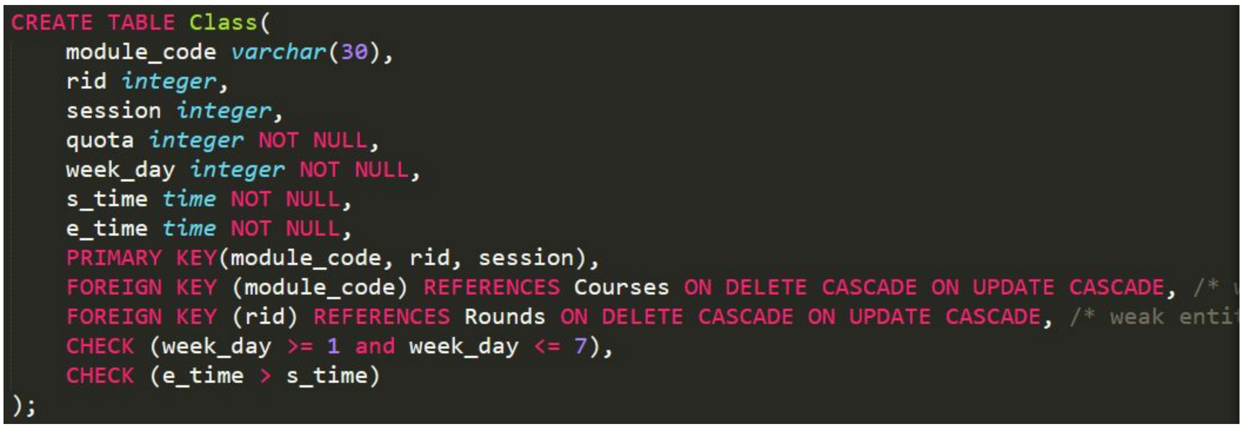
This table captures *prerequisite* relationship between *Courses* and *Courses*.



```
CREATE TABLE Prerequisite(
    module_code varchar(10) REFERENCES Courses,
    require varchar(10) REFERENCES Courses,
    PRIMARY KEY(module_code, require),
    CHECK (module_code <> require)
);
```

Class

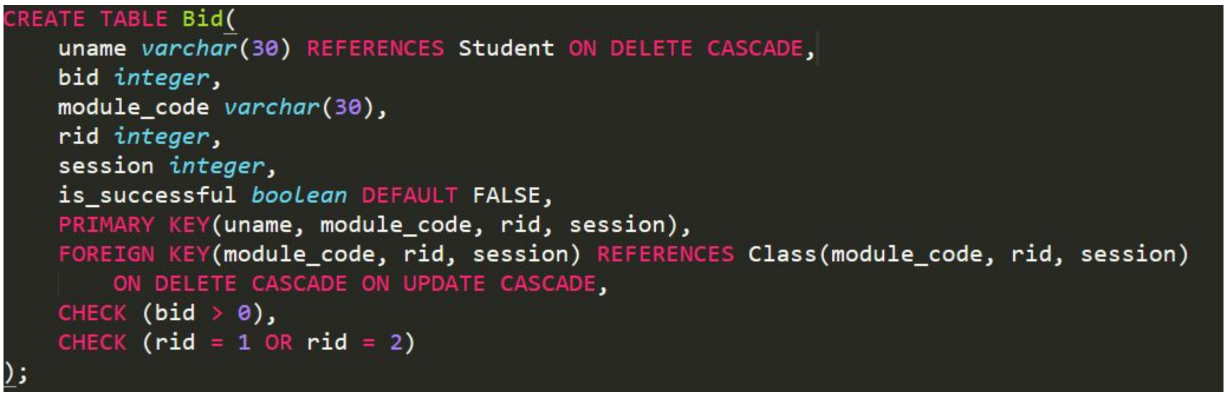
This table captures *Class* entity, *Has* relationship between *Courses* and *Class* and *In* relationship between *Rounds* and *Class* .



```
CREATE TABLE Class(
    module_code varchar(30),
    rid integer,
    session integer,
    quota integer NOT NULL,
    week_day integer NOT NULL,
    s_time time NOT NULL,
    e_time time NOT NULL,
    PRIMARY KEY(module_code, rid, session),
    FOREIGN KEY (module_code) REFERENCES Courses ON DELETE CASCADE ON UPDATE CASCADE, /* w
    FOREIGN KEY (rid) REFERENCES Rounds ON DELETE CASCADE ON UPDATE CASCADE, /* weak enti
    CHECK (week_day >= 1 and week_day <= 7),
    CHECK (e_time > s_time)
);
```

Bid

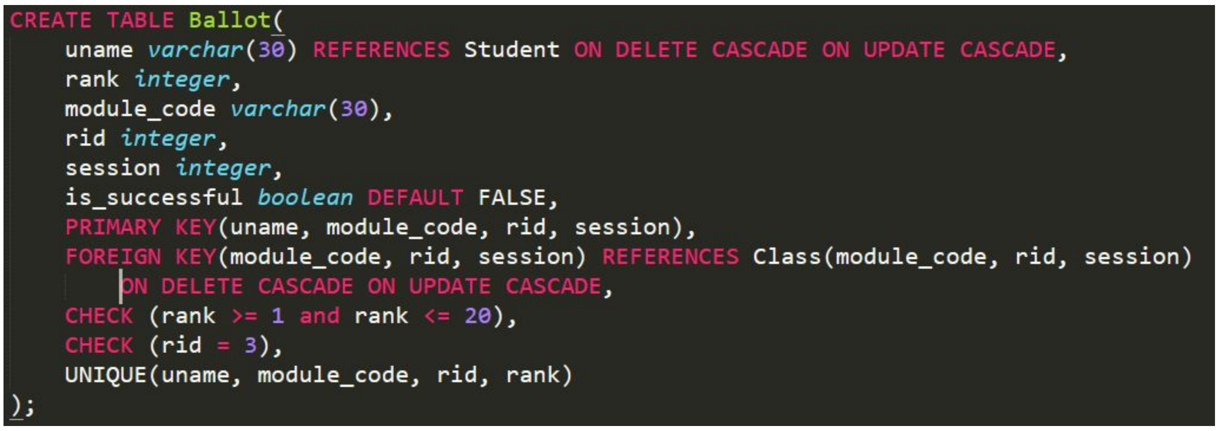
This table captures *bid* relationship between *Class* and *Student*.



```
CREATE TABLE Bid(
    uname varchar(30) REFERENCES Student ON DELETE CASCADE,
    bid integer,
    module_code varchar(30),
    rid integer,
    session integer,
    is_successful boolean DEFAULT FALSE,
    PRIMARY KEY(uname, module_code, rid, session),
    FOREIGN KEY(module_code, rid, session) REFERENCES Class(module_code, rid, session)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK (bid > 0),
    CHECK (rid = 1 OR rid = 2)
);
```

Ballot

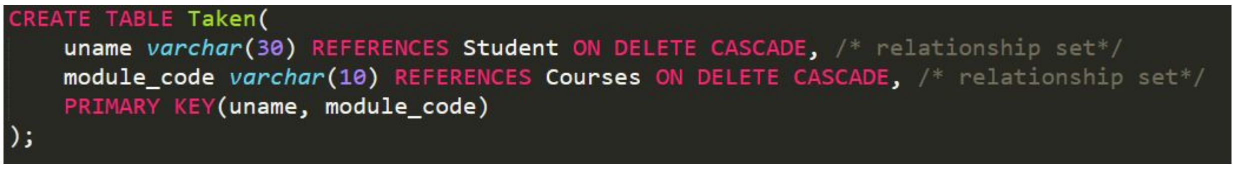
This table captures *Ballot* relationship between *Class* and *Student*.



```
CREATE TABLE Ballot(
    uname varchar(30) REFERENCES Student ON DELETE CASCADE ON UPDATE CASCADE,
    rank integer,
    module_code varchar(30),
    rid integer,
    session integer,
    is_successful boolean DEFAULT FALSE,
    PRIMARY KEY(uname, module_code, rid, session),
    FOREIGN KEY(module_code, rid, session) REFERENCES Class(module_code, rid, session)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CHECK (rank >= 1 and rank <= 20),
    CHECK (rid = 3),
    UNIQUE(uname, module_code, rid, rank)
);
```

Taken

This table captures *Taken* relationship between *Courses* and *Student*.



```
CREATE TABLE Taken(
    uname varchar(30) REFERENCES Student ON DELETE CASCADE, /* relationship set*/
    module_code varchar(10) REFERENCES Courses ON DELETE CASCADE, /* relationship set*/
    PRIMARY KEY(uname, module_code)
);
```

2.4 Unreflected constraints

2.4.1 Logical constraints not reflected in both ER diagram and relational schema

The following are the constraints not reflected in both ER diagram and relational schema, and not reported as one of the non-trivial tigger constraint.

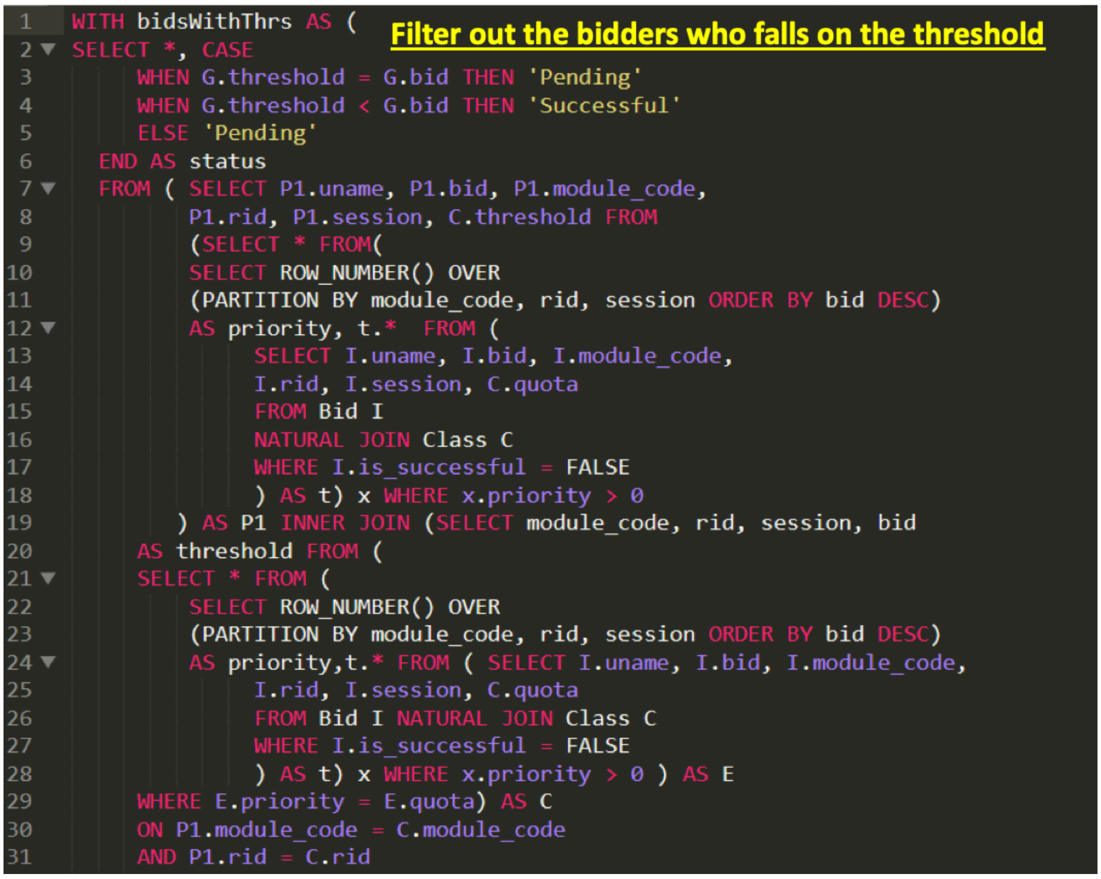
- Student can only ballot a tutorial if he has successfully bidded its corresponding lecture in previous rounds. This constraint is enforced with ***`lecture_before_tut`*** trigger (attached in appendix)
- Student cannot ballot a tutorial if the class time clash with one of the lectures the student has got in previous rounds. This constraint is enforced with ***`tutorial_clash_with_lecture`*** trigger (attached in appendix)
- Student cannot get a tutorial if it clashes with other tutorials he successfully balloted. This constraint is enforced with ***`prevent_tutorial_clash`*** trigger (attached in appendix)
- There is overlapping constraint in Users. A user can either be admin or student, but not both. This constraint is enforced in the Web.
- Student cannot bid a module that he has taken before. This constraint is enforced in the Web.

2.5 Complex queries and advanced SQL features

Feature 1

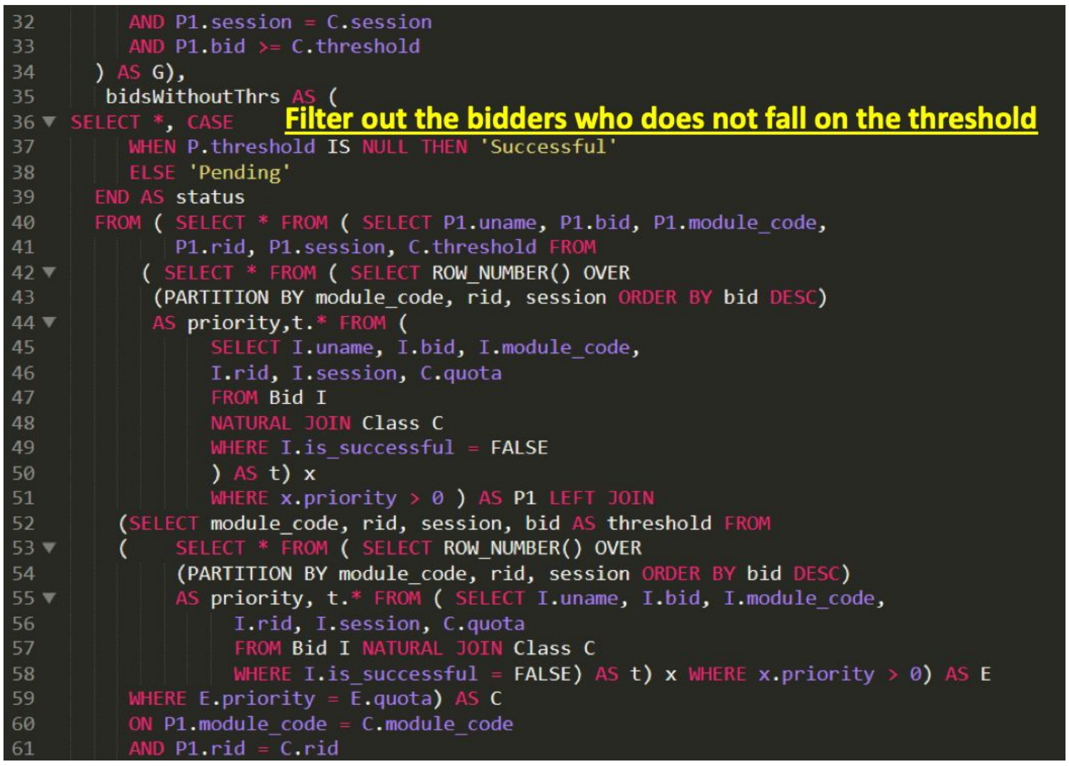
A view that contains all the information for potential successful bidders and those bidders who falls on the threshold of the minimum successful bidding line (Students whose bid point is lower than the threshold will be filtered out of the table), report a summary consisting of the following information:

1. The module and session that has been chosen by the student
2. The faculty and CAP of the student
3. The bidding point from each student
4. The minimum successful bidding line (“threshold”)
5. The current bid status of the student (Either “successful” for those who will be guaranteed a position in this module or “pending” for those who falls on the threshold with the same bid point and require another round of review from Admin)



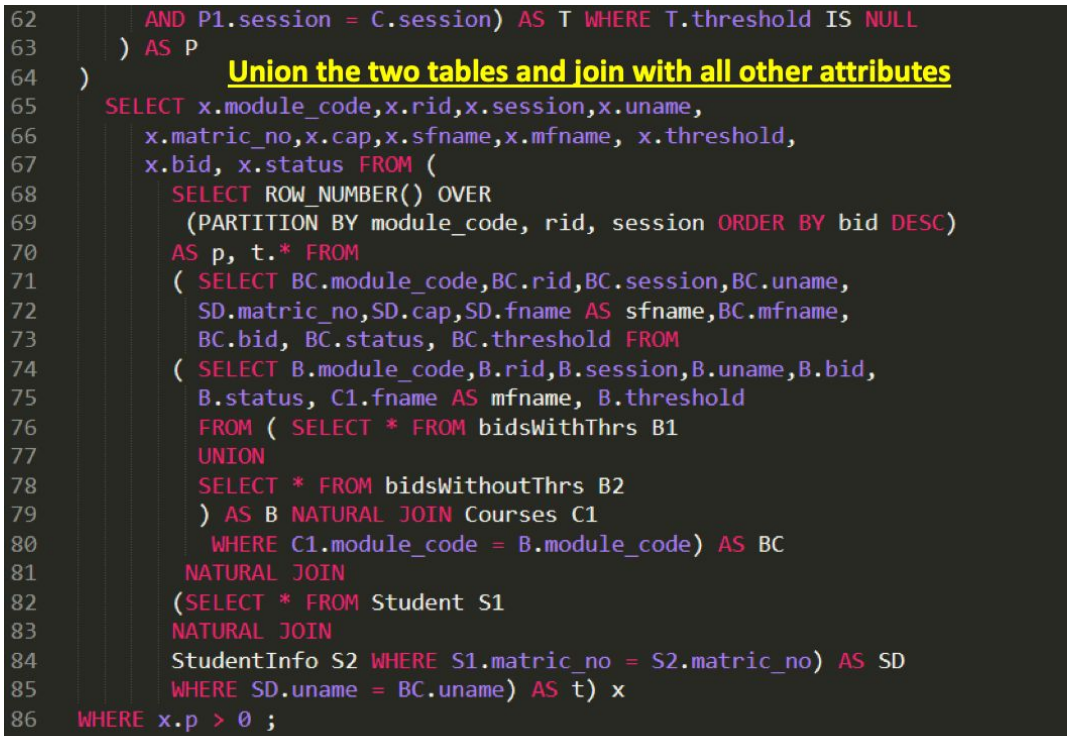
Filter out the bidders who falls on the threshold

```sql
WITH bidsWithThrs AS (
SELECT *, CASE
    WHEN G.threshold = G.bid THEN 'Pending'
    WHEN G.threshold < G.bid THEN 'Successful'
    ELSE 'Pending'
  END AS status
  FROM ( SELECT P1.uname, P1.bid, P1.module_code,
        P1.rid, P1.session, C.threshold FROM
        (SELECT * FROM(
        SELECT ROW_NUMBER() OVER
        (PARTITION BY module_code, rid, session ORDER BY bid DESC)
        AS priority, t.*  FROM (
             SELECT I.uname, I.bid, I.module_code,
             I.rid, I.session, C.quota
             FROM Bid I
             NATURAL JOIN Class C
             WHERE I.is_successful = FALSE
             ) AS t) x WHERE x.priority > 0
      ) AS P1 INNER JOIN (SELECT module_code, rid, session, bid
    AS threshold FROM (
    SELECT * FROM (
        SELECT ROW_NUMBER() OVER
        (PARTITION BY module_code, rid, session ORDER BY bid DESC)
        AS priority,t.* FROM ( SELECT I.uname, I.bid, I.module_code,
             I.rid, I.session, C.quota
             FROM Bid I NATURAL JOIN Class C
             WHERE I.is_successful = FALSE
             ) AS t) x WHERE x.priority > 0 ) AS E
    WHERE E.priority = E.quota) AS C
    ON P1.module_code = C.module_code
    AND P1.rid = C.rid
```



Filter out the bidders who does not fall on the threshold

```sql
        AND P1.session = C.session
        AND P1.bid >= C.threshold
    ) AS G),
     bidsWithoutThrs AS (
SELECT *, CASE
        WHEN P.threshold IS NULL THEN 'Successful'
        ELSE 'Pending'
    END AS status
    FROM ( SELECT * FROM ( SELECT P1.uname, P1.bid, P1.module_code,
                P1.rid, P1.session, C.threshold FROM
          ( SELECT * FROM ( SELECT ROW_NUMBER() OVER
           (PARTITION BY module_code, rid, session ORDER BY bid DESC)
           AS priority,t.* FROM (
                SELECT I.uname, I.bid, I.module_code,
                I.rid, I.session, C.quota
                FROM Bid I
                NATURAL JOIN Class C
                WHERE I.is_successful = FALSE
                ) AS t) x
                WHERE x.priority > 0 ) AS P1 LEFT JOIN
        (SELECT module_code, rid, session, bid AS threshold FROM
        (     SELECT * FROM ( SELECT ROW_NUMBER() OVER
              (PARTITION BY module_code, rid, session ORDER BY bid DESC)
              AS priority, t.* FROM ( SELECT I.uname, I.bid, I.module_code,
                   I.rid, I.session, C.quota
                   FROM Bid I NATURAL JOIN Class C
                   WHERE I.is_successful = FALSE) AS t) x WHERE x.priority > 0) AS E
        WHERE E.priority = E.quota) AS C
        ON P1.module_code = C.module_code
        AND P1.rid = C.rid
```



Union the two tables and join with all other attributes

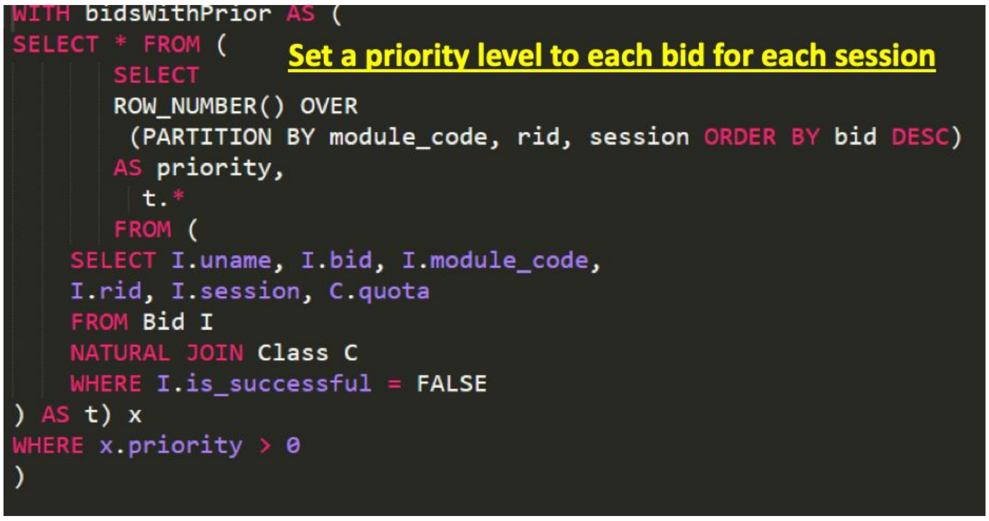
```sql
        AND P1.session = C.session) AS T WHERE T.threshold IS NULL
      ) AS P
  )
   SELECT x.module_code,x.rid,x.session,x.uname,
      x.matric_no,x.cap,x.sfname,x.mfname, x.threshold,
      x.bid, x.status FROM (
        SELECT ROW_NUMBER() OVER
         (PARTITION BY module_code, rid, session ORDER BY bid DESC)
        AS p, t.* FROM
        ( SELECT BC.module_code,BC.rid,BC.session,BC.uname,
          SD.matric_no,SD.cap,SD.fname AS sfname,BC.mfname,
          BC.bid, BC.status, BC.threshold FROM
        ( SELECT B.module_code,B.rid,B.session,B.uname,B.bid,
          B.status, C1.fname AS mfname, B.threshold
          FROM ( SELECT * FROM bidsWithThrs B1
          UNION
          SELECT * FROM bidsWithoutThrs B2
          ) AS B NATURAL JOIN Courses C1
           WHERE C1.module_code = B.module_code) AS BC
         NATURAL JOIN
        (SELECT * FROM Student S1
        NATURAL JOIN
        StudentInfo S2 WHERE S1.matric_no = S2.matric_no) AS SD
        WHERE SD.uname = BC.uname) AS t) x
  WHERE x.p > 0 ;
```

This view helps the Admin to quickly update each bid status from all the students after every round. For those students who fulfilled the same bidding criteria and falls on the threshold, Admin can easily access them through this view, and decide whether he should increase the quota to accept them all or just select a few among these students based on their faculty or CAP.

Feature 2

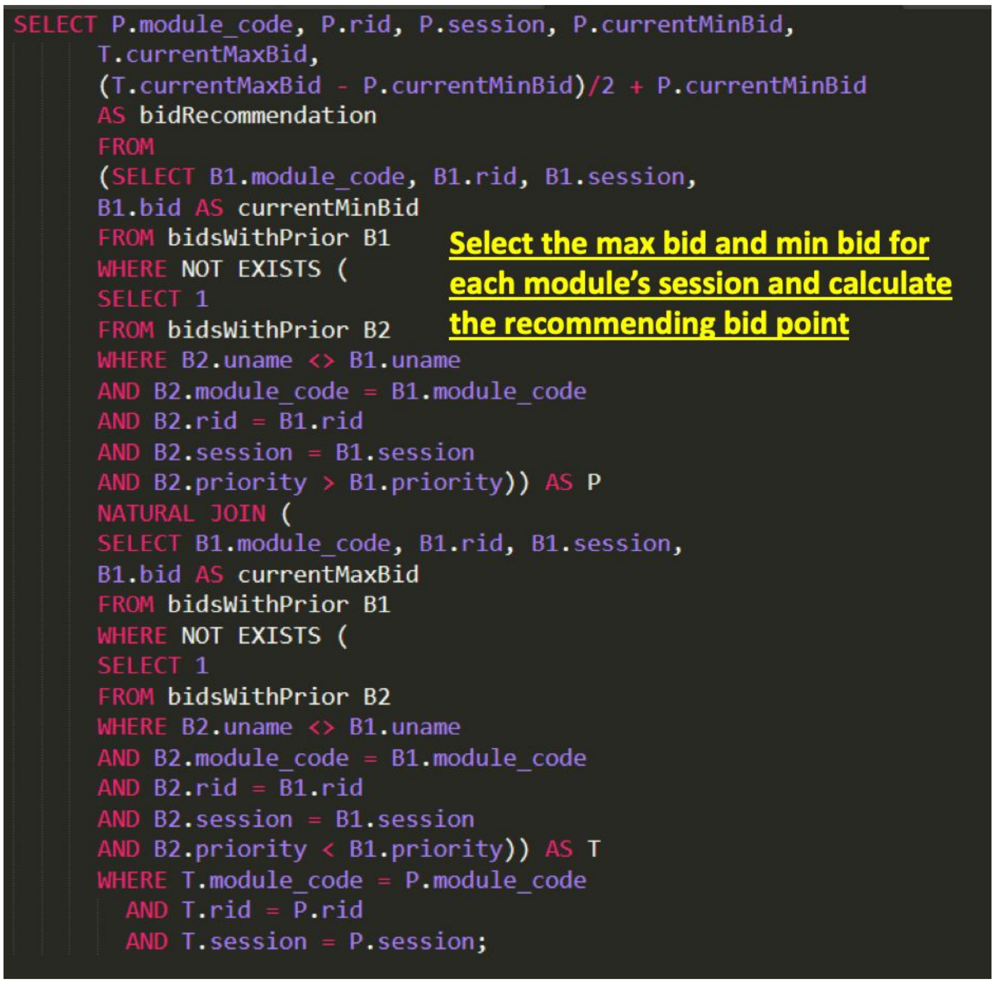
The user (students or admin) can review the current max bid point , current lowest bid point of each session of the modules as well as recommending a bid point that will potentially helps the student to win the bid.

This query can be used for recommending a suitable bid point to the bidders and helps them to evaluate how they should spend their remaining bid point.



Set a priority level to each bid for each session

```
WITH bidsWithPrior AS (
SELECT * FROM (
       SELECT
       ROW_NUMBER() OVER
        (PARTITION BY module_code, rid, session ORDER BY bid DESC)
       AS priority,
         t.*
       FROM (
    SELECT I.uname, I.bid, I.module_code,
    I.rid, I.session, C.quota
    FROM Bid I
    NATURAL JOIN Class C
    WHERE I.is_successful = FALSE
) AS t) x
WHERE x.priority > 0
)
```



Select the max bid and min bid for each module's session and calculate the recommending bid point

```
SELECT P.module_code, P.rid, P.session, P.currentMinBid,
      T.currentMaxBid,
      (T.currentMaxBid - P.currentMinBid)/2 + P.currentMinBid
      AS bidRecommendation
      FROM
      (SELECT B1.module_code, B1.rid, B1.session,
      B1.bid AS currentMinBid
      FROM bidsWithPrior B1
      WHERE NOT EXISTS (
      SELECT 1
      FROM bidsWithPrior B2
      WHERE B2.uname <> B1.uname
      AND B2.module_code = B1.module_code
      AND B2.rid = B1.rid
      AND B2.session = B1.session
      AND B2.priority > B1.priority)) AS P
      NATURAL JOIN (
      SELECT B1.module_code, B1.rid, B1.session,
      B1.bid AS currentMaxBid
      FROM bidsWithPrior B1
      WHERE NOT EXISTS (
      SELECT 1
      FROM bidsWithPrior B2
      WHERE B2.uname <> B1.uname
      AND B2.module_code = B1.module_code
      AND B2.rid = B1.rid
      AND B2.session = B1.session
      AND B2.priority < B1.priority)) AS T
      WHERE T.module_code = P.module_code
        AND T.rid = P.rid
        AND T.session = P.session;
```

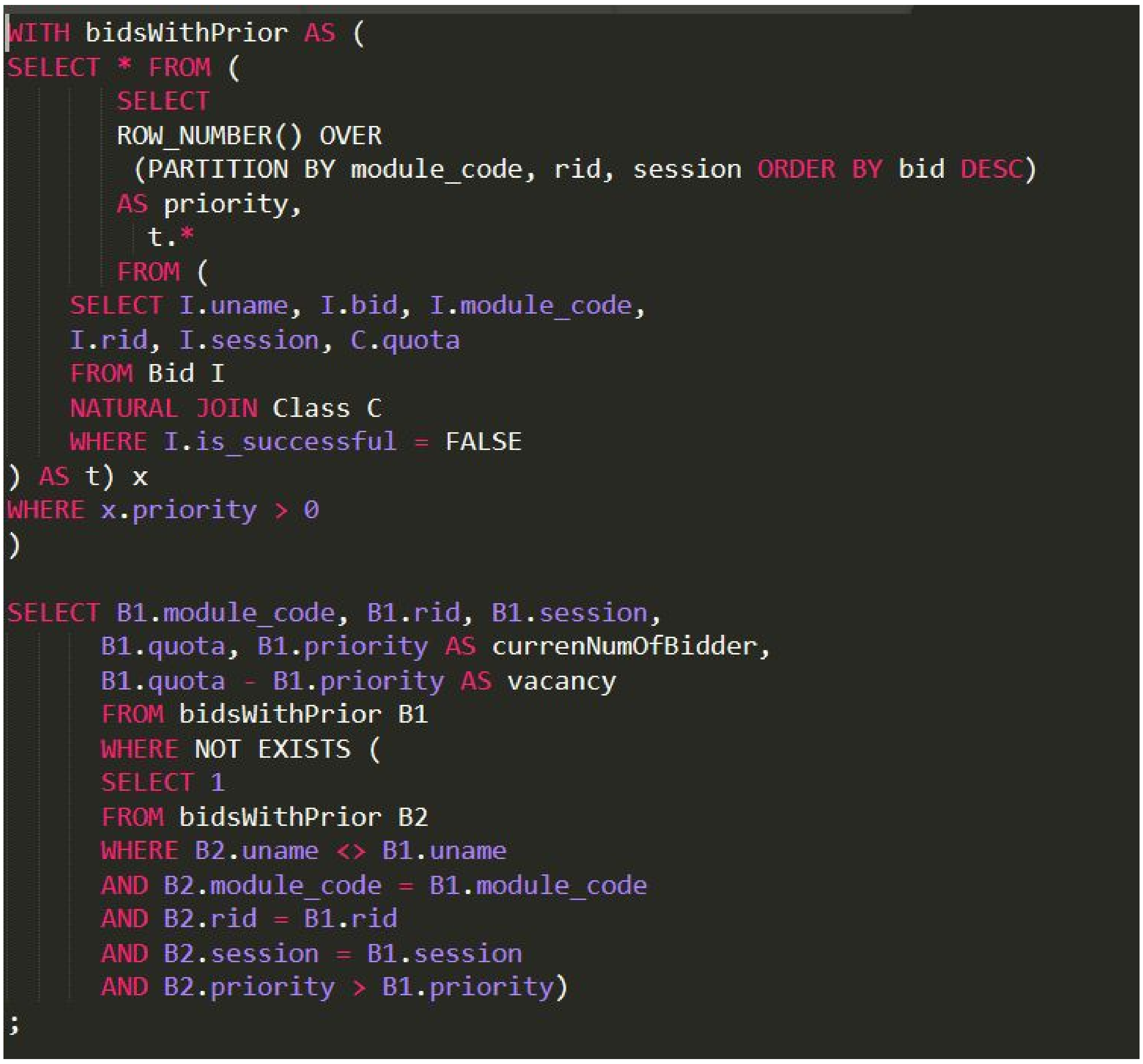
Feature 3

This query shows the user how many people are competing for the same module's session at the same time:

It contains the following information:

1. The total number of students who is bidding for the same session
2. The quota of each module's session
3. The slots that are left for the students to bid

This query will helps the student to evaluate the bidding competitiveness of the module, so they can estimate the chance of winning the bid.

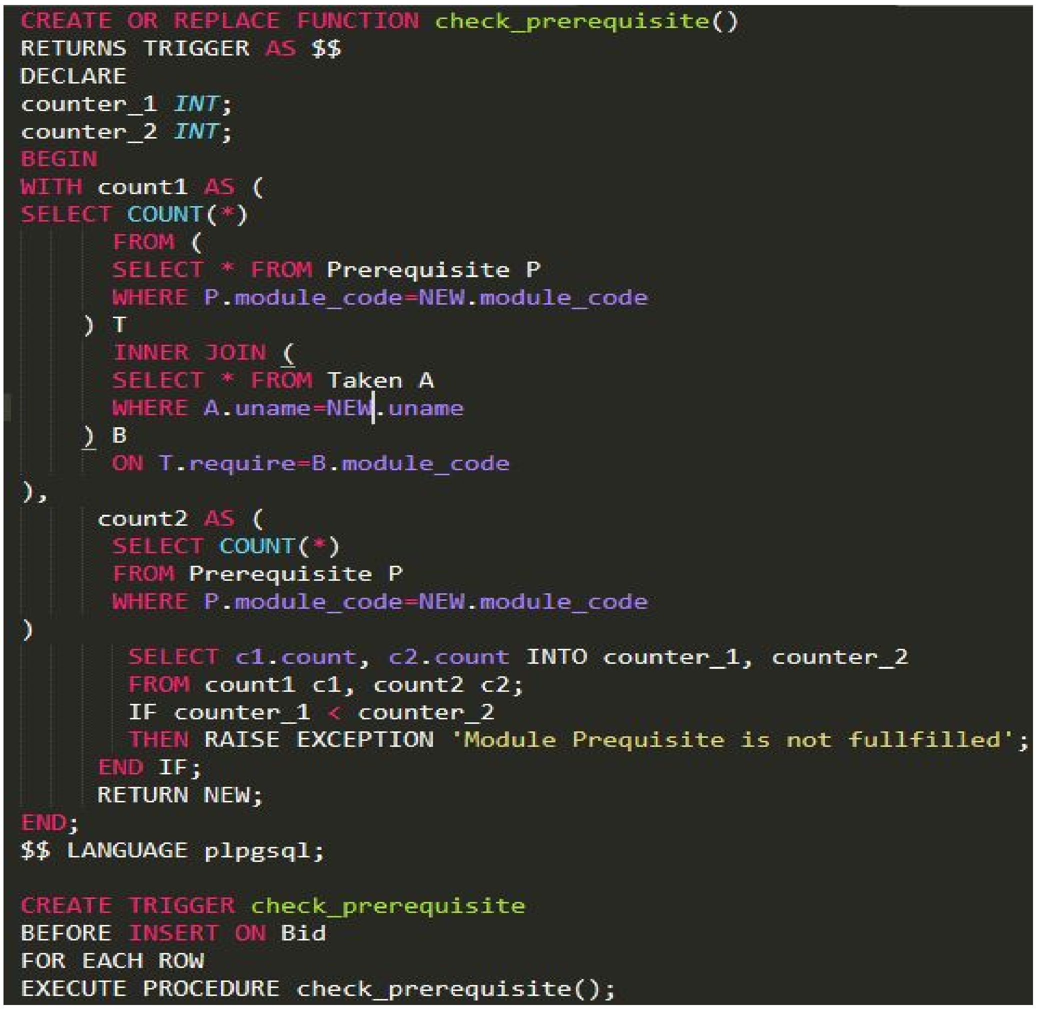


```
WITH bidsWithPrior AS (
SELECT * FROM (
        SELECT
        ROW_NUMBER() OVER
         (PARTITION BY module_code, rid, session ORDER BY bid DESC)
        AS priority,
          t.*
        FROM (
    SELECT I.uname, I.bid, I.module_code,
    I.rid, I.session, C.quota
    FROM Bid I
    NATURAL JOIN Class C
    WHERE I.is_successful = FALSE
) AS t) x
WHERE x.priority > 0
)

SELECT B1.module_code, B1.rid, B1.session,
      B1.quota, B1.priority AS currenNumOfBidder,
      B1.quota - B1.priority AS vacancy
      FROM bidsWithPrior B1
      WHERE NOT EXISTS (
      SELECT 1
      FROM bidsWithPrior B2
      WHERE B2.uname <> B1.uname
      AND B2.module_code = B1.module_code
      AND B2.rid = B1.rid
      AND B2.session = B1.session
      AND B2.priority > B1.priority)
;
```

2.6 Non-trivial integrity constraint/trigger

Constraint 1

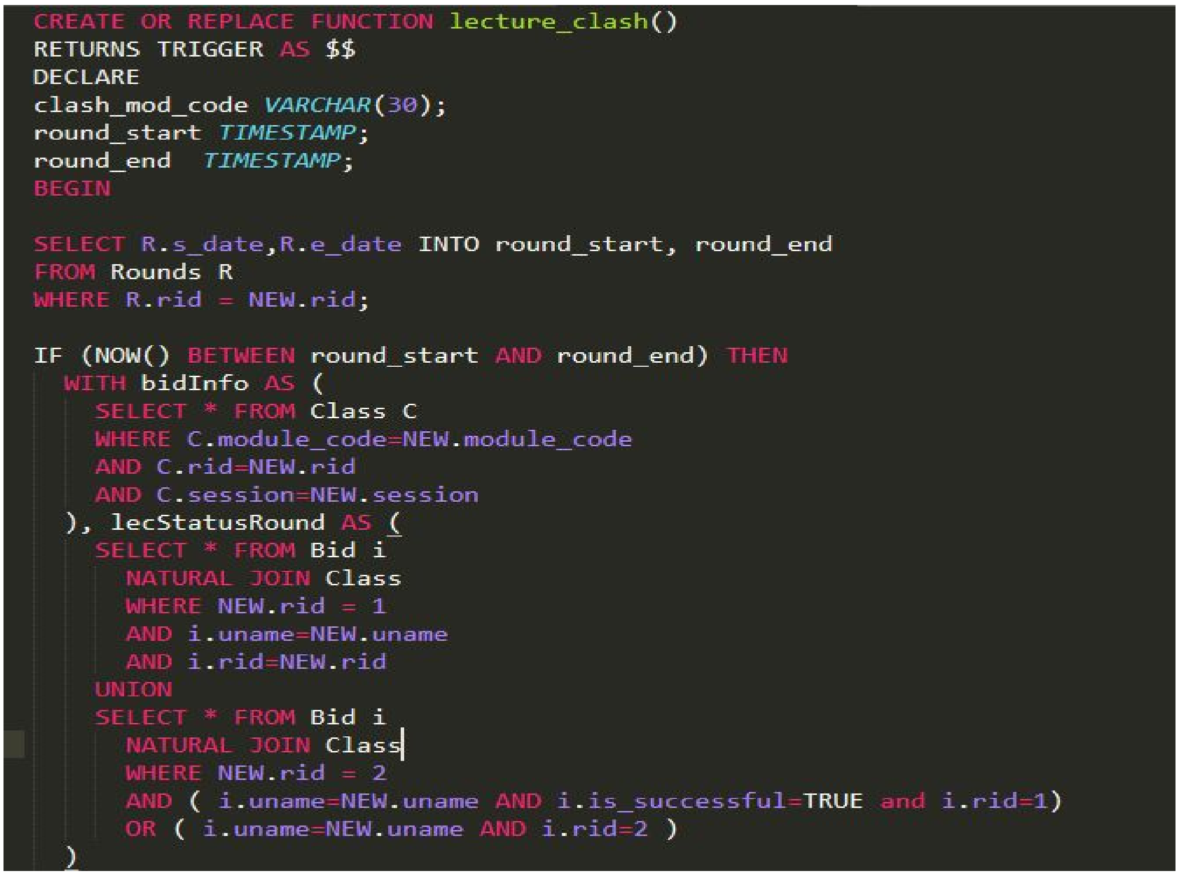
A student can only bid the lectures which they have fulfilled the prerequisite.



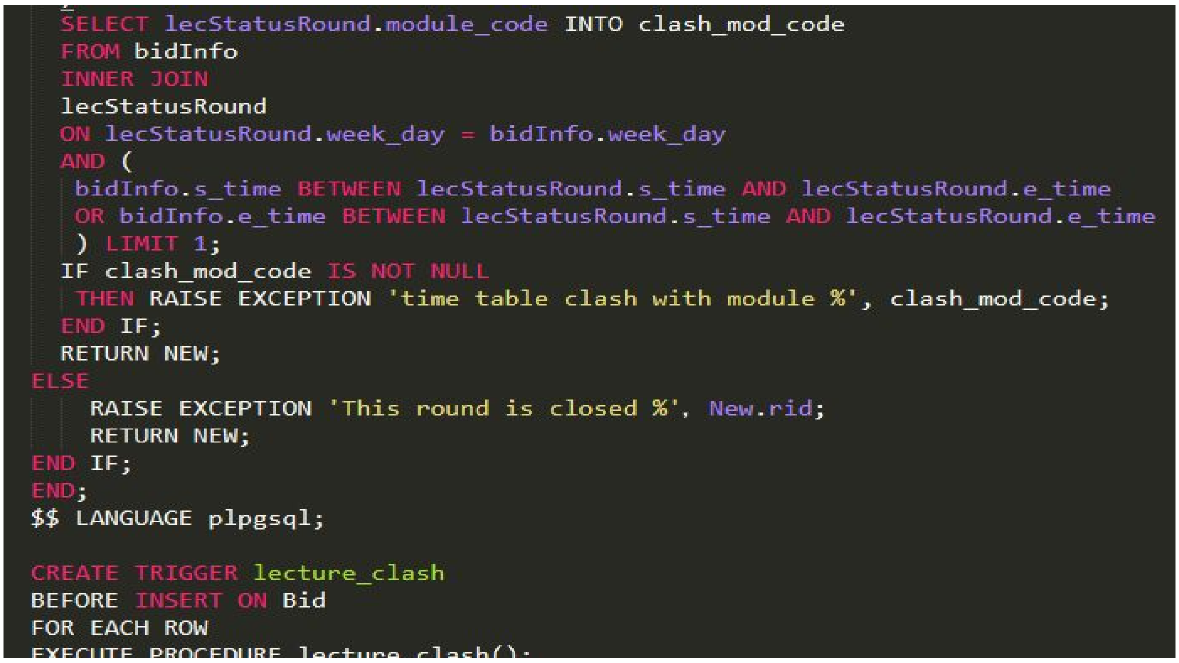
```
CREATE OR REPLACE FUNCTION check_prerequisite()
RETURNS TRIGGER AS $$
DECLARE
counter_1 INT;
counter_2 INT;
BEGIN
WITH count1 AS (
SELECT COUNT(*)
        FROM (
        SELECT * FROM Prerequisite P
        WHERE P.module_code=NEW.module_code
      ) T
        INNER JOIN (
        SELECT * FROM Taken A
        WHERE A.uname=NEW.uname
      ) B
        ON T.require=B.module_code
),
     count2 AS (
        SELECT COUNT(*)
        FROM Prerequisite P
        WHERE P.module_code=NEW.module_code
)
         SELECT c1.count, c2.count INTO counter_1, counter_2
         FROM count1 c1, count2 c2;
         IF counter_1 < counter_2
         THEN RAISE EXCEPTION 'Module Prequisite is not fullfilled';
       END IF;
       RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_prerequisite
BEFORE INSERT ON Bid
FOR EACH ROW
EXECUTE PROCEDURE check_prerequisite();
```

Constraint 2

The students are only allowed to bid within the period of each round. Student cannot bid a lecture which clashes with other lectures that he has successfully bidded in previous rounds or is bidding in the current round.



```
CREATE OR REPLACE FUNCTION lecture_clash()
RETURNS TRIGGER AS $$
DECLARE
clash_mod_code VARCHAR(30);
round_start TIMESTAMP;
round_end  TIMESTAMP;
BEGIN

SELECT R.s_date,R.e_date INTO round_start, round_end
FROM Rounds R
WHERE R.rid = NEW.rid;

IF (NOW() BETWEEN round_start AND round_end) THEN
    WITH bidInfo AS (
        SELECT * FROM Class C
        WHERE C.module_code=NEW.module_code
        AND C.rid=NEW.rid
        AND C.session=NEW.session
    ), lecStatusRound AS (
        SELECT * FROM Bid i
            NATURAL JOIN Class
            WHERE NEW.rid = 1
            AND i.uname=NEW.uname
            AND i.rid=NEW.rid
        UNION
        SELECT * FROM Bid i
            NATURAL JOIN Class
            WHERE NEW.rid = 2
            AND ( i.uname=NEW.uname AND i.is_successful=TRUE and i.rid=1)
            OR ( i.uname=NEW.uname AND i.rid=2 )
    )
```



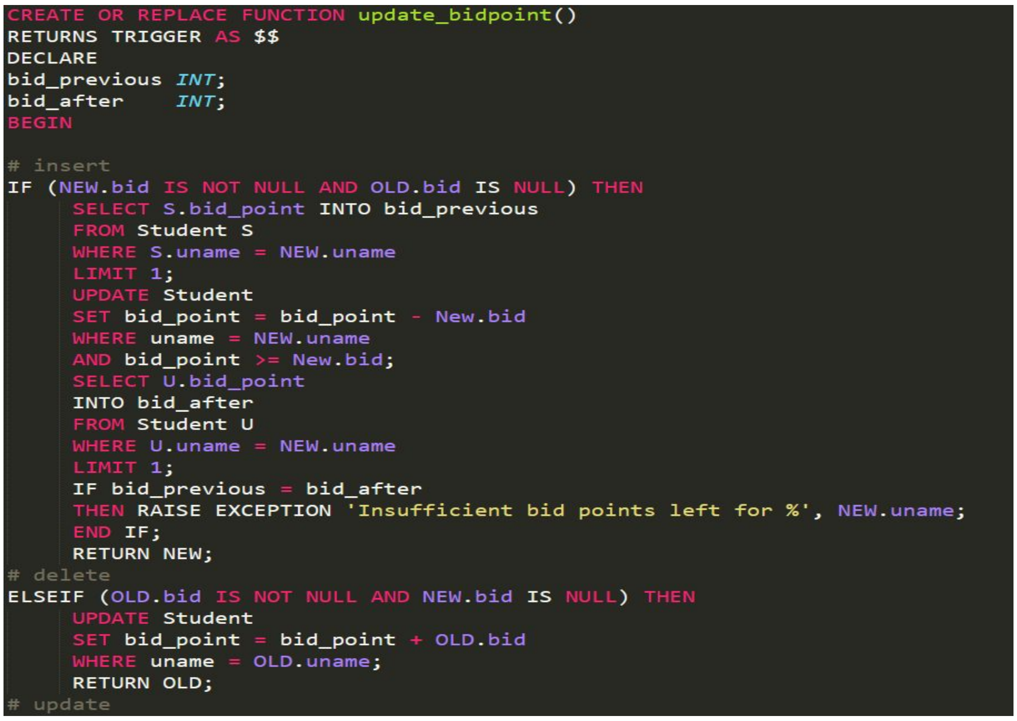
```
    SELECT lecStatusRound.module_code INTO clash_mod_code
    FROM bidInfo
    INNER JOIN
    lecStatusRound
    ON lecStatusRound.week_day = bidInfo.week_day
    AND (
     bidInfo.s_time BETWEEN lecStatusRound.s_time AND lecStatusRound.e_time
     OR bidInfo.e_time BETWEEN lecStatusRound.s_time AND lecStatusRound.e_time
     ) LIMIT 1;
    IF clash_mod_code IS NOT NULL
     THEN RAISE EXCEPTION 'time table clash with module %', clash_mod_code;
    END IF;
    RETURN NEW;
ELSE
    RAISE EXCEPTION 'This round is closed %', New.rid;
    RETURN NEW;
END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER lecture_clash
BEFORE INSERT ON Bid
FOR EACH ROW
EXECUTE PROCEDURE lecture_clash();
```

Constraint 3

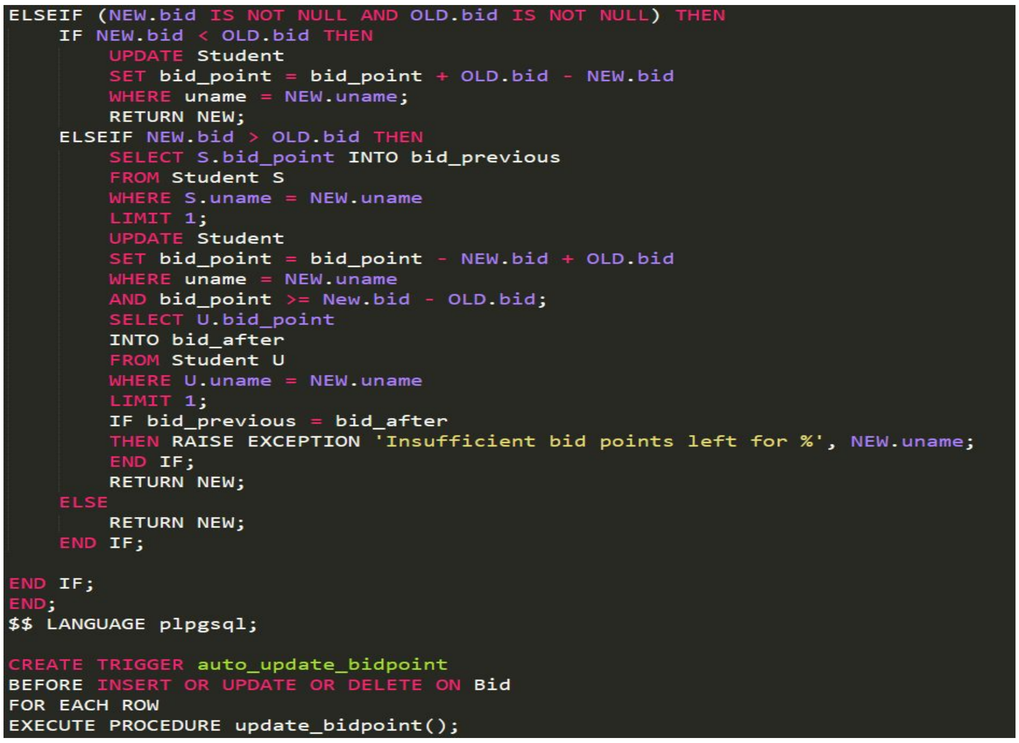
Student's bid point will be automatically updated whenever his bidding information changed.

There are three cases:

1. Student place bid on a lecture: This will check if the student has sufficient bid point for this bidding. If yes, deduct from his total bid point accordingly. If no, reject the bidding.
2. Student change bid on a lecture: If student chooses to place fewer bid, the extra bid will be refunded. If students choose to place more bid, he will go through the process as above.
3. Student withdraw bid on a lecture: The bid will be refunded.



```
CREATE OR REPLACE FUNCTION update_bidpoint()
RETURNS TRIGGER AS $$
DECLARE
bid_previous INT;
bid_after    INT;
BEGIN

# insert
IF (NEW.bid IS NOT NULL AND OLD.bid IS NULL) THEN
     SELECT S.bid_point INTO bid_previous
     FROM Student S
     WHERE S.uname = NEW.uname
     LIMIT 1;
     UPDATE Student
     SET bid_point = bid_point - New.bid
     WHERE uname = NEW.uname
     AND bid_point >= New.bid;
     SELECT U.bid_point
     INTO bid_after
     FROM Student U
     WHERE U.uname = NEW.uname
     LIMIT 1;
     IF bid_previous = bid_after
     THEN RAISE EXCEPTION 'Insufficient bid points left for %', NEW.uname;
     END IF;
     RETURN NEW;
# delete
ELSEIF (OLD.bid IS NOT NULL AND NEW.bid IS NULL) THEN
     UPDATE Student
     SET bid_point = bid_point + OLD.bid
     WHERE uname = OLD.uname;
     RETURN OLD;
# update
```



```
ELSEIF (NEW.bid IS NOT NULL AND OLD.bid IS NOT NULL) THEN
    IF NEW.bid < OLD.bid THEN
        UPDATE Student
        SET bid_point = bid_point + OLD.bid - NEW.bid
        WHERE uname = NEW.uname;
        RETURN NEW;
    ELSEIF NEW.bid > OLD.bid THEN
        SELECT S.bid_point INTO bid_previous
        FROM Student S
        WHERE S.uname = NEW.uname
        LIMIT 1;
        UPDATE Student
        SET bid_point = bid_point - NEW.bid + OLD.bid
        WHERE uname = NEW.uname
        AND bid_point >= New.bid - OLD.bid;
        SELECT U.bid_point
        INTO bid_after
        FROM Student U
        WHERE U.uname = NEW.uname
        LIMIT 1;
        IF bid_previous = bid_after
        THEN RAISE EXCEPTION 'Insufficient bid points left for %', NEW.uname;
        END IF;
        RETURN NEW;
    ELSE
        RETURN NEW;
    END IF;

END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER auto_update_bidpoint
BEFORE INSERT OR UPDATE OR DELETE ON Bid
FOR EACH ROW
EXECUTE PROCEDURE update_bidpoint();
```

3. Application Functionalities

This chapter details the functions ShadowModreg has, along with screenshots of ShadowModreg in operation. All insertion, update and deletion of records mentioned are done via stored functions in database.

3.1 For normal users

3.1.1 Registration, Sign-In and Password Reset

Anonymous visitors to ShadowModreg will be redirected to sign-in page, from which they can register (sign up) or sign in before they can access any information on ShadowModreg.

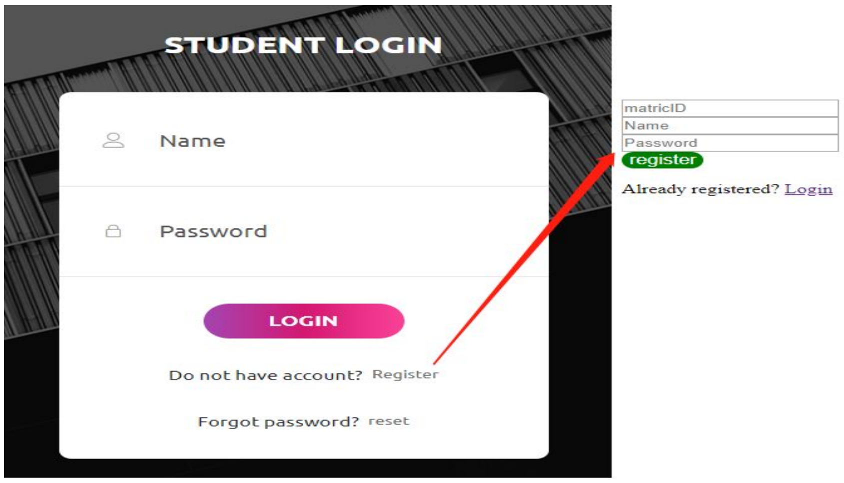
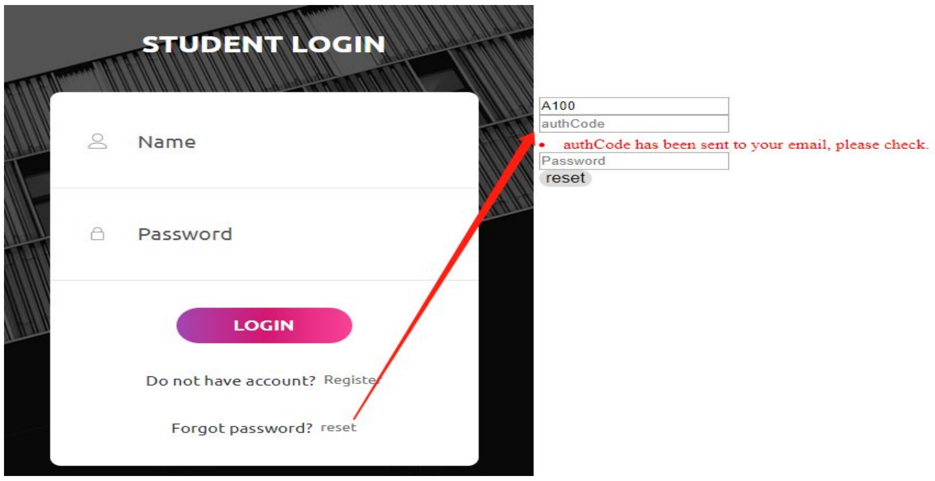


Figure Sign-in & Register

As shown above, if a student has not registered an account on the website, he could register an account by providing his matricID to register an account.



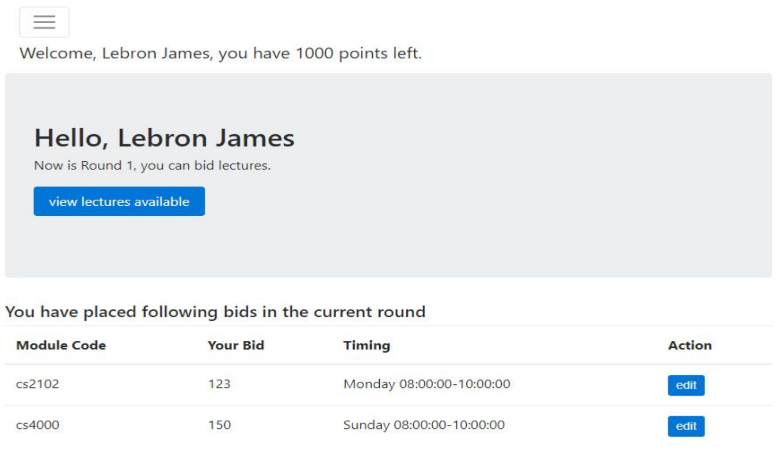
authCode has been sent to your email, please check.

Password Reset

As shown above, if a student forgot his password, he could provide his matricID, and the server will generate an authCode of 10-character length and send it to the students' email, after which student could use this authCode to reset his password.

3.1.2 Welcome Page

Once a student login to the ShadowModreg successfully, he will be redirected to the welcome page, where he could see his real name, bidding points, and situation of the current round.



Welcome, Lebron James, you have 1000 points left.

Hello, Lebron James

Now is Round 1, you can bid lectures.

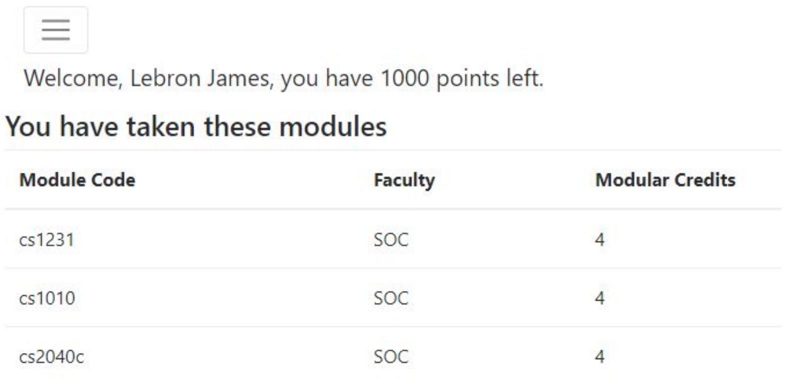
You have placed following bids in the current round

Module Code	Your Bid	Timing	Action
cs2102	123	Monday 08:00:00-10:00:00	edit
cs4000	150	Sunday 08:00:00-10:00:00	edit

Welcome Page Screenshot

3.1.3 Password changing and viewing modules taken

A student of ShadowModreg can view the module he has taken as well as change his password.



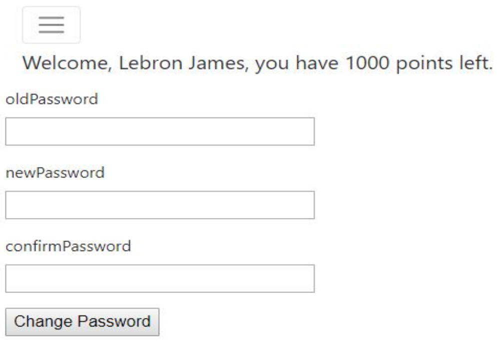
Welcome, Lebron James, you have 1000 points left.

You have taken these modules

Module Code	Faculty	Modular Credits
cs1231	SOC	4
cs1010	SOC	4
cs2040c	SOC	4

Viewing modules taken Screenshot

Students can view the modules he has taken, so he would know if he has fulfilled the prerequisite of a specific module.



Welcome, Lebron James, you have 1000 points left.

oldPassword

newPassword

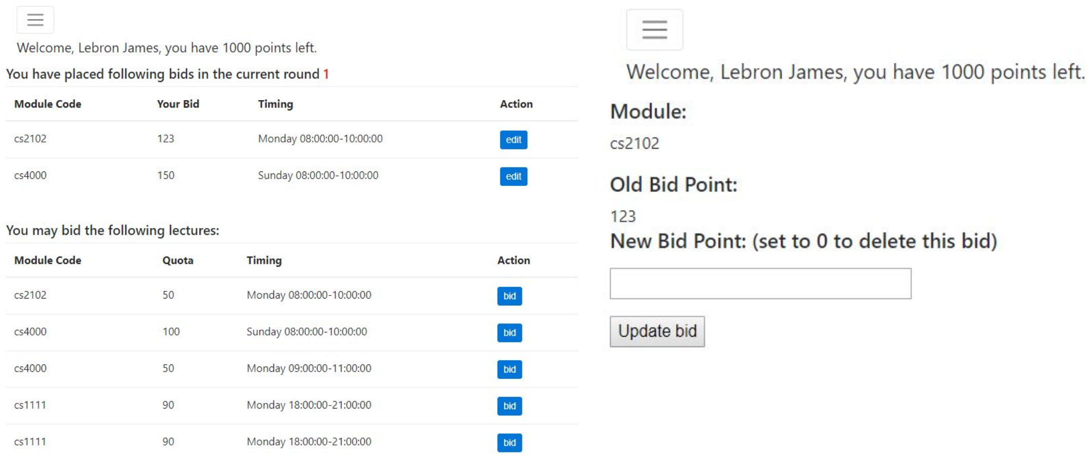
confirmPassword

Changing Password Screenshot

Students can change their passwords by providing the current password they are using.

3.1.4 Lecture bidding, editing bid, and dropping bid

ShadowModreg provides students a user-friendly lecture bidding page as shown below. Students can easily bid a lecture, edit their bidding points and drop their bids.



Welcome, Lebron James, you have 1000 points left.

You have placed following bids in the current round 1

Module Code	Your Bid	Timing	Action
cs2102	123	Monday 08:00:00-10:00:00	edit
cs4000	150	Sunday 08:00:00-10:00:00	edit

You may bid the following lectures:

Module Code	Quota	Timing	Action
cs2102	50	Monday 08:00:00-10:00:00	bid
cs4000	100	Sunday 08:00:00-10:00:00	bid
cs4000	50	Monday 09:00:00-11:00:00	bid
cs1111	90	Monday 18:00:00-21:00:00	bid
cs1111	90	Monday 18:00:00-21:00:00	bid

Welcome, Lebron James, you have 1000 points left.

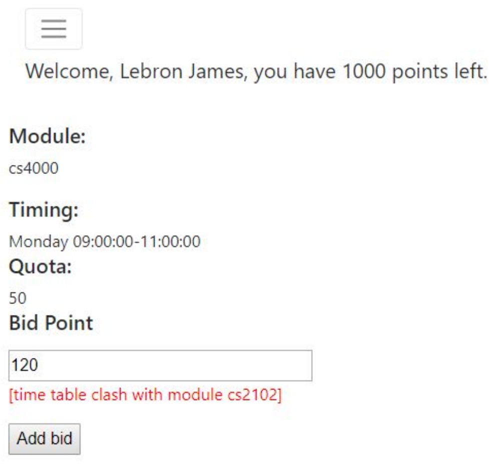
Module:
cs2102

Old Bid Point:
123

New Bid Point: (set to 0 to delete this bid)

Lecture Bidding/Bid Editing

Students can edit his bidding points for a specific lecture as well as dropping a bid by setting the bidding point to 0. After editing the bidding points for a specific lecture, his total bidding points will be adjusted accordingly.



Welcome, Lebron James, you have 1000 points left.

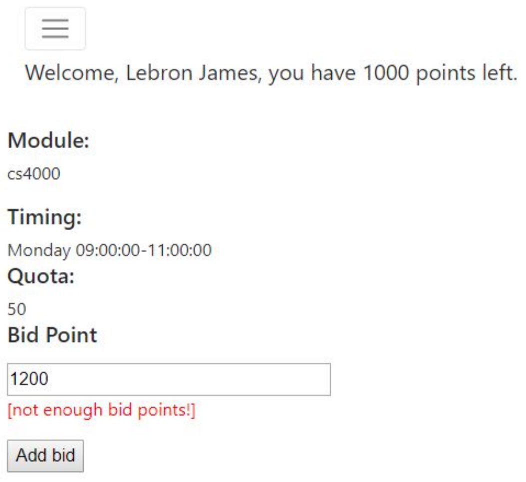
Module:
cs4000

Timing:
Monday 09:00:00-11:00:00

Quota:
50

Bid Point
120

[time table clash with module cs2102]



Welcome, Lebron James, you have 1000 points left.

Module:
cs4000

Timing:
Monday 09:00:00-11:00:00

Quota:
50

Bid Point
1200

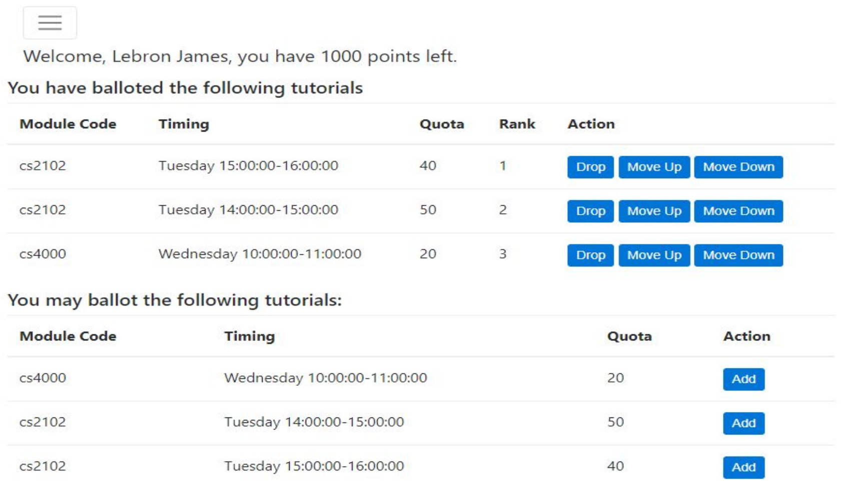
[not enough bid points!]

Lecture Bidding Page Screenshot

Students can bid a lecture and will be prompted if it clashes with any modules or his bidding points is not enough to place the bid.

3.1.5 Tutorial balloting, ranking and dropping ballot

Students may ballot their corresponding tutorial slots in round 3 after successfully bidding the lectures.



Welcome, Lebron James, you have 1000 points left.

You have balloted the following tutorials

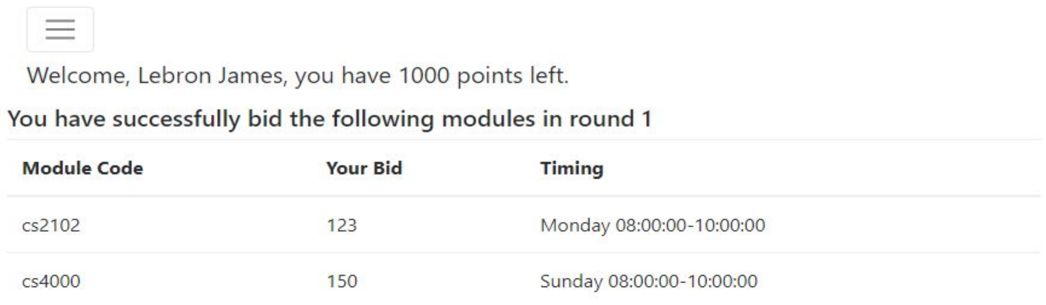
Module Code	Timing	Quota	Rank	Action
cs2102	Tuesday 15:00:00-16:00:00	40	1	Drop Move Up Move Down
cs2102	Tuesday 14:00:00-15:00:00	50	2	Drop Move Up Move Down
cs4000	Wednesday 10:00:00-11:00:00	20	3	Drop Move Up Move Down

You may ballot the following tutorials:

Module Code	Timing	Quota	Action
cs4000	Wednesday 10:00:00-11:00:00	20	Add
cs2102	Tuesday 14:00:00-15:00:00	50	Add
cs2102	Tuesday 15:00:00-16:00:00	40	Add

Tutorial Balloting Page Screenshot

3.1.6 View results in the ended round



Welcome, Lebron James, you have 1000 points left.

You have successfully bid the following modules in round 1

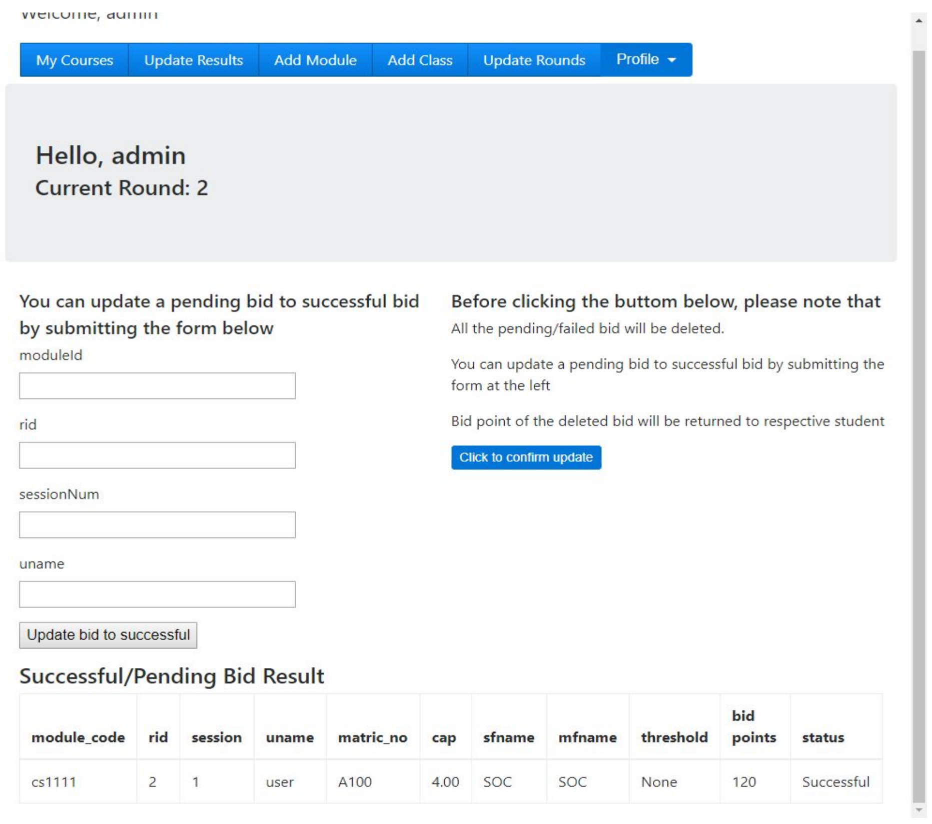
Module Code	Your Bid	Timing
cs2102	123	Monday 08:00:00-10:00:00
cs4000	150	Sunday 08:00:00-10:00:00

Students can view the results for their bidding in the previous round.

3.2 For admin users

3.2.1 Generate bidding/balloting results after each round

Admins are given authority to update the bid to successful automatically based on predefined criteria after the round ends or manually update the specific bid to successful. Admins are not supposed to update the bidding/balloting result within the round period so that to be fair to students who have yet to bid/ballot for that round.



Hello, admin
Current Round: 2

You can update a pending bid to successful bid by submitting the form below

moduleId

rid

sessionNum

uname

Before clicking the buttom below, please note that

All the pending/failed bid will be deleted.

You can update a pending bid to successful bid by submitting the form at the left

Bid point of the deleted bid will be returned to respective student

Successful/Pending Bid Result

module_code	rid	session	uname	matric_no	cap	sfname	mfname	threshold	bid points	status
cs1111	2	1	user	A100	4.00	SOC	SOC	None	120	Successful

Admin Update Result Page

3.2.2 Add module/class

Admins are allowed to create module and class by filling in relevant info. The classes created (including tutorial class) will be used in bidding/bollating.

3.2.3 View my module/class

Admin can view all modules/classes managed by him/her. All modules/classes are displayed in table form in the My Courses Page.

3.2.4 Update round

Admins can change the starting and ending time of the round, such that extending the ending time of a round is possible for any unexpected situation.

4. Conclusion and Reflection

In summary, we have presented the basic functions and interfaces of ShadowModreg, a workable prototype of module registration website application, along with certain implementation details. This chapter discusses the three major challenges we encountered during development.

Initially, we had difficulty in coming up with the sufficient number of entities and relationship sets to meet the requirement. We were constrained by the scope of project functions stated in the requirements but later on was able to overcome by recalling the past CORS system, which uses a bidding system for lectures and balloting system for tutorials.

The second challenge we were facing is that none of us has web development experience, so we split into two groups, those who are more familiar with python will do the web development, and the rest will do the sql development.

In conclusion, the project has deepened our understanding about database schema design and implementation. Many of the lessons in lectures are used when we are working on the project. We are grateful to this learning journey