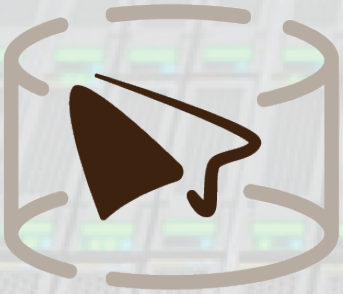
IUB



SPM

Student Performance Monitor

FINAL PROJECT REPORT

GROUP-1

Project Title: Student Performance Monitoring

Prepared By:

Naimul Aziz (1822240),

Tahzeeb Ahmed (1930234)

Nibrash Kazi Subah (1720158)

Final Project Report: Table of Contents –

INTRODUCTION

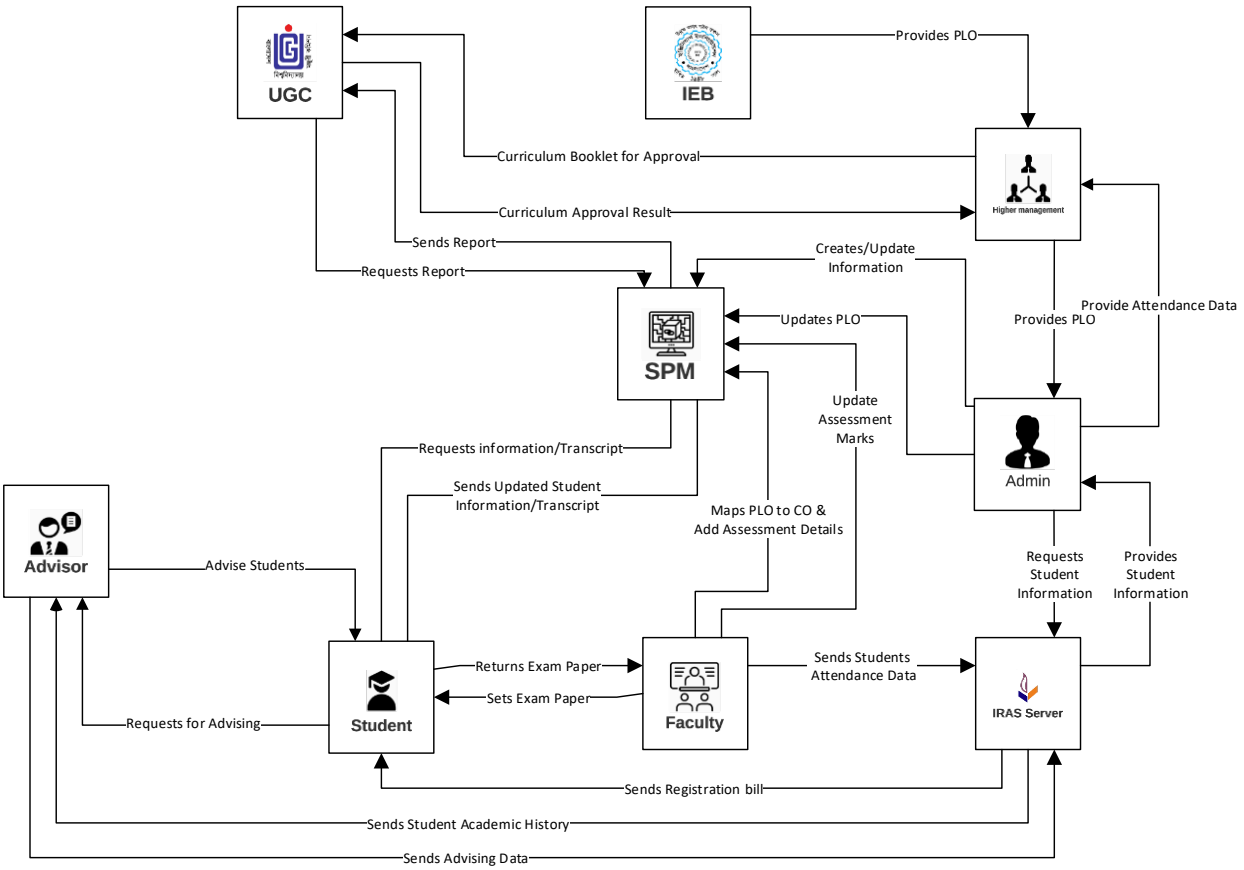
Background of the Project

The upgraded student performance monitor (SPM) is designed with comprehensiveness and scalability in mind to adapt to the ever-changing standards of the educational industry around the world. This database software allows the performance and achievement of students to be tracked and monitored at real time within historical timeframes. Our interactive and easy to use interface allows all users including higher management to gain convenience and strengthens their capabilities while conducting research and studying the enrollment trends. The unique database design allows the software to consider changes within the curriculum or syllabus of the University academic system. Various stakeholders which include students, faculties, higher management and even guardians or parents to gain access to the vast array of functionalities this software has to offer. The clean statistical analysis tools allow higher management to track all students' academic progress allowing them to see a bigger picture of the overall academic performance in a university. Students can monitor their own performances as well.

Objective of the Project

This project aims to make advances in the education sector by making it easy and accessible for authorities to make positive impactful changes within the curricula. It is designed to make the lives of all its users within the education community more seamless. within this age where information is key this software will not only empower its users to gain access to insightful feedback but also allow them to improve past mistakes and misconceptions.

Rich Picture (AS-IS):



Six Element Analysis (AS-IS):

Process	System Roles					
	Human	**Non-Comp Hardware**	**Computing Hardware**	**Software**	**Database**	**Network & Communication**
1. Admin creates user accounts	**Admin** a) Extracts students' and faculties' information from iRAS b) Updates information into the SPM Database c) Creates new Student/Faculty accounts when required. **Developing Team and IT Experts** a) Builds and maintains the SPM system. **Internet Service Providers** a) Provides the Internet service to the data sources, SPM users and SPM system.	**Paper & Stationary** a) Used to collect information in forms from users which do not have accounts in iRAS, i.e. UGC. **Access Request Form** a) Used by others for requesting access the system.	**Computers** a) SPM admin will use computers to access iRAS to collect data. b) Users will use the computer to view the data. c) Faculties will use the computer to view and update. **Database Server** a) Used by iRAS to store data and by SPM developers to collect data. **Networking Devices (Router, Switch, Bridge, Hub):** a) Used to access iRAS, SPM	**Operating Software** a) Used by iRAS and SPM **iRAS** a) Used as a source of data from which the accounts will be created. **SPM** a) The software for which the admin will create accounts.	**iRAS Database** a) Used by the Admin as a source of information for user accounts. **Excel Files** a) User account data may be stored in excel files which will then be used by SPM. **SPM Database** a) The user account information will be stored here – usernames / account names, etc.	**Internet** a) It is used to access and store data from iRAS to SPM.

2. Retrieve PLO and updates PLO mapped CO to SPM	**IEB** Provides PLO to Higher Management **Faculty** Updates mapped CO to SPM. **Admin** 1) Adds PLO to SPM 2) Maps PLO to CO. **Higher Management** Provides PLO to Admin.	**Book** a) Contains details of the PLO. **Pen and Paper** Used for mapping all the CO with the received PLO from IEB.	**Computer** a) IEB will use the computer to access SPM and update when needed. b) Faculties will use the computer to view course details and map the CO. c) IEB uses the computer to create PLO which will be given to the universities. **Mobile** a) Used for communication between Faculty, IEB. **Networking Devices (Router, Switch, Bridge, Hub):** a) Used to access the Internet	**SPM** Used to create, read, update, and/or delete details about PLOs and COs. **Email Software** Used for communication between Faculties and IEB. **Operating System** a) Any OS used by the users, Windows, Mac.	**SPM Database** 1) For storing the mapped COs. 2) For storing the PLO. **IEB Database** Retrieving the PLO details from IEB.	**Internet** 1) Used by faculties to access IEB website and update SPM. 2) Used by IEB to update and store PLO in their database.

4. Set Question Papers according to COs and conduct exams	**Faculty** 1) Retrieve COs from software. 2) Format and set question papers for examinations according to COs collected 3) Set classroom schedule for exams. 4) Conduct examinations and collect test papers. **Student** Sit for examinations and submit attempted test papers to faculty.	**Table & Chair** a) To use during exams. **Pen and Paper** 1) For attempting the exams. 2) Questions may be printed on paper. **Clock** Setting time for the exam. **Room** Specific room for exams.	**Computer** 1) Used by faculties to access the COs from the software. 2) Faculties also use it to take online exams and interact with students 3) Students may use it to attend online exams. **Mobile Phones** Some exams may allow mobile phones for scanning and uploading pdfs to virtual examinations. **Printer** a) Used by faculties to print out question papers for students. **Database Server** a) Used by faculties to access and collect the COs to set questions. **Networking Devices (Router, Switch, Bridge, Hub):** a) Used by faculties and students to access the Internet.	**SPM** a) The software from which the faculty will collect COs. **Google Classroom** a) Used by faculties and students during exams. **Operating system** a) Any OS used by the users such as Windows and Mac. **Printing Software** Used to run the printer for printing the question paper. **PDF Viewer** a) To view the questions or send the answers in PDF format.	**SPM Database** Faculty members access COs from this. .	**Internet** 1) Used by faculties to access SPM software and its database. 2) Used by faculties and students during exams.

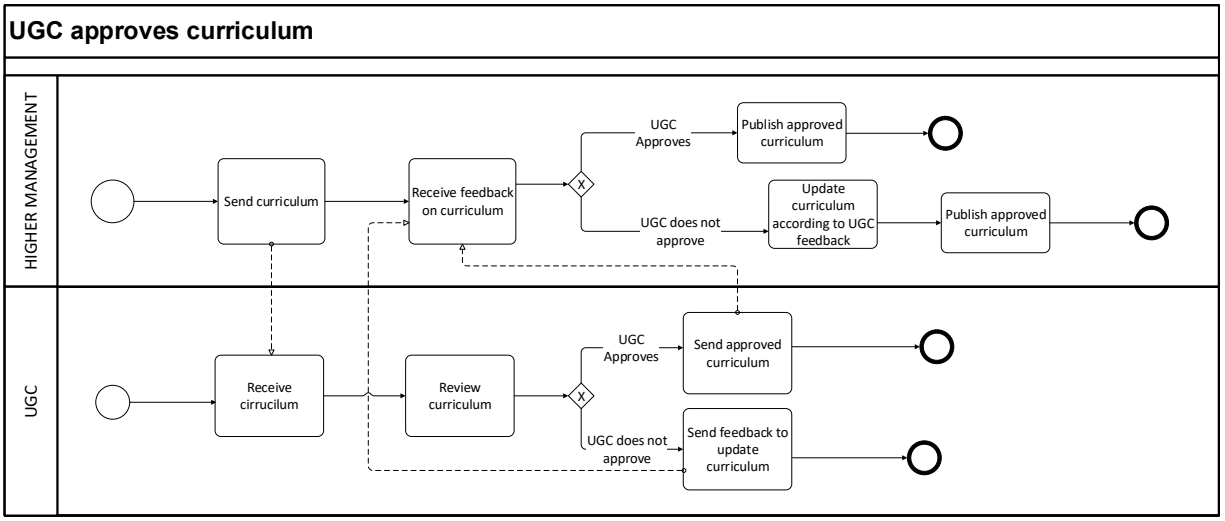
5. Teachers evaluate students and update marks in SPM	**Faculty** 1) Update database with the achieved COs marks of the student. 2) Send the Exam paper back to the Student. **Student** Request faculty to return graded exam papers.	**Pen and Paper** a) The exam paper is marked with a red pen.	**Computer** Used by faculty members to store the exam result and update the database with achieved COs. **Database Server** Used by faculty members to access and store or update the database. **Mobile** Used for communication between faculty members and students. **Networking Devices (Router, Switch, Bridge, Hub):** Used by faculty members to access the Internet.	**SPM** It is used to store and update the Student profile. **Operating System** Any OS used by the users, e.g., Windows, Mac	**SPM Database** It is used to store the updated data of the Student.	**Internet Connection** It is used by the faculty to access the SPM software and the database.

6. Monitor Attendance	**Faculty** Manually takes attendance in classroom by ticking the checkbox in IRAS online attendance sheet for those students who respond as present in the classroom. **Student** Responds to faculty's attendance call-out as present for marking their presence in the online attendance sheet in IRAS. **Admin** Compile attendance data based on various parameters, organizes them and sends attendance data reports for HM. **HM** Retrieve and conduct studies on attendance records and metrics.	**Paper** 1) Used by faculties for jotting down names of students who have not been marked presence and the network is down. This is also a backup method in case of system failure. 2) Used by higher manageme nt to print out attendance records and metric reports. **Pen** Used for marking presence of students in case of manual attendance sheets instead of online normally occurring from a downed system.	**Computer** 1) Used by faculties for marking attendance into the IRAS system during classroom sessions. 2) Used by admin to compile reports on attendance data. 3) Used by student to view their attendance records as well as other relevant and personal information. **Printer** 1) Used by teachers to print attendance sheets for manual attendance. 2) Used by higher management or admin for printing out metric reports on attendance data.	**IRAS** Used to manage student records including attendance, grades, CGPA, personal information, etc. **Printing Software** Used to print out attendance sheets from IRAS and metric reports by the admin and higher management. **Microsoft Excel** Used by administrators to compile and graph subsets of data requested by higher management to create reports on attendance metrics. **Operating System** Any OS that the devices running IRAS is operating under such as Mac, Windows, Linux, etc.	**IRAS Database** Used to insert, update, and retrieve attendance records for IUB students, faculties, parents, admin, and HM.	**Internet** Used for accessing the IRAS database server for accessing student attendance information.

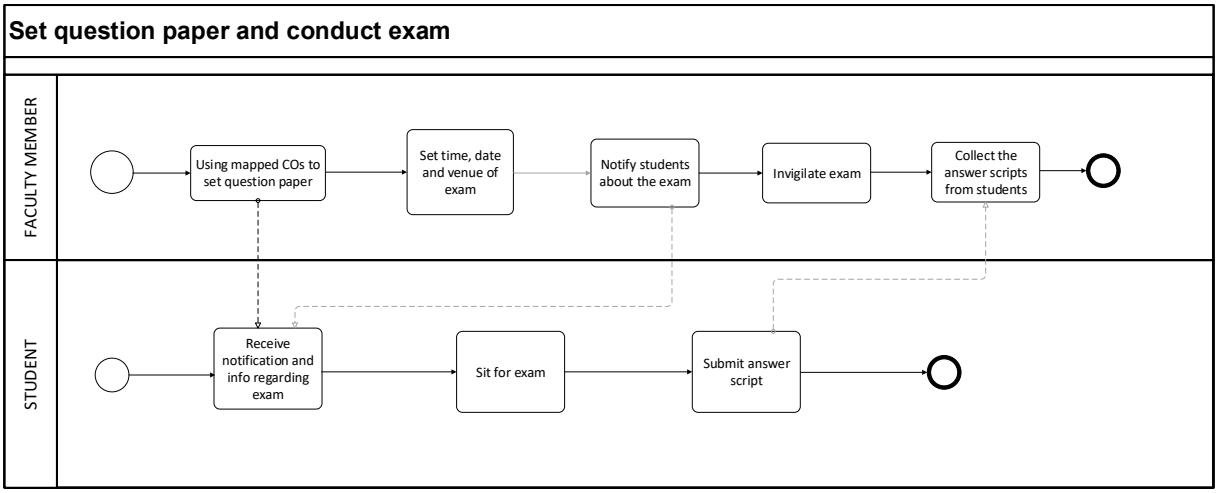
7. Request, Transcripts	**Student** 1) Request the system to generate a transcript form the database. 2) Receive the transcript and view/print it.	**Paper** Used in case the student wants to print the transcript.	**Computer** 1) Used to request the transcript. 2) View the transcript **Printer** 1) Print the transcript on a piece of paper. **Networking devices (Router, Switch, Bridge, Hub):** a) Used to access the Internet	**SPM** a) Queries the database to collect the required and relevant information of the student to generate a transcript, e.g. marks, grades, courses, credits, etc. b) Calculate the relevant information from the student's data, e.g., CGPA. c) Create a web page for the student to see their transcript. **Operating System** Any OS used by the users, Windows, Mac. **Printing Software** Printing software used for printing progress reports transcript. **PDF Viewer** To view transcript in PDF format.	**SPM Database** Provide the SPM software with the information it queries from the database.	**Internet** To access the SPM software.

8. Update Curriculum in SPM	**Admin** Updates changes in the curriculum	**Paper** Used in case the updates are noted.	**Computer / Phone** Used to update changes. **Printer** Print the curriculum data report. **Networking devices (Router, Switch, Bridge, Hub)** Used to access the Internet.	**SPM** SPM will contain the updated data. **Operating System** Any OS used by the users, e.g., Windows, Mac.	**SPM Database** Provide the SPM software with the information it queries from the database.	**Internet** Used to access the SPM software.
9. UGC Requests Reports from SPM to evaluate Department / University and Generate Report	**UGC** 1) UGC may request student performance reports based on a specific department or the whole university. 2) They may provide feedback regarding their evaluations. **Admin** 1) Admin will prepare a students' performance based on given parameters.	**Paper and Stationary** UGC may send out an application requesting to evaluate IUB using SPM. **Memo** Used to send messages to the SPM development team regarding UGC requests.	**Printer / Fax** 1) Used to print out reports to send to UGC. 2) Reports may also be faxed. **Laptop / PC & Other devices** 1) UGC will need a computer or phone to access SPM 2) UGC may use on for evaluating and sending results. **Networking Devices (Router, Switch, Bridge, Hub)** Used to access the Internet.	**SPM** 1) SPM will contain all individual student evaluations. 2) Organizes and calculates individual data to show various trends of the department, school, or university. **Email Software** Requests for reports may be sent via email. **Operating System** Any OS used by the users, e.g., Windows, Mac. **Printing Software** Used for running the printer. **PDF Viewer** To view report in PDF.	**SPM Database** Will contain all data regarding COs and PLOs for individual students.	**Internet** 1) To access SPM. 2) Reports may be requested or sent via email which is accessed by using the Internet.

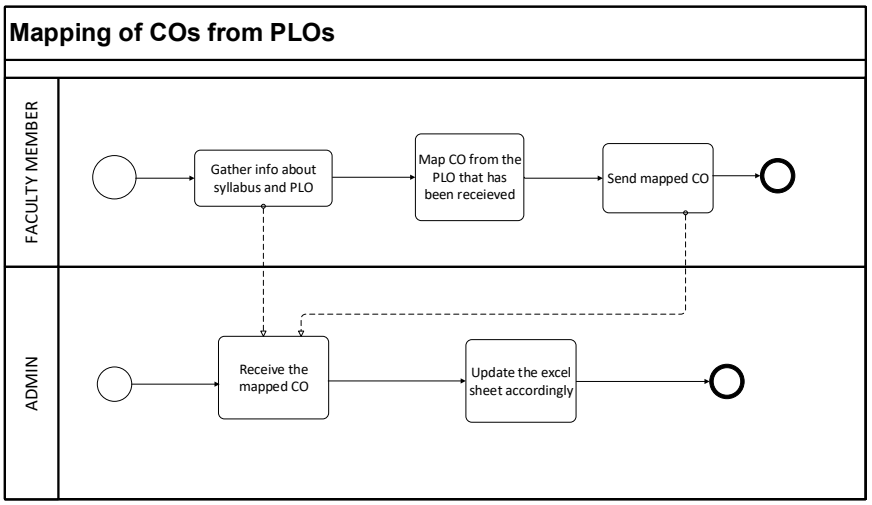
BPMN 2.0 Diagram (AS-IS)



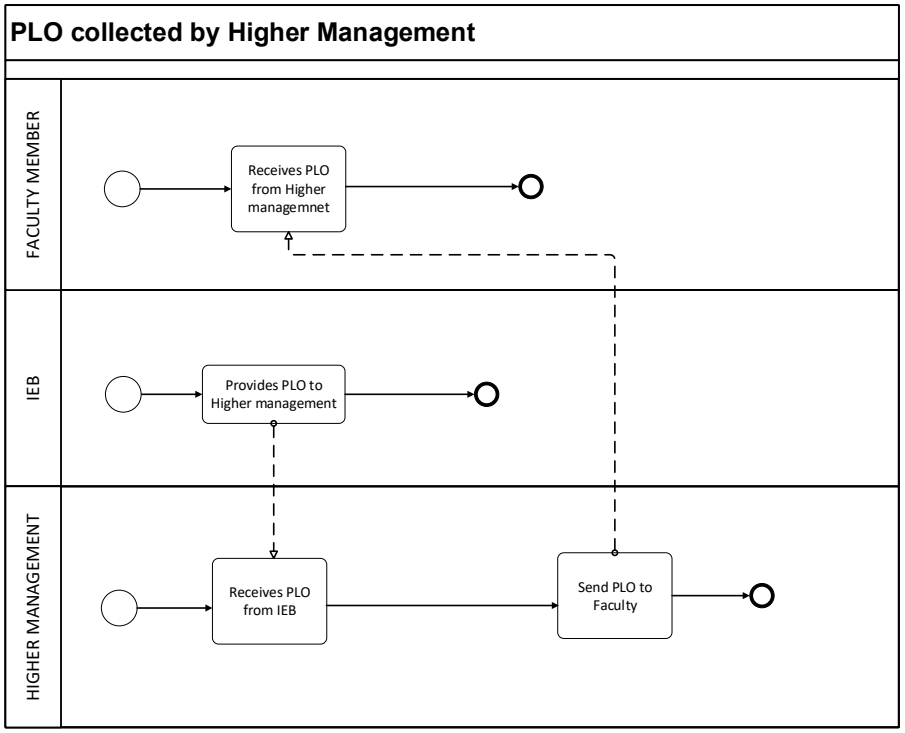
Process diagram for UGC approving curriculum (AS-IS)



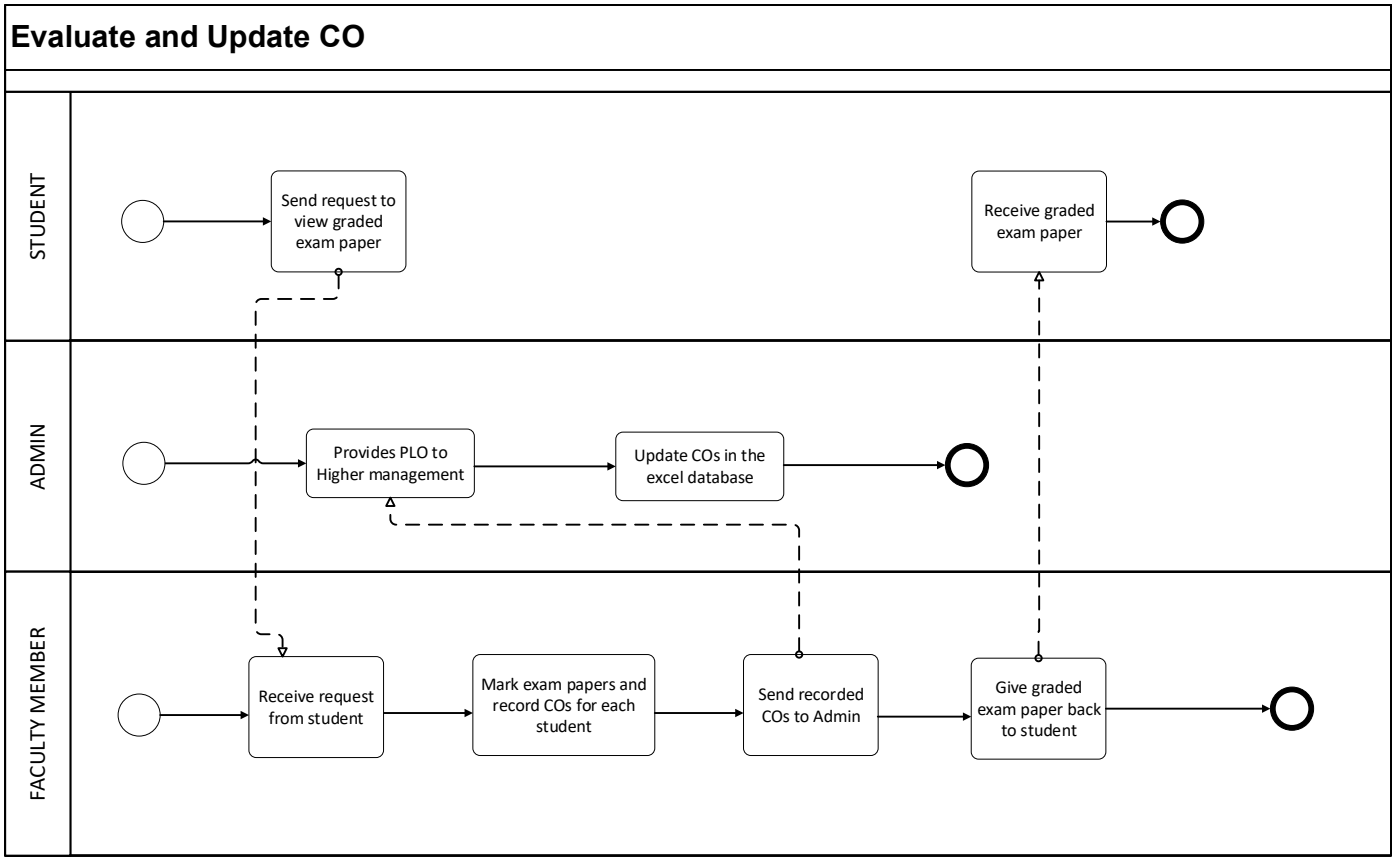
Process diagram for Setting question papers and coducting exams (AS-IS)



Process diagram for Mapping COs from PLOs (AS-IS)



Process diagram for PLO collected by Higher Management (AS-IS)



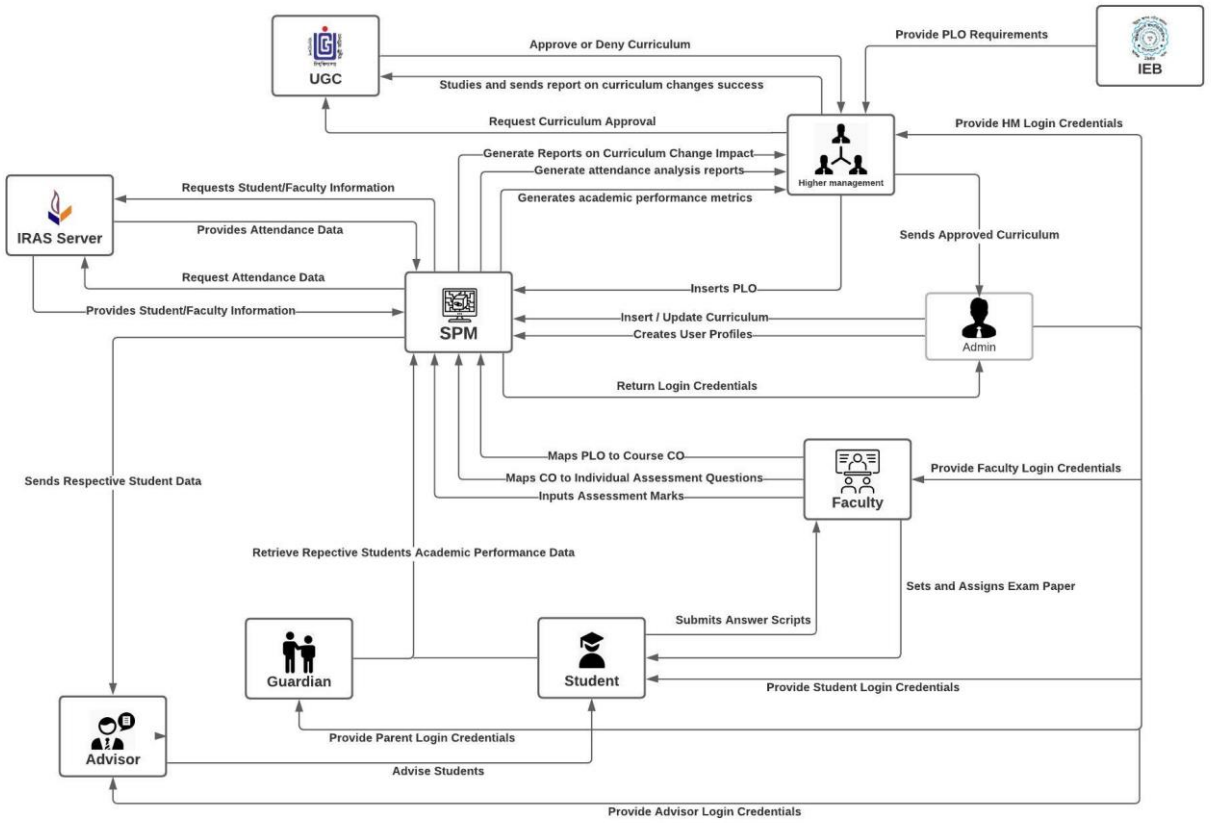
Process diagram for Ecaluate and Update CO (AS-IS)

Problem Analysis

Process Name	Stakeholders	Concern (Problems)	Analysis (Reason of the Problem)	Proposed Solution
Admin creates user accounts.	1. Administrator 2. Guardians 3. Students 4. Faculty 5. Advisors 6. Higher Management	Administrators only create accounts for faculty members, students, and Higher Management. Furthermore, parents are not able to view the progress reports of their children who are students at the university. This may lead them to worry about their education and study efforts. Additionally, the current system provides no user profiles for Advisors.	The reason of the problem is that the parents are dependent upon the students to provide them with the proper results, grades, and progress of their coursework. Some students hesitate to share their achievements due to personal reasons. In regard to advisors, the current system doesn't allow advisors to have access to students' PLO progress.	There is an option for parents to apply for a user account by filling out a form to the institution. The information is reviewed before opening an account and allowing access to the parent. Both the student and guardian must sign the account request form. Additionally, by developing an Advisor's portal in the SPM, the advisor can pull their respective students' PLO progress. with access to these data Advisors will be able to provide a more holistic prognosis to their respective students
Updating Curriculum in SPM	1. Admin 2. Higher Management 3. SPM 4. UGC	The current system only stores data from the latest update of the PLOs and Curriculum, historical data and changes are not stored and/or utilized.	There are a lot of changes in curriculum that may arise from improvement of educational standards or introduction of new PLOs. This may be easy to implement through the system but the changes made have a crucial impact on academic performance which is vital to track. Without managing the different curriculum version, this is nearly impossible.	A Curriculum Management Subsystem (CMS) will be integrated into the entire SPM which will be solely responsible for managing the changes in curriculum according to the date it has been proposed, the day it took effect. The different versions can also be traced and their impact on PLO and CO achievement can be studied by observing historical data over time.
Request Transcript from SPM	1. Student 2. Parents	Students and parents are only allowed to view their students' academic progress through transcripts. The only problem is that scores and marks for a specific course is not published until semester ends. Moreover, only the grade itself is published. The performance details are not known by the students or parents within the duration of a course.	The problem arises because there is a lack of managing student performance records such as scores of specific assessments and attendance records. The inability of the system to allow access for parents to view these records lower their ability to track the achievement for each course through the entire semester.	A system feature that will be implemented to allow user accounts to view the respective progress of their courses throughout the semester. Faculties will update the system frequently as assessments are graded, projects are submitted, and attendance is entered. This will be virtually accessible to students, administrators, teachers and occasionally to the parents who have requested access to their child's records.

Teachers will evaluate students and update marks in SPM	1. Faculty 2. Student 3. SPM	Upon receiving exam papers from students, the faculty member must manually calculate the total CO marks achieved by a student based on the teachers on mapping of each question to a CO, this is done simultaneously while they mark answer scripts. This process becomes time consuming on the faculty's end and is prone to human-error.	Current process consumes too much as the mapping of questions to COs isn't arranged in the system and tallying the CO achievements for each student is a manual task to be carried out concurrently while grading papers. A single CO can be mapped to two questions with one in the mid-term and the other in the final term. The tally for the CO achievement is done by faculty members as she grades her papers. This step is redundant to the evaluation process posing higher costs from being prone to human error and wasted time on an unnecessary step of the procedure. The lack of questions to CO mapping may lead to insertion anomalies and ergo inconsistency in the data.	We can eliminate this manual and separate task of tallying CO evaluation alongside marking papers, by allowing our software to directly map CO outcomes to questions set in the Midterm and Final term examinations. By doing so, all the faculty has to carry out an initial mapping and then only input the total marks for each question on a paper and input the marks achieved by students per question per paper, the SPM can calculate and evaluate the CO automatically for each student based on their achieved marks on each question of their mid-term and final-term answer scripts.
Report Generation Requests from UGC for University or Department Evaluation	1. UGC 2. University Higher Management 3. Admin	Currently, the SPM accepts requests for data reports on student performance and achievement. These requests are managed manually by administrators who obtain, arrange and compile data manually to create reports according to the parameters they are given by the organizations. This requires excess time, effort and thereby imposes higher cost on the university.	The current system is lacking the analytic features that are required for graphing and tabulating information and data. Due to maintain scalability of the product, there is a necessity for the system to withhold data illustration features such as graphical report generation software.	A graphical report generation software will consist of various functionalities such as data gathering and processing as well as organizing and depicting them in various illustrations such as graphs, tables, illustrations, etc.

Rich Picture (TO-BE)



Six Element Analysis (TO-BE)

Process	System Roles					
	Human	**Non-Comp Hardware**	**Computing Hardware**	**Software**	**Database**	**Network & Communication**
1. Admin creates user accounts	**Admin** 1) Extracts students' and faculties' information from iRAS 2) Updates information into the SPM Database 3) Creates new user accounts upon request **Development Team and IT Experts** Builds and maintains the SPM system. **Internet Service Providers** Provides the Internet service to the data sources, SPM users and SPM system.	**Paper & Stationary** Used to collect information in forms from users which do not have accounts in iRAS such as parents and UGC. **Access Request Form** Used by others for requesting access to the system such as parents or external analysts.	**Computers** 1) SPM administrators will use computers to access iRAS for data collection. 2) New account users (parents and analysts) will use the computer to view the data. 3) Faculties will use the computer to read and update. **Database Server** Used by iRAS to store data and by SPM developers to collect data. **Networking Devices (Router, Switch, Bridge, Hub):** Used to access iRAS, SPM	**Operating Software** Used by iRAS and SPM **iRAS** Used as a source of data from which the accounts will be created. **SPM** Educational program management system that the administrators will create the accounts for its respective users.	**iRAS Database** Used by the Admin as a source of information for user accounts. **Excel Files** User account data may be stored in excel files which will then be used by SPM. **SPM Database** The user account information will be stored here including usernames / account names, etc.	**Internet** It is used to access and store data from iRAS to SPM.

2. Retrieve, Insert or Update PLOs and COs into SPM	**IEB** Provides PLO or any updates to Higher Mgt. **Higher Management** Inserts PLO into SPM. **Admin** Adds or updates PLO to SPM. **Faculty** Studies PLO and CO mapping.	**Book** Contains details of the PLO. **Pen and Paper** For mapping all the CO with the received PLO from IEB.	**Computer** 1) IEB will use the computer to create PLO which will be provided to the universities. 2) IEB can also mail higher management of universities about PLOs or any changes made to it. 3) Faculties view current PLOs and the mapped COs. **Mobile** Used for communication between stakeholders. **Networking Devices (Router, Switch, Bridge, Hub):** Used to access the Internet	**SPM** Used to create, read, update, and/or delete details about PLOs and COs. **Outlook** Used for communication between Faculties and IEB. **Operating System** Any OS used by the users, Windows, Mac. **CMS** A subsystem dedicated to commit associated changes to SPM through the CMS database as instances of curriculum versions.	**SPM Database** For storing mapping of PLOs to COs. **IEB Database** Retrieving the PLO details from IEB. **CMS Database** Send curriculum changes into the SPM.	**Internet** 1) Used by faculties to access IEB website and update SPM. 2) Used by IEB to update and store PLO in their database.

3. Set or Change Curriculum based on PLO	**Higher Management** 1) Consult with board on designing program curriculum to cover current PLOs set forth by IEB. 2) Send defined curriculum or changes to UGC for approval **UGC** 1) Review the curriculum for adherence to PLOs. 2) Approve or deny the curriculum. **Admin** Make changes to curriculum based on change request approvals from higher management.	**Paper and Stationary** 1) Conduct drawings and mappings to design curriculum 2) Taking important notes and drawing rough sketches during board meetings. **Curriculum Booklet** Used to provide a manual version of curriculum for students, parents, UGC, etc. **Board Room** Specific room designated for board meetings. **Change Request Form** Printed version of a change request that is to be implemented in the database.	**Computer** 1) Higher Management and designated faculties use computers to determine and create curriculum report for specific programs offered by the university. 1) UGC will use the computer to view current curriculum of various programs and provide multiple feedback if necessary. **Mobile** Stakeholders have reliable communication through phones. **Printer** Used for printing change request forms, templates, curriculum booklets, course information, etc. **Networking Devices (Router, Switch, Bridge, Hub):** Used to access the Internet for viewing online materials.	**Skype** Members of educational institutions, UGC, IEB and others can hold teleconferences. **Email** 1) Used for minor communication among members such as sending important files and scheduling appointments. 2) UGC sends feedback to institutions regarding curriculum approval requests or current flaws or appraisals on the current curriculum. **SPM** Used for reviewing PLOs for mapping all COs to them. **Curriculum Management System (CMS)** 1) Add new courses as due to PLO update. 2) Make changes to course information as a result from new research and policies.	**SPM Database** Use the database to obtain the PLOs. **CMS Database** Use the database for version management of curriculum as changes are made to program requirements and PLOs through the future.	**Internet** 1) Used by UGC to view shared updates to curriculum. 2) Higher Management communicates files, requests, and changes to curriculum through the network.

4. Set Question Papers according to COs and conduct exams	**Faculty** 1) Retrieve COs associated with specific course. 2) Format and set question papers for examinations according to the collected COs. 3) Set classroom schedule for assessments. 4) Conduct examinations and collect test papers. Student Sit for examinations and submit attempted test papers to faculty.	**Table & Chair** To use during exams. **Pen and Paper** 1) For attempting the exams. 2) Questions may be printed on paper. **Clock** Setting time for the exam. **Room** Specific room for exams.	**Computer** 1) Used by faculties to access the COs from the software. 2) Faculties also use it to take online exams and interact with students when required. 3) Students use it to attend online exams if necessary. **Mobile Phones** Virtual examinations may require mobile phones for scanning and uploading pdfs to virtual examinations. **Printer** Used by faculties to print out question papers for students. **Database Server** Used by faculties to access and collect the COs to set questions. **Networking Devices (Router, Switch, Bridge, Hub):** Used by faculties and students to access the Internet.	**SPM** The software from which the faculty will collect COs. **Google Classroom** Used by faculties and students during exams. **Operating system** Any OS used by the users such as Windows and Mac. **Printing Software** To view question in PDF to send the answer in PDF. **PDF Viewer** To view the questions or send the answers in PDF format.	**SPM Database** 1) Students and faculties can access COs from this. 2) Students can view current COs and skills required for achieving those goals.	**Internet** 1) Used by faculties to access SPM software and its database. 2) Used by faculties and students during exams.

5. Teachers evaluate students and update marks in SPM	**Faculty** 1) Update database with the achieved COs marks of the student. 2) Send the Exam paper back to the Student. **Student** Request faculty to return graded exam papers.	**Pen and Paper** The exam paper is graded virtually or marked with a red pen.	**Computer** Used by faculty members to store the exam result and update the database with achieved COs. **Database Server** Used by faculty members to access and store or update the database. **Mobile** Used for communication between faculty members and students. **Networking Devices (Router, Switch, Bridge, Hub):** Used by faculty members to access the Internet.	**SPM** Store the updated data of the student. **OS** Any OS such as Windows, Mac	**SPM Database** It is used to store the updated data of the Student.	**Internet Connection** It is used by the faculty to access the SPM software and the database.

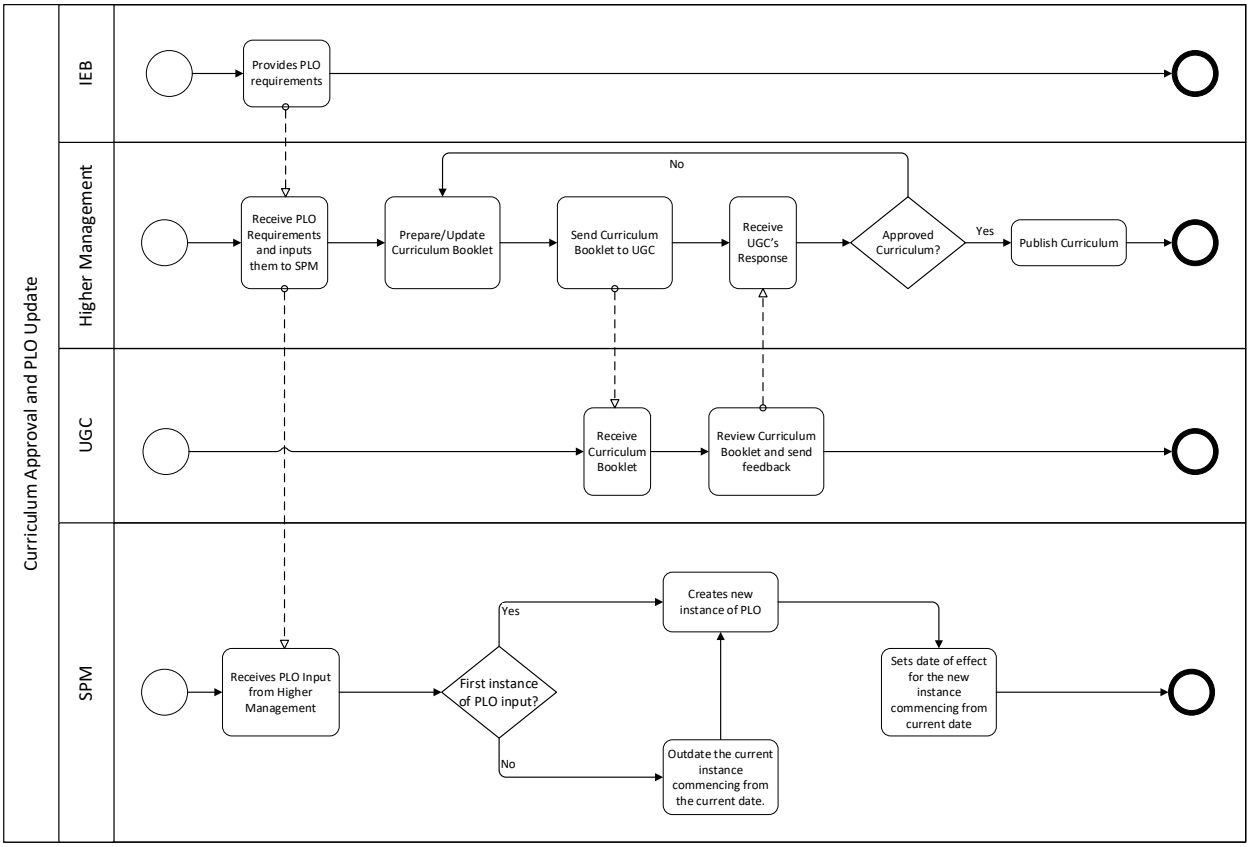
6. Tally Attendance, Highlight Assessment Dates, View and Manage Records	**Faculty** Take attendance in classroom, either mark them tardy, present or leave blank in case of absence. **Student** Check their attendance record and take precautions as to not go over the maximum limit of classes missed for a specific course. **Admin** View and update changes to attendance records on occasional requests.	**Paper** Needed for printed attendance records to higher authority for assuring student and faculty punctuality.	**Computer** 1) Used by faculties in classrooms to record attendance on the attendance management system. 2) Used by administrators to make any requested changes to existing attendance records. **Printer** Print attendance records upon request for reference or as needed. **Networking Devices (Router, Switch, Bridge, Hub):** Used to access the Internet for recording attendance on the system.	**Attendance Management System (AMS)** Used to manage attendance records in the AMS Database by faculties and administrators. **SPM** Retrieve attendance records from AMS to account for as a factor to the overall calculation of student grades and thereby also their CGPA for each semester.	**AMS Database** Used to record attendance for a long lifetime. **SPM Database** Used to clone attendance data from AMS to generate student grades and calculate CGPA.	**Internet** Used To access both, SPM and AMS servers for database management.

7. Request, Progress Reports, Report Cards, or Transcripts from SPM	**Student** 1) Request for specific progress Reports based on midterm, final, or semester results. 2) They can also request for transcript based on current GPA and credits earned.	**Paper** Students may want to print a hard copy of the transcript.	**Computer** 1) Used for requesting progress reports or transcripts. 2) View progress reports, transcripts. **Printer** Print hard copy of transcript **Networking devices (Router, Switch, Bridge, Hub):** Used to access the Internet	**SPM** 1) Queries the database to collect the required and relevant information of the student to generate a progress report or transcript, e.g. marks, grades, courses, credits, etc. 2) Calculate the relevant information from the student's data such as marks or grades. 3) Create a web page for the student to view and download electronic copies of their transcript. **Operating System** Any OS used by the users, Windows, Mac. **Printing Software** Printing software used for printing progress reports transcript. **PDF Viewer** To view reports or transcript in PDF format.	**SPM Database** Provide the SPM software with the information it queries from the database.	**Internet** To access the SPM software.

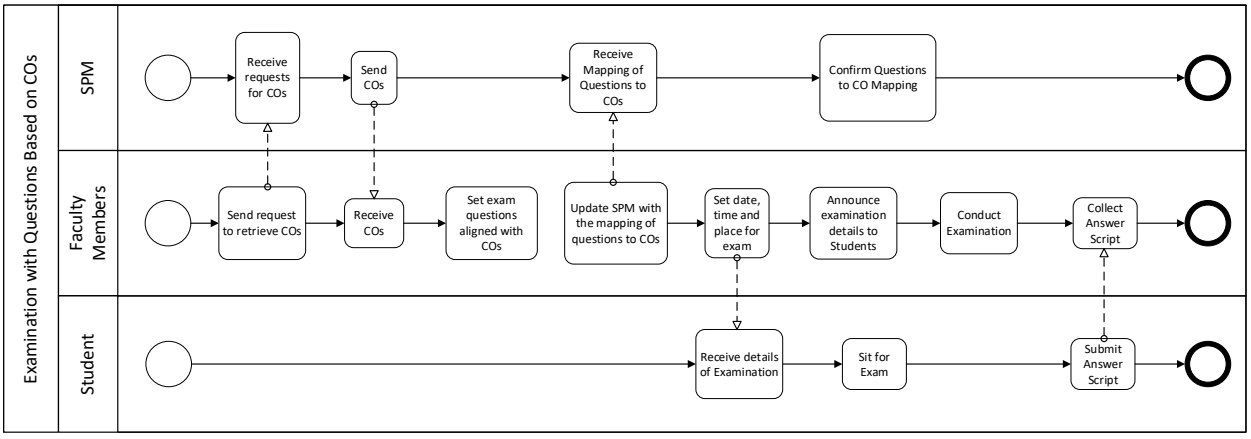
8. UGC Request Reports on Student Success Info for performance appraisal of departments or universities.	**UGC** 1) UGC may request student performance reports based on a particular department or the whole university. 2) They may provide feedback regarding their evaluations. **Data Analyst** Admin will prepare a report on students' performance based on given parameters.	**Paper and Stationary** a) UGC may send out an application requesting to evaluate using SPM. **Information Request Form** Used for requesting SPM development team for information and data on specific universities, depts, etc.	**Printer/Fax** 1) Used to print out reports to send to UGC. 2) Reports may also be faxed. **Laptop/PC & other devices** 1) UGC will need a computer or phone to access SPM. 2) UGC may use one for evaluating and sending results. **Networking Devices (Router, Switch, Bridge, Hub):** Used to access the Internet.	**SPM** 1) SPM will contain all individual student evaluations. 2) Organizes and calculates individual data to show various trends of the department, school, or university. **Operating System** Any OS used by the users, e.g. Windows, Mac. **Printing Software** Printing software used for printing the curriculum.	**SPM Database** Will contain all data regarding COs and PLOs for individual students.	**Internet** To access SPM. b) Reports may be requested or sent via email which is accessed by using the Internet.

9. Arrange, compare, and depict data through graphs, tables, and other illustrations.	**Higher Management** Conduct studies on student performance records, attendance records, and faculty evaluations. 2) Create and print reports using a graphical report generation software. **Admin** Perform studies and create reports based on given parameters as required or ordered from higher management.	**Paper** Needed for printing out hard copies of generated reports.	**Computer** 1) Used for viewing and analyzing data through SPM. 2) Administrators use computers to design special reports based on given parameters. **Printers** Used for printing hard copy of reports.	**SPM** Retrieves performance records for analysis. **AMS** Retrieves attendance records for assessing student punctuality. **CMS** Retrieve data on curriculum changes to compare versions. **Graphical Report Generation Software** Create graphs, tables and other illustrations based on manual input or requests from IRAS, SPM, AMS or CMS. **Outlook** Send reports to those with a shared interest. **Operating System** Any OS used by the users, e.g. Windows, Mac. **Printing Software** Used for running the printer printing the curriculum. **File Format Conversion Software** Used for converting reports into different format for supporting various views.	**SPM Database** 1) Conduct studies based on student performance records such as PLOs, COs, grades, GPA. 2) Conduct studies on historical records to detect process performance improvement, process bottlenecks, course content difficulty and managerial effectiveness in PLOs. **AMS Database** 1) Gather attendance records for creating graphs based on student punctuality. 2) Compare punctuality ratios with other ratios published by universities. **CMS Database** Compare the changes in curriculum and its impact on student performance based on historical records in the SPM Database.	**Internet** Used to access all the databases on the server for information retrieval and data analysis.

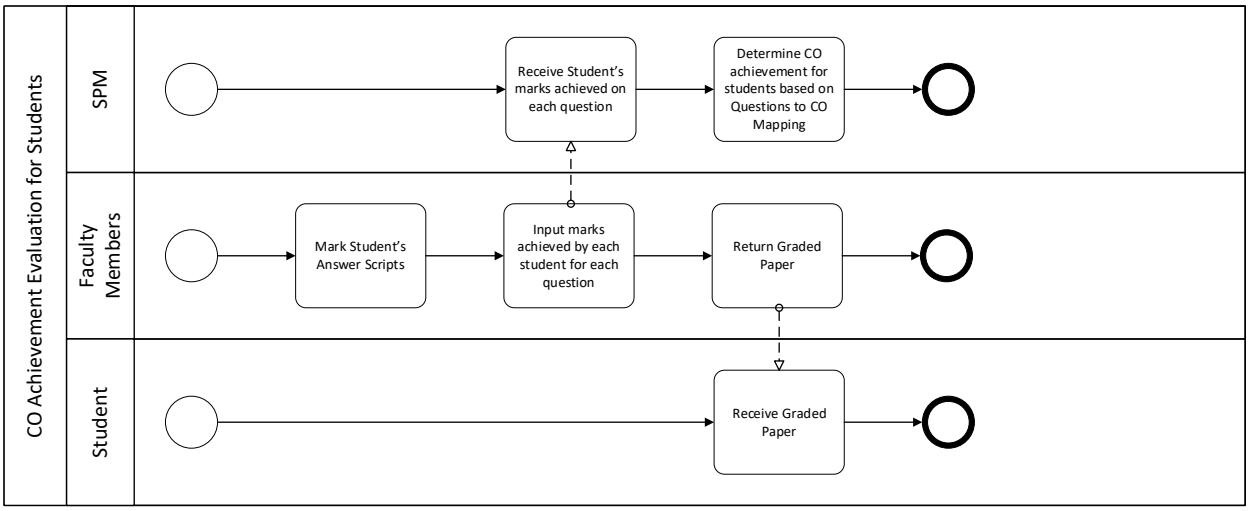
BPMN 2.0 Diagram (TO-BE):



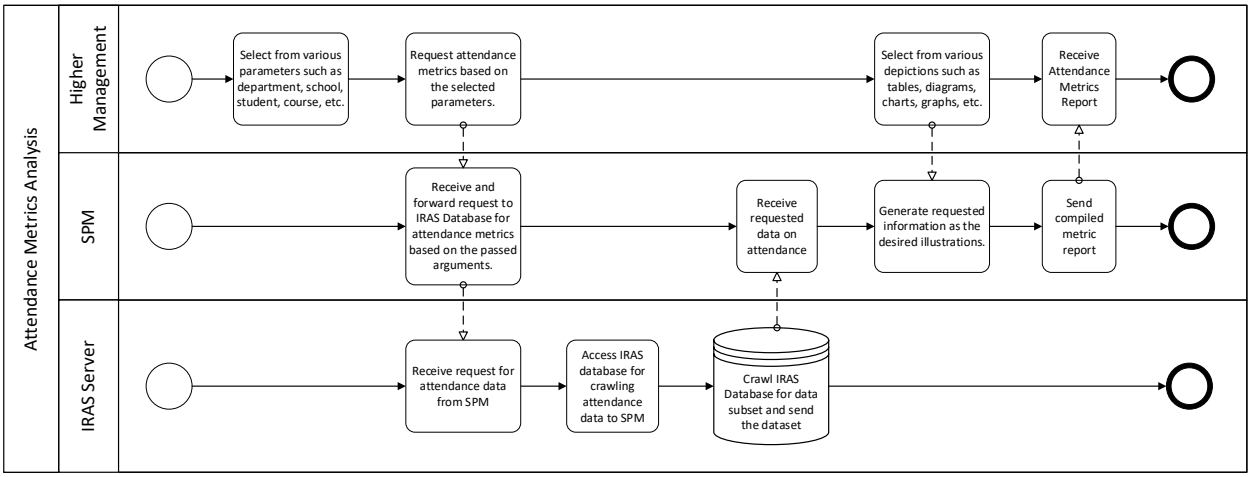
Process Diagram for Curriculum Approval and PLO Update



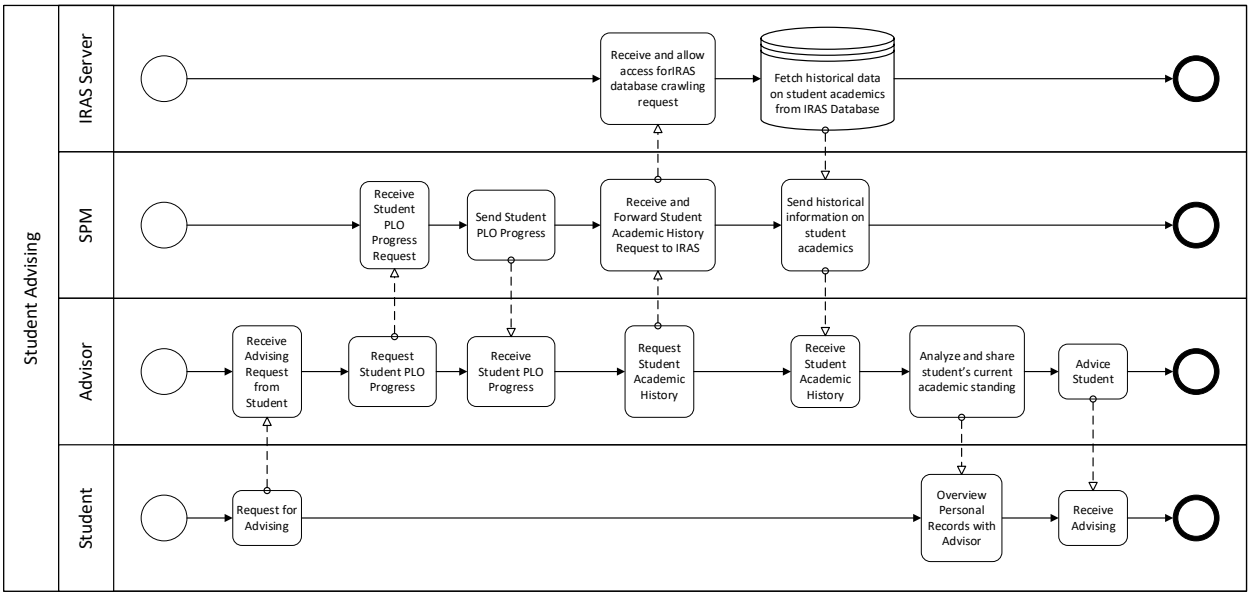
Process Diagram on Basing Examination Questions according to COs



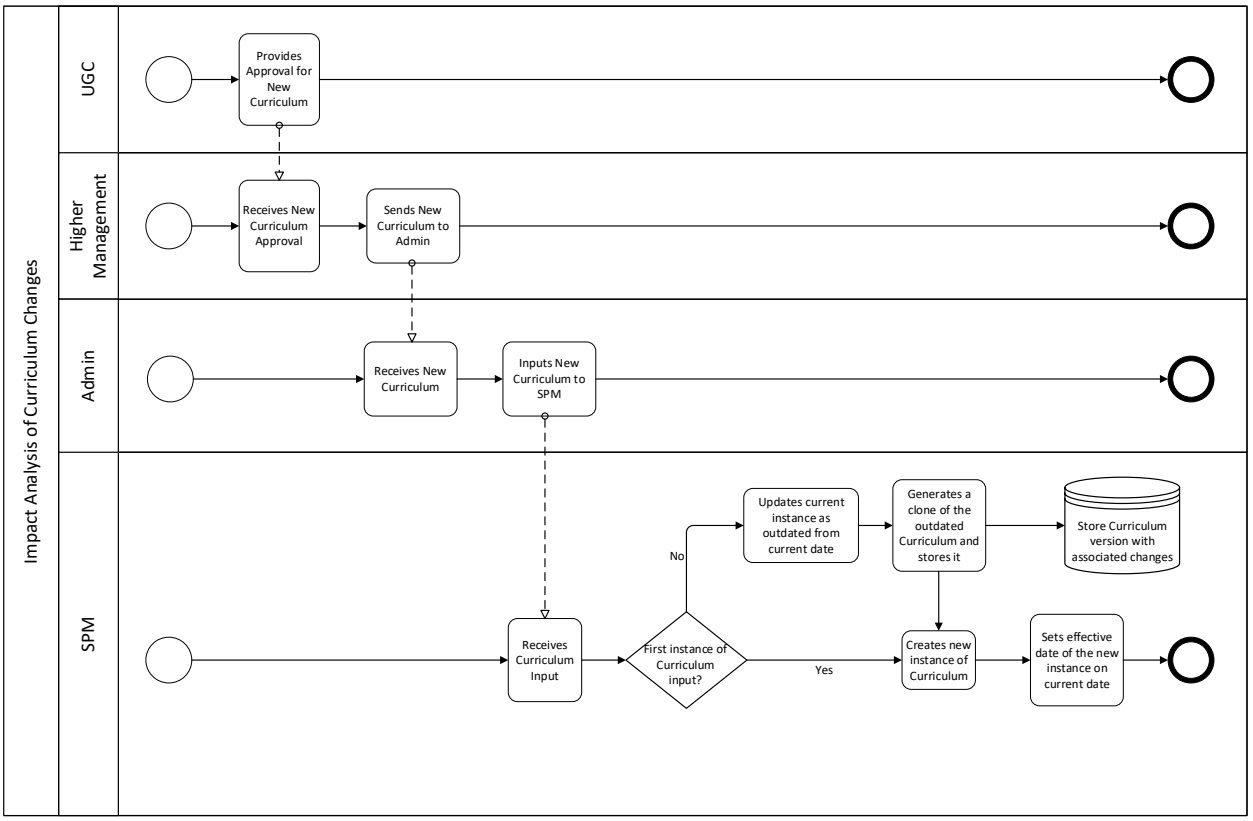
Process Diagram on CO Achievement for Students



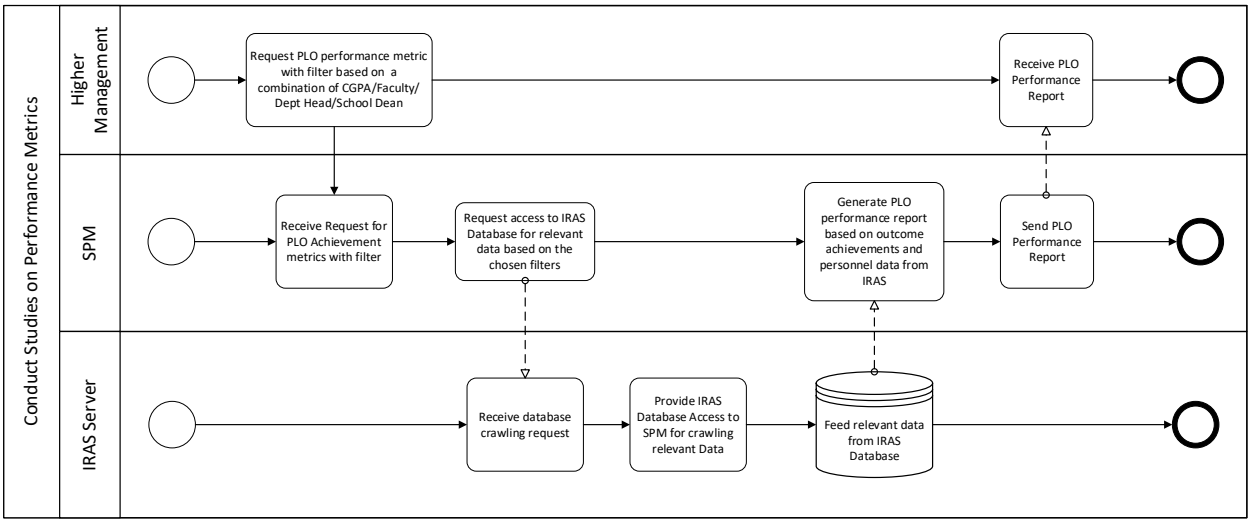
Process Diagram on Attendance Metrics Analysis



Process Diagram on Student Advising



Process Diagram on Impact Analysis of Curriculum Changes

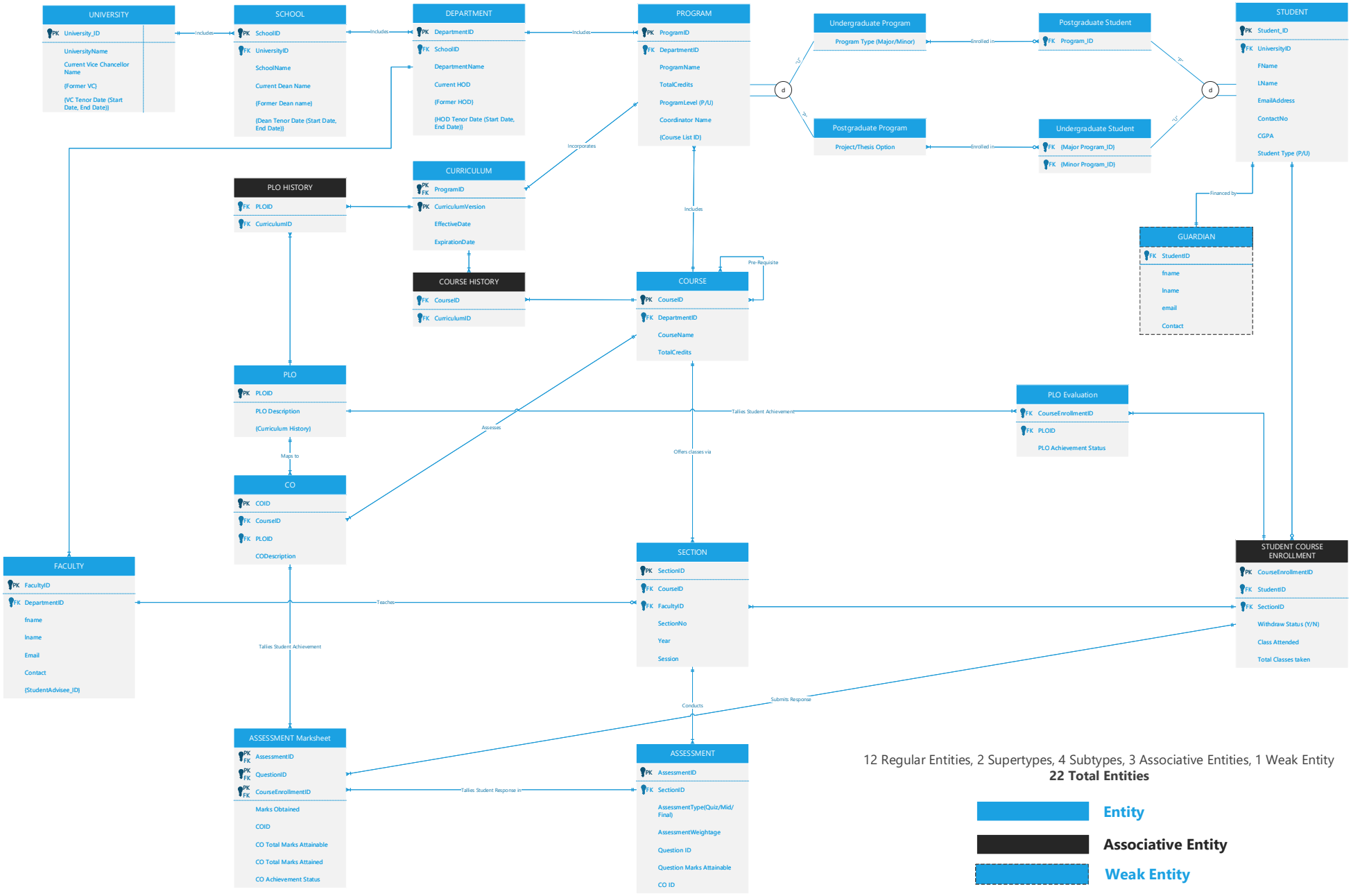


Process Diagram on Performance Metrics Analysis

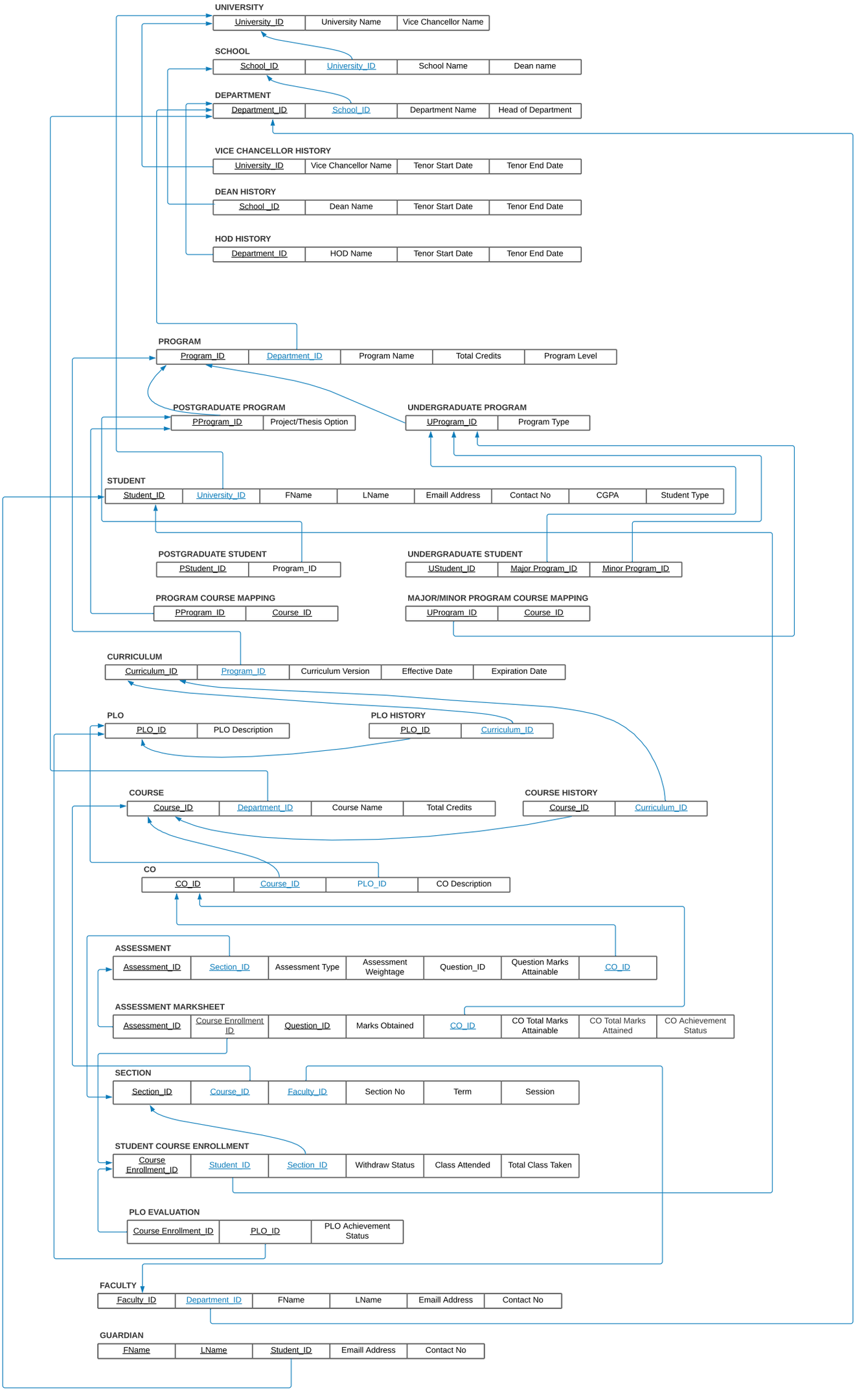
Business rules

- A department may include many programs or at least one program
- A school may include many departments or at least one department
- An university may include many or at least one school
- A student must be either a Postgraduate Student or a Undergraduate Student; therefore must be enrolled in the Postgraduate program or the Undergraduate program respectively
- A program whether it be Undergraduate or Postgraduate can have zero or more student enrolled under it
- A undergraduate student must have a major and a minor program under their degree
- A program incorporates at least one or more curricula
- A department is delegated to at least one or more Faculty members who are in turn incharge of teaching zero or more sections of a course under the same department via classes
- Each section has one or more assessments conducted by the faculty. One or more assessment responses by the students are tallied via the Assessment marksheet.
- One Assessment marksheet can include one or more assessments but one assessment can tally up in one and only one assessment marksheet
- Student achievement can be tallied up via one or many assessment marksheet to get the total CO (course outcome)
- One or many CO can map to get a PLO (program learning outcome) which can later be evaluated by the higher authority
- One course can have one or more COs

Enhanced Entity Relationship Diagram



University	University_ID	u1
	University Name	u2
	Vice Chancellor Name	u3
School	School_ID	s1
	University_ID	u1
	School Name	s2
	Dean Name	s3
Department	Department_ID	d1
	School_ID	s1
	Department Name	d2
	Head of Department	d3
Vice Chancellor History	University_ID	u1
	Vice Chancellor Name	v1
	Tenor Start Date	v2
	Tenor End Date	v3
Dean History	School_ID	s1
	Dean Name	dh1
	Tenor Start Date	dh2
	Tenor End Date	dh3
HOD History	Department_ID	d1
	HOD Name	h1
	Tenor Start Date	h2
	Tenor End Date	h3
Program	Program_ID	p1
	Department_ID	d1
	Program Name	p2
	Total Credits	p3
	Program level	p4
Postgraduate Program	PProgram_ID	p1
	Project/Thesis Option	pp1
Undergraduate Program	UProgram_ID	p1
	Program Type	up1
Curriculum	Curriculum_ID	cu1
	Program_ID	p1
	Curriculum Version	cu2
	Effective Date	cu3
	Expiration Date	cu4
PLO	PLO_ID	pl1
	PLO Description	pl2
PLO History	PLO_ID	pl1
	Curriculum_ID	cu1
Course	Course_ID	c1
	Department_ID	d1
	Course Name	c2
	Total Credits	c3
Course History	Course_ID	c1
	Curriculum_ID	cu1
Faculty	Faculty_ID	f1
	Department_ID	d1
	Fname	f2
	Lname	f3
	Email Address	f4
	Contact No	f5
Student	Student_ID	st1
	University_ID	u1
	Fname	st2
	Lname	st3
	Email Address	st4
	Contact No	st5
	CGPA	st6
	Student Type	st7
Postgraduate Student	PStudent_ID	st1
	Program_ID	p1
Undergraduate Student	UStudent_ID	st1
	Major Program_ID	p1
	Minor Program_ID	p1
CO	CO_ID	co1
	Course_ID	c1
	PLO_ID	pl1
	CO Description	co2
Assessment	Assessment_ID	a1
	Section_ID	sc1
	Assesment Type	a2
	Assessment Weightage	a3
	Question_ID	q1
	Question Marks Attainable	q2
	CO_ID	co1
Assessment Marksheet	Assessment_ID	a1
	Course Enrollment ID	ce1
	Question_ID	q1
	Marks Obtained	am1
	CO_ID	co1
	CO Total Marks Attainable	am2
	CO Total Marks Attained	am3
	CO Achievement Status	am4
Section	Section_ID	sc1
	Course_ID	c1
	Faculty_ID	f1
	Section No	sc2
	Term	sc3
	Session	sc4
Student Course Enrollment	Course Enrollment ID	ce1
	Student ID	st1
	Section ID	sc1
	Withdraw Status	ce2
	Class Attended	ce3
	Total Class Taken	ce4
PLO Evaluation	Course Enrollment ID	ce1
	PLO_ID	pl1
	PLO Achievement Status	plo1
Guardian	Fname	g1
	Lname	g2
	Student_ID	st1
	Email Address	g3
	Contact No	g4


UNIVERSITY
University_ID
University Name
Vice Chancellor Name
SCHOOL
School_ID
University_ID
School Name
Dean name
DEPARTMENT
Department_ID
School_ID
Department Name
Head of Department
VICE CHANCELLOR HISTORY
University_ID
Vice Chancellor Name
Tenor Start Date
Tenor End Date
DEAN HISTORY
School _ID
Dean Name
Tenor Start Date
Tenor End Date
HOD HISTORY
Department_ID
HOD Name
Tenor Start Date
Tenor End Date
PROGRAM
Program_ID
Department_ID
Program Name
Total Credits
Program Level
POSTGRADUATE PROGRAM
PProgram_ID
Project/Thesis Option
UNDERGRADUATE PROGRAM
UProgram_ID
Program Type
STUDENT
Student_ID
University_ID
FName
LName
Emaill Address
Contact No
CGPA
Student Type
POSTGRADUATE STUDENT
PStudent_ID
Program_ID
UNDERGRADUATE STUDENT
UStudent_ID
Major Program_ID
Minor Program_ID
PROGRAM COURSE MAPPING
PProgram_ID
Course_ID
MAJOR/MINOR PROGRAM COURSE MAPPING
UProgram_ID
Course_ID
CURRICULUM
Curriculum_ID
Program_ID
Curriculum Version
Effective Date
Expiration Date
PLO
PLO_ID
PLO Description
PLO HISTORY
PLO_ID
Curriculum_ID
COURSE
Course_ID
Department_ID
Course Name
Total Credits
COURSE HISTORY
Course_ID
Curriculum_ID
CO
CO_ID
Course_ID
PLO_ID
CO Description
ASSESSMENT
Assessment_ID
Section_ID
Assessment Type
Assessment Weightage
Question_ID
Question Marks Attainable
CO_ID
ASSESSMENT MARKSHEET
Assessment_ID
Course Enrollment ID
Question_ID
Marks Obtained
CO_ID
CO Total Marks Attainable
CO Total Marks Attained
CO Achievement Status
SECTION
Section_ID
Course_ID
Faculty_ID
Section No
Term
Session
STUDENT COURSE ENROLLMENT
Course Enrollment_ID
Student_ID
Section_ID
Withdraw Status
Class Attended
Total Class Taken
PLO EVALUATION
Course Enrollment_ID
PLO_ID
PLO Achievement Status
FACULTY
Faculty_ID
Department_ID
FName
LName
Emaill Address
Contact No
GUARDIAN
FName
LName
Student_ID
Emaill Address
Contact No

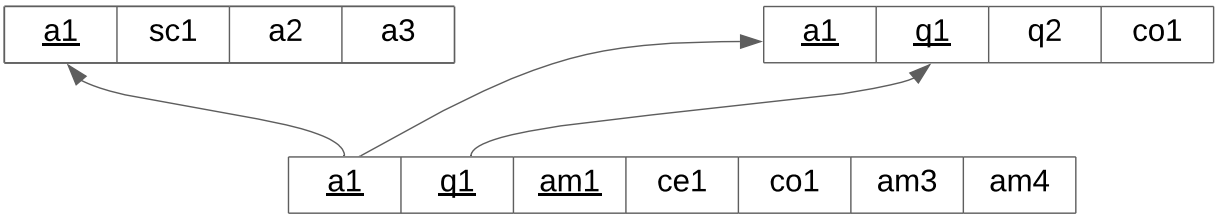
Functional Dependencies

a1 →	sc1, a2, a3
a1, q1 →	q2, co1
a1, q1, am1 →	ce1
ce1 →	co1, am2, am3, am4

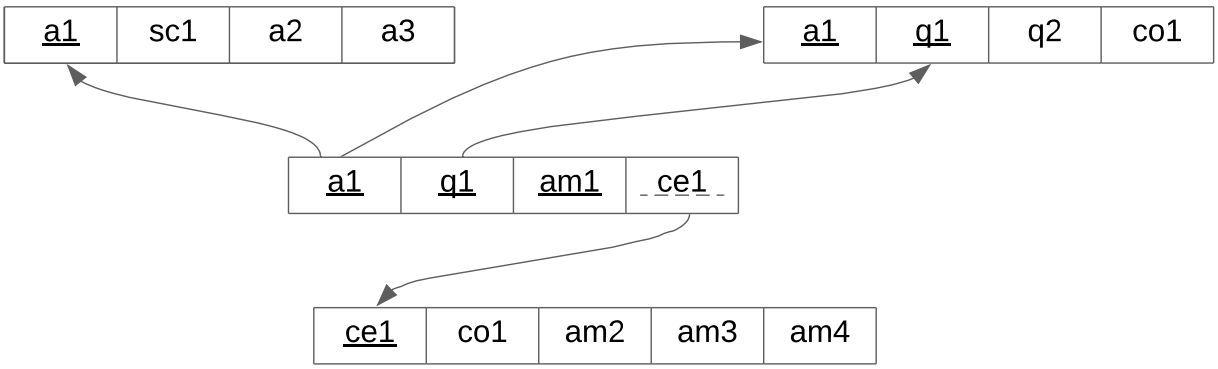
1NF:

a1	q1	am1	a2	a3	q2	co1	ce1	sc1	am2	am3	am4

2NF:



3NF:



BCNF:

No non-key can identify any primary key or part of the primary key. Therefore, all the relations are in BCNF.

Program_T

Name	Data Type	Size	Remark
Program_ID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
Department_ID	GUID	14	This is the foreigh key from Department Table.
ProgramName	VARCHAR	40	This is the name of the degree program. Example: "Computer Science and Engineering"
TotalCredits	INTEGER	3	This is the total require credits for this program. Example: 120
ProgramLevel	CHAR	1	This is a type discriminator between postgraduate programs and undergraduate programs. Example: "U" or "P"

PostgradProgram_T

Name	Data Type	Size	Remark
PProgram_ID	GUID	14	This is a foreign key from Program Table
ProjectThesisOption	CHAR	1	This is the discriminator for master's program which either offer a project stream or a thesis stream to finish the program. Example: "P" or "T"

UndergradProgram_T

Name	Data Type	Size	Remark
UProgram_ID	GUID	14	This is a foreign key from Program Table
ProgramType	CHAR	2	This is a type discriminator for the focus level of a program, a program can either be a minor or a major. Example: "MJ" or "MN"

ProgramCourseMapping_T

Name	Data Type	Size	Remark
Program_ID	GUID	14	This is a foreign key from Program Table
Course_ID	VARCHAR	10	This is a foreign key from the course table

Course_T

Name	Data Type	Size	Remark
Course_ID	**VARCHAR**	**10**	**This will serve as the unique identfier for each course . Example: "IUBCSE303".**
Department_ID	GUID	14	This is the foreigh key from Department Table.
CourseName	VARCHAR	40	This is the name of the course. Example: "Business Finance I"
Total Credits	INTEGER	1	This is the total require credits for this course. Example: 3

Curriculum_T

Name	Data Type	Size	Remark
Curriculum_ID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
Program_ID	GUID	14	This a foreign key from the program table.
CurriculumVersion	CHAR	2	This is the version number of curriculum deployed. Example: "V1"
EffectiveDate	DATE	YYYY-MM-DD	This is the date from which this curriculum was integrated into practice. Example: "2001-01-01"
ExpirationDate	DATE	YYYY-MM-DD	This is the date on which the curriculum was rendered obsolete and replaced by a newer version. Example: "2019-03-02"

PLO_T

Name	Data Type	Size	Remark
PLO_ID	**VARCHAR**	**13**	**This will serve as the unique identfier for each PLO, composed of the shorthands for University name, Program level, Program Name and PLO Number. Example: "IUBUCSE01".**
PLOName	VARCHAR	15	This stores the name of the PLO. Example: "Knowledge"
PLODescription	TEXT	200	This stores the description of the PLO. Example: "Knowledge: An ability to select and apply the knowledge, techniques, skills, and modern tools of the computer science and engineering discipline."

PLOHistory_T

Name	Data Type	Size	Remark
PLO_ID	VARCHAR	13	This is a foreign key from the PLO Table
Curriculum_ID	GUID	14	This is a foreign key from the Curriculum table

CourseHistory_T

Name	Data Type	Size	Remark
Course_ID	VARCHAR	10	This is a foreign key from the Course table
Curriculum_ID	GUID	14	This is a foreign key from the Curriculum table

CO_T

Name	Data Type	Size	Remark

CO_ID	**VARHAR**	**14**	**This will serve as the primary key composed of the course id and CO version. Example: "IUBCSE101CO1"**
Course_ID	VARCHAR	10	This a foreign key from the Course table.
PLO_ID	VARCHAR	13	This is a foreign key from PLO table
CODescription	TEXT	200	This stores the description of the CO. Example: "Use Kirchhoff's laws, circuit theorems and node voltage methodology to solve simple DC as well as AC circuits."

C

Section_T

Name	Data Type	Size	Remark
Section_ID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
Course_ID	VARCHAR	10	This a foreign key from the Course table.
Faculty_ID	VARCHAR	7	This is a foreign key from the Faculty table
SectionNo	INT	2	This is the section number. Example: 02
TimeSlot	TIME	HH-MM-SS	This is the start time for the section's scheduled class.
Days	VARCHAR	2	This represent the weekdays on which the classes are scheduled to be taken. Example: "ST", "MW"
Year	INT	4	This is the year on which this section's classes were scheduled on. Example: "2019"
Session	VARCHAR	8	This is the semester session out of the three trimesters the section's classes were scheduled on. Example: "Autumn"

University_T

Name	Data Type	Size	Remark
University_ID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
University_Name	VARCHAR	30	This will store the name of the university. Example: "Independent University Bangladesh"
VCFname	VARCHAR	15	This will store the first name of the current Vice Chancellor. Example: "Milan"
VCLname	VARCHAR	15	This will store the last name of the current Vice Chancellor. Example: "Pagon"

VCHistory_T

Name	Data Type	Size	Remark
University_ID	GUID	14	This is a foreign key from the University Table
VCFname	VARCHAR	15	This will store the first name of a Vice Chancellor. Example: "Milan"
VCLname	VARCHAR	15	This will store the last name of a Vice Chancellor. Example: "Pagon"
TenorStartDate	DATE	YYYY-MM-DD	This will store the starting date of the VC's tenor.
TenorEndDate	DATE	YYYY-MM-DD	This will store the ending date of the VC's tenor.

School_T

Name	Data Type	Size	Remark
School_ID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
University_ID	GUID	14	This is a foreign key from the University Table
SchoolName	VARCHAR	15	This will store the name of the school. Example: "School of Business and Entreprenuership"
DeanFname	VARCHAR	15	This will store the first name of the current Dean. Example: "Milan"
DeanLname	VARCHAR	15	This will store the last name of the current Dean. Example: "Pagon"

DeanHistory_T

Name	Data Type	Size	Remark
School_ID	GUID	14	This is a foreign key from the School Table
DeanFname	VARCHAR	15	This will store the first name of a Dean. Example: "Milan"
DeanLname	VARCHAR	15	This will store the last name of a Dean. Example: "Pagon"
TenorStartDate	DATE	YYYY-MM-DD	This will store the starting date of the Dean's tenor.
TenorEndDate	DATE	YYYY-MM-DD	This will store the ending date of the Dean's tenor.

Deparment_T

Name	Data Type	Size	Remark
Deparment_T	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
School_ID	GUID	14	This is a foreign key from the School Table
SchoolName	VARCHAR	15	This will store the name of the school. Example: "Department of Computer Science and Engineering"

HODFname	VARCHAR	15	This will store the first name of the current Dean. Example: "Dr. Mahady"
HODLname	VARCHAR	15	This will store the last name of the current HOD. Example: "Hasan"

HODHistory_T

Name	Data Type	Size	Remark
Deparment_T	GUID	14	This is a foreign key from the Department Table
HODFname	VARCHAR	15	This will store the first name of a HOD. Example: "Dr. Mahady"
HODLname	VARCHAR	15	This will store the last name of a HOD. Example: "Hasan"
TenorStartDate	DATE	YYYY-MM-DD	This will store the starting date of the HOD's tenor.
TenorEndDate	DATE	YYYY-MM-DD	This will store the ending date of the HOD's tenor.

Student_T

Name	Data Type	Size	Remark
Student_ID	**VARCHAR**	**7**	**This will serve as the primary for this table. Example: "1930234"**
University_ID	GUID	14	This a foreign key from the University table.
Fname	VARCHAR	15	This will be the student's first name. Example: "Tahzeeb"
Lname	VARCHAR	15	This will be the student's last name. Example: "Ahmed"
Email	VARCHAR	20	This will store the student's email address. Example: "tahzeeb2g@gmail.com"
Contact	VARCHAR	13	This will store the student's contact number. Example: "01914602860"
CGPA	DECIMAL	3	This will store the student's current CGPA. Example: "3.92"
StudentType	CHAR	1	This will be a type discriminator between Postgraduate and Undergraduate students. Example: "U" or "P"

UndergradStudent_T

Name	Data Type	Size	Remark
Student_ID	VARCHAR	7	This is a foreign key from the Student table
MajorProgramID	GUID	14	This is a foreign key from the Program table.
MinorProgramID	GUID	14	This is a foreign key from the Program table.

PostgradStudent_T

Name	Data Type	Size	Remark

Student_ID	VARCHAR	7	This is a foreign key from the Student table
Program_ID	GUID	14	This is a foreign key from the Program table.

Guardian_T

Name	Data Type	Size	Remark
Student_ID	VARCHAR	7	This is a foreign key from the Student table
Fname	VARCHAR	15	This will be the guardian's first name. Example: "Ruhin"
Lname	VARCHAR	15	This will be the guardian's last name. Example: "Illahi"
Email	VARCHAR	20	This will store the student's email address. Example: "tahzeeb2g@gmail.com"
Contact	VARCHAR	13	This will store the student's contact number. Example: "01914602860"

StudentCourseEnrollment_T

Name	Data Type	Size	Remark
CourseEnrollmentID	**GUID**	**14**	**This will serve as the primary key composed auto generated by the NewSequentialID() function.**
Student_ID	VARCHAR	7	This is a foreign key from the Student table
Section_ID	GUID	14	This is a foreign key from the Section table.
ClassAttended	INT	2	This will record the number of classess attended by a student in a section. Example: 18
TotalClasses	INT	2	This will record the number of classess taken by a section. Example: 18

Faculty_T

Name	Data Type	Size	Remark
Faculty_ID	**VARCHAR**	**7**	**This will serve as the primary for this table. Example: "1630234"**
Department_ID	GUID	14	This a foreign key from the Department table.
Fname	VARCHAR	15	This will be the Faculty's first name. Example: "MD Abu"
Lname	VARCHAR	15	This will be the Faculty's last name. Example: "Sayed"
Email	VARCHAR	20	This will store the faculty's email address. Example: "a. sayed@gmail.com"
Contact	VARCHAR	13	This will store the faculty's contact number. Example: "01914602860"

Assessment_T

Name	Data Type	Size	Remark
Assessment_ID	**GUID**	**14**	**This will serve as the primary for this table.**
Section_ID	GUID	14	This a foreign key from the Section table.
AssessmentType	CHAR	1	This will serve as the assessment discriminator between quizzes, midterms and finals. Example: "Q" or "M" or "F"
AssessmentWeightage	DECIMAL	3	This will store the assessment's total obtainable marks as a weight of the entire course. Example: "0.25"

Question_T

Name	Data Type	Size	Remark
Question_ID	**GUID**	**14**	**This will serve as the primary for this table.**
Assessment_ID	GUID	14	This a foreign key from the Assessment table.
CO_ID	VARHAR	14	This is a foreign key from the CO table
MarksAttainable	INT	2	This will store the total attainable marks for each question. Example: 5

AssessmentMarksheet_T

Name	Data Type	Size	Remark
Assessment_ID	GUID	14	This is a foreign key from the Assessment table.
Question_ID	GUID	14	This is a foreign key form the Question table
CourseEnrollmentID	GUID	14	This is the foreign key from the StudentCourseEnrollment table
MarksObtained	INT	2	This will store the marks obtained on each question by the student.

COEvaluation_T

Name	Data Type	Size	Remark
CourseEnrollmentID	GUID	14	This is the foreign key from the StudentCourseEnrollment table
CO_ID	VARHAR	14	This is a foreign key from the CO table
COMarksAttainable	INT	3	This will store the total attainable marks for each course outcome for each section. Example: 120
COMarksObtained	INT	3	This will store the total marks obtained by a student for the current CO for each section. Example: 113
COBenchmark	DECIMAL	3	This will store the minimum percentage required by a student in a course to achieve a course outcome.

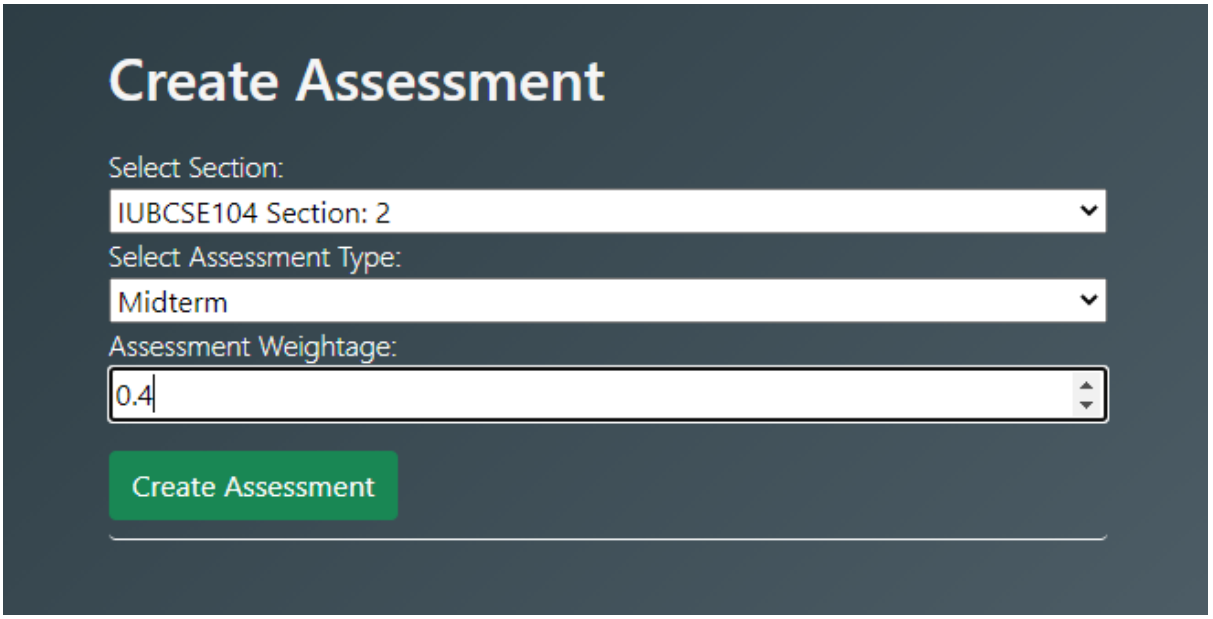
COAchivementStatus	CHAR	1	This will store the CO achievement status for a student in a course, it can either be "Y" or "N"

PLOEvaluation_T

Name	Data Type	Size	Remark
CourseEnrollmentID	GUID	14	This is the foreign key from the StudentCourseEnrollment table
PLO_ID	VARCHAR	13	This is a foreign key from the PLO Table
PLOAchivementStatus	CHAR	1	This will store the PLO achievement status for a student in a course, it can either be "Y" or "N"

Input Forms:

Creating Assessment



views.py

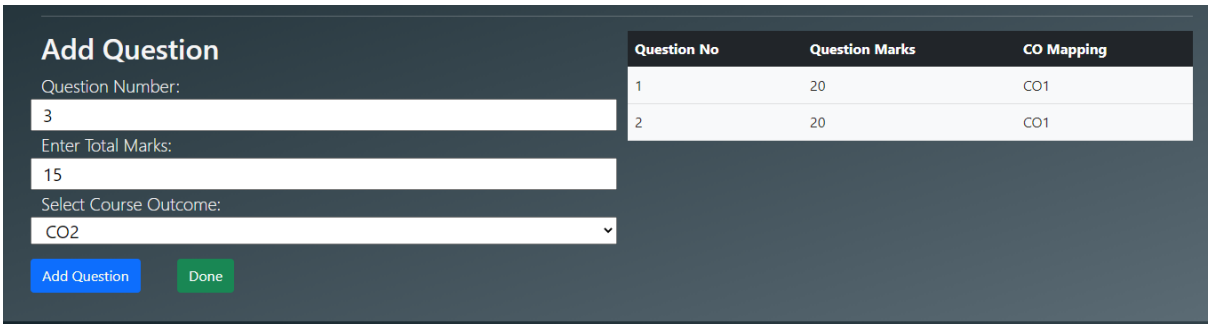
```
    class NewAssessmentForm(forms.Form):
        section = forms.ChoiceField(
            choices=sectionChoices, label="Select Section")
        assessmentType = forms.ChoiceField(
            choices=assessmentChoices, label="Select Assessment Type")
        assessmentWeight = forms.DecimalField(label="Assessment
Weightage")
    if request.method == "POST":
        form = NewAssessmentForm(request.POST)
        if form.is_valid():
            sectionID = form.cleaned_data["section"]
            assessmentType = form.cleaned_data["assessmentType"]
            assessmentWeightage = form.cleaned_data["assessmentWeight"]
            message = "Assessment Created Successfully!"

            # To pass data into Next view
            request.session["assessmentID"] =
createAssessment(str(sectionID), assessmentType,

assessmentWeightage)
            # To pass data into Next view
            request.session["courseID"] = getCourse(str(sectionID))
```

```
        else:
            message = "Invalid input, Assessment not created."

        # For table update in real time.
        assessments = getCurrentAssessments(facultyid)

        return mapQuestions(request)
```

facultyqueries.py

```
def createAssessment(sectionID, assessmentType, assessmentWeight):
    with connection.cursor() as cursor:
        cursor.execute('''INSERT INTO assessment_t VALUES (UUID(), %s,
%s, %s);''', (
            sectionID, assessmentType, assessmentWeight))

    with connection.cursor() as cursor:
        cursor.execute('''SELECT assessment_t.assessment_id FROM
assessment_t WHERE assessment_t.section_id="{}" AND
assessment_t.assessmentType="{}" AND assessment_t.assessmentWeightage =
{};'''.format(
            sectionID, assessmentType, assessmentWeight))
        assessmentID = cursor.fetchone()
    return assessmentID[0]
```

Mapping Questions to COs



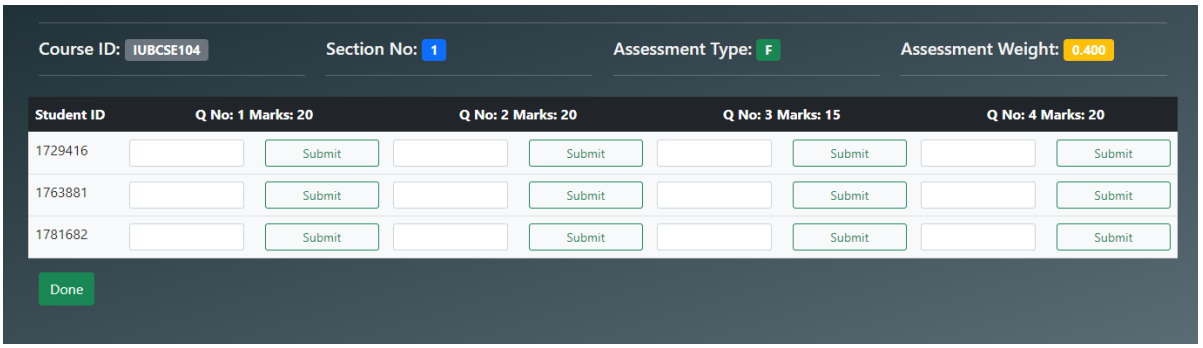
views.py

```
    class NewQuestion(forms.Form):
        qNo = forms.IntegerField(label="Question Number")
        marksAttainable = forms.IntegerField(label="Enter Total Marks")
        coID = forms.ChoiceField(
            choices=coChoices, label="Select Course Outcome")
    if request.method == "POST":
        form = NewQuestion(request.POST)

        if form.is_valid():
            qNo = form.cleaned_data["qNo"]
            marks = form.cleaned_data["marksAttainable"]
            coID = form.cleaned_data["coID"]
            createQuestion(qNo, request.session["assessmentID"], coID,
marks)
            questionList =
getQuestionList(request.session["assessmentID"])

    return render(request, "SPM/mapQuestions.html", {
        "form2": NewQuestion,
        "courseID": request.session["courseID"],
        "assessmentID": request.session["assessmentID"],
        "questionList": questionList
    })
```

facultyqueries.py

```
def createQuestion(qNo, assessmentID, coID, marks):
    with connection.cursor() as cursor:
        cursor.execute(
            """INSERT INTO question_t VALUES(UUID(),%s, %s, %s,
%s);""", (qNo, assessmentID, coID, marks))
```

Insert Students' Marksheet for each Question



views.py

```
    if request.method == "POST":
        # Getting questionID and ceID
        keys = request.POST.keys()
        keys = list(keys)

        # Getting marks
        marks = request.POST[keys[1]]

        # Splitting IDs
        IDs = keys[1].split(",")
        questionID = IDs[0]
        ceID = IDs[1]

        # Updating marks to spm db
        entryQuestionMark(assessmentID, questionID, ceID, marks)

    #--------------------For front-end interactions--------------#
        # forming buttonID
        buttonID = ""
        buttonID = buttonID+ceID+","+questionID

        # Appending CellIDs and ButtonIDs
        cellIDs.append(str(keys[1]))
        buttonIDs.append(str(buttonID))

        # Converting to JSON
        data1 = dumps(cellIDs)
        data2 = dumps(buttonIDs)
    #------------------------------------------------------------#
        return render(request, "SPM/marksheet.html", {
            "assessmentID": assessmentID,
            "assessmentInfo": assessmentInfo,
```

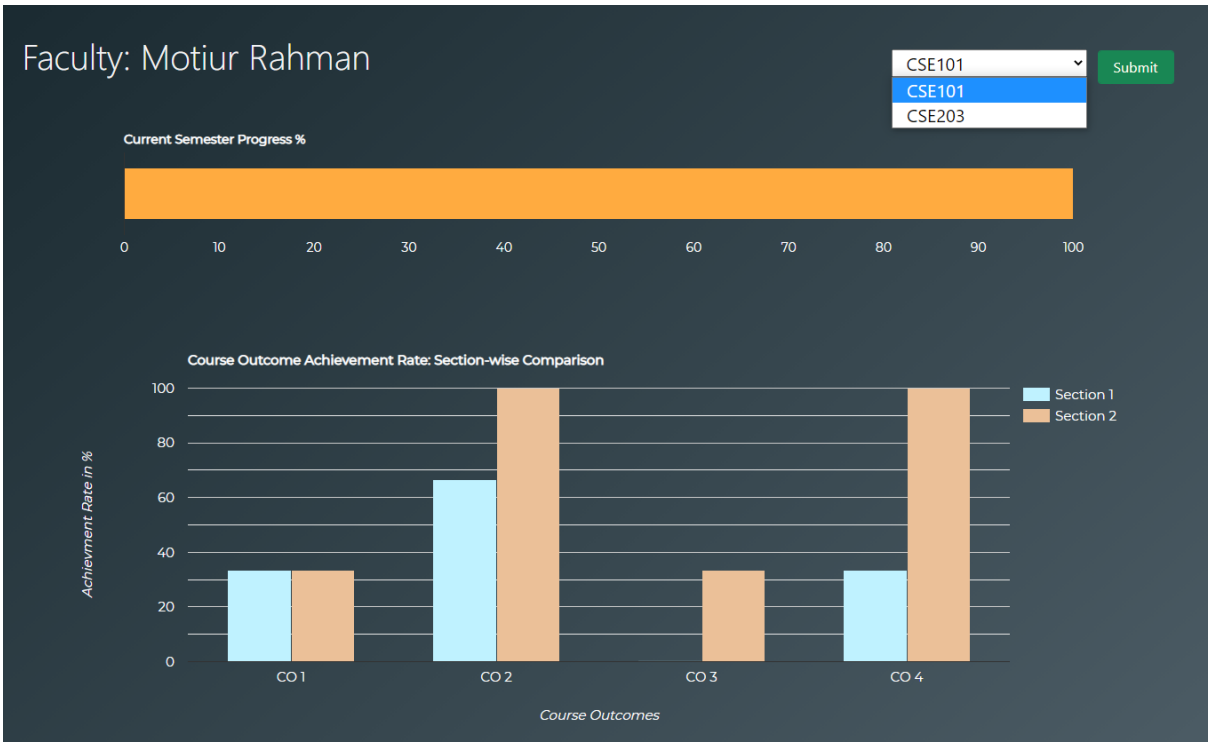
```
            "courseEnrollmentIDs": courseEnrollmentIDs,
            "questions": questions,
            "cellIDs": data1,
            "buttonIDs": data2,
        })
```

facultyqueries.py

```
def entryQuestionMark(assessmentID, questionID, ceID, marks):
    with connection.cursor() as cursor:
        cursor.execute(
            """INSERT INTO assessmentmarksheet_t VALUES (%s, %s, %s,
%s);""", (assessmentID, questionID, ceID, marks))
```

Output Forms

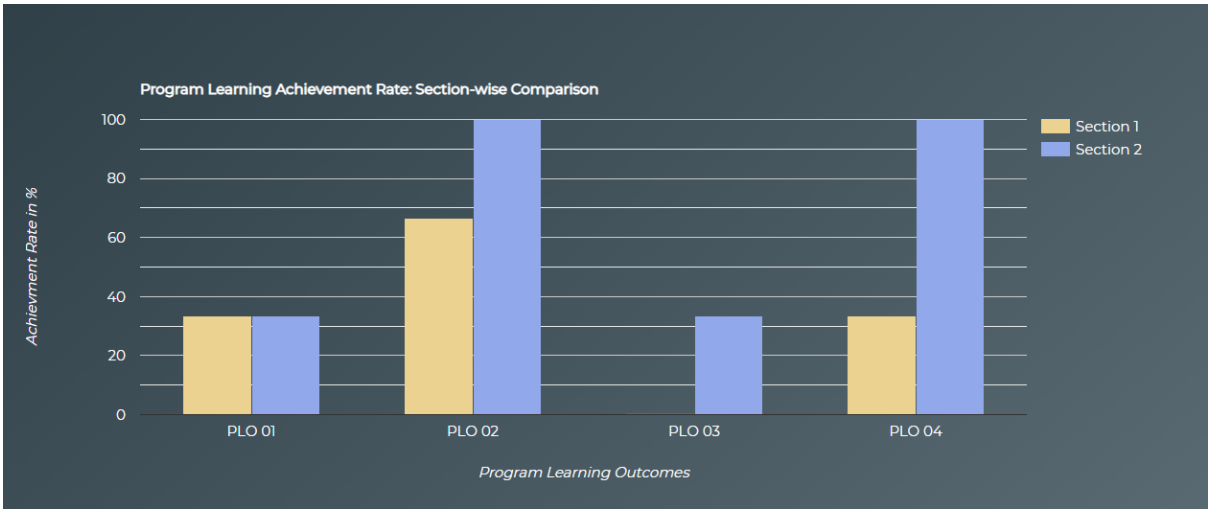
Faculty-wise current semester's section CO Achievement Rate Comparisons



graphqueries.py

```
def getCOEvaluationRate(sectionID, coID):
    sql_query = '''SELECT ((SELECT COUNT(*) FROM coevaluation_t,
studentcourseenrollment_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = "{}" AND coevaluation_t.co_id =
"{}" AND coevaluation_t.coAchievementStatus = 'Y')/(SELECT COUNT(*)
FROM coevaluation_t, studentcourseenrollment_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = "{}" AND coevaluation_t.co_id =
"{}"));'''
    with connection.cursor() as cursor:
        cursor.execute(sql_query.format(sectionID, coID, sectionID,
coID))
        row = cursor.fetchall()
        rate = round((float(row[0][0])), 3)*100
    return rate
```

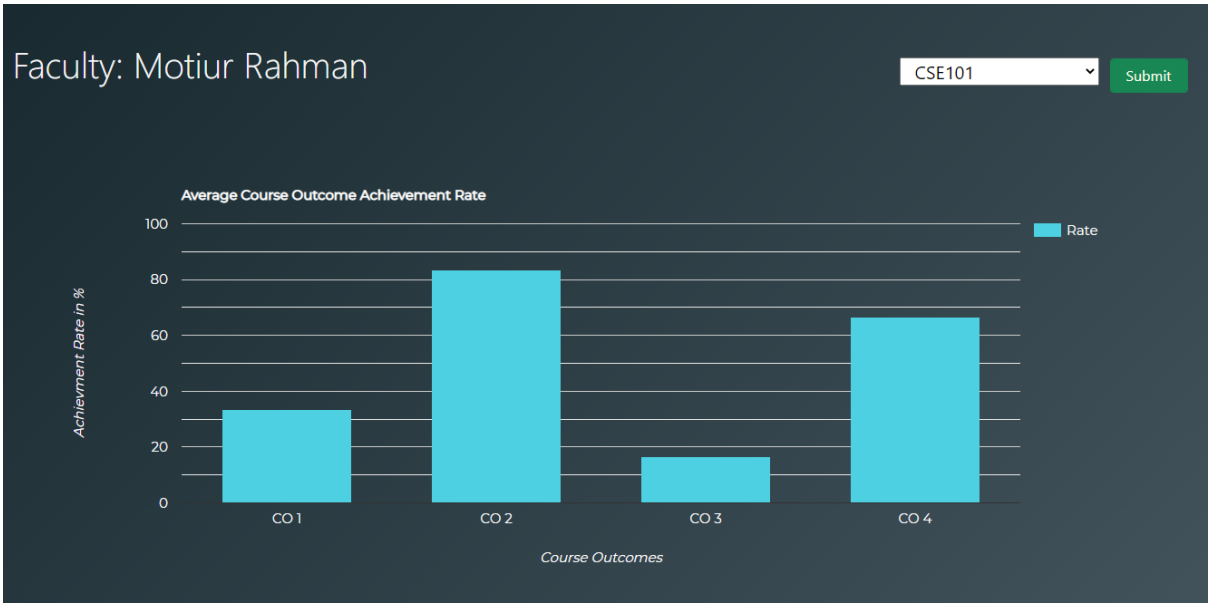
Faculty-wise current semester's section PLO Achievement Rate section-wise comparison



graphqueries.py

```
def getPLOEvaluationRate(sectionID, ploID):
    sql_query = '''SELECT(SELECT COUNT(*) FROM ploevaluation_t,
studentcourseenrollment_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = "{}" AND ploevaluation_t.plo_id
= "{}" AND ploevaluation_t.ploAchievementStatus = 'Y')/(SELECT COUNT(*)
FROM ploevaluation_t, studentcourseenrollment_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = "{}" AND ploevaluation_t.plo_id
= "{}");'''

    with connection.cursor() as cursor:
        cursor.execute(sql_query.format(sectionID, ploID, sectionID,
ploID))
        row = cursor.fetchall()
        rate = round((float(row[0][0])), 3)*100
    return rate
```

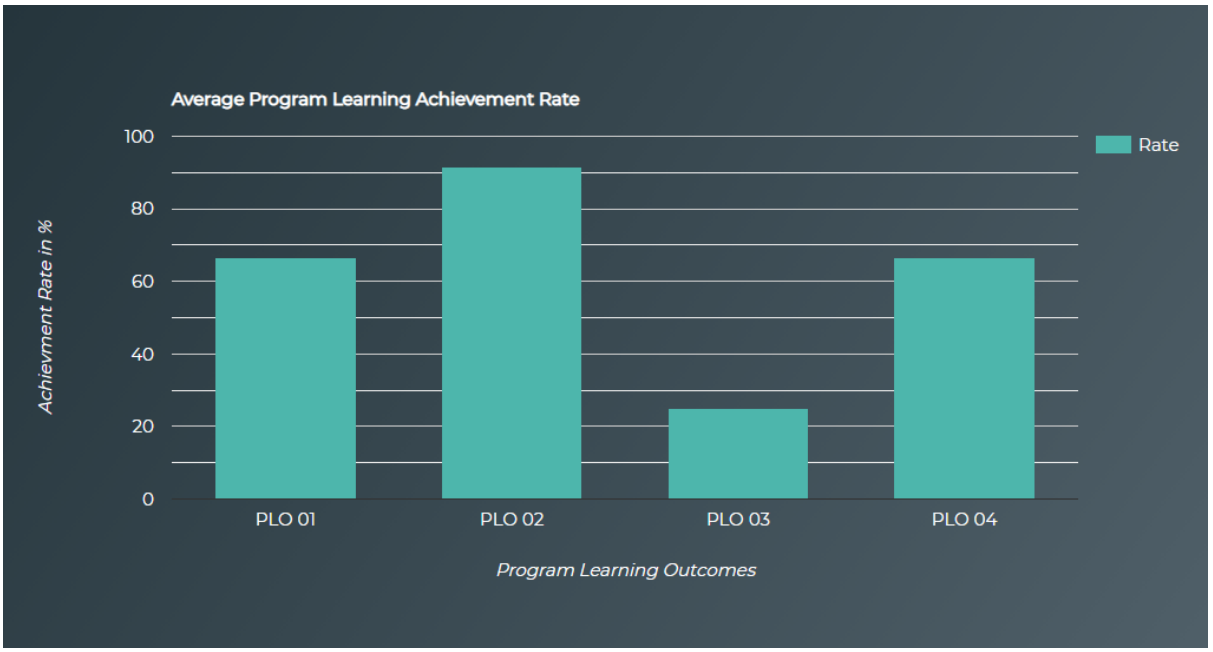
Faculty's historical Course-wise Average CO achievement rate



graphqueries.py

```
def getAverageCOEvaluationRate(facultyID, coID):
    sql_query = '''SELECT (SELECT
COUNT(DISTINCT(coevaluation_t.courseenrollment_id)) FROM
coevaluation_t, studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.faculty_id = {} AND coevaluation_t.co_id = "{}" AND
coevaluation_t.coAchievementStatus = 'Y')/(SELECT
COUNT(DISTINCT(coevaluation_t.courseenrollment_id)) FROM
coevaluation_t, studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.faculty_id = {} AND coevaluation_t.co_id = "{}");'''

    rate = 0.0
    with connection.cursor() as cursor:
        cursor.execute(sql_query.format(facultyID, coID, facultyID,
coID))
        row = cursor.fetchone()
        if row[0] != None:
            rate = round(float(row[0]), 3)*100
    return rate
```

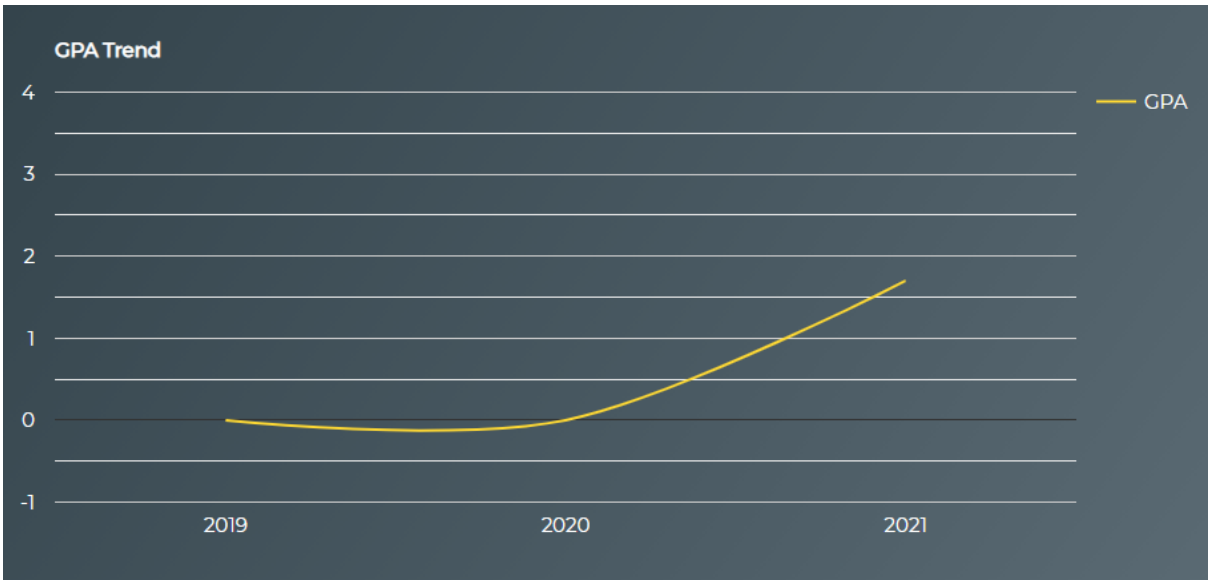
Faculty's historical Course-wise Average PLO achievement rate



graphqueries.py

```
def getAveragePLOEvaluationRate(facultyID, ploID):
    sql_query = '''SELECT (SELECT COUNT(*) FROM ploevaluation_t,
studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.faculty_id = {} AND ploevaluation_t.plo_id = "{}" AND
ploevaluation_t.ploAchievementStatus = 'Y')/(SELECT COUNT(*) FROM
ploevaluation_t, studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.faculty_id = {} AND ploevaluation_t.plo_id = "{}");'''

    rate = 0.0
    with connection.cursor() as cursor:
        cursor.execute(sql_query.format(facultyID, ploID, facultyID,
ploID))
        row = cursor.fetchone()
        if row[0] != None:
            rate = round(float(row[0]), 3)*100

    return rate
```

Faculty Historical Course-wise Yearly Average GPA

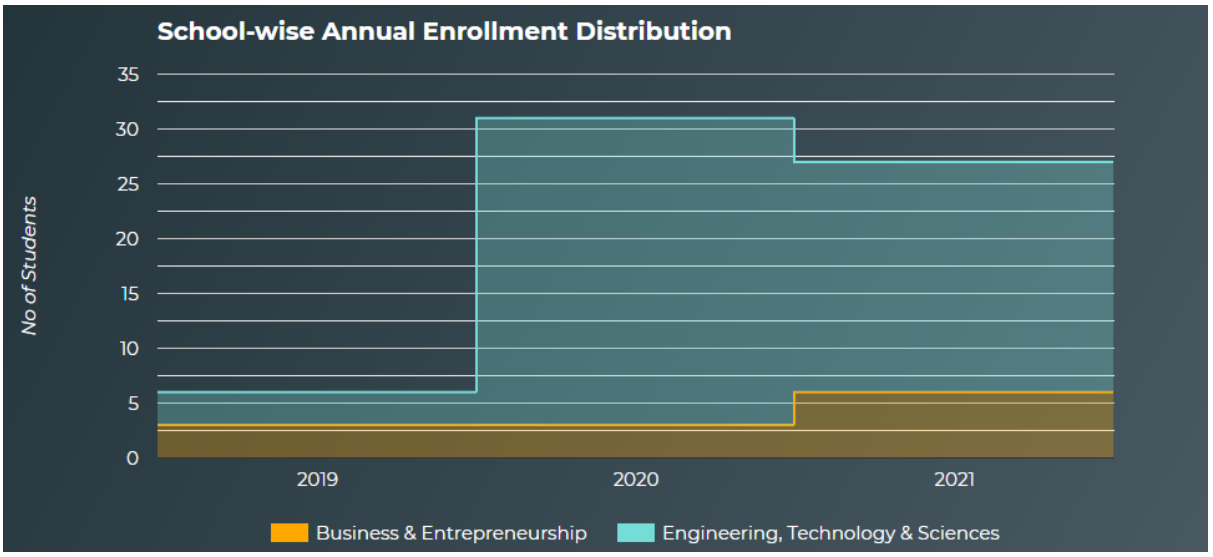


graphqueries.py

```
def getAverageGPA(facultyID, courseID):
    sql_query = '''SELECT (SELECT AVG(coevaluation_t.coMarksObtained)
FROM coevaluation_t, studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND section_t.faculty_id = {} AND
section_t.course_id = "{}")/(SELECT
AVG(coevaluation_t.coMarksAttainable) FROM coevaluation_t,
studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND section_t.faculty_id = {} AND
section_t.course_id = "{}")*100;
'''

    gpaList = []
    years = [2019, 2020, 2021]
    for year in years:
        buffer = []
        buffer.append(str(year))
        with connection.cursor() as cursor:
            cursor.execute(sql_query.format(year, facultyID,
                           courseID, year, facultyID, courseID))
            row = cursor.fetchone()
```

```
            if row[0] != None:
                if row[0] <= 49:
                    buffer.append(1.0)
                elif row[0] > 49 and row[0] <= 54:
                    buffer.append(1.3)
                elif row[0] > 54 and row[0] <= 59:
                    buffer.append(1.7)
                elif row[0] > 59 and row[0] <= 64:
                    buffer.append(2.0)
                elif row[0] > 64 and row[0] <= 69:
                    buffer.append(2.3)
                elif row[0] > 69 and row[0] <= 74:
                    buffer.append(2.7)
                elif row[0] > 74 and row[0] <= 79:
                    buffer.append(3.0)
                elif row[0] > 79 and row[0] <= 84:
                    buffer.append(3.3)
                elif row[0] > 84 and row[0] <= 89:
                    buffer.append(3.7)
                elif row[0] > 89 and row[0] <= 100:
                    buffer.append(4.0)
            else:
                buffer.append(0.0)
        gpaList.append(buffer)

    return gpaList
```

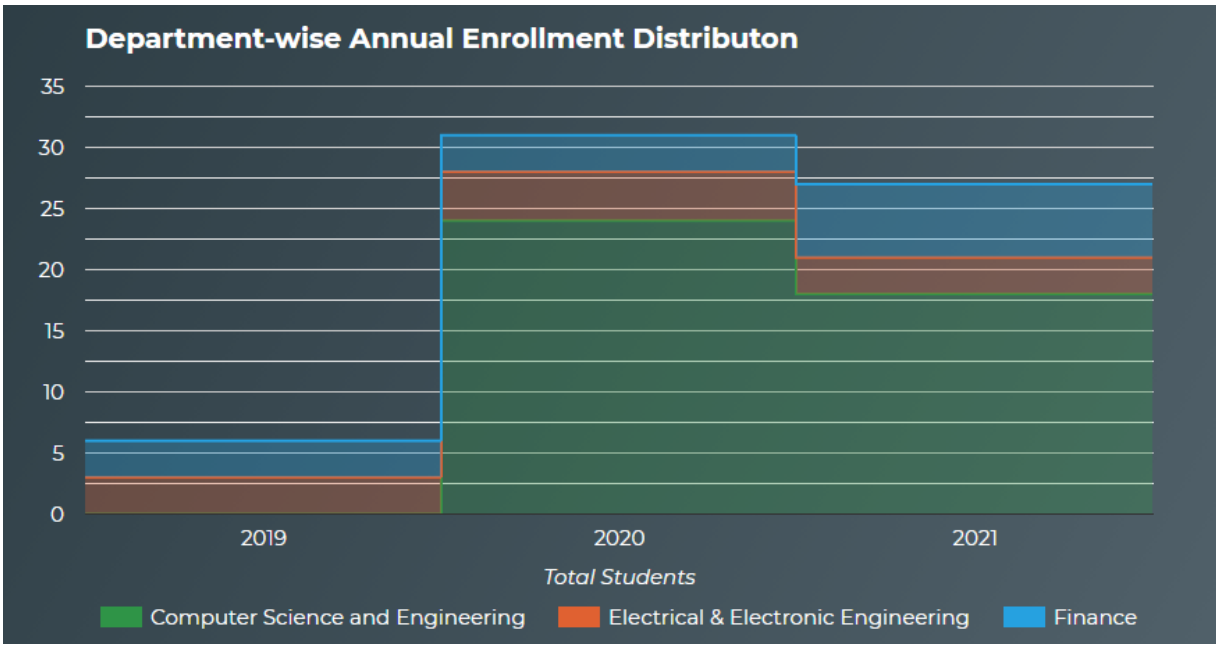
School-wise Annual Enrollment Distribution



graphqueries.py

```
def SchoolyearWiseStudentEnrollment(schoolList):
    sql_query = '''SELECT school_t.schoolName,
COUNT(DISTINCT(studentcourseenrollment_t.student_id)) FROM school_t,
studentcourseenrollment_t, section_t, course_t, department_t WHERE
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND course_t.department_id =
department_t.department_id AND department_t.school_id =
school_t.school_id AND section_t.year = {} AND school_t.schoolName =
"{}" GROUP BY school_t.school_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for school in schoolList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, school[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(row[1])
                else:
                    yearTbl.append(0)
        arrTable.append(yearTbl)

    return arrTable
```

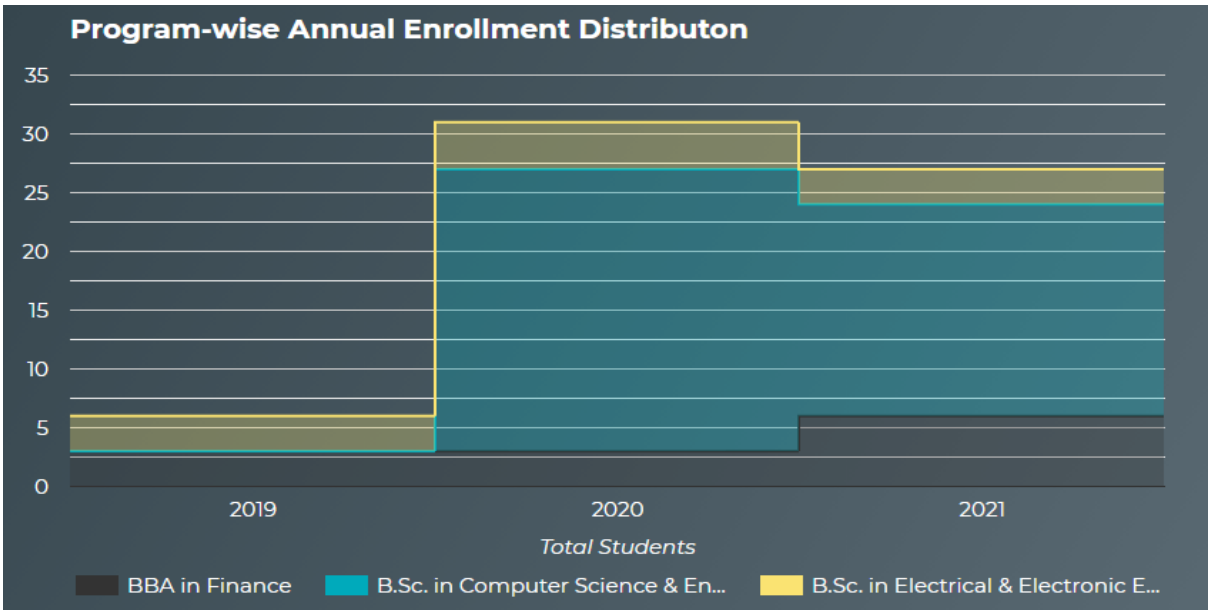
Department-wise Annual Enrollment Distribution



graphqueries.py

```
def DPTyearWiseStudentEnrollment(dptList):
    sql_query = '''SELECT department_t.deparmentName,
COUNT(DISTINCT(studentcourseenrollment_t.student_id)) FROM
studentcourseenrollment_t, section_t, course_t, department_t WHERE
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND course_t.department_id =
department_t.department_id AND section_t.year = {} AND
department_t.deparmentName = "{}" GROUP BY
department_t.department_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for dpt in dptList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, dpt[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(row[1])
                else:
                    yearTbl.append(0)
        arrTable.append(yearTbl)
    return arrTable
```

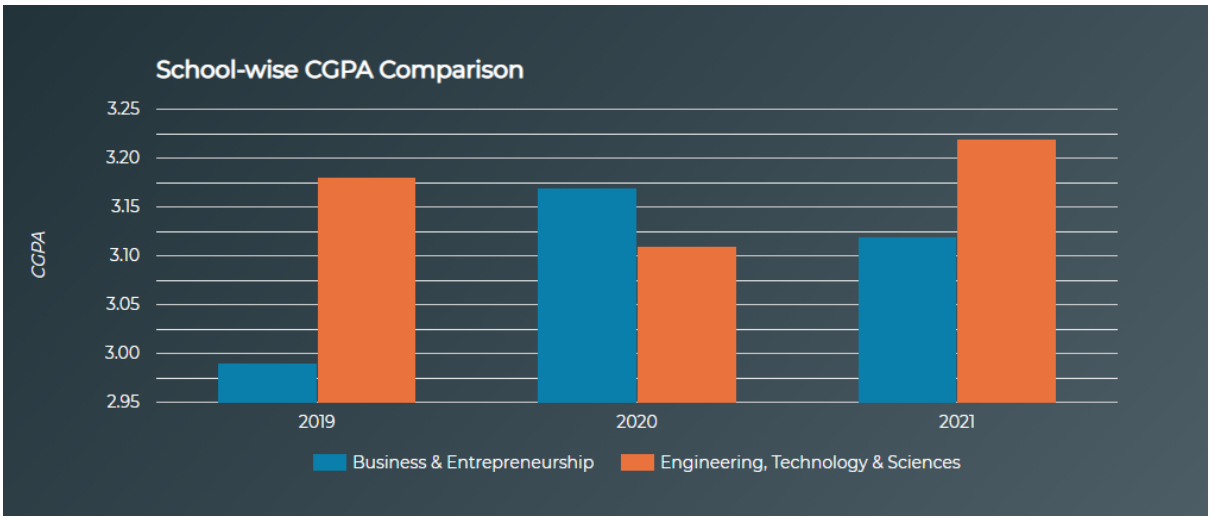
Program-wise Annual Enrollment Distribution



graphqueries.py

```
def ProgramYearWiseStudentEnrollment(programList):
    sql_query = '''SELECT program_t.programName,
COUNT(DISTINCT(studentcourseenrollment_t.student_id)) FROM
department_t, program_t, studentcourseenrollment_t, section_t,
programcoursemapping_t WHERE studentcourseenrollment_t.section_id =
section_t.section_id AND section_t.course_id =
programcoursemapping_t.course_id AND program_t.program_id =
programcoursemapping_t.program_id AND program_t.department_id=
department_t.department_id AND section_t.year={} AND
program_t.programName = "{}" GROUP BY program_t.program_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for program in programList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, program[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(row[1])
                else:
                    yearTbl.append(0)
        arrTable.append(yearTbl)
    return arrTable
```

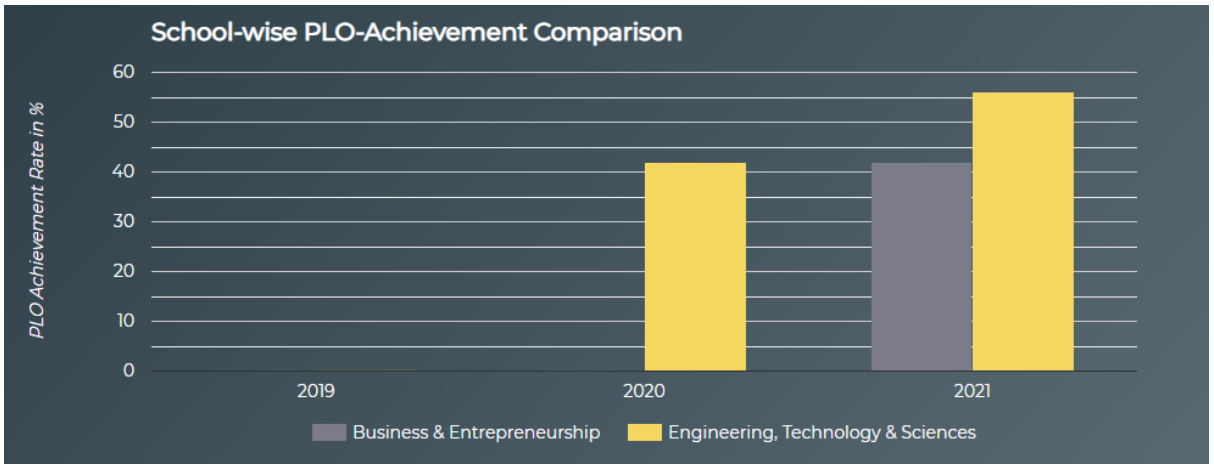
School-wise Annual CGPA Comparison



graphqueries.py

```
def getSchoolWiseAverageCGPA(schoolList):
    sql_query = '''SELECT school_t.schoolName, AVG(student_t.cgpa) FROM
student_t, studentcourseenrollment_t, section_t,
course_t,department_t,school_t WHERE student_t.student_id =
studentcourseenrollment_t.student_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND course_t.department_id =
department_t.department_id AND department_t.school_id =
school_t.school_id AND section_t.year = {} AND school_t.schoolName="{}"
GROUP BY school_t.school_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for school in schoolList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, school[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(round(float(row[1]), 2))
                else:
                    yearTbl.append(0.00)
        arrTable.append(yearTbl)

    return arrTable
```

School-wise Annual PLO-Achievement Rate Comparison

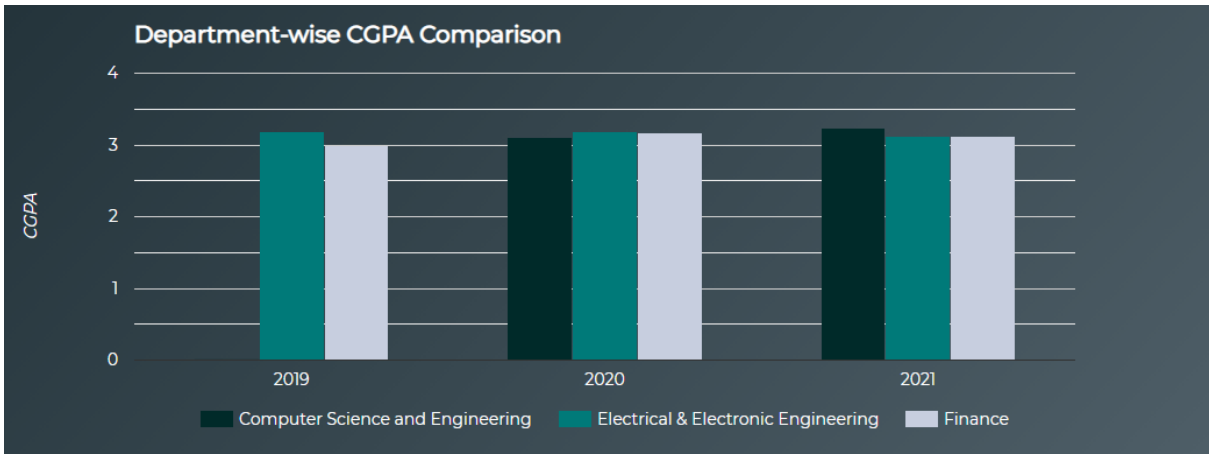


graphqueries.py

```
def getSchoolWisePLORate(schoolList):
    sql_query = '''SELECT school_t.schoolName,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
program_t, curriculum_t, plohistory_t, department_t, school_t,
studentcourseenrollment_t, section_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND ploevaluation_t.plo_id = plohistory_t.plo_id
AND plohistory_t.curriculum_id = curriculum_t.curriculum_id AND
(curriculum_t.expirationDate IS NULL OR curriculum_t.expirationDate =
'') AND curriculum_t.program_id = program_t.program_id AND
program_t.department_id = department_t.department_id AND
department_t.school_id = school_t.school_id AND school_t.schoolName =
"{}" GROUP BY school_t.school_id;'''
    sql_query2 = '''SELECT school_t.schoolName,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
program_t, curriculum_t, plohistory_t, department_t, school_t,
studentcourseenrollment_t, section_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND ploevaluation_t.plo_id = plohistory_t.plo_id
AND plohistory_t.curriculum_id = curriculum_t.curriculum_id AND
(curriculum_t.expirationDate IS NULL OR curriculum_t.expirationDate =
'') AND curriculum_t.program_id = program_t.program_id AND
program_t.department_id = department_t.department_id AND
department_t.school_id = school_t.school_id AND
```

```
ploevaluation_t.ploAchievementStatus = 'Y' AND school_t.schoolName="{}"
GROUP BY school_t.school_id;'''

    years = [2019, 2020, 2021]
    arrTable = []

    for year in years:
        yearTbl = []
        yearTbl.append(str(year))

        for school in schoolList:
            attempted = 0
            achieved = 0
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, school[0]))
                row = cursor.fetchone()
                if row != None:
                    attempted = row[1]
                else:
                    yearTbl.append(0.0)
                    continue
            with connection.cursor() as cursor:
                cursor.execute(sql_query2.format(year, school[0]))
                row = cursor.fetchone()
                if row != None:
                    achieved = row[1]
                else:
                    yearTbl.append(0.0)
                    continue
            yearTbl.append(round(float(achieved/attempted), 2)*100)

        arrTable.append(yearTbl)

    return arrTable
```

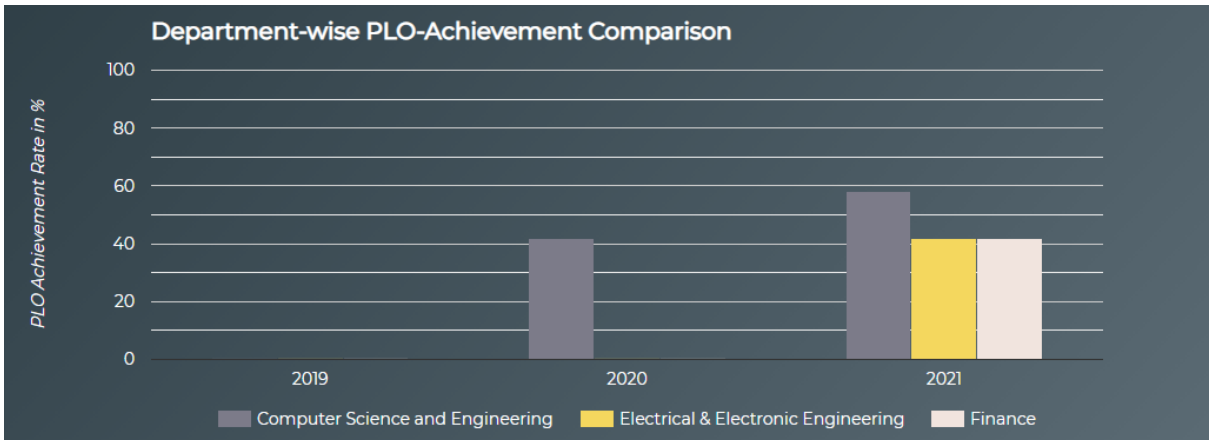
Department-wise Annual CGPA Comparison



graphqueries.py

```
def getDepartmentWiseAverageCGPA(dptList):
    sql_query = '''SELECT department_t.deparmentName,
AVG(student_t.cgpa) FROM student_t, studentcourseenrollment_t,
section_t, course_t, department_t WHERE student_t.student_id =
studentcourseenrollment_t.student_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND course_t.department_id =
department_t.department_id AND section_t.year = {} AND
department_t.deparmentName = "{}" GROUP BY
department_t.department_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for dpt in dptList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, dpt[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(round(float(row[1]), 2))
                else:
                    yearTbl.append(0.00)
        arrTable.append(yearTbl)

    return arrTable
```

Department-wise Annual PLO-Achievement Rate Comparison

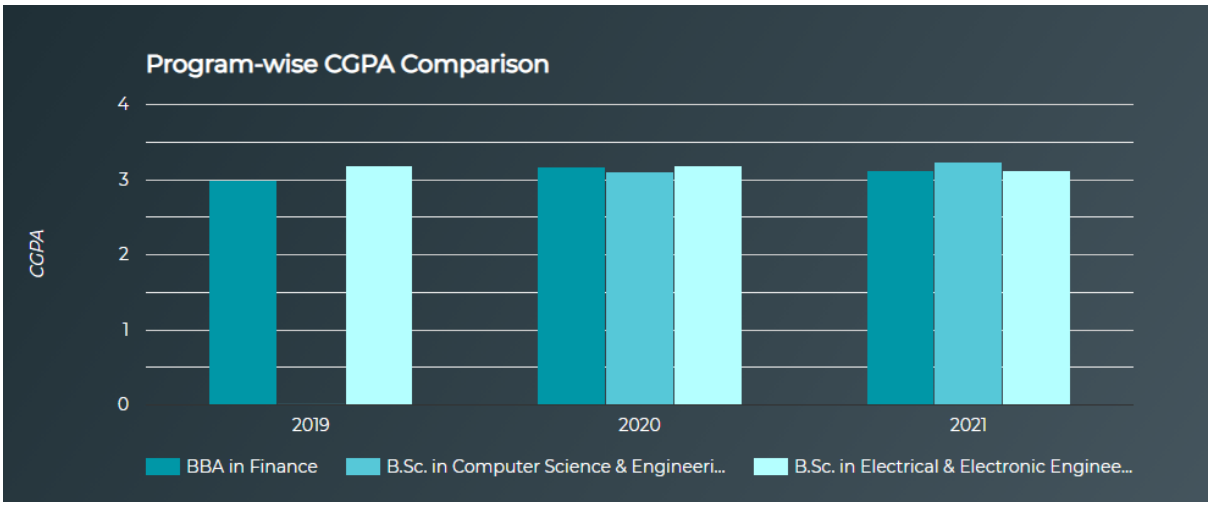


graphqueries.py

```
def getDeparmentWisePLORate(dptList):
    sql_query = '''SELECT department_t.department_id,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
program_t, curriculum_t, plohistory_t, department_t,
studentcourseenrollment_t, section_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
ploevaluation_t.plo_id = plohistory_t.plo_id AND
plohistory_t.curriculum_id = curriculum_t.curriculum_id AND
(curriculum_t.expirationDate IS NULL OR curriculum_t.expirationDate =
'') AND curriculum_t.program_id = program_t.program_id AND
program_t.department_id = department_t.department_id AND section_t.year
= {} AND department_t.deparmentName = "{}" GROUP BY
department_t.department_id;'''
    sql_query2 = '''SELECT department_t.department_id,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
program_t, curriculum_t, plohistory_t, department_t,
studentcourseenrollment_t, section_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
ploevaluation_t.plo_id = plohistory_t.plo_id AND
plohistory_t.curriculum_id = curriculum_t.curriculum_id AND
(curriculum_t.expirationDate IS NULL OR curriculum_t.expirationDate =
'') AND curriculum_t.program_id = program_t.program_id AND
program_t.department_id = department_t.department_id AND section_t.year
= {} AND department_t.deparmentName = "{}" AND
```

```
ploevaluation_t.ploAchievementStatus = 'Y' GROUP BY
department_t.department_id;'''

    years = [2019, 2020, 2021]
    arrTable = []

    for year in years:
        yearTbl = []
        yearTbl.append(str(year))

        for dpt in dptList:
            attempted = 0
            achieved = 0
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, dpt[0]))
                row = cursor.fetchone()
                if row != None:
                    attempted = row[1]
                else:
                    yearTbl.append(0.0)
                    continue
            with connection.cursor() as cursor:
                cursor.execute(sql_query2.format(year, dpt[0]))
                row = cursor.fetchone()
                if row != None:
                    achieved = row[1]
                else:
                    yearTbl.append(0.0)
                    continue
            yearTbl.append(round(float(achieved/attempted), 2)*100)

        arrTable.append(yearTbl)

    return arrTable
```

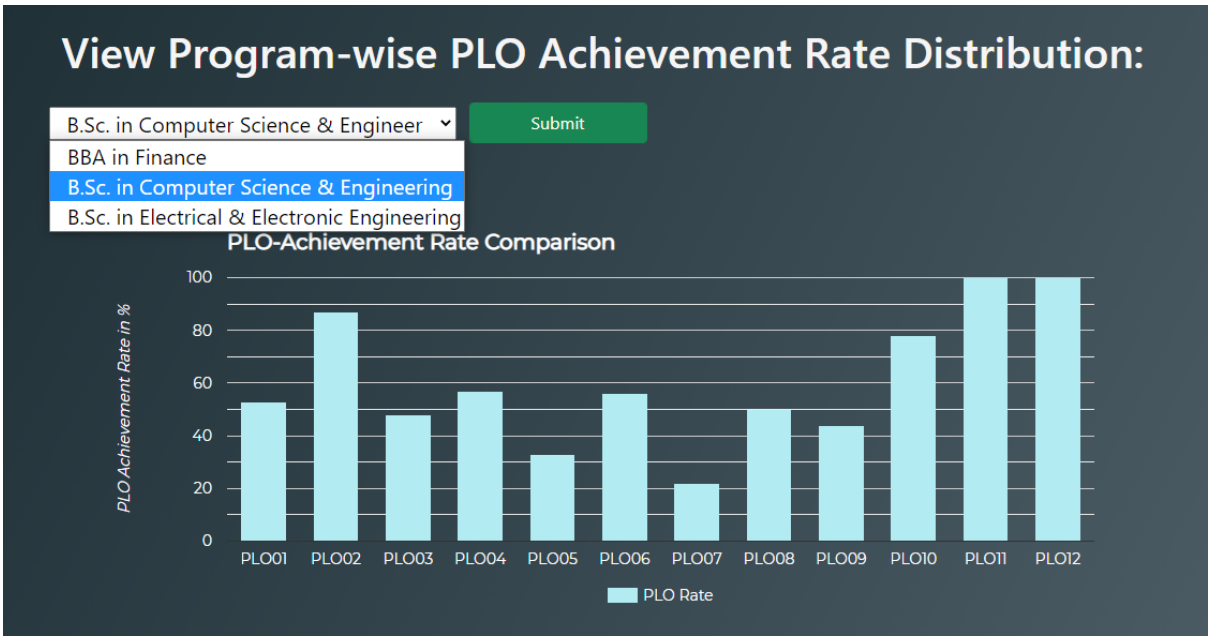
Program-wise Annual CGPA Comparison



graphqueries.py

```
def getProgramWiseCGPA(programList):
    sql_query = '''SELECT program_t.programName, AVG(student_t.cgpa)
FROM program_t, studentcourseenrollment_t, section_t,
programcoursemapping_t, student_t WHERE student_t.student_id =
studentcourseenrollment_t.student_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = programcoursemapping_t.course_id AND
program_t.program_id = programcoursemapping_t.program_id AND
section_t.year = {} AND program_t.programName = "{}" GROUP BY
program_t.program_id;'''
    years = [2019, 2020, 2021]
    arrTable = []
    for year in years:
        yearTbl = []
        yearTbl.append(str(year))
        for program in programList:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(year, program[0]))
                row = cursor.fetchone()
                if row != None:
                    yearTbl.append(round(float(row[1]), 2))
                else:
                    yearTbl.append(0.00)
        arrTable.append(yearTbl)
    return arrTable
```

Program-wise Individual PLO Average Achievement Rate

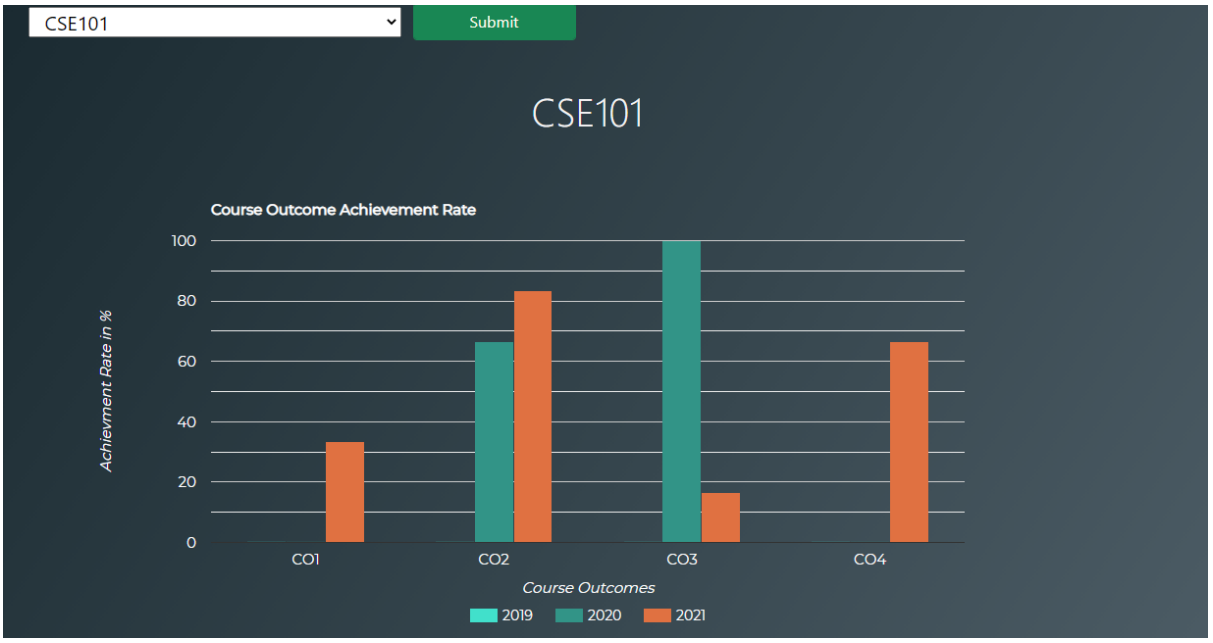


graphqueries.py

```
def getProgramWisePLORate(programName):
    sql_query1 = '''SELECT ploevaluation_t.plo_id,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
plohistory_t, curriculum_t, program_t WHERE ploevaluation_t.plo_id =
plohistory_t.plo_id AND plohistory_t.curriculum_id =
curriculum_t.curriculum_id AND (curriculum_t.expirationDate IS NULL or
curriculum_t.expirationDate = "") AND curriculum_t.program_id =
program_t.program_id AND program_t.programName = "{}" GROUP BY
plo_id;'''
    sql_query2 = '''SELECT ploevaluation_t.plo_id,
COUNT(ploevaluation_t.courseenrollment_id) FROM ploevaluation_t,
plohistory_t, curriculum_t, program_t WHERE ploevaluation_t.plo_id =
plohistory_t.plo_id AND plohistory_t.curriculum_id =
curriculum_t.curriculum_id AND (curriculum_t.expirationDate IS NULL or
curriculum_t.expirationDate = "") AND curriculum_t.program_id =
program_t.program_id AND program_t.programName = "{}" AND
ploevaluation_t.ploAchievementStatus = "Y" GROUP BY plo_id;'''

    with connection.cursor() as cursor:
        cursor.execute(sql_query1.format(programName))
        attempted = cursor.fetchall()

    with connection.cursor() as cursor:
        cursor.execute(sql_query2.format(programName))
        achieved = cursor.fetchall()
```

```
    dataTable = []
    dict = {}
    for i in range(len(attempted)):
        data = []
        PLOrate = 0.0
        programName = attempted[i][0]
        dict[programName] = False
        for j in range(len(achieved)):
            if attempted[i][0] == achieved[j][0]:
                PLOrate = round(float(achieved[j][1]/attempted[i][1]),
2)
                dict[programName] = True
                programName = achieved[j][0]
                break

        if dict[programName] == False:
            programName = attempted[i][0]
            data.append(
                "PLO" +
programName[len(programName)-2:len(programName)])
            data.append(0)
        else:
            data.append(
                "PLO" +
programName[len(programName)-2:len(programName)])
            data.append(PLOrate*100)
        dataTable.append(data)

    return dataTable
```

Course-wise CO Average Achievement Rate

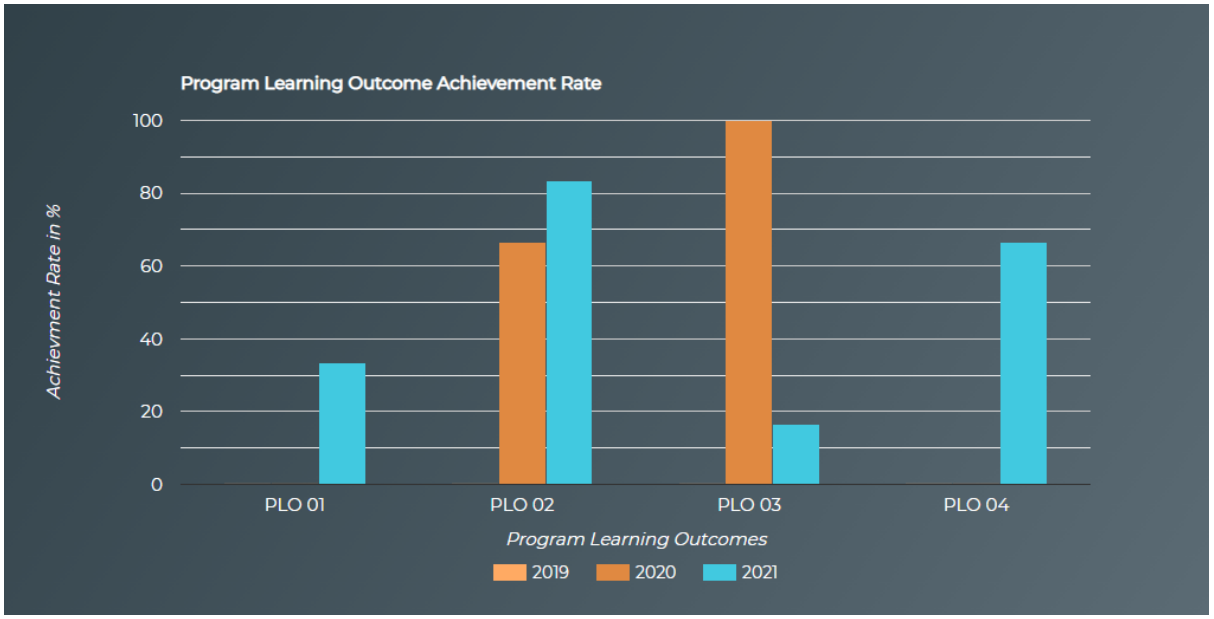


graphqueries.py

```
def getCourseWiseCORate(courseID, coList):
    sql_query = '''SELECT (SELECT COUNT(*) FROM coevaluation_t,
studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND section_t.course_id = "{}" AND
coevaluation_t.co_id = "{}" AND coevaluation_t.coAchievementStatus =
"Y" GROUP BY coevaluation_t.co_id)/(SELECT COUNT(*) FROM
coevaluation_t, studentcourseenrollment_t, section_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
coevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.year = {} AND section_t.course_id = "{}" AND
coevaluation_t.co_id = "{}" GROUP BY coevaluation_t.co_id);'''
    years = [2019, 2020, 2021]
    arrTable = []

    for coID in coList:
        yearTbl = []
        coName = coID[len(coID)-3:len(coID)]
        yearTbl.append(coName)
        for year in years:
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(
```

```
                    year, courseID, coID, year, courseID, coID))
                row = cursor.fetchone()
                if row[0] == None:
                    yearTbl.append(0.0)
                else:
                    yearTbl.append(round(float(row[0]*100), 2))
        arrTable.append(yearTbl)

    return arrTable
```

Course-wise PLO Average Achievement Rate

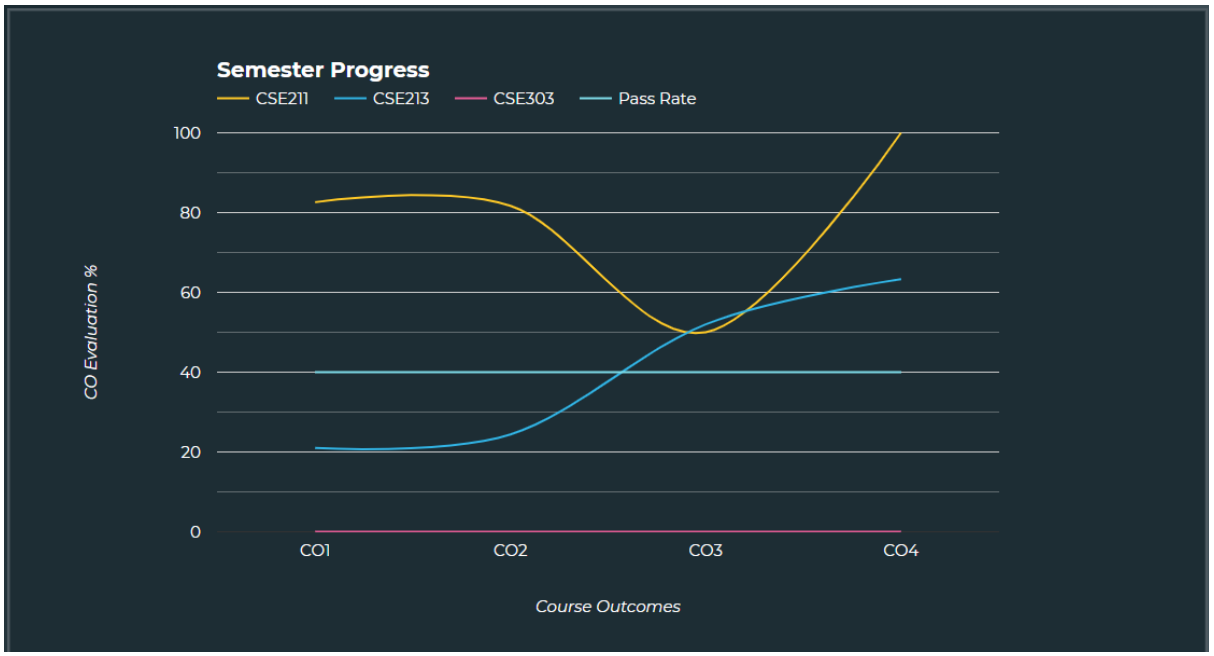


graphqueries.py

```
def getCourseWisePLORate(courseID, ploList):
    sql_query = '''SELECT (SELECT COUNT(*) FROM ploevaluation_t,
studentcourseenrollment_t, section_t, course_t WHERE
studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND
ploevaluation_t.ploAchievementStatus = "Y" AND section_t.year = {} AND
course_t.course_id = "{}" AND ploevaluation_t.plo_id = "{}")/(SELECT
COUNT(*) FROM ploevaluation_t, studentcourseenrollment_t, section_t,
course_t WHERE studentcourseenrollment_t.courseEnrollment_id =
ploevaluation_t.courseenrollment_id AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND section_t.year = {} AND
course_t.course_id = "{}" AND ploevaluation_t.plo_id = "{}");'''
    years = [2019, 2020, 2021]
    arrTable = []

    for ploID in ploList:
        yearTbl = []
        ploName = "PLO " + ploID[len(ploID)-2:len(ploID)]
        yearTbl.append(ploName)
        for year in years:
            with connection.cursor() as cursor:
```

```
                cursor.execute(sql_query.format(
                    year, courseID, ploID, year, courseID, ploID))
                row = cursor.fetchone()
                if row[0] == None:
                    yearTbl.append(0.0)
                else:
                    yearTbl.append(round(float((row[0])*100), 2))
        arrTable.append(yearTbl)

    return arrTable
```

Student Current Semester Progress Chart

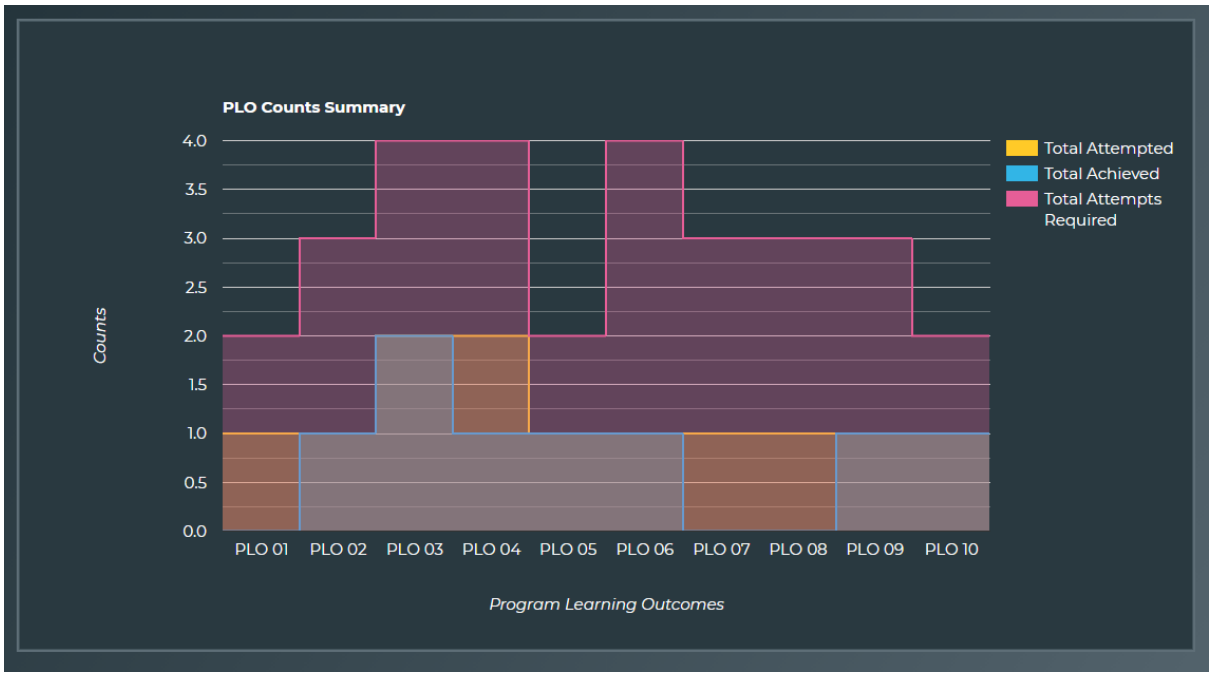


graphqueries.py

```
def getStudentSemProgress(courseIDs, studentID):
    sql_query = '''SELECT coevaluation_t.co_id,
ROUND((coevaluation_t.coMarksObtained /
coevaluation_t.coMarksAttainable)*100,1) FROM coevaluation_t, co_t,
studentcourseenrollment_t WHERE coevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.student_id = "{}" AND coevaluation_t.co_id =
co_t.co_id AND co_t.course_id = "{}" GROUP BY coevaluation_t.co_id;'''

    dataTable = [[], [], [], [], []]
    dataTable[0].append("COs")
    for i in range(len(dataTable)-1):
        dataTable[i+1].append("CO"+str(i+1))
        for courseID in courseIDs:
            if i == 0:

dataTable[0].append(courseID[(len(courseID)-6):len(courseID)])
            with connection.cursor() as cursor:
                cursor.execute(sql_query.format(studentID, courseID))
                rows = cursor.fetchall()
                if not rows:
                    dataTable[i+1].append(0.0)
                else:
                    dataTable[i+1].append(rows[i][1])
```

```
    dataTable[0].append("Pass Rate")
    for i in range(len(dataTable)):
        if i != 0:
            dataTable[i].append(40.0)

    return dataTable
```

Student Course-wise PLO Summary Table

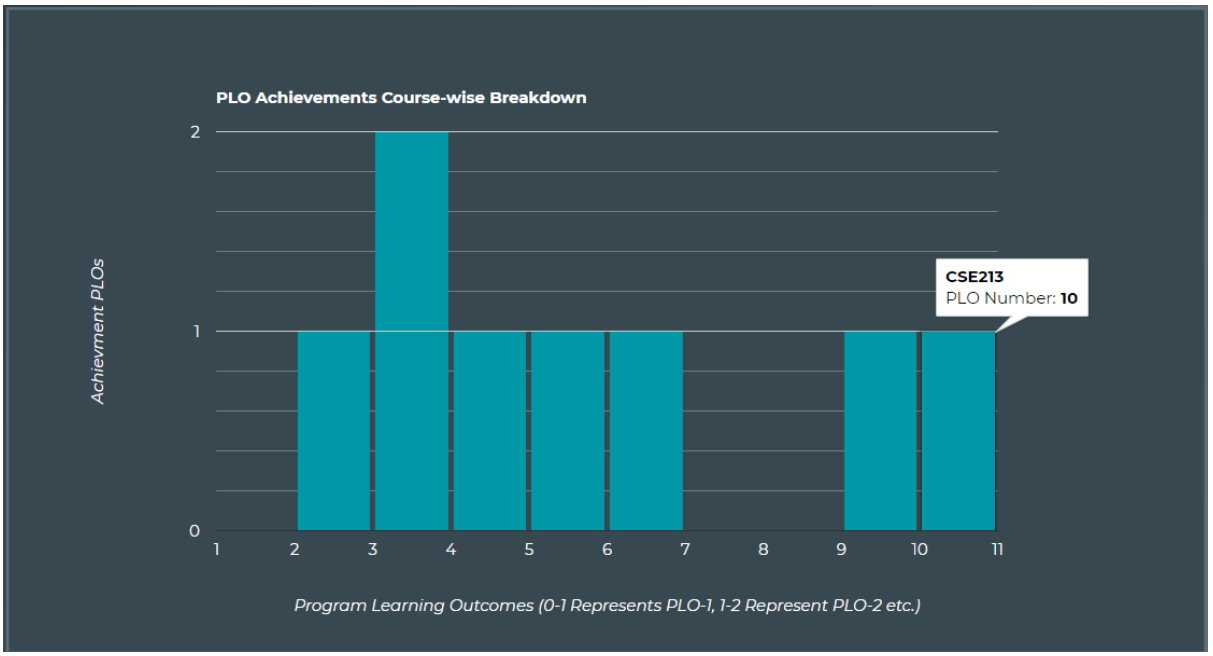


graphqueries.py

```
def getStudentPLOStats(studentID):

    sql_query1 = '''SELECT ploevaluation_t.plo_id,
COUNT(ploevaluation_t.plo_id) FROM ploevaluation_t,
studentcourseenrollment_t, section_t, course_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.student_id = "{}" AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id GROUP BY
ploevaluation_t.plo_id;'''
    sql_query2 = '''SELECT COUNT(ploevaluation_t.plo_id) FROM
ploevaluation_t, studentcourseenrollment_t, section_t, course_t WHERE
ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.student_id = "{}" AND
ploevaluation_t.ploAchievementStatus = "Y" AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id AND ploevaluation_t.plo_id =
"{}";
'''
    sql_query3 = '''SELECT COUNT(co_t.plo_id) FROM co_t WHERE
co_t.plo_id = "{}";'''
    dataTable = []
    with connection.cursor() as cursor:
```

```
        cursor.execute(sql_query1.format(studentID))
        rows = cursor.fetchall()
        for row in rows:
            data = []
            ploID = "PLO " + row[0][(len(row[0])-2):len(row[0])]
            data.append(ploID)
            data.append(row[1])
            cursor.execute(sql_query2.format(studentID, row[0]))
            achieved = cursor.fetchone()
            data.append(achieved[0])
            cursor.execute(sql_query3.format(row[0]))
            attemptRequired = cursor.fetchone()
            data.append(attemptRequired[0])
            dataTable.append(data)

    return dataTable
```

Student Course-wise PLO Achievements Course-wise breakdown



graphqueries.py

```
def getStudentPLOcourseWiseHistory(studentID):
    sql_query = '''SELECT course_t.course_id, ploevaluation_t.plo_id
FROM ploevaluation_t, studentcourseenrollment_t, section_t, course_t
WHERE ploevaluation_t.courseenrollment_id =
studentcourseenrollment_t.courseEnrollment_id AND
studentcourseenrollment_t.student_id = "{}" AND
ploevaluation_t.ploAchievementStatus = "Y" AND
studentcourseenrollment_t.section_id = section_t.section_id AND
section_t.course_id = course_t.course_id;'''
    dataTable = []
    with connection.cursor() as cursor:
        cursor.execute(sql_query.format(studentID))

        for row in cursor.fetchall():
            data = []
            courseName = row[0][(len(row[0])-6):len(row[0])]
            data.append(courseName)
            ploID = row[1][(len(row[1])-2):len(row[1])]
            ploID = int(ploID)
            data.append(ploID)
            dataTable.append(data)

    return dataTable
```

Conclusion:

Problems and Solution:

1. **Problem:** CGPA calculations don't update based on current semester marks input
   Solution: We can easily add in this feature as the database allows storage of the necessary data

2. **Problem:** User-interface isn't necessarily intuitive due to the limited timeframe
   Solution: we hope to improve on this on next iterations given more time

Additional Features and Future Developments:

1. The database is design to scaled for individual universities there each university can have its separate portal in the SPM software we've developed

2. Users will also be expanded to add administration and different managerial positions

3. Attendance Statistical analysis can be added

4. Data-entry for program and courses, can even allow for proper Program-course mapping leading to additional features

Conclusion and recommendations:

We are optimistic that the capabilities of this SPM software have potential to help organize a strong foundation for universities and other educational institutions alike especially with the surplus of data, information and student enrollments globally. Not to mention the current pandemic situation urges our civilization to move forward into a more virtual activity-based world. Whether that is good or bad thing long term is still at question however, what we can tell for sure is that this software will shape this virtual based education. We believe that a great future holds for such a software.