CS301, DBMS
Database Management and Systems

Indian Institute of Technology, Ropar

Design Documentation for Phase A of the project

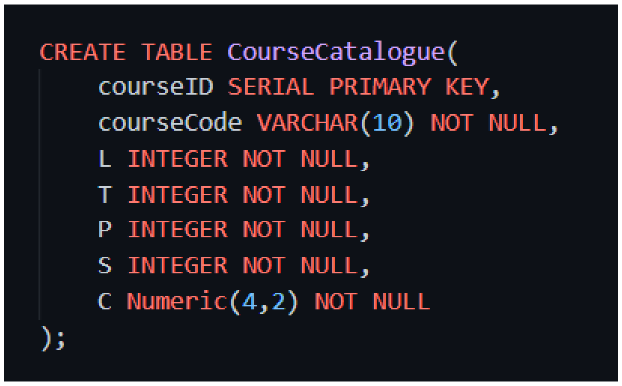
Topic: Design of an Academic System

Instructor: Dr. Vishwanath Gunturi
Date of Submission: 28-10-2021

Team Members
Vasu Bansal (2019csb1130@iitrpr.ac.in)
Pradeep Kumar (2019csb1107@iitrpr.ac.in)
Anant Dhakkad (2019csb1070@iitrpr.ac.in)

Details about the Tables and their related Stored Procedures

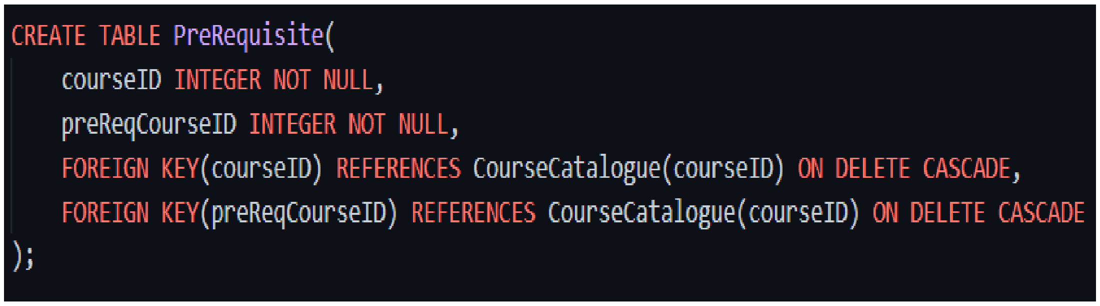
1. `CourseCatalogue`



```
CREATE TABLE CourseCatalogue(
    courseID SERIAL PRIMARY KEY,
    courseCode VARCHAR(10) NOT NULL,
    L INTEGER NOT NULL,
    T INTEGER NOT NULL,
    P INTEGER NOT NULL,
    S INTEGER NOT NULL,
    C Numeric(4,2) NOT NULL
);
```

- This table stores a list of all the courses along with their ***LTPSC*** structure and their ***course code***.
- Each tuple in this table is uniquely identified by its ***courseID*** which is also the **primary key** in this table.
- To list all the entries in this table, a stored procedure named **`viewCourseCatalogue`** can be used. To add a new course to the CourseCatalogue one can use the stored procedure named **`updateCourseCatalogue`**.
- In case the LTPSC structure for a course is updated, the course is added as a new entry in this table with a unique *courseID*. So in future, both the tuples for the same course can be distinguished using the *courseID* and the required version can be used.

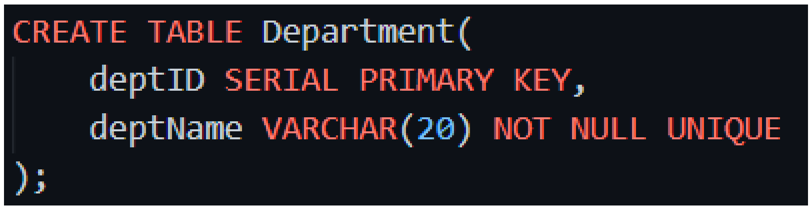
2. `PreRequisite`



```
CREATE TABLE PreRequisite(
    courseID INTEGER NOT NULL,
    preReqCourseID INTEGER NOT NULL,
    FOREIGN KEY(courseID) REFERENCES CourseCatalogue(courseID) ON DELETE CASCADE,
    FOREIGN KEY(preReqCourseID) REFERENCES CourseCatalogue(courseID) ON DELETE CASCADE
);
```

- This table stores a list of all the courses along with their required prerequisites. In case of multiple prerequisites of a course they are added as multiple tuples to this table. For example if `CS301` has prerequisites as `CS101 and CS201`, then two tuples i.e. *(CS301,CS101) and (CS301,CS201)* would be added to this tuple.
- The `courseID` and `preReqCourseID` fields must be valid courses as they are declared as a foreign key and referenced from the CourseCatalogue table.
- The attributes obey the property of *ON DELETE CASCADE*, i.e. if a course is deleted from the CourseCatalogue, then it is deleted from the PreRequisite table as well. This is because if a course no longer exists then maintaining its preRequisites is of no use further.

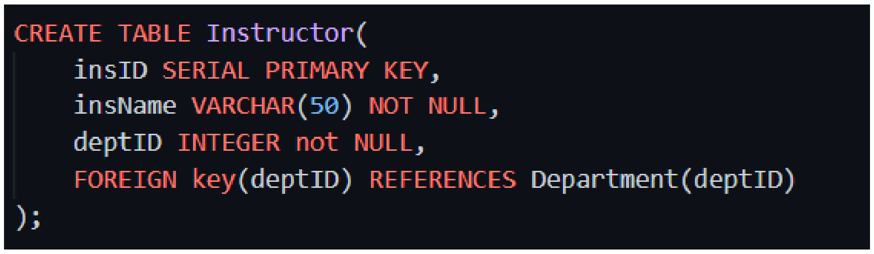
3. **Department**



```
CREATE TABLE Department(
    deptID SERIAL PRIMARY KEY,
    deptName VARCHAR(20) NOT NULL UNIQUE
);
```

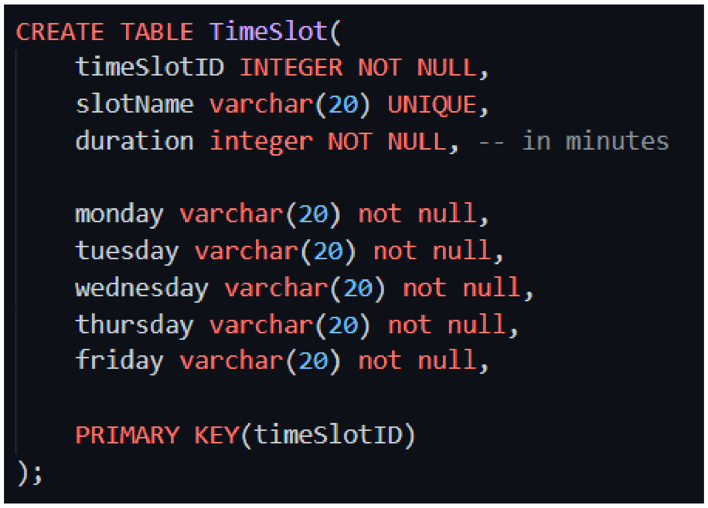
- Each department (with deptName for example CSE, EE) is given a unique ***deptID*** which is used to identify the department.

4. **Instructor**



```
CREATE TABLE Instructor(
    insID SERIAL PRIMARY KEY,
    insName VARCHAR(50) NOT NULL,
    deptID INTEGER not NULL,
    FOREIGN key(deptID) REFERENCES Department(deptID)
);
```

5. TimeSlot



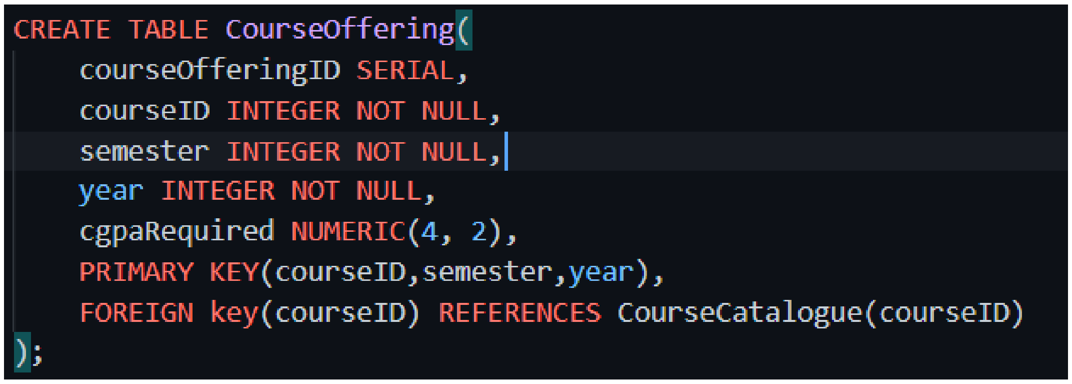
```sql
CREATE TABLE TimeSlot(
    timeSlotID INTEGER NOT NULL,
    slotName varchar(20) UNIQUE,
    duration integer NOT NULL, -- in minutes

    monday varchar(20) not null,
    tuesday varchar(20) not null,
    wednesday varchar(20) not null,
    thursday varchar(20) not null,
    friday varchar(20) not null,

    PRIMARY KEY(timeSlotID)
);
```

- This table stores the time slots. Each slot is composed of the ***slotName***, its ***duration*** in minutes, along with the slot timings for each day of the week. Each slot is uniquely identified by a ***timeSlotID***.

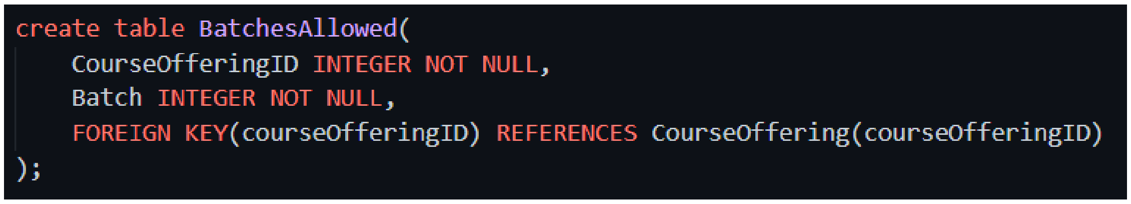
6. CourseOffering



```sql
CREATE TABLE CourseOffering(
    courseOfferingID SERIAL,
    courseID INTEGER NOT NULL,
    semester INTEGER NOT NULL,
    year INTEGER NOT NULL,
    cgpaRequired NUMERIC(4, 2),
    PRIMARY KEY(courseID,semester,year),
    FOREIGN key(courseID) REFERENCES CourseCatalogue(courseID)
);
```

- When a course is offered a tuple is added to this table. For example, if CS301 is offered in semester 1 of the year 2021 and the minimum CGPA requirement to enrol in this course is 5 then we add a tuple ***`(CS301,1,2021,5)`*** to this table.

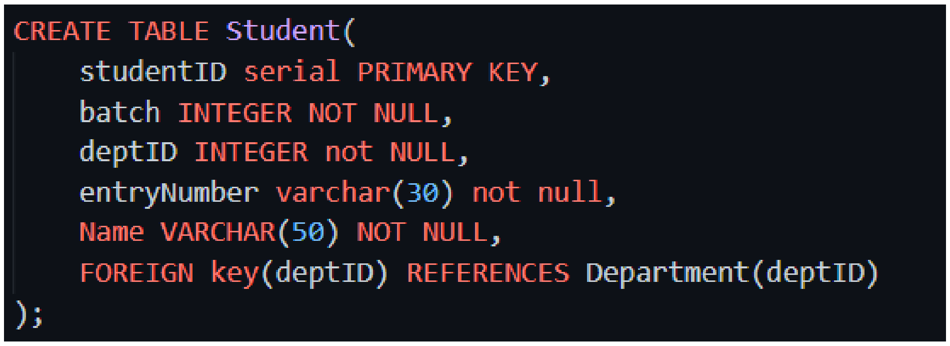
7. BatchesAllowed



```
create table BatchesAllowed(
    CourseOfferingID INTEGER NOT NULL,
    Batch INTEGER NOT NULL,
    FOREIGN KEY(courseOfferingID) REFERENCES CourseOffering(courseOfferingID)
);
```

- This table contains for each course the list of batches that are allowed to enrol for this course. For example, if CS201 is allowed to be enrolled by batch 2018 and 2019, and let us assume that the courseOfferingID is x for CS201 then this table would contain (x,2018) and (x,2019) in this table.
- Note that **courseOfferingID** is a foreign key from the **courseOffering table** thus ensuring a proper mapping.

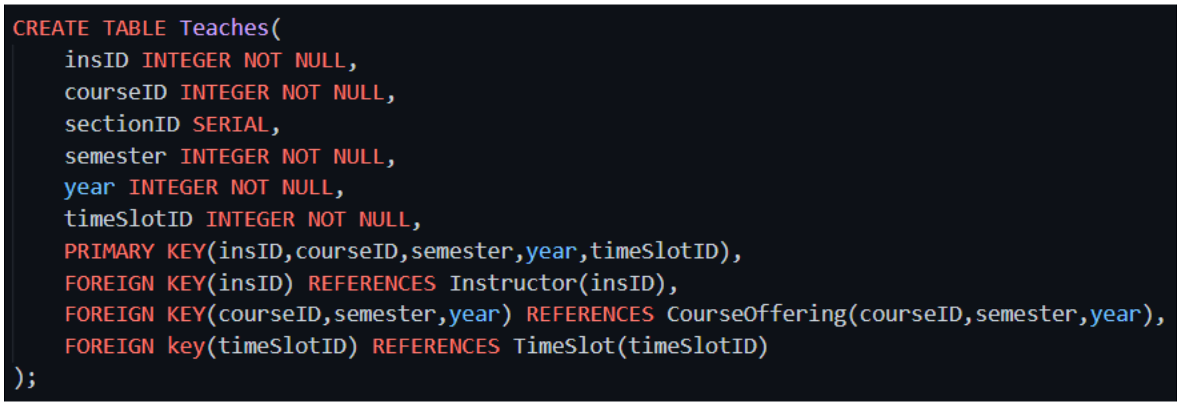
8. Student



```
CREATE TABLE Student(
    studentID serial PRIMARY KEY,
    batch INTEGER NOT NULL,
    deptID INTEGER not NULL,
    entryNumber varchar(30) not null,
    Name VARCHAR(50) NOT NULL,
    FOREIGN key(deptID) REFERENCES Department(deptID)
);
```

- This is a very important table in our database. It stores the records of the students. Each student is assigned a unique entry number, a department, a batch (for example 2018 or 2019) and a unique studentID.

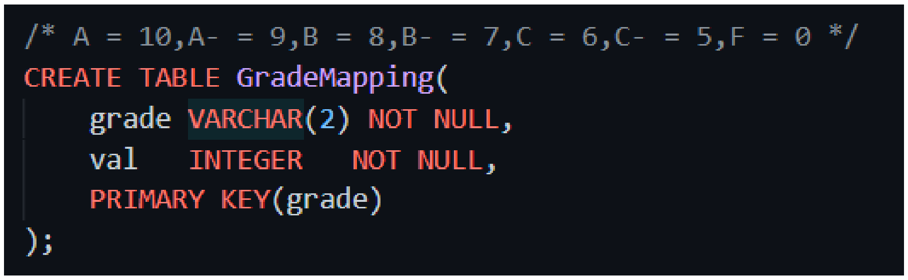
9. Teaches



```
CREATE TABLE Teaches(
    insID INTEGER NOT NULL,
    courseID INTEGER NOT NULL,
    sectionID SERIAL,
    semester INTEGER NOT NULL,
    year INTEGER NOT NULL,
    timeSlotID INTEGER NOT NULL,
    PRIMARY KEY(insID,courseID,semester,year,timeSlotID),
    FOREIGN KEY(insID) REFERENCES Instructor(insID),
    FOREIGN KEY(courseID,semester,year) REFERENCES CourseOffering(courseID,semester,year),
    FOREIGN key(timeSlotID) REFERENCES TimeSlot(timeSlotID)
);
```

- This table is the heart of the database. It defines how a particular course is offered and enables the route for the students to enrol in an offered course.
- Each row in the teaches table is a combination of (instructor ID, *courseID*, *semester*, *year*, *timeSlotID*). When a student wants to enrol for a course it checks the sectionID and sends the request for enrollment in this course. Then upon further checks, he/she can either enrol or raise a ticket for the same.

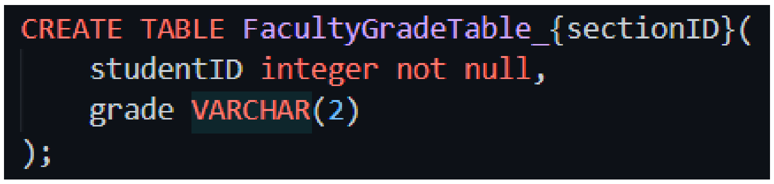
10.GradeMapping



```
/* A = 10,A- = 9,B = 8,B- = 7,C = 6,C- = 5,F = 0 */
CREATE TABLE GradeMapping(
    grade VARCHAR(2) NOT NULL,
    val   INTEGER    NOT NULL,
    PRIMARY KEY(grade)
);
```

- Used to map letter grades to their decimal equivalents. This is helpful in the `calculate_CGPA` stored procedure.

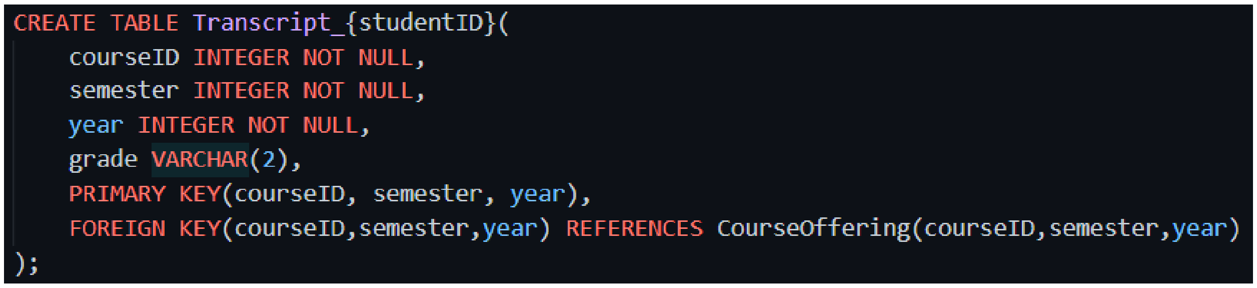
11.FacultyGradeTable_{sectionID} @Dynamic_Table



```
CREATE TABLE FacultyGradeTable_{sectionID}(
    studentID integer not null,
    grade VARCHAR(2)
);
```

- Grade table corresponding to each section. This table is created dynamically when a row is inserted into the Teaches Table.

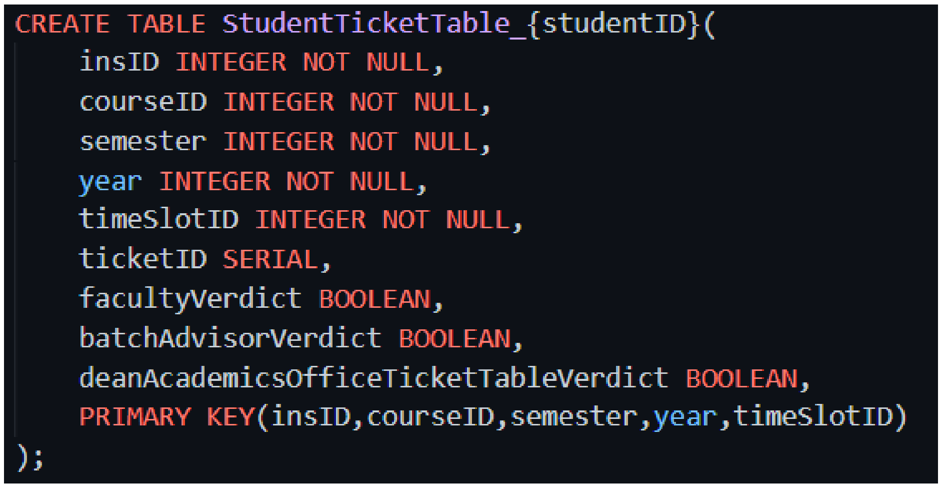
12. **Transcript_{studentID}** **@Dynamic_Table**



```
CREATE TABLE Transcript_{studentID}(
    courseID INTEGER NOT NULL,
    semester INTEGER NOT NULL,
    year INTEGER NOT NULL,
    grade VARCHAR(2),
    PRIMARY KEY(courseID, semester, year),
    FOREIGN KEY(courseID,semester,year) REFERENCES CourseOffering(courseID,semester,year)
);
```

- Transcript table corresponding to each student. This table is created dynamically when a row is inserted into the Student Table.
- It is created by the trigger ***postInsertStudent***. More details about this trigger can be found in the subsequent section.

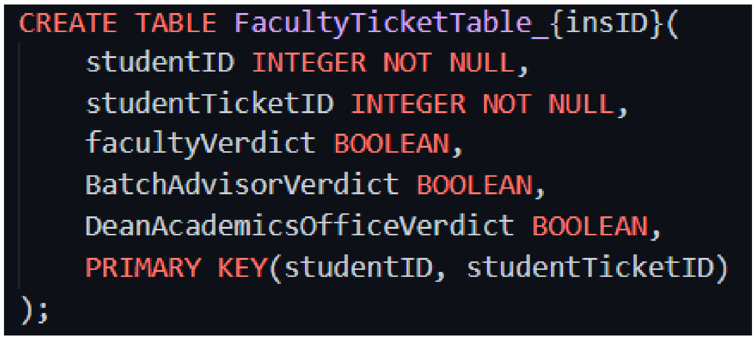
13. **StudentTicketTable_{studentID}** **@Dynamic_Table**



```
CREATE TABLE StudentTicketTable_{studentID}(
    insID INTEGER NOT NULL,
    courseID INTEGER NOT NULL,
    semester INTEGER NOT NULL,
    year INTEGER NOT NULL,
    timeSlotID INTEGER NOT NULL,
    ticketID SERIAL,
    facultyVerdict BOOLEAN,
    batchAdvisorVerdict BOOLEAN,
    deanAcademicsOfficeTicketTableVerdict BOOLEAN,
    PRIMARY KEY(insID,courseID,semester,year,timeSlotID)
);
```

- It denotes the Student Ticket table corresponding to each student. This table is created dynamically when a row is inserted into the Student Table.
- It is created by the trigger ***postInsertStudent***. More details about this trigger can be found in the subsequent section.

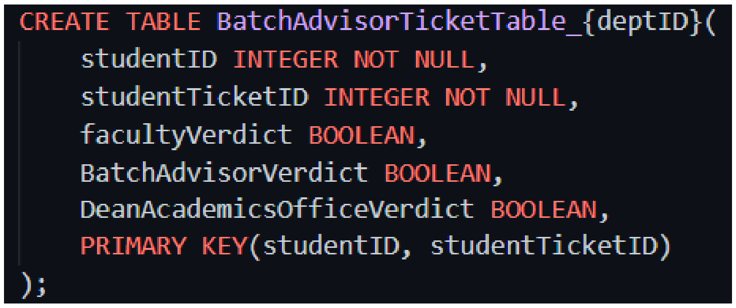
14.FacultyTicketTable_{insID} @Dynamic_Table



```sql
CREATE TABLE FacultyTicketTable_{insID}(
    studentID INTEGER NOT NULL,
    studentTicketID INTEGER NOT NULL,
    facultyVerdict BOOLEAN,
    BatchAdvisorVerdict BOOLEAN,
    DeanAcademicsOfficeVerdict BOOLEAN,
    PRIMARY KEY(studentID, studentTicketID)
);
```

- It denotes the Faculty Ticket table corresponding to each faculty. This table is created dynamically when a row is inserted into the Instructor Table.
- It is created by the trigger ***postInsertInstructor***. More details about this trigger can be found in the subsequent section.

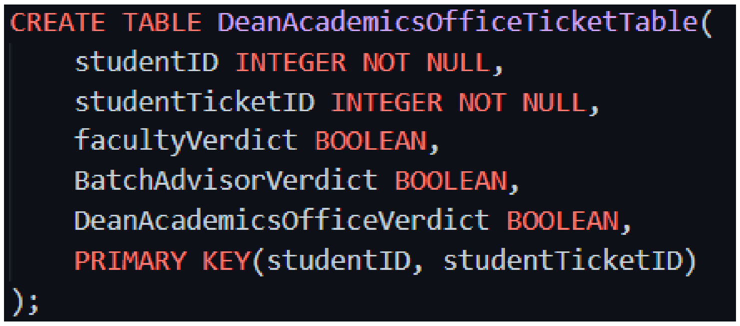
15.BatchAdvisorTable_{deptID} @Dynamic_Table



```sql
CREATE TABLE BatchAdvisorTicketTable_{deptID}(
    studentID INTEGER NOT NULL,
    studentTicketID INTEGER NOT NULL,
    facultyVerdict BOOLEAN,
    BatchAdvisorVerdict BOOLEAN,
    DeanAcademicsOfficeVerdict BOOLEAN,
    PRIMARY KEY(studentID, studentTicketID)
);
```

- Ticket table for each Batch Advisor.

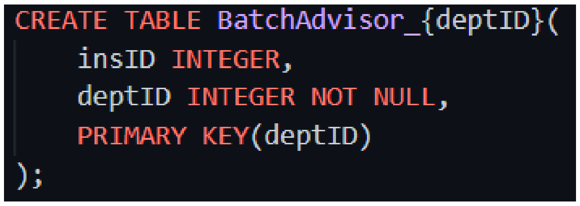
16.DeanAcademicsOfficeTicketTable



```sql
CREATE TABLE DeanAcademicsOfficeTicketTable(
    studentID INTEGER NOT NULL,
    studentTicketID INTEGER NOT NULL,
    facultyVerdict BOOLEAN,
    BatchAdvisorVerdict BOOLEAN,
    DeanAcademicsOfficeVerdict BOOLEAN,
    PRIMARY KEY(studentID, studentTicketID)
);
```

- Ticket table for each Dean Academics Office.

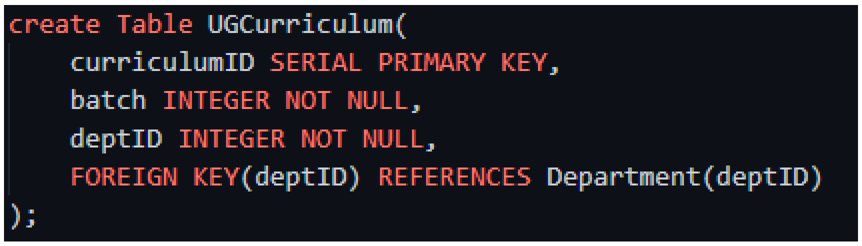
17. BatchAdvisor_{deptID} @Dynamic_Table



```sql
CREATE TABLE BatchAdvisor_{deptID}(
    insID INTEGER,
    deptID INTEGER NOT NULL,
    PRIMARY KEY(deptID)
);
```

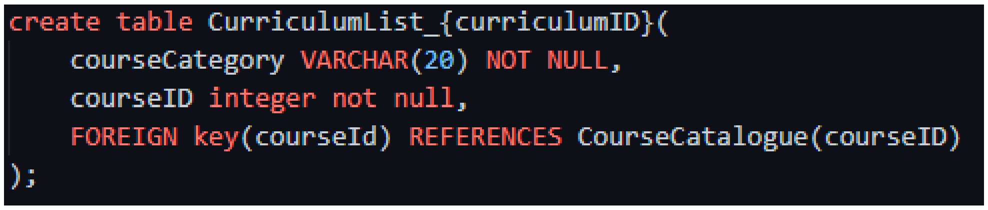
- For each department, a dynamic table using the deptID is created. It contains the id of the instructor who is the batch advisor for the given department.

18. UGCurriculum



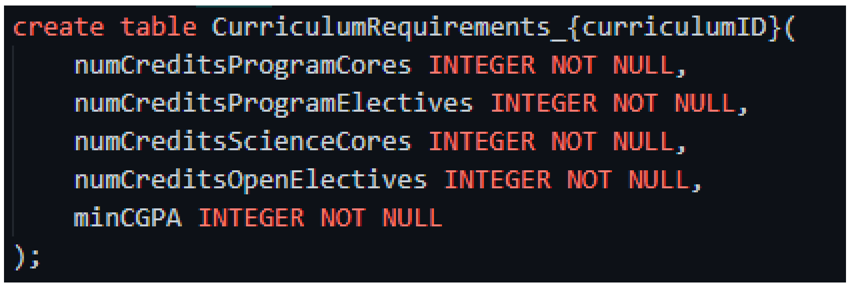
```sql
create Table UGCurriculum(
    curriculumID SERIAL PRIMARY KEY,
    batch INTEGER NOT NULL,
    deptID INTEGER NOT NULL,
    FOREIGN KEY(deptID) REFERENCES Department(deptID)
);
```

19. CurriculumList_{curriculumID} @Dynamic_Table



```sql
create table CurriculumList_{curriculumID}(
    courseCategory VARCHAR(20) NOT NULL,
    courseID integer not null,
    FOREIGN key(courseId) REFERENCES CourseCatalogue(courseID)
);
```

20. CurriculumRequirements_{curriculumID} @Dynamic_Table



```sql
create table CurriculumRequirements_{curriculumID}(
    numCreditsProgramCores INTEGER NOT NULL,
    numCreditsProgramElectives INTEGER NOT NULL,
    numCreditsScienceCores INTEGER NOT NULL,
    numCreditsOpenElectives INTEGER NOT NULL,
    minCGPA INTEGER NOT NULL
);
```

Triggers :

1. **GenerateFacultyGradeTable:** This trigger generates a grade table for a particular course. A separate table is created for each section of the same course. This table contains the **`studentId`** and their **`course grade`**. The instructor of a course can update the grade of a student enrolled in the given course in this table. As soon as the grade is updated in this table, a trigger is launched which updates the grade for this course in the transcript table of that particular student automatically. This approach has a **major security advantage**. Suppose for instance an instructor account is compromised and the hacker tries to change the grades of a student in its other courses. But he will not be able to do so since grades are updated in the transcript table via a trigger and not directly.

2. **postInsertingStudent:** When a new student enrols in the institute, this trigger is triggered. For that particular student, a dedicated Transcript table and a dedicated Ticket Table are created.

3. **postInsertingInstructor:** When a new faculty is added to the Faculty table, this trigger is triggered, resulting in the creation of a new Ticket Table for that faculty.

4. **postInsertingDepartment:** This trigger is triggered when a new department is added to the department table. It generates a new batchAdvisorTable for that department as well as a new Ticket table for the Batch Advisor of this department.

Stored Procedure :

1. **OfferCourse:** This procedure is invoked whenever a new course offering is floated. It takes CourseID, semester, year, and the CGPA criteria as input. It will then perform the necessary checks, such as determining whether the CGPA criteria and *courseID* are valid. The tuple will be successfully inserted into the **CourseOffering** table if all of the checks are successful.

2. **InsertIntoTeaches:** This procedure is used to insert a new tuple into the **Teaches** table. It accepts the following parameters as input: *instructorID*, *CourseID*, *sectionID*, *semester, year*, and *slotName*. After validating the *CourseID* and *slotName*, it inserts the tuple into the Teaches Table. A unique *sectionID* is also generated in this case.

3. **RegisterStudent:** This stored procedure is responsible for the student's enrollment in a specific course. It accepts the following parameters: *studentID*, *courseCode*, *semester*, *year*, *insName*, and *slotName* as input. Before enrolling the student in that course, it determines whether or not the total credits registered for this semester will satisfy the 1.25 rule after adding this course. If the preceding condition is met, it checks to see if the student has completed all prerequisite courses and also meets the CGPA requirements to enrol in this course. If the aforementioned condition is not met, the enrollment is rejected; otherwise, another check is performed to see if the student is already enrolled in a course with the same time slot as this one. If the student meets all of the prerequisites, he or she will be successfully enrolled in the course.

4. **RaiseTicket:** If a student is unable to register for a course due to any of the reasons listed above, he or she may raise a Ticket. This procedure runs the necessary checks to see if this ticket has already been raised. If this is not the case, the ticket table for the student, faculty for this course, and the Dean is updated accordingly.

5. **MakeBatchAdvisor:** The Dean can use this procedure to appoint a faculty batch advisor for a department.

6. **UploadTimeTableSlots**: This procedure can be used by academic staff to upload a timetable in the form of a csv file.

7. **PrintGradeSheet**: This stored procedure allows you to print a specific student's grade sheet (dumped into an external csv file). This function can be accessed by the DeanAcademicsOffice, faculty, batchAdvisor, and the student himself.

8. **UploadCourseGrades** (importing from a csv file): Through this procedure, the instructor of a course can upload grades for all enrolled students.

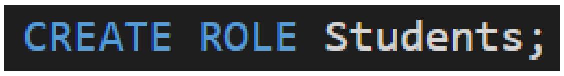
9. **CalculateCurrentCGPA**: This function returns the current CGPA of the specified student by using its transcript table.

10. **CanGraduate:** This function takes a studentID as input and returns whether the student is eligible to graduate. It checks if the student has completed all of the courses listed in his or her specific **UG Curriculum**, as well as whether or not the student meets the **minimum CGPA requirement for graduation**.

Roles and Permissions

We have created four different types of roles in our academic database system. They are as follows

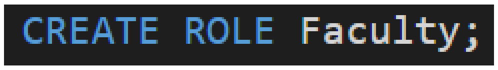
Role_1



```
CREATE ROLE Students;
```

A user with a role assigned as Student can only read his transcript table but can not perform a write operation on it. Almost all the other tables should have only read permissions assigned for the student category. A student should have permission to insert a ticket into the **`FacultyTicketTable`**, **`BatchAdvisorTicketTable`** and **`DeanAcademicsOfficeTicketTable`**.

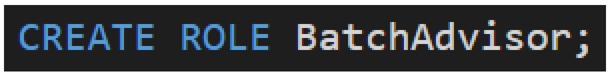
Role_2



```
CREATE ROLE Faculty;
```

Faculty can read the CourseOffering table. He can write in the Teaches Table and has full access to the FacultyGradeTable_{sectionID} and FacultyTicketTable_{insID}.

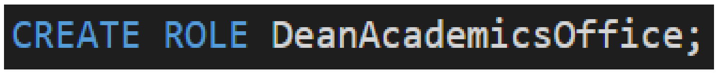
Role_3



```
CREATE ROLE BatchAdvisor;
```

BatchAdvisor can read and write on BatchAdvisorTicketTable, FacultyTicketTable and DeanAcademicsOfficeTicketTable.

Role_4



```
CREATE ROLE DeanAcademicsOffice;
```

DeanAcademicsOffice has read permission on all Tables but writes on few. He can write on DeanAcademicsOfficeTicketTable, Student Table, can update FacultyTicketTable and BatchAdvisorTicketTable.